angr

The angr Project

Jun 04, 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 Project . . . 223
10.2 Plugin Ecosystem . . . 235
10.3 Program State . . . 238
10.4 Storage . . . 423
10.5 Memory Mixins . . . 456
10.6 Concretization Strategies . . . 541
10.7 Simulation Manager . . . 544
10.8 Exploration Techniques . . . 552
10.9 Simulation Engines . . . 587
10.10 Simulation Logging . . . 638
10.11 Procedures . . . 642
10.12 Calling Conventions and Types . . . 658
10.13 Knowledge Base . . . 711
10.14 Serialization . . . 870
10.15 Analysis . . . 873
10.16 SimOS . . . 1325

10.17 Function Signature Matching . . . 1342
10.18 Utils . . . 1344
10.19 Errors . . . 1355
10.20 Distributed analysis . . . 1361
10.21 angr Intermediate Language . . . 1364

11 Indices and tables **1411**

Python Module Index **1413**

Index **1419**

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 on 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` (this is the default), 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`, 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.:raw-html-m2r:<*br*> 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. TODO: link

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 <./examples.md#reverseme-modern-binary-exploitation—csci-4968>*:

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("State %s 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`!

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)` *
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)

>>> state = angr.SimState(arch='AMD64')
>>> 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 Widening

There is another kind of merging called *widening* which takes several states and produces a more general state. It is used during static analysis.

Todo

Explain what this means

6.2.5 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.

6.2.6 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)
```

6.2.7 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_UNCONST	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**(*when=When.BEFORE*, *enabled=True*, *condition=None*, *action=<function BP_IPDB>*, ***kwargs*)

Bases: object

A breakpoint.

Parameters

- **when** (When)
- **enabled** (*bool*)
- **condition** (Callable[[SimState], bool] | None)
- **action** (Action)
- **kwargs** (*dict[str, Any]*)

__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.**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**(*missing_type*)

Bases: *AngrTypeError*

Parameters
missing_type (*str*)

__init__(*missing_type*)

Parameters
missing_type (*str*)

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**(*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*)

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.

Parameters

- **graph** (*networkx.DiGraph*)
- **dst_run** (*int*)
- **dst_stmt_idx** (*int*)
- **direction** (*str*)
- **ignore_sp** (*bool*)
- **ignore_bp** (*bool*)
- **max_level** (*int*)
- **stop_at_calls** (*bool*)
- **max_predecessors** (*int*)
- **include_imarks** (*bool*)
- **control_dependence** (*bool*)

__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**(*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*)

Bases: *Serializable*

Represents a basic block in a binary or a program.

Parameters
arch (*Arch | None*)

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

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

class angr.**Emulator**(*engine*, *init_state*)

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.

Parameters

- **engine** (SuccessorsEngine)
- **init_state** (SimState)

__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**(**values*)

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**(*project*, *obj=None*, *name=None*)

Bases: `object`

Represents a "model" of knowledge about an artifact.

Contains things like a CFG, data references, etc.

functions: *FunctionManager*

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**(*model*, *api_key=None*, *api_base=None*, *max_tokens=4096*, *temperature=0.0*)

Bases: `object`

A client for interacting with LLMs via pydantic-ai. Used by the decompiler to suggest improved variable names, function names, and types.

Parameters

- **model** (*str*)
- **api_key** (*str | None*)
- **api_base** (*str | None*)
- **max_tokens** (*int*)
- **temperature** (*float*)

__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**(*base*, *sim_state*, *heap=None*)

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*)

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*, *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).

is_free()

Returns a concrete determination as to whether the chunk is free.

data_ptr()

Returns the address of the payload of the chunk.

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

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. 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

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

exception angr.**PathUnreachableError**

Bases: *AngrPathError*

class angr.**PointerWrapper**(*value*, *buffer=False*)

Bases: object

__init__(*value*, *buffer=False*)

class angr.**Project**(*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*, *concrete_target=None*, *eager_ifunc_resolution=None*, *cache_limits=None*, *rustc_version=None*, *rustc_optimization_level=None*, ***kwargs*)

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 retranslating 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).
- **load_options** (*dict[str, Any] | None*)
- **support_selfmodifying_code** (*bool | None*)
- **cache_limits** (*dict[str, int | None] | None*)

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.

Parameters

- **arch** (*Arch*)
- **load_options** (*dict[str, Any] | None*)
- **selfmodifying_code** (*bool*)
- **support_selfmodifying_code** (*bool | None*)
- **cache_limits** (*dict[str, int | None] | None*)

__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*, *concrete_target=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().

languages()

Return type
list[str]

property is_rust_binary: bool

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(*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*)

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**(*arch*)

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.

Parameters
arch (*archinfo.Arch*)

__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

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(*cc*)

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*)

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.
- **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**(*name=None*, *content=None*, *size=None*, *has_end=None*, *seekable=True*, *writable=True*, *ident=None*, *concrete=None*, ***kwargs*)

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

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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*, *events=True*, ***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.

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.

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.**SimFileBase**(*name=None*, *writable=True*, *ident=None*, *concrete=False*, *file_exists=True*, ***kwargs*)

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.

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.

class angr.**SimFileDescriptor**(*simfile*, *flags=0*)

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*)

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

eof()

Return the EOF status. May be a symbolic boolean.

tell()

Return the current position, or None if the concept doesn't make sense for the given file.

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.

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.

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.**SimFileDescriptorDuplex**(*read_file*, *write_file*)

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*)

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

set_state(*state*)

Sets a new state (for example, if the state has been branched)

eof()

Return the EOF status. May be a symbolic boolean.

tell()

Return the current position, or None if the concept doesn't make sense for the given file.

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

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.

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.

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.

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

exception angr.**SimFileError**

Bases: *SimMemoryError*, *SimFilesystemError*

class angr.**SimFileStream**(*name=None*, *content=None*, *pos=0*, ***kwargs*)

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*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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(*_*, *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.

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.

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

exception angr.**SimFilesystemError**

Bases: *SimError*

class angr.**SimHeapBrk**(*heap_base=None*, *heap_size=None*)

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*)

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.

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.

Parameters
sim_size – a size specifying how much to decrease the break pointer by (may be symbolic or not)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

exception angr.**SimHeapError**

Bases: *SimStateError*

class angr.**SimHeapPTMalloc**(*heap_base=None*, *heap_size=None*)

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*)

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.

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 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

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

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

init_state()

Use this function to perform any initialization on the state at plugin-add time

class angr.**SimHostFilesystem**(*host_path=None*, ***kwargs*)

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*)

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.

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**(*missing_addr*, *missing_size*, **args*)

Bases: *SimMemoryError*

__init__(*missing_addr*, *missing_size*, **args*)

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**(*project*, *name=None*)

Bases: object

A class describing OS/arch-level configuration.

Parameters

- **project** (angr.Project)
- **name** (*str | None*)

__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**(*name*, *write_mode=None*, *content=None*, *writable=True*, *ident=None*, ***kwargs*)

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*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

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.

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.**SimPacketsStream**(*name*, *pos=0*, ***kwargs*)

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*)

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(_, *data*, *size=None*, ***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

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.

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

exception angr.**SimPosixError**

Bases: *SimStateError*

class angr.**SimProcedure**(*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*)

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
- **project** (`angr.Project` | *None*)
- **prototype** (*str* | `SimTypeFunction` | *None*)

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.

Parameters

- **project** (`angr.Project` | *None*)
- **cc** (`angr.SimCC` | *None*)
- **prototype** (*str* | `SimTypeFunction` | *None*)

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`

Parameters

- **project** (`angr.Project` | *None*)
- **cc** (`angr.SimCC` | *None*)
- **prototype** (*str* | `SimTypeFunction` | *None*)

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** (`None` | *ArgSession* | `int`) – 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.

Parameters

- **project** (angr.Project | *None*)
- **cc** (angr.SimCC | *None*)
- **prototype** (*str* | SimTypeFunction | *None*)

__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**(*state*)

Bases: *SimEngineError*

__init__(*state*)

angr.**SimSegfaultError**

alias of *SimSegfaultException*

exception angr.**SimSegfaultException**(*addr*, *reason*, *original_addr=None*)

Bases: *SimException*, *SimMemoryError*

__init__(*addr*, *reason*, *original_addr=None*)

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**(*project=None*, *arch=None*, *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*)

Bases: *PluginHub*[*SimStatePlugin*], Generic

The SimState represents the state of a program, including its memory, registers, and so forth.

Parameters

- **project** (*Project* | None) – The project instance.
- **arch** (Arch | None) – The architecture of the state.
- **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*)

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*

registers: *DefaultMemory*

regs: *SimRegNameView*

memory: *DefaultMemory*

callstack: *CallStack*

mem: *SimMemView*

history: *SimStateHistory*

inspect: *SimInspector*

jni_references: *SimStateJNIReferences*

scratch: *SimStateScratch*

__init__(*project=None*, *arch=None*, *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 | None*)
- **arch** (*Arch | None*)
- **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

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.

has_plugin(*name*)

Return whether or not a plugin with the name `name` is currently active.

register_plugin(*name*, *plugin*, *inhibit_init=False*)

Add a new plugin `plugin` with name `name` to the active plugins.

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)

widen(**others*)

Perform a widening between self and other states :type others: :param others: :return:

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

property with_condition

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.

STRONGREF_STATE = False

__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

set_strongref_state(*state*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others (Iterable[*SimStatePlugin*]) – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

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**(*expr_type*, *expr*)

Bases: *SimExpressionError*

__init__(*expr_type*, *expr*)

exception angr.**SimUnsatError**

Bases: *SimValueError*

exception angr.**SimUnsupportedError**

Bases: *SimError*

exception angr.**SimValueError**

Bases: *SimSolverError*

exception angr.**SimZeroDivisionException**

Bases: *SimException*, *SimOperationError*

class angr.**SimulationManager**(*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*)

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*)

10.1 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**(*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*, *concrete_target=None*, *eager_ifunc_resolution=None*, *cache_limits=None*, *rustc_version=None*, *rustc_optimization_level=None*, ***kwargs*)

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).
- **load_options** (*dict[str, Any] | None*)
- **support_selfmodifying_code** (*bool | None*)
- **cache_limits** (*dict[str, int | None] | None*)

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.

Parameters

- **arch** (*Arch*)
- **load_options** (*dict[str, Any] | None*)
- **selfmodifying_code** (*bool*)
- **support_selfmodifying_code** (*bool | None*)
- **cache_limits** (*dict[str, int | None] | None*)

__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*, *concrete_target=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().

languages()

Return type
`list[str]`

property is_rust_binary: bool

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.factory.**AngrObjectFactory**(*project*, *default_engine=None*)

Bases: `object`

This factory provides access to important analysis elements.

Parameters

- **project** (Project)
- **default_engine** (type[SimEngine] | None)

__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*)

class angr.block.**DisassemblerBlock**(*addr*, *insns*, *thumb*, *arch*)

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**(*addr*, *insns*, *thumb*, *arch*)

Bases: *DisassemblerBlock*

Deep copy of the capstone blocks, which have serious issues with having extended lifespans outside of capstone itself

class angr.block.**CapstoneInsn**(*capstone_insn*)

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**(*addr*, *insns*, *thumb*, *arch*)

Bases: *DisassemblerBlock*

Deep copy of the pcode blocks

class angr.block.**PCodeInsn**(*pcode_insn*)

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**(*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*)

Bases: *Serializable*

Represents a basic block in a binary or a program.

Parameters

arch (*Arch | None*)

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

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

class angr.block.**SootBlock**(*addr*, ** (Keyword-only parameters separator (PEP 3102))*, *project*, *arch*)

Bases: object

Represents a Soot IR basic block.

Parameters

- **project** (Project)
- **arch** (*Arch*)

__init__(*addr*, ***, *project*, *arch*)

Parameters

- **project** (Project)
- **arch** (*Arch*)

property soot

property size

property codenode

10.2 Plugin Ecosystem

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.

10.3 Program State

angr.sim_state.**arch_overridable**(*f*)

class angr.sim_state.**SimState**(*project=None*, *arch=None*, *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*)

Bases: *PluginHub*[*SimStatePlugin*], Generic

The SimState represents the state of a program, including its memory, registers, and so forth.

Parameters

- **project** (*Project* | None) – The project instance.
- **arch** (Arch | None) – The architecture of the state.
- **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*)

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*

registers: *DefaultMemory*

regs: *SimRegNameView*

memory: *DefaultMemory*

callstack: *CallStack*

mem: *SimMemView*

history: *SimStateHistory*

inspect: *SimInspector*

jni_references: *SimStateJNIReferences*

scratch: *SimStateScratch*

__init__(*project=None*, *arch=None*, *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 | None*)
- **arch** (*Arch | None*)
- **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

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.

has_plugin(*name*)

Return whether or not a plugin with the name `name` is currently active.

register_plugin(*name*, *plugin*, *inhibit_init=False*)

Add a new plugin `plugin` with name `name` to the active plugins.

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)

widen(**others*)

Perform a widening between self and other states :type others: :param others: :return:

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

property with_condition

class angr.sim_state_options.**StateOption**(*name*, *types*, *default='_NO_DEFAULT_VALUE'*, *description=None*)

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**(*thing*)

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': <O ABSTRACT_MEMORY[bool]>, 'ABSTRACT_SOLVER': <O
ABSTRACT_SOLVER[bool]>, 'ACTION_DEPS': <O ACTION_DEPS[bool]>, 'ADD_AUTO_REFS': <O
ADD_AUTO_REFS[bool]>, 'ALLOW_SEND_FAILURES': <O ALLOW_SEND_FAILURES[bool]>,
'ALL_FILES_EXIST': <O ALL_FILES_EXIST[bool]>, 'ANY_FILE_MIGHT_EXIST': <O
ANY_FILE_MIGHT_EXIST[bool]>, 'APPROXIMATE_FIRST': <O APPROXIMATE_FIRST[bool]>,
'APPROXIMATE_GUARDS': <O APPROXIMATE_GUARDS[bool]>, 'APPROXIMATE_MEMORY_INDICES': <O
APPROXIMATE_MEMORY_INDICES[bool]>, 'APPROXIMATE_MEMORY_SIZES': <O
APPROXIMATE_MEMORY_SIZES[bool]>, 'APPROXIMATE_SATISFIABILITY': <O
APPROXIMATE_SATISFIABILITY[bool]>, 'AST_DEPS': <O AST_DEPS[bool]>, 'AUTO_REFS': <O
AUTO_REFS[bool]>, 'AVOID_MULTIVALUED_READS': <O AVOID_MULTIVALUED_READS[bool]>,
'AVOID_MULTIVALUED_WRITES': <O AVOID_MULTIVALUED_WRITES[bool]>,
'BEST_EFFORT_MEMORY_STORING': <O BEST_EFFORT_MEMORY_STORING[bool]>,
'BYPASS_ERRORED_IRCCALL': <O BYPASS_ERRORED_IRCCALL[bool]>, 'BYPASS_ERRORED_IROP':
<O BYPASS_ERRORED_IROP[bool]>, 'BYPASS_ERRORED_IRSTMT': <O
BYPASS_ERRORED_IRSTMT[bool]>, 'BYPASS_UNSUPPORTED_IRCCALL': <O
BYPASS_UNSUPPORTED_IRCCALL[bool]>, 'BYPASS_UNSUPPORTED_IRDIRTY': <O
BYPASS_UNSUPPORTED_IRDIRTY[bool]>, 'BYPASS_UNSUPPORTED_IREXPR': <O
BYPASS_UNSUPPORTED_IREXPR[bool]>, 'BYPASS_UNSUPPORTED_IROP': <O
BYPASS_UNSUPPORTED_IROP[bool]>, 'BYPASS_UNSUPPORTED_IRSTMT': <O
BYPASS_UNSUPPORTED_IRSTMT[bool]>, 'BYPASS_UNSUPPORTED_SYSCALL': <O
BYPASS_UNSUPPORTED_SYSCALL[bool]>, 'BYPASS_VERITESTING_EXCEPTIONS': <O
BYPASS_VERITESTING_EXCEPTIONS[bool]>, 'CACHELESS_SOLVER': <O
CACHELESS_SOLVER[bool]>, 'CALLLESS': <O CALLLESS[bool]>, 'CGC_ENFORCE_FD': <O
CGC_ENFORCE_FD[bool]>, 'CGC_NON_BLOCKING_FDS': <O CGC_NON_BLOCKING_FDS[bool]>,
'CGC_NO_SYMBOLIC_RECEIVE_LENGTH': <O CGC_NO_SYMBOLIC_RECEIVE_LENGTH[bool]>,
'COMPOSITE_SOLVER': <O COMPOSITE_SOLVER[bool]>, 'CONCRETIZE': <O CONCRETIZE[bool]>,
'CONCRETIZE_SYMBOLIC_FILE_READ_SIZES': <O
CONCRETIZE_SYMBOLIC_FILE_READ_SIZES[bool]>, 'CONCRETIZE_SYMBOLIC_WRITE_SIZES': <O
CONCRETIZE_SYMBOLIC_WRITE_SIZES[bool]>, 'CONSERVATIVE_READ_STRATEGY': <O
CONSERVATIVE_READ_STRATEGY[bool]>, 'CONSERVATIVE_WRITE_STRATEGY': <O
CONSERVATIVE_WRITE_STRATEGY[bool]>, 'CONSTRAINT_TRACKING_IN_SOLVER': <O
CONSTRAINT_TRACKING_IN_SOLVER[bool]>, 'COPY_STATES': <O COPY_STATES[bool]>,
'CPUID_SYMBOLIC': <O CPUID_SYMBOLIC[bool]>, 'DOWNSIZE_Z3':  <O DOWNSIZE_Z3[bool]>,
'DO_CCALLS': <O DO_CCALLS[bool]>, 'DO_RET_EMULATION': <O DO_RET_EMULATION[bool]>,
'EFFICIENT_STATE_MERGING': <O EFFICIENT_STATE_MERGING[bool]>, 'ENABLE_NX': <O
ENABLE_NX[bool]>, 'EXCEPTION_HANDLING': <O EXCEPTION_HANDLING[bool]>,
'EXTENDED_IROP_SUPPORT': <O EXTENDED_IROP_SUPPORT[bool]>, 'FAST_MEMORY': <O
FAST_MEMORY[bool]>, 'FAST_REGISTERS': <O FAST_REGISTERS[bool]>, 'FILES_HAVE_EOF': <O
FILES_HAVE_EOF[bool]>, 'HYBRID_SOLVER': <O HYBRID_SOLVER[bool]>,
'JAVA_IDENTIFY_GETTER_SETTER': <O JAVA_IDENTIFY_GETTER_SETTER[bool]>,
'JAVA_TRACK_ATTRIBUTES': <O JAVA_TRACK_ATTRIBUTES[bool]>, 'KEEP_IP_SYMBOLIC': <O
KEEP_IP_SYMBOLIC[bool]>, 'LAZY_SOLVES': <O LAZY_SOLVES[bool]>,
'MEMORY_CHUNK_INDIVIDUAL_READS': <O MEMORY_CHUNK_INDIVIDUAL_READS[bool]>,
'MEMORY_FIND_STRICT_SIZE_LIMIT': <O MEMORY_FIND_STRICT_SIZE_LIMIT[bool]>,
'MEMORY_SYMBOLIC_BYTES_MAP': <O MEMORY_SYMBOLIC_BYTES_MAP[bool]>,
'NO_CROSS_INSN_OPT': <O NO_CROSS_INSN_OPT[bool]>, 'NO_IP_CONCRETIZATION': <O
NO_IP_CONCRETIZATION[bool]>, 'NO_SYMBOLIC_JUMP_RESOLUTION': <O
NO_SYMBOLIC_JUMP_RESOLUTION[bool]>, 'NO_SYMBOLIC_SYSCALL_RESOLUTION': <O
NO_SYMBOLIC_SYSCALL_RESOLUTION[bool]>, 'OPTIMIZE_IR': <O OPTIMIZE_IR[bool]>,
'PRODUCE_ZERODIV_SUCCESSORS': <O PRODUCE_ZERODIV_SUCCESSORS[bool]>,
'REGION_MAPPING': <O REGION_MAPPING[bool]>, 'REPLACEMENT_SOLVER': <O
REPLACEMENT_SOLVER[bool]>, 'REVERSE_MEMORY_HASH_MAP': <O
REVERSE_MEMORY_HASH_MAP[bool]>, 'REVERSE_MEMORY_NAME_MAP': <O
REVERSE_MEMORY_NAME_MAP[bool]>, 'RUN_HOOKS_AT_PLT': <O RUN_HOOKS_AT_PLT[bool]>,
'SHORT_READS': <O SHORT_READS[bool]>, 'SIMPLIFY_CONSTRAINTS': <O
SIMPLIFY_CONSTRAINTS[bool]>, 'SIMPLIFY_EXIT_GUARD': <O SIMPLIFY_EXIT_GUARD[bool]>,
'SIMPLIFY_EXIT_STATE': <O SIMPLIFY_EXIT_STATE[bool]>, 'SIMPLIFY_EXIT_TARGET': <O
SIMPLIFY_EXIT_TARGET[bool]>, 'SIMPLIFY_EXPRS': <O SIMPLIFY_EXPRS[bool]>,
'SIMPLIFY_MEMORY_READS': <O SIMPLIFY_MEMORY_READS[bool]>, 'SIMPLIFY_MEMORY_WRITES':
<O SIMPLIFY_MEMORY_WRITES[bool]>, 'SIMPLIFY_MERGED_CONSTRAINTS': <O
```

__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

`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**(*omit_fp=False*, *adjust_stack=False*)

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.

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.

class angr.state_plugins.**CallStack**(*call_site_addr=0*, *func_addr=0*, *stack_ptr=0*, *ret_addr=0*, *jumpkind='Ijk_Call'*, *next_frame=None*, *invoke_return_variable=None*)

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.

Parameters
next_frame (*Self | None*)

__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(*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.

copy_without_tail(*memo*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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,
```

(continues on next page)

(continued from previous page)

```
    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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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**(*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*)

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.

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*)

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**(*base*, *sim_state*, *heap=None*)

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*)

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*, *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).

is_free()

Returns a concrete determination as to whether the chunk is free.

data_ptr()

Returns the address of the payload of the chunk.

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

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. 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

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.**PTChunkIterator**(*chunk*, *cond=<function PTChunkIterator.<lambda>>*)

Bases: object

__init__(*chunk*, *cond=<function PTChunkIterator.<lambda>>*)

class angr.state_plugins.**PosixDevFS**

Bases: *SimMount*

get(*path*)

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*, *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*)

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(_)

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

merge(*others*, *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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

copy(_)

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.

class angr.state_plugins.**PosixProcFS**

Bases: *SimMount*

The virtual file system mounted at /proc (as of now, on Linux).

get(*path*)

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*, *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*)

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(_)

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

merge(*others*, *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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

copy(_)

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.

class angr.state_plugins.**SimAction**(*state*, *region_type*)

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**(*state*, *constraint*, *condition=None*)

Bases: *SimAction*

A constraint action represents an extra constraint added during execution of a path.

__init__(*state*, *constraint*, *condition=None*)

Initializes the SimAction.

Parameters
state – the state that's the SimAction is taking place in.

property all_objects

property is_symbolic

class angr.state_plugins.**SimActionData**(*state*, *region_type*, *action*, *tmp=None*, *addr=None*, *size=None*, *data=None*, *condition=None*, *fallback=None*, *fd=None*)

Bases: *SimAction*

A Data action represents a read or a write from memory, registers or a file.

READ = 'read'

WRITE = 'write'

OPERATE = 'operate'

__init__(*state*, *region_type*, *action*, *tmp=None*, *addr=None*, *size=None*, *data=None*, *condition=None*, *fallback=None*, *fd=None*)

Initializes the SimAction.

Parameters
state – the state that's the SimAction is taking place in.

downsize()

Clears some low-level details (that take up memory) out of the SimAction.

property all_objects

property is_symbolic

property tmp_deps

property reg_deps

property storage

class angr.state_plugins.**SimActionExit**(*state*, *target*, *condition=None*, *exit_type=None*)

Bases: *SimAction*

An Exit action represents a (possibly conditional) jump.

CONDITIONAL = 'conditional'

DEFAULT = 'default'

__init__(*state*, *target*, *condition=None*, *exit_type=None*)

Initializes the SimAction.

Parameters
state – the state that's the SimAction is taking place in.

property all_objects

property is_symbolic

class angr.state_plugins.**SimActionObject**(*ast*, *reg_deps=frozenset({})*, *tmp_deps=frozenset({})*, *deps=frozenset({})*, *state=None*)

Bases: object

A SimActionObject tracks an AST and its dependencies.

Parameters

- **ast** (*claripy.ast.Base*)
- **reg_deps** (*frozenset[SimActionData | SimActionOperation]*)
- **tmp_deps** (*frozenset[SimActionData | SimActionOperation]*)
- **deps** (*frozenset*)
- **state** (*SimState | None*)

__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[Base | bool | int | float | str | FSort | tuple[Base | bool | int | float | str | FSort | tuple[ArgType] | None] | None, ...]

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**(*state*, *op*, *exprs*, *result*)

Bases: *SimAction*

An action representing an operation between variables and/or constants.

__init__(*state*, *op*, *exprs*, *result*)

Initializes the SimAction.

Parameters
state – the state that's the SimAction is taking place in.

property all_objects

property is_symbolic

class angr.state_plugins.**SimDebugVariable**(*state*, *addr*, *var_type*)

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.

Parameters

- **state** (SimState)
- **var_type** (VariableType)

__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**(*state*, *event_type*, ***kwargs*)

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**(*files=None*, *pathsep=None*, *cwd=None*, *mountpoints=None*)

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
- **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*)

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.

property unlinks

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

chdir(*path*)

Changes the current directory to the given path

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.

class angr.state_plugins.**SimHeapBase**(*heap_base=None*, *heap_size=None*)

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 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

__init__(*heap_base=None*, *heap_size=None*)

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.

init_state()

Use this function to perform any initialization on the state at plugin-add time

class angr.state_plugins.**SimHeapBrk**(*heap_base=None*, *heap_size=None*)

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*)

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.

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.

Parameters

sim_size – a size specifying how much to decrease the break pointer by (may be symbolic or not)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.state_plugins.**SimHeapLibc**(*heap_base=None*, *heap_size=None*)

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**(*heap_base=None*, *heap_size=None*)

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*)

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.

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 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

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

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

init_state()

Use this function to perform any initialization on the state at plugin-add time

class angr.state_plugins.**SimHostFilesystem**(*host_path=None*, ***kwargs*)

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*)

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.

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

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.

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()
```

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

set_state(*state*)

Sets a new state (for example, if the state has been branched)

class angr.state_plugins.**SimJavaVmClassloader**(*initialized_classes=None*)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.state_plugins.**SimLightRegisters**(*reg_map=None*, *registers=None*)

Bases: *SimStatePlugin*

__init__(*reg_map=None*, *registers=None*)

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.

set_state(*state*)

Sets a new state (for example, if the state has been branched)

resolve_register(*offset*, *size*)

load(*offset*, *size=None*, ***kwargs*)

store(*offset*, *value*, *size=None*, *endness=None*, ***kwargs*)

class angr.state_plugins.**SimMemView**(*ty=None*, *addr=None*, *state=None*)

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'
```

Parameters

state (SimState[*Any*, *Any*])

__init__(*ty=None*, *addr=None*, *state=None*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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*

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

get(*reg_name*)

class angr.state_plugins.**SimSolver**(*solver=None*, *all_variables=None*, *temporal_tracked_variables=None*, *eternal_tracked_variables=None*)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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*)

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]

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.

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*)

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

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.

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*)

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

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!

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*)

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]

Evaluate an expression to get at most *n* possible solutions. Errors if either none or more than *n* solutions are returned.

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*)

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]

Evaluate an expression to get at least *n* possible solutions. Errors if less than *n* solutions were found.

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*)

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]

Evaluate an expression to get exactly the *n* possible solutions. Errors if any number of solutions other than *n* was found to exist.

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__()

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.

peek_input()

discard_input(*num_bytes*)

peek_output()

discard_output(*num_bytes*)

addr_invalid(*a*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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**(*edge_hitmap=None*)

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.

Parameters
edge_hitmap (bytes | *None*)

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.

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

widen(*others*)

Widening operation for static analysis.

Parameters
others – Other plugin instances.

Returns
False, widening is not meaningful for hitmaps.

class angr.state_plugins.**SimStateGlobals**(*backer=None*)

Bases: *SimStatePlugin*

__init__(*backer=None*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

keys()

values()

items()

get(*k*, *alt=None*)

pop(*k*, *alt=None*)

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.

class angr.state_plugins.**SimStateHistory**(*parent=None*, *clone=None*)

Bases: *SimStatePlugin*

This class keeps track of historically-relevant information for paths.

STRONGREF_STATE = True

__init__(*parent=None*, *clone=None*)

init_state()

Use this function to perform any initialization on the state at plugin-add time

set_strongref_state(*state*)

property addr

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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.

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**(*vm_ref=None*, *generation=None*, *base_translation_data=None*, *translation_data=None*, *dirty_pages=None*)

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.

Parameters

- **vm_ref** (*IcicleVMRef | None*)
- **generation** (*int | None*)
- **base_translation_data** (*IcicleStateTranslationData | None*)
- **translation_data** (*IcicleStateTranslationData | None*)
- **dirty_pages** (*set[int] | None*)

__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.

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.state_plugins.**SimStateJNIReferences**(*local_refs=None*, *global_refs=None*)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\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',
```

```
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__()

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

property errno

ret_errno(*val*)

class angr.state_plugins.**SimStateLog**(*log=None*)

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

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

clear()

class angr.state_plugins.**SimStateLoopData**(*back_edge_trip_counts=None*, *header_trip_counts=None*, *current_loop=None*)

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).

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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.

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.

STRONGREF_STATE = False

__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`

set_strongref_state(*state*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others (Iterable[*SimStatePlugin*]) – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

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**(*constrained_addrs=None*)

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*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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.

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**(*scratch=None*)

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

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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__()

init_state()

Use this function to perform any initialization on the state at plugin-add time

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.

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.

class angr.state_plugins.**SimSystemPosix**(*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*)

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*)

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.

property closed_fds

init_state()

Use this function to perform any initialization on the state at plugin-add time

set_brk(*new_brk*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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**(*man=None*)

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

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.

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

set_state(*state*)

Sets a new state (for example, if the state has been branched)

class angr.state_plugins.**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*)

Bases: `tuple`

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**(*view*)

Bases: object

__init__(*view*)

class angr.state_plugins.**Unicorn**(*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*)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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*)

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.

STRONGREF_STATE = False

__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

set_strongref_state(*state*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others (Iterable[*SimStatePlugin*]) – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

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.inspect.**EventType**(**values*)

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'

VFG_WIDEN_STATE = 'vfg_widen_state'

ENGINE_PROCESS = 'engine_process'

MEMORY_PAGE_MAP = 'memory_page_map'

class angr.state_plugins.inspect.**InspectAttrs**(*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*)

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.

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*)

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**(**values*)

Bases: StrEnum

When to trigger breakpoints.

BEFORE = 'before'

AFTER = 'after'

BOTH = 'both'

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**(*when=When.BEFORE*, *enabled=True*, *condition=None*, *action=<function BP_IPDB>*, ***kwargs*)

Bases: object

A breakpoint.

Parameters

- **when** (When)
- **enabled** (*bool*)
- **condition** (*Callable[[SimState], bool] | None*)
- **action** (Action)
- **kwargs** (*dict[str, Any]*)

__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

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.

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()
```

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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'\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__()

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

property errno

ret_errno(*val*)

class angr.state_plugins.posix.**PosixDevFS**

Bases: *SimMount*

get(*path*)

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*, *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*)

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(_)

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

merge(*others*, *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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

copy(_)

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.

class angr.state_plugins.posix.**PosixProcFS**

Bases: *SimMount*

The virtual file system mounted at /proc (as of now, on Linux).

get(*path*)

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*, *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*)

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(_)

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

merge(*others*, *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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

copy(_)

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.

class angr.state_plugins.posix.**SimSystemPosix**(*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*)

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*)

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.

property closed_fds

init_state()

Use this function to perform any initialization on the state at plugin-add time

set_brk(*new_brk*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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.filesystem.**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*)

Bases: tuple

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**(*files=None*, *pathsep=None*, *cwd=None*, *mountpoints=None*)

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
- **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*)

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.

property unlinks

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

chdir(*path*)

Changes the current directory to the given path

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.

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**(*pathsep='/'*)

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='/'*)

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

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.

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.state_plugins.filesystem.**SimHostFilesystem**(*host_path=None*, ***kwargs*)

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*)

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.

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**(*solver=None*, *all_variables=None*, *temporal_tracked_variables=None*, *eternal_tracked_variables=None*)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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*)

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]

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.

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*)

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

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.

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*)

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

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!

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*)

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]

Evaluate an expression to get at most *n* possible solutions. Errors if either none or more than *n* solutions are returned.

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*)

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]

Evaluate an expression to get at least *n* possible solutions. Errors if less than *n* solutions were found.

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*)

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]

Evaluate an expression to get exactly the *n* possible solutions. Errors if any number of solutions other than *n* was found to exist.

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.log.**SimStateLog**(*log=None*)

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

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

clear()

class angr.state_plugins.callstack.**CallStack**(*call_site_addr=0, func_addr=0, stack_ptr=0, ret_addr=0, jumpkind='Ijk_Call', next_frame=None, invoke_return_variable=None*)

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.

Parameters
next_frame (*Self | None*)

__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(*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.

copy_without_tail(*memo*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

`widen`(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
`others` – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

`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**(*callstack_hash*, *callstack_depth*, *action*, *callframe=None*, *ret_site_addr=None*)

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*)

class angr.state_plugins.light_registers.**SimLightRegisters**(*reg_map=None*, *registers=None*)

Bases: *SimStatePlugin*

__init__(*reg_map=None*, *registers=None*)

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.

set_state(*state*)

Sets a new state (for example, if the state has been branched)

resolve_register(*offset*, *size*)

load(*offset*, *size=None*, ***kwargs*)

store(*offset*, *value*, *size=None*, *endness=None*, ***kwargs*)

class angr.state_plugins.history.**SimStateHistory**(*parent=None*, *clone=None*)

Bases: *SimStatePlugin*

This class keeps track of historically-relevant information for paths.

STRONGREF_STATE = True

__init__(*parent=None*, *clone=None*)

jump_guard: BV | None

jumpkind: str | None

edge_hitmap: *bytes* | None

init_state()

Use this function to perform any initialization on the state at plugin-add time

set_strongref_state(*state*)

property addr

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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.

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**(*start*, *end=None*)

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**(*start*, *end=None*)

Bases: *TreeIter*

class angr.state_plugins.history.**LambdaAttrIter**(*start*, *f*, ***kwargs*)

Bases: *TreeIter*

__init__(*start*, *f*, ***kwargs*)

class angr.state_plugins.history.**LambdaIterIter**(*start*, *f*, *reverse=True*, ***kwargs*)

Bases: *LambdaAttrIter*

__init__(*start*, *f*, *reverse=True*, ***kwargs*)

class angr.state_plugins.gdb.**GDB**(*omit_fp=False*, *adjust_stack=False*)

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.

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.

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__()

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.

peek_input()

discard_input(*num_bytes*)

peek_output()

discard_output(*num_bytes*)

addr_invalid(*a*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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.

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**(*addr*, *func_name*, *str_arg_num*, *base*, *base_arg*, *allows_negative*)

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**(*addr*, *func_name*, *int_arg_num*, *str_dst_num*, *base*, *base_arg*)

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**(*addr*, *func_name*, *check_symbolic_arg*)

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__()

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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**(*max_depth=13*)

Bases: *SimStatePlugin*

__init__(*max_depth=13*)

static get_flag_rand_args(*expr*)

get_expr_depth(*expr*)

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

get_flag_bytes(*ast*)

filter_constraints(*constraints*)

analyze_transmit(*state*, *buf*)

static prep_tracer(*state*)

class angr.state_plugins.globals.**SimStateGlobals**(*backer=None*)

Bases: *SimStatePlugin*

__init__(*backer=None*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

keys()

values()

items()

get(*k*, *alt=None*)

pop(*k*, *alt=None*)

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.

class angr.state_plugins.uc_manager.**SimUCManager**(*man=None*)

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

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.

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

set_state(*state*)

Sets a new state (for example, if the state has been branched)

class angr.state_plugins.scratch.**SimStateScratch**(*scratch=None*)

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

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

clear()

class angr.state_plugins.preconstrainer.**SimStatePreconstrainer**(*constrained_addrs=None*)

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*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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.

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.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**(*addr*)

Bases: SimplificationAvoidanceAnnotation

__init__(*addr*)

class angr.state_plugins.unicorn_engine.**Uniwrapper**(*arch*, *cache_key*, *thumb=False*)

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**(*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*)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

class angr.state_plugins.loop_data.**SimStateLoopData**(*back_edge_trip_counts=None*, *header_trip_counts=None*, *current_loop=None*)

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).

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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.

class angr.state_plugins.javavm_classloader.**SimJavaVmClassloader**(*initialized_classes=None*)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.state_plugins.jni_references.**SimStateJNIReferences**(*local_refs=None*, *global_refs=None*)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.state_plugins.heap.**PTChunk**(*base*, *sim_state*, *heap=None*)

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*)

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*, *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).

is_free()

Returns a concrete determination as to whether the chunk is free.

data_ptr()

Returns the address of the payload of the chunk.

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

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. 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

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.**PTChunkIterator**(*chunk*, *cond=<function PTChunkIterator.<lambda>>*)

Bases: object

__init__(*chunk*, *cond=<function PTChunkIterator.<lambda>>*)

class angr.state_plugins.heap.**SimHeapBase**(*heap_base=None*, *heap_size=None*)

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 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

__init__(*heap_base=None*, *heap_size=None*)

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.

init_state()

Use this function to perform any initialization on the state at plugin-add time

class angr.state_plugins.heap.**SimHeapBrk**(*heap_base=None*, *heap_size=None*)

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*)

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.

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.

Parameters

sim_size – a size specifying how much to decrease the break pointer by (may be symbolic or not)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.state_plugins.heap.**SimHeapLibc**(*heap_base=None*, *heap_size=None*)

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**(*heap_base=None*, *heap_size=None*)

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*)

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.

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 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

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

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

init_state()

Use this function to perform any initialization on the state at plugin-add time

class angr.state_plugins.heap.heap_base.**SimHeapBase**(*heap_base=None*, *heap_size=None*)

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 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

__init__(*heap_base=None*, *heap_size=None*)

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.

init_state()

Use this function to perform any initialization on the state at plugin-add time

class angr.state_plugins.heap.heap_brk.**SimHeapBrk**(*heap_base=None*, *heap_size=None*)

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*)

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.

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.

Parameters
sim_size – a size specifying how much to decrease the break pointer by (may be symbolic or not)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.state_plugins.heap.heap_freelist.**Chunk**(*base*, *sim_state*)

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**(*heap_base=None*, *heap_size=None*)

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()

class angr.state_plugins.heap.heap_libc.**SimHeapLibc**(*heap_base=None*, *heap_size=None*)

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.heap_ptmalloc.`**`PTChunk`**(*base*, *sim_state*, *heap=None*)

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*)

`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*, *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).

`is_free()`

Returns a concrete determination as to whether the chunk is free.

`data_ptr()`

Returns the address of the payload of the chunk.

`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

`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. 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

`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_ptmalloc.PTChunkIterator(chunk, cond=<function PTChunkIterator.<lambda>>)`

Bases: `object`

`__init__(chunk, cond=<function PTChunkIterator.<lambda>>)`

class angr.state_plugins.heap.heap_ptmalloc.**SimHeapPTMalloc**(*heap_base=None*, *heap_size=None*)

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*)

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.

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 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

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

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

init_state()

Use this function to perform any initialization on the state at plugin-add time

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

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__()

init_state()

Use this function to perform any initialization on the state at plugin-add time

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.

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.

class angr.state_plugins.debug_variables.**SimDebugVariable**(*state*, *addr*, *var_type*)

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.

Parameters

- **state** (`SimState`)
- **var_type** (`VariableType`)

__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

10.4 Storage

class angr.storage.**DefaultMemory**(**args*, ***kwargs*)

Bases: *HexDumperMixin*, *SmartFindMixin*, *UnwrapperMixin*, *NameResolutionMixin*, *DataNormalizationMixin*, *SimplificationMixin*, *InspectMixin*, *ActionsMixinHigh*, *UnderconstrainedMixin*, *SizeConcretizationMixin*, *SizeNormalizationMixin*, *AddressConcretizationMixin*, *ActionsMixinLow*, *ConditionalMixin*, *ConvenientMappingsMixin*, *DirtyAddrsMixin*, *StackAllocationMixin*, *ConcreteBackerMixin*, *ClemoryBackerMixin*, *DictBackerMixin*, *PrivilegedPagingMixin*, *UltraPagesMixin*, *DefaultFillerMixin*, *SymbolicMergerMixin*, *PagedMemoryMixin*

class angr.storage.**SimFile**(*name=None*, *content=None*, *size=None*, *has_end=None*, *seekable=True*, *writable=True*, *ident=None*, *concrete=None*, ***kwargs*)

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

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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*, *events=True*, ***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.

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.

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.storage.**SimMemoryObject**(*obj*, *base*, *endness*, *length=None*, *byte_width=8*)

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.state_plugins.view.**SimRegNameView**

Bases: *SimStatePlugin*

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

get(*reg_name*)

class angr.state_plugins.view.**SimMemView**(*ty=None*, *addr=None*, *state=None*)

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'
```

Parameters

state (SimState[*Any*, *Any*])

__init__(*ty=None*, *addr=None*, *state=None*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

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**(*view*)

Bases: object

__init__(*view*)

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**(*name=None*, *writable=True*, *ident=None*, *concrete=False*, *file_exists=True*, ***kwargs*)

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.

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.

class angr.storage.file.**SimFile**(*name=None*, *content=None*, *size=None*, *has_end=None*, *seekable=True*, *writable=True*, *ident=None*, *concrete=None*, ***kwargs*)

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

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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*, *events=True*, ***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.

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.

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,
```

(continues on next page)

(continued from previous page)

```
    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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.storage.file.**SimFileStream**(*name=None*, *content=None*, *pos=0*, ***kwargs*)

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*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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(_, *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.

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.

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,
```

(continues on next page)

(continued from previous page)

```
    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

class angr.storage.file.**SimPackets**(*name*, *write_mode=None*, *content=None*, *writable=True*, *ident=None*, ***kwargs*)

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*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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

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.

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.storage.file.**SimPacketsStream**(*name*, *pos=0*, ***kwargs*)

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*)

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(_, *data*, *size=None*, ***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

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.

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

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**(*simfile*, *flags=0*)

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*)

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

eof()

Return the EOF status. May be a symbolic boolean.

tell()

Return the current position, or None if the concept doesn't make sense for the given file.

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.

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.

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.storage.file.**SimFileDescriptorDuplex**(*read_file*, *write_file*)

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*)

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

set_state(*state*)

Sets a new state (for example, if the state has been branched)

eof()

Return the EOF status. May be a symbolic boolean.

tell()

Return the current position, or None if the concept doesn't make sense for the given file.

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

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.

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.

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.

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.storage.file.**SimPacketsSlots**(*name*, *read_sizes*, *ident=None*, ***kwargs*)

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*)

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.

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.

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

widen(_)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

angr.storage.memory_object.**obj_bit_size**(*o*)

class angr.storage.memory_object.**SimMemoryObject**(*obj*, *base*, *endness*, *length=None*, *byte_width=8*)

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**(*obj*, *base*, *endness*, *length=None*, *byte_width=8*, *label=None*)

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

class angr.concretization_strategies.**SimConcretizationStrategy**(*filter=None*, *exact=True*)

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**(*filter=None*, *exact=True*)

Bases: *SimConcretizationStrategy*

Concretization strategy that returns any single solution.

class angr.concretization_strategies.**SimConcretizationStrategyControlledData**(*limit*, *fixed_addrs*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that constraints the address to controlled data. Controlled data consists of symbolic data and the addresses given as arguments. memory.

__init__(*limit*, *fixed_addrs*, ***kwargs*)

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.

class angr.concretization_strategies.**SimConcretizationStrategyEval**(*limit*, ***kwargs*)

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.

__init__(*limit*, ***kwargs*)

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.

class angr.concretization_strategies.**SimConcretizationStrategyMax**(*max_addr=None*)

Bases: *SimConcretizationStrategy*

Concretization strategy that returns the maximum address.

Parameters

max_addr (*int | None*)

__init__(*max_addr=None*)

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.
- **max_addr** (*int | None*)

class angr.concretization_strategies.**SimConcretizationStrategyNonzero**(*filter=None*, *exact=True*)

Bases: *SimConcretizationStrategy*

Concretization strategy that returns any non-zero solution.

class angr.concretization_strategies.**SimConcretizationStrategyNonzeroRange**(*limit*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves a range in a non-zero location.

__init__(*limit*, ***kwargs*)

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.

class angr.concretization_strategies.**SimConcretizationStrategyNorepeats**(*repeat_expr*, *repeat_constraints=None*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses, without repeating.

__init__(*repeat_expr*, *repeat_constraints=None*, ***kwargs*)

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.

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.**SimConcretizationStrategyNorepeatsRange**(*repeat_expr*, *min=None*, *granularity=None*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves a range, with no repeats.

__init__(*repeat_expr*, *min=None*, *granularity=None*, ***kwargs*)

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.

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.**SimConcretizationStrategyRange**(*limit*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses to a range.

__init__(*limit*, ***kwargs*)

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.

class angr.concretization_strategies.**SimConcretizationStrategySingle**(*filter=None*, *exact=True*)

Bases: *SimConcretizationStrategy*

Concretization strategy that ensures a single solution for an address.

class angr.concretization_strategies.**SimConcretizationStrategySolutions**(*limit*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves an address into some limited number of solutions.

__init__(*limit*, ***kwargs*)

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.

class angr.concretization_strategies.**SimConcretizationStrategyUnlimitedRange**(*limit*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses to a range without checking if the number of possible addresses is within the limit.

__init__(*limit*, ***kwargs*)

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.

10.5 Memory Mixins

class angr.storage.memory_mixins.**AbstractMemory**(**args*, ***kwargs*)

Bases: *UnwrapperMixin*, *NameResolutionMixin*, *DataNormalizationMixin*, *SimplificationMixin*, *InspectMixin*, *ActionsMixinHigh*, *UnderconstrainedMixin*, *SizeConcretizationMixin*, *SizeNormalizationMixin*, *ActionsMixinLow*, *ConditionalMixin*, *RegionedAddressConcretizationMixin*, *RegionedMemoryMixin*

class angr.storage.memory_mixins.**AbstractMergerMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

AbstractMergerMixin handles merging initialized values.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

class angr.storage.memory_mixins.**ActionsMixinHigh**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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**(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

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*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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`

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*)

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**

Returns
A memoryview into the loaded bytes.

class angr.storage.memory_mixins.**ClemoryBackerMixin**(*cle_memory_backer=None*, ***kwargs*)

Bases: *PagedMemoryMixin*

Parameters
cle_memory_backer (*None | cle.Loader | cle.Clemory*)

__init__(*cle_memory_backer=None*, ***kwargs*)

Parameters
cle_memory_backer (*None | Loader | Clemory*)

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.

class angr.storage.memory_mixins.**ConcreteBackerMixin**(*cle_memory_backer=None*, ***kwargs*)

Bases: *ClemoryBackerMixin*

Parameters
cle_memory_backer (*None | cle.Loader | cle.Clemory*)

class angr.storage.memory_mixins.**ConditionalMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

load(*addr*, *size=None*, ***, *condition=None*, *fallback=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***, *condition=None*, ***kwargs*)

class angr.storage.memory_mixins.**ConvenientMappingsMixin**(***kwargs*)

Bases: MemoryMixin

Implements mappings between names and hashes of symbolic variables and these variables themselves.

__init__(***kwargs*)

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.

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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Normalizes the data field for a store and the fallback field for a load to be BVs.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

store(*addr*, *data*, *size=None*, ***kwargs*)

load(*addr*, *size=None*, ***, *fallback=None*, ***kwargs*)

class angr.storage.memory_mixins.**DefaultFillerMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

class angr.storage.memory_mixins.**DefaultListPagesMemory**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

Bases: *HexDumperMixin*, *SmartFindMixin*, *UnwrapperMixin*, *NameResolutionMixin*, *DataNormalizationMixin*, *SimplificationMixin*, *InspectMixin*, *ActionsMixinHigh*, *UnderconstrainedMixin*, *SizeConcretizationMixin*, *SizeNormalizationMixin*, *AddressConcretizationMixin*, *ActionsMixinLow*, *ConditionalMixin*, *ConvenientMappingsMixin*, *DirtyAddrsMixin*, *StackAllocationMixin*, *ConcreteBackerMixin*, *ClemoryBackerMixin*, *DictBackerMixin*, *PrivilegedPagingMixin*, *UltraPagesMixin*, *DefaultFillerMixin*, *SymbolicMergerMixin*, *PagedMemoryMixin*

class angr.storage.memory_mixins.**DictBackerMixin**(*dict_memory_backer=None*, ***kwargs*)

Bases: *PagedMemoryMixin*

__init__(*dict_memory_backer=None*, ***kwargs*)

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.

class angr.storage.memory_mixins.**DirtyAddrsMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: `MemoryMixin`

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

store(*addr*, *data*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.**ExplicitFillerMixin**(*uninitialized_read_handler=None*, ***kwargs*)

Bases: `MemoryMixin`

__init__(*uninitialized_read_handler=None*, ***kwargs*)

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.

class angr.storage.memory_mixins.**FastMemory**(*uninitialized_read_handler=None*, ***kwargs*)

Bases: *NameResolutionMixin*, *SimpleInterfaceMixin*, *SimplificationMixin*, *InspectMixin*, *ConditionalMixin*, *ExplicitFillerMixin*, *DefaultFillerMixin*, *SlottedMemoryMixin*

class angr.storage.memory_mixins.**HexDumperMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: `MemoryMixin`

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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**(**args*, ***kwargs*)

Bases: *RefcountMixin*, MemoryMixin

Tracks the history of memory writes.

__init__(**args*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***kwargs*)

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.

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.

parents()

changed_bytes(*other*, ***kwargs*)

Return type
set[int] | None

all_bytes_changed_in_history()

Return type
SegmentList

class angr.storage.memory_mixins.**ISPOMixin**(*memory_id=None*, *endness='Iend_BE'*)

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.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

class angr.storage.memory_mixins.**InspectMixin**(*memory_id=None*, *endness='Iend_BE'*)

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)

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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**(*memory_id='mem'*, *stack=None*, *heap=None*, *vm_static_table=None*, *load_strategies=None*, *store_strategies=None*, *max_array_size=1000*, ***kwargs*)

Bases: JavaVmMemoryMixin

class angr.storage.memory_mixins.**JavaVmMemoryMixin**(*memory_id='mem'*, *stack=None*, *heap=None*, *vm_static_table=None*, *load_strategies=None*, *store_strategies=None*, *max_array_size=1000*, ***kwargs*)

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.

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

class angr.storage.memory_mixins.**KeyValueMemory**(**args*, ***kwargs*)

Bases: *KeyValueMemoryMixin*

class angr.storage.memory_mixins.**KeyValueMemoryMixin**(**args*, ***kwargs*)

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*)

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.

Return type
`Self`

class angr.storage.memory_mixins.**LabelMergerMixin**(**args*, ***kwargs*)

Bases: MemoryMixin

A memory mixin for merging labels. Labels come from SimLabeledMemoryObject.

__init__(**args*, ***kwargs*)

copy(*memo=None*)

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.

class angr.storage.memory_mixins.**LabeledMemory**(**args*, *top_func=None*, ***kwargs*)

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**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

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*)

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.

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*)

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** – The address to start erasing.
- **size** – The number of bytes for erasing.

Return type
None

Returns
None

merge(*others*, *merge_conditions*, *common_ancestor=None*, *page_addr=None*, *memory=None*, *changed_offsets=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** (list[*ListPage*]) – 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
- **page_addr** (*int | None*)
- **changed_offsets** (*set[int] | None*)

Returns
True if the state plugins are actually merged.

Return type
bool

changed_bytes(*other*, *page_addr=None*)

Parameters

- **other** (ListPage)
- **page_addr** (*int | None*)

class angr.storage.memory_mixins.**ListPagesMixin**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

Bases: PagedMemoryMixin

PAGE_TYPE

alias of ListPage

class angr.storage.memory_mixins.**ListPagesWithLabelsMixin**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

Bases: LabeledPagesMixin, ListPagesMixin

class angr.storage.memory_mixins.**MVListPage**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

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*)

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.

Return type

MVListPage

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*)

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** – The address to start erasing.
- **size** – The number of bytes for erasing.

Return type

None

Returns
None

merge(*others*, *merge_conditions*, *common_ancestor=None*, ***, *page_addr*, *memory*, *changed_offsets=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** (`list[MVListPage]`) – 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
- **page_addr** (*int*)
- **memory** (*MemoryMixin*)
- **changed_offsets** (*set[int] | None*)

Returns
True if the state plugins are actually merged.

Return type
bool

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**(**args*, *skip_missing_values_during_merging=False*, ***kwargs*)

Bases: PagedMemoryMixin

PAGE_TYPE

alias of MVListPage

__init__(**args*, *skip_missing_values_during_merging=False*, ***kwargs*)

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.

Return type

MVListPagesMixin

class angr.storage.memory_mixins.**MVListPagesWithLabelsMixin**(**args*, *skip_missing_values_during_merging=False*, ***kwargs*)

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**(*related_function_addr=None*, ***kwargs*)

Bases: MemoryMixin

__init__(*related_function_addr=None*, ***kwargs*)

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.

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*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

dbg_print(*indent=0*)

Print out debugging information

class angr.storage.memory_mixins.**MultiValueMergerMixin**(**args*, *element_limit=5*, *annotation_limit=256*, *top_func=None*, *is_top_func=None*, *phi_maker=None*, *merge_into_top=True*, ***kwargs*)

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*)

copy(*memo=None*)

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.

class angr.storage.memory_mixins.**MultiValuedMemory**(**args*, *skip_missing_values_during_merging=False*, ***kwargs*)

Bases: *SizeNormalizationMixin*, *MVListPagesMixin*, *DefaultFillerMixin*, *MultiValueMergerMixin*, *PagedMemoryMixin*, *PagedMemoryMultiValueMixin*

class angr.storage.memory_mixins.**NameResolutionMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: `MemoryMixin`

This mixin allows you to provide register names as load addresses, and will automatically translate this to an offset and size.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

store(*addr*, *data*, *size=None*, ***kwargs*)

load(*addr*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.**PageBase**(**args*, ***kwargs*)

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**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

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*)

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.

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*)

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** – The address to start erasing.
- **size** – The number of bytes for erasing.

Return type
None

Returns
None

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

`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*)

`concrete_load`(*addr*, *size*, *writing=False*, **, *with_bitmap=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**
- **with_bitmap** (*bool*)

Returns
A memoryview into the loaded bytes.

`changed_bytes`(*other*)

Return type
set[int]

`changed_pages`(*other*)

Return type
dict[int, set[int] | None]

`copy_contents`(*dst*, *src*, *size*, *condition=None*, ***kwargs*)

Override this method to provide faster copying of large chunks of data.

Parameters

- **dst** – The destination of copying.
- **src** – The source of copying.
- **size** – The size of copying.
- **condition** – The storing condition.
- **kwargs** – Other parameters.

Returns
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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Implement optimizations and fast accessors for the MultiValues-variant of Paged Memory.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

load_annotations(*addr*, *size*, ***kwargs*)

Parameters

- **addr** (*int*)
- **size** (*int*)

class angr.storage.memory_mixins.**PermissionsMixin**(*permissions=None*, ***kwargs*)

Bases: MemoryMixin

This mixin adds a permissions_bits field and properties for extracting the read/write/exec permissions. It does NOT add permissions checking.

Parameters

permissions (*int | claripy.ast.BV | None*)

__init__(*permissions=None*, ***kwargs*)

Parameters

permissions (*int | BV | None*)

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.

property perm_read

property perm_write

property perm_exec

class angr.storage.memory_mixins.**PrivilegedPagingMixin**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

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**(***kwargs*)

Bases: `MemoryMixin`

This mixin adds a locked reference counter and methods to manipulate it, to facilitate copy-on-write optimizations.

__init__(***kwargs*)

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.

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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: `MemoryMixin`

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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**(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

Bases: MemoryMixin

__init__(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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

class angr.storage.memory_mixins.**RegionedMemory**(*related_function_addr=None*, ***kwargs*)

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**(*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*)

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().

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** (*Optional* | *None*)
- **dict_memory_backer** (*dict* | *None*)
- **regioned_memory_cls** (type | *None*)

__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*)

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.

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*)

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

find(*addr*, *data*, *max_search*, ***kwargs*)

Parameters
addr (*int* | *Bits*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

store(*addr*, *data*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.**SizeConcretizationMixin**(*concretize_symbolic_write_size=False*, *max_concretize_count=256*, *max_symbolic_size=4194304*, *raise_memory_limit_error=False*, *size_limit=257*, ***kwargs*)

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

Parameters

- **concretize_symbolic_write_size** (*bool*)
- **max_concretize_count** (*int | None*)
- **max_symbolic_size** (*int*)
- **raise_memory_limit_error** (*bool*)
- **size_limit** (*int*)

__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*)

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.

load(*addr*, *size=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***, *condition=None*, ***kwargs*)

class angr.storage.memory_mixins.**SizeNormalizationMixin**(*memory_id=None*, *endness='Iend_BE'*)

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

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

load(*addr*, *size=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.**SlottedMemoryMixin**(*width=None*, ***kwargs*)

Bases: `MemoryMixin`

__init__(*width=None*, ***kwargs*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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

load(*addr*, *size=None*, ***, *endness=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***, *endness=None*, ***kwargs*)

changed_bytes(*other*)

class angr.storage.memory_mixins.**SmartFindMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: `MemoryMixin`

Memory mixin providing basic searching over concrete and symbolic data.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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**(*special_memory_filler=None*, ***kwargs*)

Bases: MemoryMixin

__init__(*special_memory_filler=None*, ***kwargs*)

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.

class angr.storage.memory_mixins.**StackAllocationMixin**(*stack_end=None*, *stack_size=None*, *stack_perms=None*, ***kwargs*)

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*)

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.

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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: *SmartFindMixin*

Implements data finding for abstract memory.

Parameters

- **memory_id** (*str | None*)

- **endness** (*str*)

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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

class angr.storage.memory_mixins.**TopMergerMixin**(**args*, *top_func=None*, ***kwargs*)

Bases: MemoryMixin

A memory mixin for merging values in memory to TOP.

__init__(**args*, *top_func=None*, ***kwargs*)

copy(*memo=None*)

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.

class angr.storage.memory_mixins.**UltraPage**(*memory=None*, *init_zero=False*, ***kwargs*)

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*)

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.

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*)

merge(*others*, *merge_conditions*, *common_ancestor=None*, *page_addr=None*, *memory=None*, *changed_offsets=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** (`list[UltraPage]`) – 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
- **page_addr** (*int | None*)
- **changed_offsets** (*set[int] | None*)

Returns
True if the state plugins are actually merged.

Return type
bool

concrete_load(*addr*, *size*, *writing=False*, *with_bitmap=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**

Returns
A memoryview into the loaded bytes.

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**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

Bases: *PagedMemoryMixin*

PAGE_TYPE

alias of *UltraPage*

class angr.storage.memory_mixins.**UnderconstrainedMixin**(**args*, ***kwargs*)

Bases: MemoryMixin

__init__(**args*, ***kwargs*)

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.

load(*addr*, *size=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.**UnwrapperMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

This mixin processes SimActionObjects by passing on their .ast field.

Parameters

- **memory_id** (*str | None*)

- **endness** (*str*)

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*)

copy_contents(*dst*, *src*, *size*, *condition=None*, ***kwargs*)

Override this method to provide faster copying of large chunks of data.

Parameters

- **dst** – The destination of copying.
- **src** – The source of copying.
- **size** – The size of copying.
- **condition** – The storing condition.
- **kwargs** – Other parameters.

Returns

None

class angr.storage.memory_mixins.name_resolution_mixin.**NameResolutionMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

This mixin allows you to provide register names as load addresses, and will automatically translate this to an offset and size.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

store(*addr*, *data*, *size=None*, ***kwargs*)

load(*addr*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.smart_find_mixin.**SmartFindMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Memory mixin providing basic searching over concrete and symbolic data.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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.default_filler_mixin.**DefaultFillerMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

class angr.storage.memory_mixins.default_filler_mixin.**SpecialFillerMixin**(*special_memory_filler=None*, ***kwargs*)

Bases: MemoryMixin

__init__(*special_memory_filler=None*, ***kwargs*)

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.

class angr.storage.memory_mixins.default_filler_mixin.**ExplicitFillerMixin**(*uninitialized_read_handler=None*, ***kwargs*)

Bases: MemoryMixin

__init__(*uninitialized_read_handler=None*, ***kwargs*)

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.

class angr.storage.memory_mixins.bvv_conversion_mixin.**DataNormalizationMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Normalizes the data field for a store and the fallback field for a load to be BVs.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

store(*addr*, *data*, *size=None*, ***kwargs*)

load(*addr*, *size=None*, ***, *fallback=None*, ***kwargs*)

class angr.storage.memory_mixins.hex_dumper_mixin.**HexDumperMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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.underconstrained_mixin.**UnderconstrainedMixin**(**args*, ***kwargs*)

Bases: MemoryMixin

__init__(**args*, ***kwargs*)

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.

load(*addr*, *size=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.simple_interface_mixin.**SimpleInterfaceMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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.actions_mixin.**ActionsMixinHigh**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

load(*addr*, *size=None*, ***, *action=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***, *action=None*, ***kwargs*)

Parameters
action (*SimActionData | None*)

class angr.storage.memory_mixins.symbolic_merger_mixin.**SymbolicMergerMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)

- **endness** (*str*)

class angr.storage.memory_mixins.size_resolution_mixin.**SizeNormalizationMixin**(*memory_id=None*, *endness='Iend_BE'*)

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

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

load(*addr*, *size=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.size_resolution_mixin.**SizeConcretizationMixin**(*concretize_symbolic_write_size*, *max_concretize_count=256*, *max_symbolic_size=4194304*, *raise_memory_limit_error=False*, *size_limit=257*, ***kwargs*)

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

Parameters

- **concretize_symbolic_write_size** (*bool*)
- **max_concretize_count** (*int | None*)
- **max_symbolic_size** (*int*)
- **raise_memory_limit_error** (*bool*)
- **size_limit** (*int*)

__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*)

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.

load(*addr*, *size=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***, *condition=None*, ***kwargs*)

class angr.storage.memory_mixins.dirty_addrs_mixin.**DirtyAddrsMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: `MemoryMixin`

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

store(*addr*, *data*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.address_concretization_mixin.**MultiwriteAnnotation**

Bases: `Annotation`

property eliminatable

Returns whether this annotation can be eliminated in a simplification.

Returns

True if eliminatable, False otherwise

property relocateable

class angr.storage.memory_mixins.address_concretization_mixin.**AddressConcretizationMixin**(*read_strategies=N*
write_strategies=N
***kwargs*)

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*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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`

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*)

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**

Returns
A memoryview into the loaded bytes.

class angr.storage.memory_mixins.conditional_store_mixin.**ConditionalMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

load(*addr*, *size=None*, **, *condition=None*, *fallback=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, **, *condition=None*, ***kwargs*)

class angr.storage.memory_mixins.label_merger_mixin.**LabelMergerMixin**(**args*, ***kwargs*)

Bases: MemoryMixin

A memory mixin for merging labels. Labels come from SimLabeledMemoryObject.

__init__(**args*, ***kwargs*)

copy(*memo=None*)

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.

class angr.storage.memory_mixins.simplification_mixin.**SimplificationMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

store(*addr*, *data*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.unwrapper_mixin.**UnwrapperMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

This mixin processes SimActionObjects by passing on their .ast field.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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*)

copy_contents(*dst*, *src*, *size*, *condition=None*, ***kwargs*)

Override this method to provide faster copying of large chunks of data.

Parameters

- **dst** – The destination of copying.
- **src** – The source of copying.
- **size** – The size of copying.
- **condition** – The storing condition.

- **kwargs** – Other parameters.

Returns
None

class angr.storage.memory_mixins.convenient_mappings_mixin.**ConvenientMappingsMixin**(***kwargs*)

Bases: MemoryMixin

Implements mappings between names and hashes of symbolic variables and these variables themselves.

__init__(***kwargs*)

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.

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.paged_memory.pages.mv_list_page.**MVListPage**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

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*)

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.

Return type
MVListPage

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*)

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** – The address to start erasing.
- **size** – The number of bytes for erasing.

Return type
`None`

Returns
None

merge(*others*, *merge_conditions*, *common_ancestor=None*, ***, *page_addr*, *memory*, *changed_offsets=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** (list[*MVListPage*]) – 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
- **page_addr** (*int*)
- **memory** (*MemoryMixin*)
- **changed_offsets** (*set[int] | None*)

Returns
True if the state plugins are actually merged.

Return type
bool

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.multi_values.**MultiValues**(*v=None*, *offset_to_values=None*)

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.

Parameters

- **v** (*MVType | MultiValues[MVType] | None | dict[int, set[MVType]]*)
- **offset_to_values** (*dict[int, set[MVType]] | None*)

__init__(*v=None*, *offset_to_values=None*)

Parameters

- **v** (*MVType | MultiValues[MVType] | None | dict[int, set[MVType]]*)
- **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*])

class angr.storage.memory_mixins.top_merger_mixin.**TopMergerMixin**(**args*, *top_func=None*, ***kwargs*)

Bases: MemoryMixin

A memory mixin for merging values in memory to TOP.

__init__(**args*, *top_func=None*, ***kwargs*)

copy(*memo=None*)

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.

class angr.storage.memory_mixins.multi_value_merger_mixin.**MultiValueMergerMixin**(**args*, *element_limit=5*, *annotation_limit=256*, *top_func=None*, *is_top_func=None*, *phi_maker=None*, *merge_into_top=True*, ***kwargs*)

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*)

copy(*memo=None*)

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.

class angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**PagedMemoryMixin**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

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*)

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.

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*)

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** – The address to start erasing.
- **size** – The number of bytes for erasing.

Return type
None

Returns
None

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

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*)

concrete_load(*addr*, *size*, *writing=False*, **, *with_bitmap=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**
- **with_bitmap** (*bool*)

Returns
A memoryview into the loaded bytes.

changed_bytes(*other*)

Return type
set[int]

changed_pages(*other*)

Return type
dict[int, set[int] | None]

copy_contents(*dst*, *src*, *size*, *condition=None*, ***kwargs*)

Override this method to provide faster copying of large chunks of data.

Parameters

- **dst** – The destination of copying.
- **src** – The source of copying.
- **size** – The size of copying.
- **condition** – The storing condition.
- **kwargs** – Other parameters.

Returns
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**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

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**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

Bases: *PagedMemoryMixin*

PAGE_TYPE

alias of *ListPage*

class angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**MVListPagesMixin**(**args*, *skip_missing_values_during*
***kwargs*)

Bases: *PagedMemoryMixin*

PAGE_TYPE

alias of *MVListPage*

__init__(**args*, *skip_missing_values_during_merging=False*, ***kwargs*)

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.

Return type
MVListPagesMixin

class angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**ListPagesWithLabelsMixin**(*page_size=4096, default_permissions per-mis-sions_map=None page_kwargs=No **kwargs*)

Bases: *LabeledPagesMixin*, *ListPagesMixin*

class angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**MVListPagesWithLabelsMixin**(**args, skip_missing_v **kwargs*)

Bases: *LabeledPagesMixin*, *MVListPagesMixin*

class angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**UltraPagesMixin**(*page_size=4096, de-fault_permissions=3, permis-sions_map=None, page_kwargs=None, **kwargs*)

Bases: *PagedMemoryMixin*

PAGE_TYPE

alias of *UltraPage*

class angr.storage.memory_mixins.paged_memory.page_backer_mixins.**NotMemoryview**(*obj*, *offset*, *size*)

Bases: object

__init__(*obj*, *offset*, *size*)

class angr.storage.memory_mixins.paged_memory.page_backer_mixins.**ClemoryBackerMixin**(*cle_memory_backer=Non **kwargs*)

Bases: *PagedMemoryMixin*

Parameters
cle_memory_backer (*None | cle.Loader | cle.Clemory*)

__init__(*cle_memory_backer=None*, ***kwargs*)

Parameters
cle_memory_backer (*None | Loader | Clemory*)

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.

class angr.storage.memory_mixins.paged_memory.page_backer_mixins.**ConcreteBackerMixin**(*cle_memory_backer=N*
***kwargs*)

Bases: *ClemoryBackerMixin*

Parameters
cle_memory_backer (*None | cle.Loader | cle.Clemory*)

class angr.storage.memory_mixins.paged_memory.page_backer_mixins.**DictBackerMixin**(*dict_memory_backer=None*,
***kwargs*)

Bases: *PagedMemoryMixin*

__init__(*dict_memory_backer=None*, ***kwargs*)

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.

class angr.storage.memory_mixins.paged_memory.stack_allocation_mixin.**StackAllocationMixin**(*stack_end=None*,
stack_size=None,
stack_perms=No
***kwargs*)

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*)

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.

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.paged_memory.privileged_mixin.**PrivilegedPagingMixin**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

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.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**(**args*, ***kwargs*)

Bases: *RefcountMixin*, MemoryMixin

Tracks the history of memory writes.

__init__(**args*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***kwargs*)

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.

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.

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**(*memory_id=None*, *endness='Iend_BE'*)

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.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

class angr.storage.memory_mixins.paged_memory.pages.**ListPage**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

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*)

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.

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*)

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** – The address to start erasing.
- **size** – The number of bytes for erasing.

Return type
None

Returns
None

merge(*others*, *merge_conditions*, *common_ancestor=None*, *page_addr=None*, *memory=None*, *changed_offsets=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** (`list[ListPage]`) – 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
- **page_addr** (*int | None*)
- **changed_offsets** (*set[int] | None*)

Returns
True if the state plugins are actually merged.

Return type
bool

changed_bytes(*other*, *page_addr=None*)

Parameters

- **other** (ListPage)
- **page_addr** (*int | None*)

class angr.storage.memory_mixins.paged_memory.pages.**MVListPage**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

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*)

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.

Return type
MVListPage

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*)

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** – The address to start erasing.
- **size** – The number of bytes for erasing.

Return type
None

Returns
None

merge(*others*, *merge_conditions*, *common_ancestor=None*, *, *page_addr*, *memory*, *changed_offsets=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** (`list[MVListPage]`) – 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
- **page_addr** (*int*)
- **memory** (*MemoryMixin*)
- **changed_offsets** (*set[int] | None*)

Returns
True if the state plugins are actually merged.

Return type
bool

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**(**args*, ***kwargs*)

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**(*permissions=None*, ***kwargs*)

Bases: MemoryMixin

This mixin adds a permissions_bits field and properties for extracting the read/write/exec permissions. It does NOT add permissions checking.

Parameters
permissions (*int* | *claripy.ast.BV* | *None*)

__init__(*permissions=None*, ***kwargs*)

Parameters
permissions (*int* | *BV* | *None*)

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.

property perm_read

property perm_write

property perm_exec

class angr.storage.memory_mixins.paged_memory.pages.**RefcountMixin**(***kwargs*)

Bases: MemoryMixin

This mixin adds a locked reference counter and methods to manipulate it, to facilitate copy-on-write optimizations.

__init__(***kwargs*)

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.

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**(*memory=None*, *init_zero=False*, ***kwargs*)

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*)

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.

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*)

merge(*others*, *merge_conditions*, *common_ancestor=None*, *page_addr=None*, *memory=None*, *changed_offsets=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** (*list[UltraPage]*) – 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
- **page_addr** (*int | None*)
- **changed_offsets** (*set[int] | None*)

Returns
True if the state plugins are actually merged.

Return type
bool

concrete_load(*addr*, *size*, *writing=False*, *with_bitmap=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**

Returns
A memoryview into the loaded bytes.

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.paged_memory.pages.refcount_mixin.**RefcountMixin**(***kwargs*)

Bases: MemoryMixin

This mixin adds a locked reference counter and methods to manipulate it, to facilitate copy-on-write optimizations.

__init__(***kwargs*)

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.

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.permissions_mixin.**PermissionsMixin**(*permissions=None*, ***kwargs*)

Bases: MemoryMixin

This mixin adds a permissions_bits field and properties for extracting the read/write/exec permissions. It does NOT add permissions checking.

Parameters
permissions (*int | claripy.ast.BV | None*)

__init__(*permissions=None*, ***kwargs*)

Parameters
permissions (*int | BV | None*)

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.

property perm_read

property perm_write

property perm_exec

class angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.**HistoryTrackingMixin**(**args*, ***kwargs*)

Bases: *RefcountMixin*, MemoryMixin

Tracks the history of memory writes.

__init__(**args*, ***kwargs*)

store(*addr*, *data*, *size=None*, ***kwargs*)

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.

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.

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.ispo_mixin.**ISPOMixin**(*memory_id=None*, *endness='Iend_BE'*)

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.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

class angr.storage.memory_mixins.paged_memory.pages.list_page.**ListPage**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

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*)

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.

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*)

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** – The address to start erasing.
- **size** – The number of bytes for erasing.

Return type

None

Returns

None

merge(*others*, *merge_conditions*, *common_ancestor=None*, *page_addr=None*, *memory=None*, *changed_offsets=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** (`list[ListPage]`) – 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
- **page_addr** (*int | None*)
- **changed_offsets** (*set[int] | None*)

Returns
True if the state plugins are actually merged.

Return type
bool

changed_bytes(*other*, *page_addr=None*)

Parameters

- **other** (`ListPage`)
- **page_addr** (*int | None*)

class angr.storage.memory_mixins.paged_memory.pages.ultra_page.**UltraPage**(*memory=None*, *init_zero=False*, ***kwargs*)

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*)

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.

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*)

merge(*others*, *merge_conditions*, *common_ancestor=None*, *page_addr=None*, *memory=None*, *changed_offsets=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** (`list[UltraPage]`) – 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
- **page_addr** (*int | None*)
- **changed_offsets** (*set[int] | None*)

Returns
True if the state plugins are actually merged.

Return type
bool

concrete_load(*addr*, *size*, *writing=False*, *with_bitmap=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**

Returns
A memoryview into the loaded bytes.

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.regioned_memory.**AbstractMergerMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: MemoryMixin

AbstractMergerMixin handles merging initialized values.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

class angr.storage.memory_mixins.regioned_memory.**MemoryRegionMetaMixin**(*related_function_addr=None*, ***kwargs*)

Bases: MemoryMixin

__init__(*related_function_addr=None*, ***kwargs*)

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.

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*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

dbg_print(*indent=0*)

Print out debugging information

class angr.storage.memory_mixins.regioned_memory.**RegionCategoryMixin**(*memory_id=None*, *endness='Iend_BE'*)

Bases: `MemoryMixin`

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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**(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

Bases: `MemoryMixin`

__init__(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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`

class angr.storage.memory_mixins.regioned_memory.**RegionedMemoryMixin**(*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*)

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*)

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.

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*)

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`

find(*addr*, *data*, *max_search*, ***kwargs*)

Parameters
addr (*int | Bits*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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**(*memory_id=None*, *endness='Iend_BE'*)

Bases: *SmartFindMixin*

Implements data finding for abstract memory.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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.regioned_memory.regioned_memory_mixin.**RegionedMemoryMixin**(*write_targets_li*
read_targets_lin
stack_region_ma
*generic_region_*
stack_size=6553
cle_memory_ba
dict_memory_ba
re-
gioned_memory
***kwargs*)

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*)

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.

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*)

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`

find(*addr*, *data*, *max_search*, ***kwargs*)

Parameters
addr (*int | Bits*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.region_data.**AddressWrapper**(*region*, *region_base_addr*, *address*, *is_on_stack*, *function_address*)

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.

Parameters

- **region** (*str*)
- **region_base_addr** (*int*)
- **is_on_stack** (*bool*)
- **function_address** (*int | None*)

__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**(*region_id*, *base_address*, *related_function_address=None*)

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**(*is_stack*)

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.

class angr.storage.memory_mixins.regioned_memory.region_category_mixin.**RegionCategoryMixin**(*memory_id=No*
end-
ness='Iend_BE'

Bases: MemoryMixin

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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.static_find_mixin.**StaticFindMixin**(*memory_id=None*,
end-
ness='Iend_BE')

Bases: *SmartFindMixin*

Implements data finding for abstract memory.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

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.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()

class angr.storage.memory_mixins.regioned_memory.region_meta_mixin.**Segment**(*offset*, *size=0*)

Bases: object

Segment represents a continuous memory region.

__init__(*offset*, *size=0*)

class angr.storage.memory_mixins.regioned_memory.region_meta_mixin.**AbstractLocation**(*bbl_key*, *stmt_id*, *region_id*, *segment_list=None*, *region_offset=None*, *size=None*)

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**(*related_function_a*
***kwargs*)

Bases: MemoryMixin

__init__(*related_function_addr=None*, ***kwargs*)

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.

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*)

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

widen(*others*)

The widening operation for plugins. Widening is a special kind of merging that produces a more general state from several more specific states. It is used only during intensive static analysis. The same behavior regarding copying and mutation from `merge` should be followed.

Parameters
others – the other state plugins to widen with

Returns
True if the state plugin is actually widened.

Return type
bool

dbg_print(*indent=0*)

Print out debugging information

class angr.storage.memory_mixins.regioned_memory.abstract_merger_mixin.**AbstractMergerMixin**(*memory_id=No end-ness='Iend_BE'*

Bases: `MemoryMixin`

AbstractMergerMixin handles merging initialized values.

Parameters

- **memory_id** (*str | None*)
- **endness** (*str*)

class angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin.**RegionedAddressC**

Bases: `MemoryMixin`

__init__(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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`

class angr.storage.memory_mixins.slotted_memory.**SlottedMemoryMixin**(*width=None*, ***kwargs*)

Bases: `MemoryMixin`

__init__(*width=None*, ***kwargs*)

set_state(*state*)

Sets a new state (for example, if the state has been branched)

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.

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

load(*addr*, *size=None*, *, *endness=None*, ***kwargs*)

store(*addr*, *data*, *size=None*, *, *endness=None*, ***kwargs*)

changed_bytes(*other*)

10.6 Concretization Strategies

class angr.concretization_strategies.single.**SimConcretizationStrategySingle**(*filter=None*, *exact=True*)

Bases: *SimConcretizationStrategy*

Concretization strategy that ensures a single solution for an address.

class angr.concretization_strategies.eval.**SimConcretizationStrategyEval**(*limit*, ***kwargs*)

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.

__init__(*limit*, ***kwargs*)

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.

class angr.concretization_strategies.norepeats.**SimConcretizationStrategyNorepeats**(*repeat_expr*, *repeat_constraints=None*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses, without repeating.

__init__(*repeat_expr*, *repeat_constraints=None*, ***kwargs*)

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.

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.solutions.**SimConcretizationStrategySolutions**(*limit*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves an address into some limited number of solutions.

__init__(*limit*, ***kwargs*)

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.

class angr.concretization_strategies.nonzero_range.**SimConcretizationStrategyNonzeroRange**(*limit*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves a range in a non-zero location.

__init__(*limit*, ***kwargs*)

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.

class angr.concretization_strategies.range.**SimConcretizationStrategyRange**(*limit*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses to a range.

__init__(*limit*, ***kwargs*)

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.

class angr.concretization_strategies.max.**SimConcretizationStrategyMax**(*max_addr=None*)

Bases: *SimConcretizationStrategy*

Concretization strategy that returns the maximum address.

Parameters

max_addr (*int | None*)

__init__(*max_addr=None*)

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.
- **max_addr** (*int | None*)

class angr.concretization_strategies.norepeats_range.**SimConcretizationStrategyNorepeatsRange**(*repeat_expr*, *min=None*, *granularity=None*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves a range, with no repeats.

__init__(*repeat_expr*, *min=None*, *granularity=None*, ***kwargs*)

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.

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.nonzero.**SimConcretizationStrategyNonzero**(*filter=None*, *exact=True*)

Bases: *SimConcretizationStrategy*

Concretization strategy that returns any non-zero solution.

class angr.concretization_strategies.any.**SimConcretizationStrategyAny**(*filter=None*, *exact=True*)

Bases: *SimConcretizationStrategy*

Concretization strategy that returns any single solution.

class angr.concretization_strategies.controlled_data.**SimConcretizationStrategyControlledData**(*limit*, *fixed_addrs*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that constraints the address to controlled data. Controlled data consists of symbolic data and the addresses given as arguments. memory.

__init__(*limit*, *fixed_addrs*, ***kwargs*)

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.

class angr.concretization_strategies.unlimited_range.**SimConcretizationStrategyUnlimitedRange**(*limit*, ***kwargs*)

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses to a range without checking if the number of possible addresses is within the limit.

__init__(*limit*, ***kwargs*)

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.

10.7 Simulation Manager

class angr.sim_manager.**SimulationManager**(*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*)

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**(*state*, *error*, *traceback*)

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.

Parameters

- **state** (SimState)
- **error** (*Exception*)
- **traceback** (*TracebackType*)

__init__(*state*, *error*, *traceback*)

state: *SimState*

error: Exception

traceback: TracebackType

debug()

Launch a postmortem debug shell at the site of the error.

reraise()

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)

10.8 Exploration Techniques

class angr.exploration_techniques.**DFS**(*deferred_stash='deferred'*)

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'*)

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*)

class angr.exploration_techniques.**Bucketizer**

Bases: *ExplorationTechnique*

Loop bucketization: Pick log(n) paths out of n possible paths, and stash (or drop) everything else.

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)

class angr.exploration_techniques.**CallFunctionGoal**(*function*, *arguments*)

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**(*peek_blocks=100*, *peek_functions=5*, *goals=None*, *cfg_keep_states=False*, *goal_satisfied_callback=None*, *num_fallback_states=5*)

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**(*trace*, *fuzz_bitmap=None*)

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.

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*)

class angr.exploration_techniques.**ExecuteAddressGoal**(*addr*)

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**(*find=None*, *avoid=None*, *find_stash='found'*, *avoid_stash='avoid'*, *cfg=None*, *num_find=1*, *avoid_priority=False*)

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*)

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)

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.**LengthLimiter**(*max_length*, *drop=False*)

Bases: *ExplorationTechnique*

Length limiter on paths.

__init__(*max_length*, *drop=False*)

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*)

class angr.exploration_techniques.**LocalLoopSeer**(*bound=None*, *bound_reached=None*, *discard_stash='spinning'*)

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.

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

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)

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)

class angr.exploration_techniques.**LoopSeer**(*cfg=None*, *functions=None*, *loops=None*, *use_header=False*, *bound=None*, *bound_reached=None*, *discard_stash='spinning'*, *limit_concrete_loops=True*)

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)

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

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)

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)

class angr.exploration_techniques.**ManualMergepoint**(*address*, *wait_counter=10*, *prune=True*)

Bases: *ExplorationTechnique*

__init__(*address*, *wait_counter=10*, *prune=True*)

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

mark_nofilter(*simgr*, *stash*)

mark_okfilter(*simgr*, *stash*)

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*)

class angr.exploration_techniques.**MemoryWatcher**(*min_memory=512*, *memory_stash='lowmem'*)

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'*)

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*)

class angr.exploration_techniques.**Oppologist**

Bases: *ExplorationTechnique*

The Oppologist is an exploration technique that forces uncooperative code through qemu.

__init__()

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)

class angr.exploration_techniques.**Slicecutor**(*annotated_cfg*, *force_taking_exit=False*, *force_sat=False*)

Bases: *ExplorationTechnique*

The Slicecutor is an exploration that executes provided code slices.

Parameters
force_sat (*bool*)

__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.

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

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)

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)

class angr.exploration_techniques.**Spiller**(*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*)

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.

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*)

static state_priority(*state*)

class angr.exploration_techniques.**StochasticSearch**(*start_state*, *restart_prob=0.0001*)

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).

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*)

class angr.exploration_techniques.**StubStasher**

Bases: *ExplorationTechnique*

Stash states that reach a stub SimProcedure.

static post_filter(*state*)

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*)

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__()

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*)

static report(*state*, *event*)

class angr.exploration_techniques.**TechniqueBuilder**(*setup=None*, *step_state=None*, *step=None*, *successors=None*, *filter=None*, *selector=None*, *complete=None*)

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**(*threads=8*, *local_stash='thread_local'*)

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'*)

step(*simgr*, *stash='active'*, *error_list=None*, *target_stash=None*, ***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*)

inner_step(*state*, *simgr*, ***kwargs*)

class angr.exploration_techniques.**Timeout**(*timeout=None*)

Bases: *ExplorationTechnique*

Timeout exploration technique that stops an active exploration if the run time exceeds a predefined timeout

__init__(*timeout=None*)

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*)

class angr.exploration_techniques.**Tracer**(*trace=None*, *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*)

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** – 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=None*, *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*)

set_fd_data(*fd_data*)

Set concrete bytes of various fds read by the program

Parameters

fd_data (*dict[int, bytes]*)

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

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)

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)

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*)

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)

classmethod crash_windup(*state*, *crash_addr*)

class angr.exploration_techniques.**UniqueSearch**(*similarity_func=None*, *deferred_stash='deferred'*)

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.

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*)

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**(***options*)

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*)

step_state(*simgr*, *state*, *successor_func=None*, ***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)

class angr.exploration_techniques.timeout.**Timeout**(*timeout=None*)

Bases: *ExplorationTechnique*

Timeout exploration technique that stops an active exploration if the run time exceeds a predefined timeout

__init__(*timeout=None*)

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*)

class angr.exploration_techniques.dfs.**DFS**(*deferred_stash='deferred'*)

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'*)

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*)

class angr.exploration_techniques.explorer.**Explorer**(*find=None*, *avoid=None*, *find_stash='found'*, *avoid_stash='avoid'*, *cfg=None*, *num_find=1*, *avoid_priority=False*)

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*)

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)

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.lengthlimiter.**LengthLimiter**(*max_length*, *drop=False*)

Bases: *ExplorationTechnique*

Length limiter on paths.

__init__(*max_length*, *drop=False*)

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*)

class angr.exploration_techniques.manual_mergepoint.**ManualMergepoint**(*address*, *wait_counter=10*, *prune=True*)

Bases: *ExplorationTechnique*

__init__(*address*, *wait_counter=10*, *prune=True*)

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

mark_nofilter(*simgr*, *stash*)

mark_okfilter(*simgr*, *stash*)

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*)

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__()

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.**PickledStatesDb**(*db_str='sqlite:///:memory:'*)

Bases: *PickledStatesBase*

Database-backed pickled state storage.

__init__(*db_str='sqlite:///:memory:'*)

sort()

Sort pickled states.

add(*prio*, *sid*, *taken=False*, *stash='spilled'*)

Add a newly pickled state.

Parameters

- **prio** (*int*) – Priority of the state.
- **sid** (*str*) – Persistent ID of the state.

Returns
None

pop_n(*n*, *stash='spilled'*)

Pop the top N states.

Parameters
n (*int*) – Number of states to take.

Returns
A list of states.

get_recent_n(*n*, *stash='spilled'*)

count()

class angr.exploration_techniques.spiller.**Spiller**(*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*)

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.

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*)

static state_priority(*state*)

class angr.exploration_techniques.spiller_db.**PickledState**(***kwargs*)

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.

class angr.exploration_techniques.threading.**Threading**(*threads=8*, *local_stash='thread_local'*)

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'*)

step(*simgr*, *stash='active'*, *error_list=None*, *target_stash=None*, ***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*)

inner_step(*state*, *simgr*, ***kwargs*)

class angr.exploration_techniques.veritesting.**Veritesting**(***options*)

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*)

step_state(*simgr*, *state*, *successor_func=None*, ***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)

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**(*msg*, *deviating_addr=None*, *deviating_trace_idx=None*)

Bases: *AngrTracerError*

An error class to report tracing Tracing desyncronization error

__init__(*msg*, *deviating_addr=None*, *deviating_trace_idx=None*)

class angr.exploration_techniques.tracer.**RepHook**(*mnemonic*)

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**(*trace=None*, *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*)

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** – 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=None*, *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*)

predecessors: list[*SimState*]

set_fd_data(*fd_data*)

Set concrete bytes of various fds read by the program

Parameters

fd_data (*dict[int, bytes]*)

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

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)

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)

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*)

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)

classmethod crash_windup(*state*, *crash_addr*)

class angr.exploration_techniques.driller_core.**DrillerCore**(*trace*, *fuzz_bitmap=None*)

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.

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*)

class angr.exploration_techniques.slicecutor.**Slicecutor**(*annotated_cfg*, *force_taking_exit=False*, *force_sat=False*)

Bases: *ExplorationTechnique*

The Slicecutor is an exploration that executes provided code slices.

Parameters
force_sat (*bool*)

__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.

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

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)

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)

class angr.exploration_techniques.director.**BaseGoal**(*sort*)

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**(*addr*)

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**(*function*, *arguments*)

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**(*peek_blocks=100*, *peek_functions=5*, *goals=None*, *cfg_keep_states=False*, *goal_satisfied_callback=None*, *num_fallback_states=5*)

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.oppologist.**Oppologist**

Bases: *ExplorationTechnique*

The Oppologist is an exploration technique that forces uncooperative code through qemu.

__init__()

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)

class angr.exploration_techniques.loop_seer.**LoopSeer**(*cfg=None*, *functions=None*, *loops=None*, *use_header=False*, *bound=None*, *bound_reached=None*, *discard_stash='spinning'*, *limit_concrete_loops=True*)

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)

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

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)

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)

class angr.exploration_techniques.local_loop_seer.**LocalLoopSeer**(*bound=None*, *bound_reached=None*, *discard_stash='spinning'*)

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.

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

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)

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)

class angr.exploration_techniques.stochastic.**StochasticSearch**(*start_state*, *restart_prob=0.0001*)

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).

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*)

class angr.exploration_techniques.unique.**UniqueSearch**(*similarity_func=None*, *deferred_stash='deferred'*)

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.

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*)

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.tech_builder.**TechniqueBuilder**(*setup=None*, *step_state=None*, *step=None*, *successors=None*, *filter=None*, *selector=None*, *complete=None*)

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*)

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.

class angr.exploration_techniques.memory_watcher.**MemoryWatcher**(*min_memory=512*, *memory_stash='lowmem'*)

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'*)

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*)

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.

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)

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__()

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*)

static report(*state*, *event*)

10.9 Simulation Engines

class angr.engines.**HeavyResilienceMixin**(*project*, ***kwargs*)

Bases: *VEXResilienceMixin*, *ClaripyDataMixin*

class angr.engines.**HeavyVEXMixin**(*project*)

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)

Parameters

project (angr.Project)

process_successors(*successors*, *irsb=None*, *insn_bytes=None*, *thumb=False*, *size=None*, *num_inst=None*, *extra_stop_points=None*, *opt_level=None*, *strict_block_end=None*, ***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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.**HooksMixin**(*project*)

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

Parameters
project (angr.Project)

process_successors(*successors*, *, *procedure=None*, **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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.**ProcedureEngine**(*project*)

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.

Parameters
project (angr.Project)

process_successors(*successors*, *procedure=None*, **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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

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**(*project*)

Bases: Generic

A SimEngine is a type which understands how to perform execution on a state.

Parameters
project (angr.Project)

state: StateType

__init__(*project*)

Parameters
project (Project)

class angr.engines.**SimEngineFailure**(*project*)

Bases: *SuccessorsEngine*, *ProcedureMixin*

Parameters
project (angr.Project)

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.**SimEngineSyscall**(*project*)

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.

Parameters
project (angr.Project)

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.**SimEngineUnicorn**(*project*)

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

Parameters
project (angr.Project)

__init__(*project*)

Parameters
project (Project)

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.**SimInspectMixin**(*project*, ***kwargs*)

Bases: *VEXMixin*

handle_vex_block(*irsb*)

class angr.engines.**SimSuccessors**(*addr*, *initial_state*)

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

Parameters

- **addr** (*int | SootAddressDescriptor | None*)
- **initial_state** (HeavyState *| None*)

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[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – The "normal" successors. IP may be symbolic, but must have reasonable number of solutions
- **unsat_successors** (list[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – Any successor which is unsatisfiable after its guard condition is added.
- **all_successors** (list[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – successors + unsat_successors
- **flat_successors** (list[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – 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[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – Any state for which during the flattening process we find too many solutions.

Parameters

- **addr** (*int | SootAddressDescriptor | None*)
- **initial_state** (HeavyState *| None*)

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**(*project*)

Bases: *SuccessorsEngine*, *ProcedureMixin*

Execution engine based on Soot.

Parameters
project (angr.Project)

lift_soot(*addr=None*, *the_binary=None*, ***kwargs*)

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

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**(*project*)

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.

Parameters
project (angr.Project)

__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** (SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]) – 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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

Bases: *HeavyVEXMixin*

__init__(**args*, ***kwargs*)

handle_vex_block(*irsb*)

class angr.engines.**UberEngine**(*project*)

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.

Parameters
project (angr.Project)

class angr.engines.**UberEnginePcode**(**args*, ***kwargs*)

Bases: *SimEngineFailure*, *SimEngineSyscall*, *HooksMixin*, *HeavyPcodeMixin*

class angr.engines.engine.**SimEngine**(*project*)

Bases: Generic

A SimEngine is a type which understands how to perform execution on a state.

Parameters
project (angr.Project)

state: **StateType**

__init__(*project*)

Parameters
project (Project)

class angr.engines.successors.**SimSuccessors**(*addr*, *initial_state*)

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

Parameters

- **addr** (*int | SootAddressDescriptor | None*)
- **initial_state** (HeavyState | *None*)

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[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – The "normal" successors. IP may be symbolic, but must have reasonable number of solutions
- **unsat_successors** (list[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – Any successor which is unsatisfiable after its guard condition is added.
- **all_successors** (list[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – successors + unsat_successors
- **flat_successors** (list[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – 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[SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]]) – Any state for which during the flattening process we find too many solutions.

Parameters

- **addr** (*int | SootAddressDescriptor | None*)
- **initial_state** (HeavyState | *None*)

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**(*project*)

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.

Parameters
project (angr.Project)

__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** (SimState[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]) – 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.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**(*project*)

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.

Parameters
project (angr.Project)

process_successors(*successors*, *procedure=None*, ***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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.hook.**HooksMixin**(*project*)

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

Parameters
project (angr.Project)

process_successors(*successors*, *, *procedure=None*, **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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.syscall.**SimEngineSyscall**(*project*)

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.

Parameters
project (angr.Project)

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.failure.**SimEngineFailure**(*project*)

Bases: *SuccessorsEngine*, *ProcedureMixin*

Parameters
project (angr.Project)

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.vex.**ClaripyDataMixin**(*project*, ***kwargs*)

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**(*project*, ***kwargs*)

Bases: *VEXResilienceMixin*, *ClaripyDataMixin*

class angr.engines.vex.**HeavyVEXMixin**(*project*)

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)

Parameters

project (angr.Project)

process_successors(*successors*, *irsb=None*, *insn_bytes=None*, *thumb=False*, *size=None*, *num_inst=None*, *extra_stop_points=None*, *opt_level=None*, *strict_block_end=None*, ***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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.vex.**SimInspectMixin**(*project*, ***kwargs*)

Bases: *VEXMixin*

handle_vex_block(*irsb*)

class angr.engines.vex.**SuperFastpathMixin**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

Bases: *HeavyVEXMixin*

__init__(**args*, ***kwargs*)

handle_vex_block(*irsb*)

class angr.engines.vex.**VEXLifter**(*project*, *use_cache=None*, *cache_size=5000*, *default_opt_level=1*, *selfmodifying_code=None*, *single_step=False*, *default_strict_block_end=False*, ***kwargs*)

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**(*project*, ***kwargs*)

Bases: *SimEngine*

__init__(*project*, ***kwargs*)

handle_vex_block(*irsb*)

Parameters
irsb (*IRSB*)

class angr.engines.vex.**VEXResilienceMixin**(*project*, ***kwargs*)

Bases: *VEXMixin*

class angr.engines.vex.**VEXSlicingMixin**(**args*, ***kwargs*)

Bases: *VEXMixin*

__init__(**args*, ***kwargs*)

process(*state*, *block=None*, *skip_stmts=0*, *last_stmt=None*, *whitelist=None*, ***kwargs*)

handle_vex_block(*irsb*)

class angr.engines.soot.**SootMixin**(*project*)

Bases: *SuccessorsEngine*, *ProcedureMixin*

Execution engine based on Soot.

Parameters
project (angr.Project)

lift_soot(*addr=None*, *the_binary=None*, ***kwargs*)

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

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.soot.engine.**SootMixin**(*project*)

Bases: *SuccessorsEngine*, *ProcedureMixin*

Execution engine based on Soot.

Parameters

project (angr.Project)

lift_soot(*addr=None*, *the_binary=None*, ***kwargs*)

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

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.unicorn.**SimEngineUnicorn**(*project*)

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

Parameters
project (angr.Project)

__init__(*project*)

Parameters
project (Project)

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

class angr.engines.pcode.**HeavyPcodeMixin**(**args*, ***kwargs*)

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*)

process_successors(*successors*, *irsb=None*, *insn_bytes=None*, *thumb=False*, *size=None*, *num_inst=None*, *extra_stop_points=None*, ***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.
- **irsb** (*IRSB | None*)
- **insn_bytes** (*bytes | None*)
- **thumb** (*bool*)
- **size** (*int | None*)
- **num_inst** (*int | None*)
- **extra_stop_points** (*Iterable[int] | None*)

Return type
None

angr.engines.pcode.**register_pcode_arch_default_cc**(*arch*)

Parameters
arch (*ArchPcode*)

class angr.engines.pcode.engine.**HeavyPcodeMixin**(**args*, ***kwargs*)

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*)

process_successors(*successors*, *irsb=None*, *insn_bytes=None*, *thumb=False*, *size=None*, *num_inst=None*, *extra_stop_points=None*, ***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.
- **irsb** (*IRSB | None*)
- **insn_bytes** (*bytes | None*)
- **thumb** (*bool*)
- **size** (*int | None*)
- **num_inst** (*int | None*)
- **extra_stop_points** (*Iterable[int] | None*)

Return type
None

class angr.engines.pcode.lifter.**ExitStatement**(*dst*, *jumpkind*)

Bases: object

This class exists to ease compatibility with CFGFast's processing of exit_statements. See _scan_irsb method.

Parameters

- **dst** (*int | None*)
- **jumpkind** (*str*)

__init__(*dst*, *jumpkind*)

Parameters

- **dst** (*int | None*)
- **jumpkind** (*str*)

dst: int | None

jumpkind: str

class angr.engines.pcode.lifter.**PcodeDisassemblerBlock**(*addr*, *insns*, *thumb*, *arch*)

Bases: *DisassemblerBlock*

Helper class to represent a block of disassembled target architecture instructions

addr

arch

insns

thumb

class angr.engines.pcode.lifter.**PcodeDisassemblerInsn**(*pcode_insn*)

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**(*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*)

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

Parameters

- **data** (*str* | bytes | *None*)
- **mem_addr** (*int*)
- **arch** (*Arch*)
- **max_inst** (*int* | *None*)

- **max_bytes** (*int | None*)
- **bytes_offset** (*int*)
- **traceflags** (*int*)
- **opt_level** (*int*)
- **num_inst** (*int | None*)
- **num_bytes** (*int | None*)
- **strict_block_end** (*bool*)
- **skip_stmts** (*bool*)
- **collect_data_refs** (*bool*)

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**(*arch*, *addr*)

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

Parameters

- **arch** (*Arch*)
- **addr** (*int*)

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*)

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
- **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**(*arch*)

Bases: object

Lifts basic blocks to P-code

Parameters
arch (*archinfo.Arch*)

__init__(*arch*)

Parameters
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**(*arch*, *addr*)

Bases: *Lifter*

Handles calling into pypcode to lift a block

Parameters

- **arch** (*Arch*)
- **addr** (*int*)

addr: int

allow_arch_optimizations: bool | None

arch: Arch

bytes_offset: int | None

collect_data_refs: bool

data: str | *bytes* | None

irsb: *IRSB*

max_bytes: int | None

max_inst: int | None

opt_level: int

skip_stmts: bool

strict_block_end: bool | None

traceflags: int | None

classmethod get_lifter(*arch*)

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

class angr.engines.pcode.lifter.**PcodeLifterEngineMixin**(*project=None*, *use_cache=None*, *cache_size=50000*, *default_opt_level=1*, *selfmodifying_code=None*, *single_step=False*, *default_strict_block_end=False*, ***kwargs*)

Bases: *SimEngine*

Lifter mixin to lift from machine code to P-Code.

Parameters

- **use_cache** (*bool | None*)
- **cache_size** (*int*)
- **default_opt_level** (*int*)
- **selfmodifying_code** (*bool | None*)
- **single_step** (*bool*)

- **default_strict_block_end** (*bool*)

__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

class angr.engines.pcode.emulate.**PcodeEmulatorMixin**(**args*, ***kwargs*)

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.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**(*opcode*, *is_unary*, *is_special=False*)

Bases: object

Base class for all operation behaviors.

Parameters

- **opcode** (*OpCode*)
- **is_unary** (*bool*)
- **is_special** (*bool*)

__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*)

class angr.engines.pcode.cc.**SimCCPCodeBase**(*arch*)

Bases: *SimCC*

Base class for all pcode calling conventions.

Parameters
arch (*archinfo.Arch*)

LANGUAGE = None

classmethod ARCH()

class angr.engines.pcode.cc.**SimCCM68k**(*arch*)

Bases: *SimCCPCodeBase*

Default CC for M68k

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCPCodeBase*

Default CC for RISCV

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCPCodeBase*

Default CC for SPARC

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCPCodeBase*

Default CC for SH4

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCPCodeBase*

Default CC for PARISC

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCPCodeBase*

Default CC for PowerPC

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCPCodeBase*

Default CC for Xtensa

Parameters
arch (*archinfo.Arch*)

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*)

10.10 Simulation Logging

class angr.state_plugins.sim_action.**SimAction**(*state*, *region_type*)

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**(*state*, *target*, *condition=None*, *exit_type=None*)

Bases: *SimAction*

An Exit action represents a (possibly conditional) jump.

CONDITIONAL = 'conditional'

DEFAULT = 'default'

__init__(*state*, *target*, *condition=None*, *exit_type=None*)

Initializes the SimAction.

Parameters
state – the state that's the SimAction is taking place in.

property all_objects

property is_symbolic

class angr.state_plugins.sim_action.**SimActionConstraint**(*state*, *constraint*, *condition=None*)

Bases: *SimAction*

A constraint action represents an extra constraint added during execution of a path.

__init__(*state*, *constraint*, *condition=None*)

Initializes the SimAction.

Parameters
state – the state that's the SimAction is taking place in.

property all_objects

property is_symbolic

class angr.state_plugins.sim_action.**SimActionOperation**(*state*, *op*, *exprs*, *result*)

Bases: *SimAction*

An action representing an operation between variables and/or constants.

__init__(*state*, *op*, *exprs*, *result*)

Initializes the SimAction.

Parameters
state – the state that's the SimAction is taking place in.

property all_objects

property is_symbolic

class angr.state_plugins.sim_action.**SimActionData**(*state*, *region_type*, *action*, *tmp=None*, *addr=None*, *size=None*, *data=None*, *condition=None*, *fallback=None*, *fd=None*)

Bases: *SimAction*

A Data action represents a read or a write from memory, registers or a file.

READ = 'read'

WRITE = 'write'

OPERATE = 'operate'

__init__(*state*, *region_type*, *action*, *tmp=None*, *addr=None*, *size=None*, *data=None*, *condition=None*, *fallback=None*, *fd=None*)

Initializes the SimAction.

Parameters

state – the state that's the SimAction is taking place in.

downsize()

Clears some low-level details (that take up memory) out of the SimAction.

property all_objects

property is_symbolic

property tmp_deps

property reg_deps

property storage

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**(*ast*, *reg_deps=frozenset({})*, *tmp_deps=frozenset({})*, *deps=frozenset({})*, *state=None*)

Bases: object

A SimActionObject tracks an AST and its dependencies.

Parameters

- **ast** (*Base*)
- **reg_deps** (*frozenset[SimActionData | SimActionOperation]*)
- **tmp_deps** (*frozenset[SimActionData | SimActionOperation]*)
- **deps** (*frozenset*)
- **state** (*SimState | None*)

__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[Base | bool | int | float | str | FSort | tuple[Base | bool | int | float | str | FSort | tuple[ArgType] | None] | None, ...]

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.sim_event.**SimEvent**(*state*, *event_type*, ***kwargs*)

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*)

10.11 Procedures

class angr.sim_procedure.**SimProcedure**(*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*)

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
- **project** (angr.Project | *None*)
- **prototype** (*str* | SimTypeFunction | *None*)

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.

Parameters

- **project** (angr.Project | *None*)
- **cc** (angr.SimCC | *None*)
- **prototype** (*str* | SimTypeFunction | *None*)

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`

Parameters

- **project** (angr.Project | *None*)
- **cc** (angr.SimCC | *None*)
- **prototype** (*str* | SimTypeFunction | *None*)

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** (None | *ArgSession* | int) – 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.

Parameters

- **project** (angr.Project | *None*)
- **cc** (angr.SimCC | *None*)
- **prototype** (*str* | SimTypeFunction | *None*)

__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: None | *ArgSession* | int

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

class angr.procedures.stubs.format_parser.**FormatString**(*parser*, *components*)

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**(*sss*, *length_spec*, *pad_chr*, *ty*)

Bases: object

Describes a format specifier within a format string.

Parameters
ty (SimType)

__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**(*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*)

Bases: *SimProcedure*

For SimProcedures relying on printf-style format strings.

Parameters

- **project** (angr.Project | *None*)
- **cc** (angr.SimCC | *None*)
- **prototype** (*str* | SimTypeFunction | *None*)

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)}

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**(*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*)

Bases: *FormatParser*

For SimProcedures relying on scanf-style format strings.

Parameters

- **project** (angr.Project | *None*)
- **cc** (angr.SimCC | *None*)
- **prototype** (*str* | SimTypeFunction | *None*)

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_spec = [b'e', b'E', b'f', b'F', b'g', b'G', b'a', b'A']

float_len_mod = {b'l': <class 'angr.sim_type.SimTypeDouble'>, b'll': <class 'angr.sim_type.SimTypeDouble'>}

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.

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__()

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 – The other SimLibrary

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.

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**()

10.12 Calling Conventions and Types

class angr.calling_conventions.**PointerWrapper**(*value*, *buffer=False*)

Bases: object

__init__(*value*, *buffer=False*)

class angr.calling_conventions.**AllocHelper**(*ptrsize*)

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**(*lst*)

Bases: *SerializableIterator*

__init__(*lst*)

getstate()

setstate(*state*)

class angr.calling_conventions.**SerializableCounter**(*start*, *stride*, *mapping=<function SerializableCounter.<lambda>>*)

Bases: *SerializableIterator*

__init__(*start*, *stride*, *mapping=<function SerializableCounter.<lambda>>*)

getstate()

setstate(*state*)

class angr.calling_conventions.**SimFunctionArgument**(*size*, *is_fp=False*)

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

Parameters

- **size** (*int*)
- **is_fp** (*bool*)

__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**(*reg_name*, *size*, *reg_offset=0*, *is_fp=False*, *clear_entire_reg=False*)

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

Parameters

- **reg_name** (*RegisterName*)
- **size** (*int*)

__init__(*reg_name*, *size*, *reg_offset=0*, *is_fp=False*, *clear_entire_reg=False*)

Parameters

- **reg_name** (*str*)
- **size** (*int*)

get_footprint()

Return a list of SimRegArg and SimStackArgs that are the base components used for this location

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**(*stack_offset*, *size*, *is_fp=False*)

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

Parameters

- **stack_offset** (*int*)
- **size** (*int*)
- **is_fp** (*bool*)

__init__(*stack_offset*, *size*, *is_fp=False*)

Parameters

- **stack_offset** (*int*)
- **size** (*int*)
- **is_fp** (*bool*)

stack_offset: *int*

get_footprint()

Return a list of SimRegArg and SimStackArgs that are the base components used for this location

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**(*locations*, *is_fp=False*)

Bases: *SimFunctionArgument*, *Generic*

An argument which spans multiple storage locations. Locations should be given least-significant first.

Parameters

locations (*list[T]*)

__init__(*locations*, *is_fp=False*)

Parameters

locations (*list[T]*)

get_footprint()

Return a list of SimRegArg and SimStackArgs that are the base components used for this location

set_value(*state*, *value*, ***kwargs*)

get_value(*state*, ***kwargs*)

class angr.calling_conventions.**SimStructArg**(*struct*, *locs*)

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

Parameters

- **struct** (SimStruct)
- **locs** (*dict[str, SimFunctionArgument]*)

__init__(*struct*, *locs*)

Parameters

- **struct** (SimStruct)
- **locs** (*dict[str, SimFunctionArgument]*)

get_footprint()

Return a list of SimRegArg and SimStackArgs that are the base components used for this location

get_single_footprint()

Return type
SimStackArg | *SimRegArg* | *SimComboArg*

get_value(*state*, ***kwargs*)

set_value(*state*, *value*, ***kwargs*)

class angr.calling_conventions.**SimArrayArg**(*locs*)

Bases: *SimFunctionArgument*

__init__(*locs*)

get_footprint()

Return a list of SimRegArg and SimStackArgs that are the base components used for this location

get_value(*state*, ***kwargs*)

set_value(*state*, *value*, ***kwargs*)

class angr.calling_conventions.**SimReferenceArgument**(*ptr_loc*, *main_loc*)

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)`

Parameters

- **ptr_loc** (SimFunctionArgument)
- **main_loc** (SimFunctionArgument)

__init__(*ptr_loc*, *main_loc*)

Parameters

- **ptr_loc** (SimFunctionArgument)
- **main_loc** (SimFunctionArgument)

get_footprint()

Return a list of SimRegArg and SimStackArgs that are the base components used for this location

get_value(*state*, ***kwargs*)

set_value(*state*, *value*, ***kwargs*)

class angr.calling_conventions.**ArgSession**(*cc*)

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**(*cc*)

Bases: object

An argsession for use with SimCCUsercall

__init__(*cc*)

cc

real_args

getstate()

setstate(*state*)

class angr.calling_conventions.**SimCC**(*arch*)

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.

Parameters

arch (*archinfo.Arch*)

__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

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(*cc*)

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*)

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.
- **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**(*name*, *size=8*)

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**(*arch*, *args*, *ret_loc*)

Bases: *SimCC*

__init__(*arch*, *args*, *ret_loc*)

Parameters
arch – The Archinfo arch for this CC

ArgSession

alias of *UsercallArgSession*

next_arg(*session*, *arg_type*)

Parameters
session (UsercallArgSession)

return_val(*ty*, ***kwargs*)

The location the return value is stored, based on its type.

class angr.calling_conventions.**SimCCCdecl**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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_val(*ty*, *perspective_returned=False*)

The location the return value is stored, based on its type.

return_in_implicit_outparam(*ty*)

class angr.calling_conventions.**SimCCMicrosoftCdecl**(*arch*)

Bases: *SimCCCdecl*

Parameters
arch (*archinfo.Arch*)

STRUCT_RETURN_THRESHOLD = 64

class angr.calling_conventions.**SimCCMicrosoftThiscall**(*arch*)

Bases: *SimCCCdecl*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCMicrosoftCdecl*

Parameters
arch (*archinfo.Arch*)

CALLEE_CLEANUP = True

class angr.calling_conventions.**SimCCMicrosoftFastcall**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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

class angr.calling_conventions.**MicrosoftAMD64ArgSession**(*cc*)

Bases: *ArgSession*

both_iter

cc

fp_iter

int_iter

class angr.calling_conventions.**SimCCMicrosoftAMD64**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

ARG_REGS: list[str] = ['rcx', 'rdx', 'r8', 'r9']

FP_ARG_REGS: list[str] = ['xmm0', 'xmm1', 'xmm2', 'xmm3']

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*)

return_val(*ty*, *perspective_returned=False*)

The location the return value is stored, based on its type.

class angr.calling_conventions.**SimCCSyscall**(*arch*)

Bases: *SimCC*

The base class of all syscall CCs.

Parameters

arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCSyscall*

Parameters

arch (*archinfo.Arch*)

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(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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_val(*ty*, *perspective_returned=False*)

The location the return value is stored, based on its type.

Parameters
ty (SimType | *None*)

return_in_implicit_outparam(*ty*)

class angr.calling_conventions.**SimCCAMD64LinuxSyscall**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCARM*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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.SimCC032(*arch*)

Bases: *SimCC*

Parameters

arch (*archinfo.Arch*)

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.SimCC032LinuxSyscall(*arch*)

Bases: *SimCCSyscall*

Parameters

arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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.**SimCCN64LinuxSyscall**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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.**SimCCPowerPC**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

ARCH

alias of ArchSoot

ARG_REGS: list[str] = []

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.

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.

class angr.calling_conventions.**SimCCUnknown**(*arch*)

Bases: *SimCC*

Represent an unknown calling convention.

Parameters
arch (*archinfo.Arch*)

class angr.calling_conventions.**SimCCS390X**(*arch*)

Bases: *SimCC*

Parameters
arch (*archinfo.Arch*)

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**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*archinfo.Arch*)

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**(*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.**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*

class angr.sim_variable.**SimVariable**(*size*, *ident=None*, *name=None*, *region=None*, *category=None*)

Bases: *Serializable*

The base class for all other classes of variables.

Parameters

- **size** (*int*)
- **ident** (*str | None*)
- **name** (*str | None*)
- **region** (*int | None*)

__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

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**(*size*, *ident=None*, *value=None*, *region=None*)

Bases: *SimVariable*

Describes a constant variable.

Parameters
size (*int*)

__init__(*size*, *ident=None*, *value=None*, *region=None*)

Parameters

- **ident** – A unique identifier provided by user or the program. Usually a string.
- **name** (*str*) – Name of this variable.
- **size** (*int*)

value

loc_repr(*arch*)

The representation that shows up in a GUI

copy()

Return type
SimConstantVariable

property key: tuple[str | int | None, ...]

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

class angr.sim_variable.**SimTemporaryVariable**(*tmp_id*, *size*)

Bases: *SimVariable*

Describes a temporary variable.

Parameters

- **tmp_id** (*int*)
- **size** (*int*)

__init__(*tmp_id*, *size*)

Parameters

- **ident** – A unique identifier provided by user or the program. Usually a string.
- **name** (*str*) – Name of this variable.
- **tmp_id** (*int*)
- **size** (*int*)

tmp_id

loc_repr(*arch*)

The representation that shows up in a GUI

copy()

Return type
SimTemporaryVariable

property key: tuple[str | int | None, ...]

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

class angr.sim_variable.**SimRegisterVariable**(*reg_offset*, *size*, *ident=None*, *name=None*, *region=None*, *category=None*)

Bases: *SimVariable*

Describes a register variable.

Parameters

- **reg_offset** (*int*)
- **size** (*int*)

__init__(*reg_offset*, *size*, *ident=None*, *name=None*, *region=None*, *category=None*)

Parameters

- **ident** – A unique identifier provided by user or the program. Usually a string.
- **name** (*str*) – Name of this variable.
- **reg_offset** (*int*)
- **size** (*int*)

reg

loc_repr(*arch*)

The representation that shows up in a GUI

property key: tuple[str | int | None, ...]

copy()

Return type
SimRegisterVariable

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

class angr.sim_variable.**SimComboRegisterVariable**(*reg_offsets*, *size*, *ident=None*, *name=None*, *region=None*, *category=None*)

Bases: *SimVariable*

A variable that spans multiple registers combined into a single value.

Parameters

- **reg_offsets** (*tuple*)
- **size** (*int*)

__init__(*reg_offsets*, *size*, *ident=None*, *name=None*, *region=None*, *category=None*)

Parameters

- **ident** – A unique identifier provided by user or the program. Usually a string.
- **name** (*str*) – Name of this variable.
- **reg_offsets** (*tuple*)
- **size** (*int*)

reg_offsets

property bits

loc_repr(*arch*)

The representation that shows up in a GUI

copy()

Return type
SimComboRegisterVariable

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

class angr.sim_variable.**SimMemoryVariable**(*addr*, *size*, *ident=None*, *name=None*, *region=None*, *category=None*)

Bases: *SimVariable*

Describes a memory variable; the base class for other types of memory variables.

Parameters
size (*int*)

__init__(*addr*, *size*, *ident=None*, *name=None*, *region=None*, *category=None*)

Parameters

- **ident** – A unique identifier provided by user or the program. Usually a string.
- **name** (*str*) – Name of this variable.
- **size** (*int*)

addr

loc_repr(*arch*)

The representation that shows up in a GUI

copy()

Return type
SimMemoryVariable

property key: tuple[str | int | None, ...]

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

class angr.sim_variable.**SimStackVariable**(*offset*, *size*, *base='bp'*, *base_addr=None*, *ident=None*, *name=None*, *region=None*, *category=None*)

Bases: *SimMemoryVariable*

Describes a stack variable.

Parameters

- **offset** (*int*)
- **size** (*int*)

__init__(*offset*, *size*, *base='bp'*, *base_addr=None*, *ident=None*, *name=None*, *region=None*, *category=None*)

Parameters

- **ident** – A unique identifier provided by user or the program. Usually a string.

- **name** (*str*) – Name of this variable.
- **offset** (*int*)
- **size** (*int*)

base

offset

base_addr

loc_repr(*arch*)

The representation that shows up in a GUI

copy()

Return type
SimStackVariable

property key: tuple[str | int | None, ...]

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

class angr.sim_type.**SimType**(*label=None*, *qualifier=None*)

Bases: object

SimType exists to track type information for SimProcedures.

Parameters
qualifier (*Iterable[str] | None*)

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*)

Parameters

arch (*Arch | 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*)

Deserialize a type class from a JSON-compatible dictionary.

Parameters

- **d** (*dict[str, Any]*)
- **type_collection** (SimTypeCollection | *None*)

- **memo** (*set[str] | None*)

class angr.sim_type.**TypeRef**(*name*, *ty*, *qualifier=None*)

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.

Parameters
qualifier (*Iterable | None*)

__init__(*name*, *ty*, *qualifier=None*)

Parameters

- **label** – the type label.
- **qualifier** (*Iterable | None*)

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*)

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

Parameters
name_parens (*bool*)

copy()

class angr.sim_type.**NamedTypeMixin**(**args*, *name=None*, ***kwargs*)

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.

Parameters
name (*str | None*)

__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**(*label=None*, *qualifier=None*)

Bases: *SimType*

SimTypeBottom basically represents a type error.

Parameters
qualifier (*Iterable[str] | None*)

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

Parameters
name_parens (*bool*)

copy()

class angr.sim_type.**SimTypeTop**(*size=None*, *label=None*, *qualifier=None*)

Bases: *SimType*

SimTypeTop represents any type (mostly used with a pointer for void*).

Parameters

- **size** (*int | None*)
- **qualifier** (*Iterable | None*)

__init__(*size=None*, *label=None*, *qualifier=None*)

Parameters

- **label** – the type label.
- **size** (*int | None*)
- **qualifier** (*Iterable | None*)

copy()

class angr.sim_type.**SimTypeReg**(*size*, *label=None*, *qualifier=None*)

Bases: *SimType*

SimTypeReg is the base type for all types that are register-sized.

Parameters

- **size** (*int | None*)
- **qualifier** (*Iterable | None*)

__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**(*size*, *signed=True*, *label=None*, *qualifier=None*)

Bases: *SimType*

SimTypeNum is a numeric type of arbitrary length

Parameters

- **size** (*int*)
- **qualifier** (*Iterable* | *None*)

__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**(*signed=True*, *label=None*, *qualifier=None*)

Bases: *SimTypeReg*

SimTypeInt is a type that specifies a signed or unsigned C integer.

Parameters
qualifier (*Iterable* | *None*)

__init__(*signed=True*, *label=None*, *qualifier=None*)

Parameters

- **signed** – True if signed, False if unsigned
- **label** – The type label
- **qualifier** (*Iterable* | *None*)

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*)

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**(*signed=True*, *label=None*, *qualifier=None*)

Bases: *SimTypeInt*

Parameters
qualifier (*Iterable | None*)

class angr.sim_type.**SimTypeLong**(*signed=True*, *label=None*, *qualifier=None*)

Bases: *SimTypeInt*

Parameters
qualifier (*Iterable | None*)

class angr.sim_type.**SimTypeLongLong**(*signed=True*, *label=None*, *qualifier=None*)

Bases: *SimTypeInt*

Parameters
qualifier (*Iterable | None*)

class angr.sim_type.**SimTypeFixedSizeInt**(*signed=True*, *label=None*, *qualifier=None*)

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.

Parameters
qualifier (*Iterable | None*)

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**(*signed=True*, *label=None*, *qualifier=None*)

Bases: *SimTypeFixedSizeInt*

Parameters
qualifier (*Iterable | None*)

class angr.sim_type.**SimTypeInt256**(*signed=True*, *label=None*, *qualifier=None*)

Bases: *SimTypeFixedSizeInt*

Parameters
qualifier (*Iterable | None*)

class angr.sim_type.**SimTypeInt512**(*signed=True*, *label=None*, *qualifier=None*)

Bases: *SimTypeFixedSizeInt*

Parameters
qualifier (*Iterable | None*)

class angr.sim_type.**SimTypeChar**(*signed=True*, *label=None*, *qualifier=None*)

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.

Parameters
qualifier (*Iterable | None*)

__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**(*signed=True*, *label=None*, *endness=Endness.BE*, *qualifier=None*)

Bases: *SimTypeReg*

SimTypeWideChar is a type that specifies a wide character (a UTF-16 character).

Parameters

- **endness** (*Endness*)
- **qualifier** (*Iterable | None*)

__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**(*signed=True*, *label=None*, *qualifier=None*)

Bases: *SimTypeReg*

Parameters
qualifier (*Iterable | None*)

__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**(*label=None*, *qualifier=None*)

Bases: *SimTypeReg*

SimTypeFd is a type that specifies a file descriptor.

Parameters
qualifier (*Iterable | None*)

__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**(**values*)

Bases: IntEnum

IN_OUT = 0

IN = 1

OUT = 2

IN_OUTMAYBE = 3

OUTMAYBE = 4

NONE = 5

UNKNOWN = 6

class angr.sim_type.**SimTypePointer**(*pts_to*, *label=None*, *offset=0*, *qualifier=None*, *disposition=PointerDisposition.UNKNOWN*)

Bases: *SimTypeReg*

SimTypePointer is a type that specifies a pointer to some other type.

Parameters

- **pts_to** (SimType)
- **qualifier** (*Iterable[str] | None*)
- **disposition** (PointerDisposition | *int*)

__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*)

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*)

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**(*refs*, *label=None*, *qualifier=None*)

Bases: *SimTypeReg*

SimTypeReference is a type that specifies a reference to some other type.

Parameters
qualifier (*Iterable | None*)

__init__(*refs*, *label=None*, *qualifier=None*)

Parameters

- **label** – the type label.
- **size** – the size of the type (e.g. 32bit, 8bit, etc.).
- **qualifier** (*Iterable | None*)

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**(*elem_type*, *length=None*, *label=None*, *qualifier=None*)

Bases: *SimType*

SimTypeArray is a type that specifies a series of data laid out in sequence.

Parameters
qualifier (*Iterable | None*)

__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**(*length=None*, *label=None*, *name=None*, *qualifier=None*)

Bases: *NamedTypeMixin*, *SimType*

SimTypeString is a type that represents a C-style string, i.e. a NUL-terminated array of bytes.

Parameters

- **length** (*int | None*)
- **name** (*str | None*)
- **qualifier** (*Iterable | None*)

__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**(*length=None*, *label=None*, *name=None*, *qualifier=None*)

Bases: *NamedTypeMixin*, *SimType*

A wide-character null-terminated string, where each character is 2 bytes.

Parameters

- **length** (*int | None*)
- **name** (*str | None*)
- **qualifier** (*Iterable | None*)

__init__(*length=None*, *label=None*, *name=None*, *qualifier=None*)

Parameters

- **label** – the type label.

- **length** (*int | None*)
- **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*)

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**(*args*, *returnty*, *label=None*, *arg_names=None*, *variadic=False*)

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.

Parameters

- **args** (*Iterable[SimType]*)
- **returnty** (*SimType | None*)
- **arg_names** (*Iterable[str] | None*)

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, ...]

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*)

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**(*args*, *returnty*, *label=None*, *arg_names=None*, *ctor=False*, *dtor=False*, *convention=None*)

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.

Parameters

- **arg_names** (*Iterable[str] | None*)
- **ctor** (*bool*)
- **dtor** (*bool*)
- **convention** (*str | None*)

__init__(*args*, *returnty*, *label=None*, *arg_names=None*, *ctor=False*, *dtor=False*, *convention=None*)

Parameters

- **label** – The type label
- **args** – A tuple of types representing the arguments to the function
- **returnty** – The return type of the function, or none for void
- **variadic** – Whether the function accepts varargs
- **arg_names** (*Iterable[str] | None*)
- **ctor** (*bool*)
- **dtor** (*bool*)
- **convention** (*str | None*)

copy()

class angr.sim_type.**SimTypeLength**(*signed=False*, *addr=None*, *length=None*, *label=None*)

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**(*size=32*, *label=None*, *qualifier=None*)

Bases: *SimTypeReg*

An IEEE754 single-precision floating point number

Parameters

qualifier (*Iterable | None*)

__init__(*size=32*, *label=None*, *qualifier=None*)

Parameters

- **label** – the type label.
- **size** – the size of the type (e.g. 32bit, 8bit, etc.).
- **qualifier** (*Iterable | None*)

sort = FLOAT

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**(*align_double=True*, *label=None*, *qualifier=None*)

Bases: *SimTypeFloat*

An IEEE754 double-precision floating point number

Parameters

qualifier (*Iterable | None*)

__init__(*align_double=True*, *label=None*, *qualifier=None*)

Parameters

- **label** – the type label.
- **size** – the size of the type (e.g. 32bit, 8bit, etc.).
- **qualifier** (*Iterable | None*)

sort = DOUBLE

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**(*fields*, *name=None*, *pack=False*, *align=None*, *anonymous=False*, *qualifier=None*)

Bases: *NamedTypeMixin*, *SimType*

Parameters

- **fields** (*dict[str, SimType] | OrderedDict[str, SimType]*)
- **anonymous** (*bool*)
- **qualifier** (*Iterable | None*)

__init__(*fields*, *name=None*, *pack=False*, *align=None*, *anonymous=False*, *qualifier=None*)

Parameters

- **label** – the type label.
- **fields** (*dict[str, SimType] | OrderedDict[str, SimType]*)
- **anonymous** (*bool*)
- **qualifier** (*Iterable | None*)

fields: OrderedDict[str, *SimType*]

property pack

property align

property packed

property offsets: dict[str, int]

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*)

extract(*state*, *addr*, *concrete=False*)

Return type
SimStructValue

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**(*struct*, *values=None*)

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**(*members*, *name=None*, *label=None*, *qualifier=None*)

Bases: *NamedTypeMixin*, *SimType*

Parameters

- **members** (*dict[str, SimType]*)
- **qualifier** (*Iterable | None*)

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**(*union*, *values=None*)

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**(*members*, *base_type=None*, *name=None*, *qualifier=None*)

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.
- **qualifier** (*Iterable[str] | None*)

__init__(*members*, *base_type=None*, *name=None*, *qualifier=None*)

Parameters

- **label** – the type label.
- **members** (*dict[str, int]*)
- **base_type** (SimType | None)
- **name** (*str | None*)
- **qualifier** (*Iterable[str] | None*)

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**(*flags*, *base_type=None*, *name=None*, *qualifier=None*)

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.
- **qualifier** (*Iterable[str] | None*)

__init__(*flags*, *base_type=None*, *name=None*, *qualifier=None*)

Parameters

- **label** – the type label.
- **flags** (*dict[str, int]*)
- **base_type** (SimType | None)
- **name** (*str | None*)
- **qualifier** (*Iterable[str] | None*)

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**(*, *unique_name=None*, *name=None*, *members=None*, *function_members=None*, *vtable_ptrs=None*, *pack=False*, *align=None*, *size=None*, *anonymous=False*)

Bases: *SimStruct*

Parameters

- **unique_name** (*str | None*)
- **name** (*str | None*)
- **members** (*dict[str, SimType] | None*)
- **function_members** (*dict[str, SimTypeCppFunction] | None*)
- **pack** (*bool*)
- **size** (*int | None*)
- **anonymous** (*bool*)

__init__(*, *unique_name=None*, *name=None*, *members=None*, *function_members=None*, *vtable_ptrs=None*, *pack=False*, *align=None*, *size=None*, *anonymous=False*)

Parameters

- **label** – the type label.
- **unique_name** (*str | None*)
- **name** (*str | None*)
- **members** (*dict[str, SimType] | None*)
- **function_members** (*dict[str, SimTypeCppFunction] | None*)
- **pack** (*bool*)
- **size** (*int | None*)
- **anonymous** (*bool*)

property members

property size

The size of the type in bits, or None if no size is computable.

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*)

extract(*state*, *addr*, *concrete=False*)

Return type
SimCppClassValue

store(*state*, *addr*, *value*)

Parameters
value (*int | BV*)

copy()

class angr.sim_type.**SimCppClassValue**(*class_type*, *values*)

Bases: *SimStructValue*

A SimCppClass type paired with some real values

Parameters
class_type (SimCppClass)

__init__(*class_type*, *values*)

Parameters

- **struct** – A SimStruct instance describing the type of this struct
- **values** – A mapping from struct fields to values
- **class_type** (SimCppClass)

copy()

class angr.sim_type.**SimTypeNumOffset**(*size*, *signed=True*, *label=None*, *offset=0*, *qualifier=None*)

Bases: *SimTypeNum*

like SimTypeNum, but supports an offset of 1 to 7 to a byte aligned address to allow structs with bitfields

Parameters
qualifier (*Iterable | None*)

__init__(*size*, *signed=True*, *label=None*, *offset=0*, *qualifier=None*)

Parameters

- **size** – The size of the integer, in bits
- **signed** – Whether the integer is signed or not
- **label** – A label for the type
- **qualifier** (*Iterable | None*)

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**(*name*, *original_type*, *qualifier=None*)

Bases: *SimType*

SimTypeRef is a to-be-resolved reference to another SimType.

SimTypeRef is not SimTypeReference.

Parameters

- **original_type** (type[SimType])
- **qualifier** (*Iterable | None*)

__init__(*name*, *original_type*, *qualifier=None*)

Parameters

- **label** – the type label.
- **original_type** (type[SimType])
- **qualifier** (*Iterable | None*)

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*)

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*)

Deserialize a type class from a JSON-compatible dictionary.

Return type
SimTypeRef

Parameters

- **d** (*dict[str, Any]*)
- **type_collection** (SimTypeCollection | *None*)
- **memo** (*set[str] | None*)

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*)

class angr.callable.**Callable**(*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*)

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.

Parameters

- **project** (Project)
- **addr** (*int*)
- **prototype** (SimTypeFunction | *None*)
- **base_state** (SimState | *None*)
- **cc** (SimCC | *None*)
- **add_options** (*set[str] | None*)
- **remove_options** (*set[str] | None*)
- **step_limit** (*int | None*)
- **techniques** (*list[ExplorationTechnique] | None*)

__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

10.13 Knowledge Base

Representing the artifacts of a project.

class angr.knowledge_base.**KnowledgeBase**(*project*, *obj=None*, *name=None*)

Bases: object

Represents a "model" of knowledge about an artifact.

Contains things like a CFG, data references, etc.

functions: *FunctionManager*

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.knowledge_plugins.**CFGManager**(*kb*)

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**(*kb*)

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.**Comments**(*kb*)

Bases: *KnowledgeBasePlugin*, dict

Tracks comments via a Dict of Address -> Text

Parameters
kb (KnowledgeBase)

copy() → a shallow copy of D

class angr.knowledge_plugins.**CustomStrings**(*kb*)

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**(*kb*)

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?

Parameters
kb (KnowledgeBase)

copy()

class angr.knowledge_plugins.**DebugVariableManager**(*kb*)

Bases: *KnowledgeBasePlugin*

Structure to manage and access variables with different visibility scopes.

Parameters
kb (KnowledgeBase)

__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**(*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*)

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.

Parameters

- **function_manager** (FunctionManager | *None*)
- **addr** (*int*)
- **is_simprocedure** (*bool* | *None*)
- **is_plt** (*bool* | *None*)
- **calling_convention** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)
- **prototype_libname** (*str* | *None*)
- **prototype_source** (PrototypeSource | *None*)
- **is_prototype_guessed** (*bool*)

__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

from_signature: str | None

binary_name

property name

property project: *Project* | None

property returning

property calling_convention: *SimCC* | None

property prototype: *SimTypeFunction* | None

property prototype_libname

property is_prototype_guessed: bool

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.

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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: `None | Symbol`

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**(*kb*, *cache_limit=None*)

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.
- **kb** (KnowledgeBase)

__init__(*kb*, *cache_limit=None*)

Parameters

- **kb** (KnowledgeBase)
- **cache_limit** (*int | None*)

set_kb(*kb*)

Invoked after __setstate__() is called.

Parameters
kb (KnowledgeBase)

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

items(*meta_only=False*)

Return type
Generator[tuple[TypeVar(K, int, SootMethodDescriptor), *Function*]]

Parameters
meta_only (*bool*)

values(*meta_only=False*)

Return type
Generator[*Function*]

Parameters
meta_only (*bool*)

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_key_func_addrs(*func_type*)

Return type
set[TypeVar(K, int, SootMethodDescriptor)]

Parameters
func_type (*str*)

add_key_func_addr(*func_type*, *addr*)

Return type
None

Parameters

- **func_type** (*str*)
- **addr** (*K*)

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**(*kb*)

Bases: *KnowledgeBasePlugin*, dict

This plugin tracks the targets of indirect jumps

__init__(*kb*)

copy() → a shallow copy of D

update_resolved_addrs(*indirect_address*, *resolved_addresses*)

Parameters

- **indirect_address** (*int*)
- **resolved_addresses** (*list[int]*)

class angr.knowledge_plugins.**KeyDefinitionManager**(*kb*)

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)

Parameters
kb (KnowledgeBase)

__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**(*kb*)

Bases: object

Parameters
kb (KnowledgeBase)

__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**(*kb*)

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**(*kb*)

Bases: *KnowledgeBasePlugin*

Store discovered information and artifacts about (string) obfuscation techniques in the project.

__init__(*kb*)

copy()

class angr.knowledge_plugins.**PatchManager**(*kb*)

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**(*kb*)

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**(*kb*, *lmdb_path=None*)

Bases: *KnowledgeBasePlugin*

External storage-backed database for angr knowledge base plugins.

Parameters

- **kb** (KnowledgeBase)

- **lmdb_path** (*str | None*)

__init__(*kb*, *lmdb_path=None*)

Parameters

- **kb** (KnowledgeBase)
- **lmdb_path** (*str | None*)

Return type
None

increase_lmdb_map_size()

Increase the LMDB map size.

Return type
None

reopen_lmdb()

Reopen the existing LMDB environment and all open databases in self._dbs.

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()

class angr.knowledge_plugins.**StructuredCodeManager**(*kb*)

Bases: *KnowledgeBasePlugin*

A knowledge base plugin to store structured code generator results.

__init__(*kb*)

get(*key*, *default=None*)

discard(*key*)

available_flavors(*item*)

all_flavors(*item*)

copy()

class angr.knowledge_plugins.**TypesStore**(*kb*)

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**(*kb*)

Bases: *KnowledgeBasePlugin*

Manage variables.

__init__(*kb*)

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**(*kb*)

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_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*]

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *cfg_model=None*, *kb=None*, ***kwargs*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

class angr.knowledge_plugins.patches.**Patch**(*addr*, *new_bytes*, *comment=None*)

Bases: object

Parameters
comment (*str | None*)

__init__(*addr*, *new_bytes*, *comment=None*)

Parameters
comment (*str | None*)

class angr.knowledge_plugins.patches.**PatchManager**(*kb*)

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.plugin.**KnowledgeBasePlugin**(*kb*)

Bases: object

Parameters
kb (KnowledgeBase)

__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.callsite_prototypes.**CallsitePrototypeKind**(**values*)

Bases: Enum

Describes the type of a callsite prototype.

INFERRED = 1

PROPAGATED = 2

MANUAL = 3

class angr.knowledge_plugins.callsite_prototypes.**CallsitePrototypes**(*kb*)

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.cfg.**BlockID**(*addr*, *callsite_tuples*, *jump_type*)

Bases: object

A context-sensitive key for a SimRun object.

Parameters

- **addr** (*int*)
- **callsite_tuples** (*tuple[int, ...] | None*)
- **jump_type** (*str*)

__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**(*addr*, *size*, *cfg*, *simprocedure_name=None*, *no_ret=False*, *function_address=None*, *block_id=None*, *irsb=None*, *instruction_addrs=None*, *thumb=False*, *byte_string=None*, *is_syscall=None*, *name=None*, *input_state=None*, *final_states=None*, *syscall_name=None*, *looping_times=0*, *depth=None*, *callstack_key=None*, *creation_failure_info=None*)

Bases: *CFGNode*

The CFGNode that is used in CFGEmulated.

__init__(*addr*, *size*, *cfg*, *simprocedure_name=None*, *no_ret=False*, *function_address=None*, *block_id=None*, *irsb=None*, *instruction_addrs=None*, *thumb=False*, *byte_string=None*, *is_syscall=None*, *name=None*, *input_state=None*, *final_states=None*, *syscall_name=None*, *looping_times=0*, *depth=None*, *callstack_key=None*, *creation_failure_info=None*)

Note: simprocedure_name is not used to recreate the SimProcedure object. It's only there for better __repr__.

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.

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *cfg=None*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

copy()

class angr.knowledge_plugins.cfg.**CFGManager**(*kb*)

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**(*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'*)

Bases: *Serializable*

This class describes a Control Flow Graph for a specific range of code.

Parameters

- **cache_limit** (*int | None*)
- **db_batch_size** (*int*)
- **edge_cache_limit** (*int | None*)
- **edge_db_batch_size** (*int*)
- **addr_type** (*CFG_ADDR_TYPES*)

__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*)

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *cfg_manager=None*, *loader=None*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

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**(*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*)

Bases: *Serializable*

This class stands for each single node in CFG.

Parameters
block_id (BlockID | *int* | *None*)

__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

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *cfg=None*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

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**(*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*, *type_=255*)

Bases: *Serializable*

Describes an indirect jump or call site.

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*)
- **type_** (*int | None*)

__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*, *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*)
- **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*)

Return type
None

Parameters

- **addr** (*int | None*)
- **size** (*int*)
- **entry_size** (*int*)
- **entries** (*list[int]*)
- **is_primary** (*bool*)

property jumptable_addr: int | None

property jumptable_size: int | None

property jumptable_entry_size: int | None

property jumptable_entries: list[int] | None

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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
IndirectJump

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**(*address*, *size*, *sort*, *pointer_addr=None*, *max_size=None*, *reference_size=None*)

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]*.

Parameters

- **address** (*int*)
- **size** (*int*)
- **sort** (*str | None*)
- **pointer_addr** (*int | None*)
- **max_size** (*int | None*)
- **reference_size** (*int | None*)

__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

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

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'

class angr.knowledge_plugins.cfg.cfg_model.**CFGModel**(*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'*)

Bases: *Serializable*

This class describes a Control Flow Graph for a specific range of code.

Parameters

- **cache_limit** (*int | None*)
- **db_batch_size** (*int*)
- **edge_cache_limit** (*int | None*)
- **edge_db_batch_size** (*int*)
- **addr_type** (*CFG_ADDR_TYPES*)

__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*)

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *cfg_manager=None*, *loader=None*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

`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.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**(*address*, *size*, *sort*, *pointer_addr=None*, *max_size=None*, *reference_size=None*)

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]*.

Parameters

- **address** (*int*)
- **size** (*int*)

- **sort** (*str* | *None*)
- **pointer_addr** (*int* | *None*)
- **max_size** (*int* | *None*)
- **reference_size** (*int*)

__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

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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

class angr.knowledge_plugins.cfg.cfg_manager.**CFGManager**(*kb*)

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.cfg_node.**CFGNodeCreationFailure**(*exc_info=None*, *to_copy=None*)

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**(*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*)

Bases: *Serializable*

This class stands for each single node in CFG.

Parameters
block_id (BlockID | *int* | *None*)

__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

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *cfg=None*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

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**(*addr*, *size*, *cfg*, *simprocedure_name=None*, *no_ret=False*, *function_address=None*, *block_id=None*, *irsb=None*, *instruction_addrs=None*, *thumb=False*, *byte_string=None*, *is_syscall=None*, *name=None*, *input_state=None*, *final_states=None*, *syscall_name=None*, *looping_times=0*, *depth=None*, *callstack_key=None*, *creation_failure_info=None*)

Bases: *CFGNode*

The CFGNode that is used in CFGEmulated.

__init__(*addr*, *size*, *cfg*, *simprocedure_name=None*, *no_ret=False*, *function_address=None*, *block_id=None*, *irsb=None*, *instruction_addrs=None*, *thumb=False*, *byte_string=None*, *is_syscall=None*, *name=None*, *input_state=None*, *final_states=None*, *syscall_name=None*, *looping_times=0*, *depth=None*, *callstack_key=None*, *creation_failure_info=None*)

Note: simprocedure_name is not used to recreate the SimProcedure object. It's only there for better __repr__.

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.

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *cfg=None*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

copy()

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**(*addr*, *size*, *entry_size*, *entries*)

Bases: object

Describes a jump table or a vtable.

Parameters

- **addr** (*int | None*)
- **size** (*int*)
- **entry_size** (*int*)
- **entries** (*list[int]*)

__init__(*addr*, *size*, *entry_size*, *entries*)

Parameters

- **addr** (*int | None*)
- **size** (*int*)
- **entry_size** (*int*)
- **entries** (*list[int]*)

addr

size

entry_size

entries

serialize_to_cmessage()

classmethod parse_from_cmessage(*cmsg*)

Return type
JumptableInfo

class angr.knowledge_plugins.cfg.indirect_jump.**IndirectJump**(*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*, *type_=255*)

Bases: *Serializable*

Describes an indirect jump or call site.

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*)
- **type_** (*int | None*)

__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*, *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*)
- **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*)

Return type
None

Parameters

- **addr** (*int | None*)
- **size** (*int*)
- **entry_size** (*int*)
- **entries** (*list[int]*)
- **is_primary** (*bool*)

property jumptable_addr: int | None

property jumptable_size: int | None

property jumptable_entry_size: int | None

property jumptable_entries: list[int] | None

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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
IndirectJump

class angr.knowledge_plugins.types.**TypesStore**(*kb*)

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.propagations.**PropagationManager**(*kb*)

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**(*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*)

Bases: *Serializable*

This class stores the propagation result that comes out of Propagator.

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*)

__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

class angr.knowledge_plugins.comments.**Comments**(*kb*)

Bases: *KnowledgeBasePlugin*, dict

Tracks comments via a Dict of Address -> Text

Parameters
kb (KnowledgeBase)

copy() → a shallow copy of D

class angr.knowledge_plugins.data.**Data**(*kb*)

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?

Parameters
kb (KnowledgeBase)

copy()

class angr.knowledge_plugins.indirect_jumps.**IndirectJumps**(*kb*)

Bases: *KnowledgeBasePlugin*, dict

This plugin tracks the targets of indirect jumps

__init__(*kb*)

copy() → a shallow copy of D

update_resolved_addrs(*indirect_address*, *resolved_addresses*)

Parameters

- **indirect_address** (*int*)
- **resolved_addresses** (*list[int]*)

class angr.knowledge_plugins.labels.**Labels**(*kb*)

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.functions.**Function**(*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*)

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.

Parameters

- **function_manager** (FunctionManager | *None*)
- **addr** (*int*)
- **is_simprocedure** (*bool* | *None*)
- **is_plt** (*bool* | *None*)
- **calling_convention** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)
- **prototype_libname** (*str* | *None*)
- **prototype_source** (PrototypeSource | *None*)
- **is_prototype_guessed** (*bool*)

__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

from_signature: str | None

binary_name

property name

property project: *Project* | None

property returning

property calling_convention: *SimCC* | None

property prototype: *SimTypeFunction* | None

property prototype_libname

property is_prototype_guessed: bool

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.

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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: `None | Symbol`

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**(*kb*, *cache_limit=None*)

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.
- **kb** (KnowledgeBase)

__init__(*kb*, *cache_limit=None*)

Parameters

- **kb** (KnowledgeBase)
- **cache_limit** (*int | None*)

set_kb(*kb*)

Invoked after __setstate__() is called.

Parameters
kb (KnowledgeBase)

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

items(*meta_only=False*)

Return type
Generator[tuple[TypeVar(K, int, SootMethodDescriptor), *Function*]]

Parameters
meta_only (*bool*)

values(*meta_only=False*)

Return type
Generator[*Function*]

Parameters
meta_only (*bool*)

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_key_func_addrs(*func_type*)

Return type
set[TypeVar(K, int, SootMethodDescriptor)]

Parameters
func_type (*str*)

add_key_func_addr(*func_type*, *addr*)

Return type
None

Parameters

- **func_type** (*str*)
- **addr** (*K*)

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.function_manager.**FunctionDictBase**(*backref*, *key_types=<class 'int'>*)

Bases: Generic

Base class for FunctionDict and SpillingFunctionDict.

Parameters

- **backref** (FunctionManager[K] | None)
- **key_types** (type)

__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**(*backref*, **args*, *key_types=<class 'int'>*, ***kwargs*)

Bases: SortedDict[K, *Function*], *FunctionDictBase*[K]

FunctionDict is a dict where the keys are function starting addresses and map to the associated Function.

Parameters

- **backref** (*FunctionManager[K] | None*)
- **key_types** (*type*)

__init__(*backref*, **args*, *key_types=<class 'int'>*, ***kwargs*)

Initialize sorted dict instance.

Optional key-function argument defines a callable that, like the *key* argument to the built-in *sorted* function, extracts a comparison key from each dictionary key. If no function is specified, the default compares the dictionary keys directly. The key-function argument must be provided as a positional argument and must come before all other arguments.

Optional iterable argument provides an initial sequence of pairs to initialize the sorted dict. Each pair in the sequence defines the key and corresponding value. If a key is seen more than once, the last value associated with it is stored in the new sorted dict.

Optional mapping argument provides an initial mapping of items to initialize the sorted dict.

If keyword arguments are given, the keywords themselves, with their associated values, are added as items to the dictionary. If a key is specified both in the positional argument and as a keyword argument, the value associated with the keyword is stored in the sorted dict.

Sorted dict keys must be hashable, per the requirement for Python's dictionaries. Keys (or the result of the key-function) must also be comparable, per the requirement for sorted lists.

```
>>> d = {'alpha': 1, 'beta': 2}
>>> SortedDict([('alpha', 1), ('beta', 2)]) == d
True
>>> SortedDict({'alpha': 1, 'beta': 2}) == d
True
>>> SortedDict(alpha=1, beta=2) == d
True
```

Parameters

- **backref** (*FunctionManager[K] | None*)

- **key_types** (type)

copy()

Return a shallow copy of the sorted dict.

Runtime complexity: *O(n)*

Return type
FunctionDict[TypeVar(K, int, SootMethodDescriptor)]

Returns
new sorted dict

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*)

Return the value for key if key is in the dictionary, else default.

floor_addr(*addr*)

Parameters
addr (*K*)

ceiling_addr(*addr*)

class angr.knowledge_plugins.functions.function_manager.**SpillingFunctionDict**(*backref*, *rtdb*, /, *key_types=<class 'int'>*, *cache_limit=1000*, *db_batch_size=100*, ***kwargs*)

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.

Parameters

- **backref** (FunctionManager[K] | None)
- **rtdb** (RuntimeDb)
- **key_types** (type)
- **cache_limit** (int)
- **db_batch_size** (int)

__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

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
meta_only (bool)

keys() → a set-like object providing a view on D's keys

items() → a set-like object providing a view on D's items

values() → an object providing a view on D's values

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*)

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**(*kb*, *cache_limit=None*)

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.
- **kb** (KnowledgeBase)

__init__(*kb*, *cache_limit=None*)

Parameters

- **kb** (KnowledgeBase)
- **cache_limit** (*int | None*)

set_kb(*kb*)

Invoked after __setstate__() is called.

Parameters
kb (KnowledgeBase)

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

items(*meta_only=False*)

Return type
Generator[tuple[TypeVar(K, int, SootMethodDescriptor), *Function*]]

Parameters
meta_only (*bool*)

values(*meta_only=False*)

Return type
Generator[*Function*]

Parameters
meta_only (*bool*)

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_key_func_addrs(*func_type*)

Return type
set[TypeVar(K, int, SootMethodDescriptor)]

Parameters
func_type (*str*)

add_key_func_addr(*func_type*, *addr*)

Return type
None

Parameters

- **func_type** (*str*)
- **addr** (*K*)

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.**dirty_func**(*func*)

class angr.knowledge_plugins.functions.function.**PrototypeSource**(**values*)

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

class angr.knowledge_plugins.functions.function.**FunctionInfo**(*func*)

Bases: UserDict

A dictionary that updates the .dirty field of the Function object when modified.

Parameters
func (Function)

__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**(*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*)

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.

Parameters

- **function_manager** (FunctionManager | *None*)
- **addr** (*int*)
- **is_simprocedure** (*bool* | *None*)
- **is_plt** (*bool* | *None*)
- **calling_convention** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)
- **prototype_libname** (*str* | *None*)
- **prototype_source** (PrototypeSource | *None*)
- **is_prototype_guessed** (*bool*)

__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

from_signature: str | None

binary_name

property name

property project: *Project* | None

property returning

property calling_convention: *SimCC* | None

property prototype: *SimTypeFunction* | None

property prototype_libname

property is_prototype_guessed: bool

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.

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

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: `None | Symbol`

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.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

class angr.knowledge_plugins.functions.soot_function.**SootFunction**(*function_manager*, *addr*, *name=None*, *syscall=None*)

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.

transition_graph

normalized

previous_names

addr

binary_name

bp_on_stack

retaddr_on_stack

sp_delta

startpoint

tags

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

evicted: bool

from_signature: str | None

is_default_name

meta_only: bool

ran_cca

class angr.knowledge_plugins.variables.**VariableManager**(*kb*)

Bases: *KnowledgeBasePlugin*

Manage variables.

__init__(*kb*)

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

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**(*variable*, *access_type*, *location*, *offset*, *atom_hash=None*)

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

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *variable_by_ident=None*, ***kwargs*)

Parse a protobuf cmessage and create a class object.

Parameters

- **cmsg** – The probobuf cmessage object.
- **variable_by_ident** (*dict[str, SimVariable] | None*)

Returns
A unserialized class object.

Return type
VariableAccess

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**(*register_region*, *stack_region*)

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**(*manager*, *func_addr=None*)

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".

Parameters
manager (VariableManager)

__init__(*manager*, *func_addr=None*)

manager: *VariableManager*

variable_to_types: dict[*SimVariable*, *SimType*]

set_manager(*manager*)

Parameters
manager (VariableManager)

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *variable_manager=None*, *func_addr=None*, ***kwargs*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
VariableManagerInternal

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*)

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 | SimRegisterVariable]

Get a list of variables.

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*)

Overloads

- **self**, **sort** (Literal['stack']) → list[SimStackVariable]
- **self**, **sort** (Literal['reg']) → list[SimRegisterVariable]
- **self**, **sort** (None) → list[SimRegisterVariable | SimRegisterVariable]

Get a list of unified variables.

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**(*kb*)

Bases: *KnowledgeBasePlugin*

Manage variables.

__init__(*kb*)

function_managers: dict[int, *VariableManagerInternal*]

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.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**(*low_pc*, *high_pc*, *cle_variable*)

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

Parameters

- **low_pc** (*int*)
- **high_pc** (*int*)
- **cle_variable** (Variable)

__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*)

from_pc(*pc*)

Returns the visible variable (if any) for a given pc address.

Return type
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**(*kb*)

Bases: *KnowledgeBasePlugin*

Structure to manage and access variables with different visibility scopes.

Parameters
kb (KnowledgeBase)

__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.structured_code.**StructuredCodeManager**(*kb*)

Bases: *KnowledgeBasePlugin*

A knowledge base plugin to store structured code generator results.

__init__(*kb*)

get(*key*, *default=None*)

discard(*key*)

available_flavors(*item*)

all_flavors(*item*)

copy()

class angr.knowledge_plugins.key_definitions.**Definition**(*atom*, *codeloc*, *dummy=False*, *tags=None*)

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.

Parameters

- **atom** (*A*)
- **codeloc** (*CodeLoc*)
- **dummy** (*bool*)

- **tags** (*set[Tag] | None*)

__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**(**values*)

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**(*kb*)

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)

Parameters
kb (KnowledgeBase)

__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**(*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*)

Bases: object

A LiveDefinitions instance contains definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

Parameters

- **arch** (*archinfo.Arch*)
- **track_tmps** (*bool*)
- **merge_into_tops** (*bool*)

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] | type[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.knowledge_plugins.key_definitions.**ReachingDefinitionsModel**(*func_addr=None*, *track_liveness=True*)

Bases: object

Models the definitions, uses, and memory of a ReachingDefinitionState object

Parameters

- **func_addr** (*int | None*)
- **track_liveness** (*bool*)

__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**(*uses_by_definition=None*, *uses_by_location=None*)

Bases: object

Describes uses (including the use location and the use expression) for definitions.

Parameters

- **uses_by_definition** (DefaultChainMapCOW | *None*)
- **uses_by_location** (DefaultChainMapCOW | *None*)

__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

class angr.knowledge_plugins.key_definitions.atoms.**AtomKind**(**values*)

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**(*size*)

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**(*target*)

Bases: *Atom*

Implements a guard use.

__init__(*target*)

Parameters
size – The size of the atom in bytes

target

class angr.knowledge_plugins.key_definitions.atoms.**ConstantSrc**(*value*, *size*)

Bases: *Atom*

Represents a constant.

Parameters

- **value** (*int*)
- **size** (*int*)

__init__(*value*, *size*)

Parameters

- **size** (int) – The size of the atom in bytes
- **value** (*int*)

value: int

class angr.knowledge_plugins.key_definitions.atoms.**Tmp**(*tmp_idx*, *size*)

Bases: *Atom*

Represents a variable used by the IR to store intermediate values.

Parameters

- **tmp_idx** (*int*)
- **size** (*int*)

__init__(*tmp_idx*, *size*)

Parameters

- **size** (int) – The size of the atom in bytes
- **tmp_idx** (*int*)

tmp_idx

class angr.knowledge_plugins.key_definitions.atoms.**Register**(*reg_offset*, *size*, *arch=None*)

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.

Parameters

- **reg_offset** (*RegisterOffset | int*)
- **size** (*int*)
- **arch** (*Arch | None*)

__init__(*reg_offset*, *size*, *arch=None*)

Parameters

- **size** (int) – The size of the atom in bytes
- **reg_offset** (*RegisterOffset | int*)
- **arch** (*Arch | None*)

reg_offset

arch

property name: str

class angr.knowledge_plugins.key_definitions.atoms.**VirtualVariable**(*varid*, *size*, *category*, *oident=None*)

Bases: *Atom*

Represents a virtual variable.

Parameters

- **varid** (*int*)
- **size** (*int*)
- **category** (*ailment.Expr.VirtualVariableCategory*)
- **oident** (*str | int | tuple | None*)

__init__(*varid*, *size*, *category*, *oident=None*)

Parameters

- **size** (int) – The size of the atom in bytes
- **varid** (*int*)
- **category** (VirtualVariableCategory)
- **oident** (*str | int | tuple | None*)

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**(*addr*, *size*, *endness=None*)

Bases: *Atom*

Represents a memory slice.

It is characterized by its address and its size.

Parameters

- **addr** (SpOffset | HeapAddress | *int*)
- **size** (*int*)
- **endness** (*Endness | None*)

__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.knowledge_plugins.key_definitions.constants.**ObservationPointType**(**values*)

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

class angr.knowledge_plugins.key_definitions.definition.**DefinitionMatchPredicate**(*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*)

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.

Parameters

- **kind** (AtomKind | type[Atom] | *None*)
- **bbl_addr** (*int | None*)
- **ins_addr** (*int | None*)
- **variable** (SimVariable | *None*)
- **variable_manager** (VariableManagerInternal | *None* | *Literal[False]*)

- **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*)

kind: *AtomKind* | *type*[*Atom*] | None = None

bbl_addr: int | None = None

ins_addr: int | None = None

variable: *SimVariable* | None = None

variable_manager: *VariableManagerInternal* | None | Literal[False] = 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 | None | Literal[False]*)
- **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**(*atom*, *codeloc*, *dummy=False*, *tags=None*)

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.

Parameters

- **atom** (*A*)
- **codeloc** (*CodeLoc*)
- **dummy** (*bool*)
- **tags** (*set[Tag] | None*)

__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.environment.**Environment**(*environment=None*)

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.

Parameters
environment (*dict[str | Undefined, set[claripy.ast.Base]] | None*)

__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)

class angr.knowledge_plugins.key_definitions.heap_address.**HeapAddress**(*value*)

Bases: object

The representation of an address on the heap.

Parameters
value (*int* | Undefined)

__init__(*value*)

Parameters
value (*int* | Undefined)

property value

class angr.knowledge_plugins.key_definitions.key_definition_manager.**RDAObserverControl**(*func_addr*, *call_site_block_addrs*, *call_site_ins_addrs*)

Bases: object

Parameters

- **func_addr** (*int*)
- **call_site_block_addrs** (*Iterable[int]*)
- **call_site_ins_addrs** (*Iterable[int]*)

__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**(*kb*)

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)

Parameters
kb (KnowledgeBase)

__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.live_definitions.**DerefSize**(**values*)

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**(*definition*)

Bases: Annotation

An annotation that attaches a *Definition* to an AST.

__init__(*definition*)

definition

property relocatable

Returns whether this annotation can be relocated in a simplification.

Returns
True if it can be relocated, false otherwise.

property eliminatable

Returns whether this annotation can be eliminated in a simplification.

Returns
True if eliminatable, False otherwise

class angr.knowledge_plugins.key_definitions.live_definitions.**LiveDefinitions**(*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*)

Bases: object

A LiveDefinitions instance contains definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

Parameters

- **arch** (*archinfo.Arch*)
- **track_tmps** (*bool*)
- **registers** (MultiValuedMemory)
- **stack** (MultiValuedMemory)
- **memory** (MultiValuedMemory)
- **heap** (MultiValuedMemory)
- **tmps** (*dict[int, set[Definition]]*)
- **others** (*dict[Atom, MultiValues]*)
- **tmp_uses** (*dict[int, set[CodeLocation]]*)
- **merge_into_tops** (*bool*)

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.rd_model.**ReachingDefinitionsModel**(*func_addr=None*, *track_liveness=True*)

Bases: object

Models the definitions, uses, and memory of a ReachingDefinitionState object

Parameters

- **func_addr** (*int | None*)
- **track_liveness** (*bool*)

__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*)

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**(*metadata=None*)

Bases: object

A tag for a Definition that can carry different kinds of metadata.

Parameters
metadata (object)

__init__(*metadata=None*)

Parameters
metadata (object)

class angr.knowledge_plugins.key_definitions.tag.**FunctionTag**(*function=None*, *metadata=None*)

Bases: *Tag*

A tag for a definition created (or used) in the context of a function.

Parameters

- **function** (*int | None*)
- **metadata** (object)

__init__(*function=None*, *metadata=None*)

Parameters

- **function** (*int | None*)
- **metadata** (object)

class angr.knowledge_plugins.key_definitions.tag.**SideEffectTag**(*function=None*, *metadata=None*)

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*.

Parameters

- **function** (*int | None*)
- **metadata** (object)

class angr.knowledge_plugins.key_definitions.tag.**ParameterTag**(*function=None*, *metadata=None*)

Bases: *FunctionTag*

A tag for a definition of a parameter.

Parameters

- **function** (*int | None*)
- **metadata** (object)

class angr.knowledge_plugins.key_definitions.tag.**LocalVariableTag**(*function=None*, *metadata=None*)

Bases: *FunctionTag*

A tag for a definition of a local variable of a function.

Parameters

- **function** (*int | None*)
- **metadata** (object)

class angr.knowledge_plugins.key_definitions.tag.**ReturnValueTag**(*function=None*, *metadata=None*)

Bases: *FunctionTag*

A tag for a definition of a return value of a function.

Parameters

- **function** (*int | None*)
- **metadata** (object)

class angr.knowledge_plugins.key_definitions.tag.**InitialValueTag**(*metadata=None*)

Bases: *Tag*

A tag for a definition of an initial value

Parameters

metadata (object)

class angr.knowledge_plugins.key_definitions.tag.**UnknownSizeTag**(*metadata=None*)

Bases: *Tag*

A tag for a definition of an initial value

Parameters

metadata (object)

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.

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.

class angr.knowledge_plugins.key_definitions.uses.**Uses**(*uses_by_definition=None*, *uses_by_location=None*)

Bases: object

Describes uses (including the use location and the use expression) for definitions.

Parameters

- **uses_by_definition** (DefaultChainMapCOW | *None*)
- **uses_by_location** (DefaultChainMapCOW | *None*)

__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

class angr.knowledge_plugins.xrefs.**XRef**(*ins_addr=None*, *block_addr=None*, *stmt_idx=None*, *insn_op_idx=None*, *memory_data=None*, *dst=None*, *xref_type=None*)

Bases: *Serializable*

XRef describes a reference to a MemoryData instance (if a MemoryData instance is available) or just an address.

Parameters

- **ins_addr** (*int* | *None*)
- **block_addr** (*int* | *None*)
- **stmt_idx** (*int* | *None*)
- **insn_op_idx** (*int* | *None*)
- **memory_data** (angr.knowledge_plugins.cfg.memory_data.MemoryData | *None*)
- **dst** (*int* | *None*)

__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

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *bits=None*, ***kwargs*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

insn_op_type

copy()

class angr.knowledge_plugins.xrefs.**XRefManager**(*kb*)

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_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*]

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *cfg_model=None*, *kb=None*, ***kwargs*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

class angr.knowledge_plugins.xrefs.**XRefType**

Bases: object

Offset = 0

Read = 1

Write = 2

static to_string(*ty*)

class angr.knowledge_plugins.xrefs.xref.**XRef**(*ins_addr=None*, *block_addr=None*, *stmt_idx=None*, *insn_op_idx=None*, *memory_data=None*, *dst=None*, *xref_type=None*)

Bases: *Serializable*

XRef describes a reference to a MemoryData instance (if a MemoryData instance is available) or just an address.

Parameters

- **ins_addr** (*int | None*)
- **block_addr** (*int | None*)
- **stmt_idx** (*int | None*)
- **insn_op_idx** (*int | None*)
- **memory_data** (angr.knowledge_plugins.cfg.memory_data.MemoryData *| None*)
- **dst** (*int | None*)

__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

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *bits=None*, ***kwargs*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

insn_op_type

copy()

class angr.knowledge_plugins.xrefs.xref_types.**XRefType**

Bases: object

Offset = 0

Read = 1

Write = 2

static to_string(*ty*)

class angr.knowledge_plugins.xrefs.xref_manager.**XrefDict**(**args*, ***kwargs*)

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**(*kb*)

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_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*]

serialize_to_cmessage()

Serialize the class object and returns a protobuf cmessage object.

Returns
A protobuf cmessage object.

Return type
protobuf.cmessage

classmethod parse_from_cmessage(*cmsg*, *cfg_model=None*, *kb=None*, ***kwargs*)

Parse a protobuf cmessage and create a class object.

Parameters
cmsg – The probobuf cmessage object.

Returns
A unserialized class object.

Return type
cls

class angr.code_location.**CodeLocation**(*block_addr*, *stmt_idx*, *sim_procedure=None*, *ins_addr=None*, *context=None*, *block_idx=None*, ***kwargs*)

Bases: Generic

Stands for a specific program point by specifying basic block address and statement ID (for IRSBs), or SimProcedure name (for SimProcedures).

Parameters

- **block_addr** (*BlockAddr*)
- **stmt_idx** (*StmtIdx*)
- **ins_addr** (*int | None*)
- **context** (*Context*)
- **block_idx** (*int | None*)

__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**(*call_string=None*)

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.

Parameters

call_string (*tuple[int, ...] | None*)

__init__(*call_string=None*)

Constructor.

Parameters

- **block_addr** – Address of the block
- **stmt_idx** – 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** – The instruction address.
- **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__.
- **call_string** (*tuple[int, ...] | None*)

call_string

class angr.code_location.**AILCodeLocation**(*addr*, *block_idx*, *stmt_idx*, *insn_addr=None*)

Bases: object

A code location that refers precisely to a statement of an AIL block, with an optional instruction address.

Parameters

- **addr** (*int*)
- **block_idx** (*int | None*)
- **stmt_idx** (*int*)
- **insn_addr** (*int | None*)

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

class angr.keyed_region.**StoredObject**(*start*, *obj*, *size*)

Bases: object

__init__(*start*, *obj*, *size*)

start

obj

size: *UnknownSize* | int

property obj_id

class angr.keyed_region.**RegionObject**(*start*, *size*, *objects=None*)

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*)

copy()

class angr.keyed_region.**KeyedRegion**(*tree=None*, *phi_node_contains=None*, *canonical_size=8*)

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

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.

10.14 Serialization

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

class angr.vaults.**VaultPickler**(*vault*, *file*, **args*, *assigned_objects=()*, ***kwargs*)

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**(*vault*, *file*, **args*, ***kwargs*)

Bases: Unpickler

__init__(*vault*, *file*, **args*, ***kwargs*)

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**(*d=None*)

Bases: *Vault*

A Vault that uses a dictionary for storage.

__init__(*d=None*)

is_stored(*i*)

Checks if the provided id is already in the vault.

keys()

Should return the IDs stored by the vault.

class angr.vaults.**VaultDir**(*d=None*)

Bases: *Vault*

A Vault that uses a directory for storage.

__init__(*d=None*)

keys()

Should return the IDs stored by the vault.

class angr.vaults.**VaultShelf**(*path=None*)

Bases: *VaultDict*

A Vault that uses a shelve.Shelf for storage.

__init__(*path=None*)

close()

class angr.vaults.**VaultDirShelf**(*d=None*)

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*)

keys()

Should return the IDs stored by the vault.

10.15 Analysis

class angr.analyses.**CDG**(*cfg*, *start=None*, *no_construct=False*)

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**(***kwargs*)

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.

__init__(***kwargs*)

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*) – 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.
- **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** – 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** – 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).
- **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** – 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.
- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.

Returns
None

class angr.analyses.**DDG**(*cfg*, *start=None*, *call_depth=None*, *block_addrs=None*)

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**(*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*)

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}.

Parameters

- **cfg** (CFGEmulated | *None*)
- **context_sensitivity_level** (*int*)

- **start** (*int | None*)
- **function_start** (*int | None*)
- **interfunction_level** (*int*)
- **initial_state** (*SimState | None*)
- **avoid_runs** (*list[int] | None*)
- **remove_options** (*set[str] | None*)
- **timeout** (*int | None*)
- **max_iterations_before_widening** (*int*)
- **max_iterations** (*int*)
- **widening_interval** (*int*)
- **final_state_callback** (*Callable[[SimState, CallStack], Any] | None*)
- **status_callback** (*Callable[[VFG], Any] | None*)
- **record_function_final_states** (*bool*)

__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**(*vfg=None*, *start_addr=None*, *interfunction_level=0*, *context_sensitivity_level=2*, *keep_data=False*)

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**(*project*)

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**(*cfg*, *cdg*, *ddg*, *targets=None*, *cfg_node=None*, *stmt_id=None*, *control_flow_slice=False*, *same_function=False*, *no_construct=False*)

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**(*other_project*, *cfg_a=None*, *cfg_b=None*)

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**(*cfg*, *techniques*)

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*)

optimize()

class angr.analyses.**BoyScout**(*cookiesize=1*)

Bases: *Analysis*

Try to determine the architecture and endieness of a binary blob

__init__(*cookiesize=1*)

class angr.analyses.**CFGArchOptions**(*arch*, ***options*)

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**(*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*)

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**(*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*, *eh_frame=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=512*, *nodecode_threshold=0.3*, *nodecode_step=16483*, *check_funcret_max_job=500*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *retedges=False*, *drop_bad_funcs=True*, *start=None*, *end=None*, *collect_data_references=None*, *extra_cross_references=None*, *elf_eh_frame=None*, ***extra_arch_options*)

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.

Parameters

- **function_prologues** (*bool | None*)
- **force_smart_scan** (*bool | None*)
- **indirect_calls_always_return** (*bool | None*)
- **jumptable_resolver_resolves_calls** (*bool | None*)

- **retedges** (*bool*)
- **drop_bad_funcs** (*bool*)

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*, *eh_frame=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=512*, *nodecode_threshold=0.3*, *nodecode_step=16483*, *check_funcret_max_job=500*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *retedges=False*, *drop_bad_funcs=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.
- **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).
- **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.

- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.
- **force_smart_scan** (*bool | None*)
- **drop_bad_funcs** (*bool*)

Returns
None

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**(*support_jni=False*, ***kwargs*)

Bases: *CFGFast*

addr_type: Literal['int', 'block_id', 'soot'] = 'soot'

__init__(*support_jni=False*, ***kwargs*)

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*) – 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.
- **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** – 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** – 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).
- **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** – 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.
- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.

Returns
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

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**(*starts=None*, *hook_all=False*)

Bases: *Analysis*

__init__(*starts=None*, *hook_all=False*)

analyze(*addr*)

class angr.analyses.**CallingConventionAnalysis**(*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*)

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.

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** (*networkx.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*)

__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**(*func*)

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**(*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*)

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.

Parameters

- **mode** (*CallingConventionAnalysisMode*)
- **cfg** (*CFGFast | CFGModel | None*)
- **analyze_callsites** (*bool*)
- **skip_signature_matched_functions** (*bool*)
- **max_function_blocks** (*int | None*)
- **max_function_size** (*int | None*)
- **workers** (*int*)
- **cc_callback** (*Callable | None*)
- **prioritize_func_addrs** (*list[int] | set[int] | None*)
- **skip_other_funcs** (*bool*)
- **auto_start** (*bool*)

- **func_graphs** (*dict[int, networkx.DiGraph] | None*)
- **target_functions** (*set[int] | None*)

__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**(*throw=False*)

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**(*end_state*, *start_from=None*, *end_at=None*, *block_addrs=None*)

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.

Parameters

- **end_state** (SimState)
- **start_from** (*int | None*)
- **end_at** (*int | None*)
- **block_addrs** (*list[int] | None*)

__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**(*func*, *cfg=None*, *options=None*, *preset=None*, *optimization_passes=None*, *sp_tracker_track_memory=True*, *variable_kb=None*, *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=True*, *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'>*)

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

Parameters

- **func** (Function | str | int)
- **cfg** (CFGFast | CFGModel | None)
- **preset** (str | DecompilationPreset | None)
- **peephole_optimizations** (_PEEPHOLE_OPTIMIZATIONS_TYPE)
- **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)

__init__(*func*, *cfg=None*, *options=None*, *preset=None*, *optimization_passes=None*, *sp_tracker_track_memory=True*, *variable_kb=None*, *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=True*, *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'>*)

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)

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**(*function=None*, *ranges=None*, *thumb=False*, *include_ir=False*, *block_bytes=None*)

Bases: *Analysis*

Produce formatted machine code disassembly.

Parameters

- **function** (*Function | None*)
- **ranges** (*Sequence[tuple[int, int]] | None*)
- **thumb** (*bool*)
- **include_ir** (*bool*)
- **block_bytes** (*bytes | None*)

__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**(*func*, *func_graph=None*, *entry=None*, *exception_edges=False*)

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.

Parameters

- **func** (Function)
- **func_graph** (networkx.DiGraph[T_co] | None)

__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**(*func*, *max_depth=100*, *track_arg_uses=False*, *track_arg_passthru=False*)

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.

Parameters

- **func** (Function)
- **max_depth** (*int*)
- **track_arg_uses** (*bool*)
- **track_arg_passthru** (*bool*)

__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*)

class angr.analyses.**FastConstantPropagation**(*func*, *blocks=None*, *vex_cross_insn_opt=False*, *load_callback=None*)

Bases: *Analysis*

An extremely fast constant propagation analysis that finds function-wide constant values with potentially high false negative rates.

Parameters

- **func** (Function)
- **blocks** (*set[Block] | None*)
- **vex_cross_insn_opt** (*bool*)
- **load_callback** (*Callable | None*)

__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**(*sig=None*, *max_mismatched_bytes=0*, *dry_run=False*, *match_named_functions=False*)

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.

Parameters

- **sig** (FlirtSignature | str | None)
- **max_mismatched_bytes** (int)
- **dry_run** (bool)
- **match_named_functions** (bool)

__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**(*order_jobs=False*, *allow_merging=False*, *allow_widening=False*, *status_callback=None*, *graph_visitor=None*)

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.

Parameters

- **status_callback** (Callable[[ForwardAnalysis], Any] | None)
- **graph_visitor** (GraphVisitor[NodeType] | None)

__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**(*cfg=None*, *require_predecessors=True*, *only_find=None*)

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**(*func=None*, *func_graph=None*, *block=None*, *max_iterations=1*, *replacements=None*, *overlay=None*, *pointers_only=False*)

Bases: *ForwardAnalysis*, *Analysis*

Finds possible initializations for global data sections and generate an overlay to be used in other analyses later on.

__init__(*func=None*, *func_graph=None*, *block=None*, *max_iterations=1*, *replacements=None*, *overlay=None*, *pointers_only=False*)

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** (*GraphVisitor or None*) – A graph visitor to provide successors.

Returns
None

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**(*cfunc*)

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

Parameters
cfunc (CFunction)

__init__(*cfunc*)

Parameters
cfunc (CFunction)

class angr.analyses.**LoopFinder**(*functions=None*, *normalize=True*)

Bases: *Analysis*

Extracts all the loops from all the functions in a binary.

__init__(*functions=None*, *normalize=True*)

class angr.analyses.**LoopUnroller**(*graph*, *loop_body*, *unroll_times*, *save_original*, *loop_body_incomplete=False*)

Bases: *Analysis*

Unroll a loop in an AIL graph for a specified number of iterations.

Parameters

- **graph** (*networkx.DiGraph*)
- **loop_body** (*set[tuple[int, int | None]]*)
- **unroll_times** (*int*)
- **save_original** (*bool*)
- **loop_body_incomplete** (*bool*)

__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**(*cfg=None*, *region_size_threshold=32*)

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.

Parameters

- **cfg** (CFGModel | *None*)

- **region_size_threshold** (*int*)

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**(*start_state*, *goal_addr*, *cfg*, *cache_size=10000*)

Bases: *Analysis*

Parameters

- **start_state** (SimState)
- **goal_addr** (*int*)
- **cfg** (CFGModel)

__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**(*func=None*, *block=None*, *func_graph=None*, *base_state=None*, *max_iterations=30*, *load_callback=None*, *stack_pointer_tracker=None*, *only_consts=False*, *completed_funcs=None*, *do_binops=True*, *store_tops=True*, *vex_cross_insn_opt=False*, *func_addr=None*, *gp=None*, *cache_results=False*, *key_prefix=None*, *profiling=False*)

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

Parameters

- **func_addr** (*int | None*)
- **gp** (*int | None*)
- **cache_results** (*bool*)
- **key_prefix** (*str | None*)
- **profiling** (*bool*)

__init__(*func=None*, *block=None*, *func_graph=None*, *base_state=None*, *max_iterations=30*, *load_callback=None*, *stack_pointer_tracker=None*, *only_consts=False*, *completed_funcs=None*, *do_binops=True*, *store_tops=True*, *vex_cross_insn_opt=False*, *func_addr=None*, *gp=None*, *cache_results=False*, *key_prefix=None*, *profiling=False*)

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** (*GraphVisitor or None*) – A graph visitor to provide successors.
- **func_addr** (*int | None*)
- **gp** (*int | None*)
- **cache_results** (*bool*)
- **key_prefix** (*str | None*)
- **profiling** (*bool*)

Returns

None

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**(*func*, *cfg_model*, *xrefs*, *decompilation=None*, *expand_funcs=None*)

Bases: *Analysis*

Generate a proximity graph.

Parameters

- **func** (*Function*)
- **cfg_model** (*CFGModel*)
- **xrefs** (*XRefManager*)
- **decompilation** (*Decompiler | None*)
- **expand_funcs** (*set[int] | None*)

__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**(*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*)

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.

Parameters

- **subject** (*Subject | ailment.Block | Block | Function | str*)
- **observation_points** (*Iterable[ObservationPoint] | None*)
- **init_state** (*ReachingDefinitionsState | None*)
- **state_initializer** (*RDAStateInitializer | None*)
- **function_handler** (*FunctionHandler | None*)
- **dep_graph** (*DepGraph | bool | None*)
- **interfunction_level** (*int*)
- **track_liveness** (*bool*)
- **func_addr** (*int | None*)
- **element_limit** (*int*)
- **merge_into_tops** (*bool*)

__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*)

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**(*syntax='intel'*, *remove_cgc_attachments=True*, *log_relocations=True*)

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:

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**(*func*, *func_graph*, *entry=None*, *func_addr=None*, *arg_vvars=None*)

Bases: *Analysis*

Calculates LiveIn and LiveOut sets for each block in a partial-SSA function.

Parameters

- **func** (Function)
- **func_graph** (*networkx.DiGraph[Block]*)
- **entry** (Block | *None*)
- **func_addr** (*int* | *None*)
- **arg_vvars** (*list[VirtualVariable]* | *None*)

__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.**SPropagatorAnalysis**(*subject*, **, ail_manager*, *func_graph=None*, *only_consts=True*, *stack_pointer_tracker=None*, *func_args=None*, *func_addr=None*, *stack_arg_offsets=None*)

Bases: *Analysis*

Constant and expression propagation that only supports SSA AIL graphs.

Parameters

- **subject** (Block | Function)
- **ail_manager** (Manager)
- **func_graph** (*networkx.DiGraph | None*)
- **only_consts** (*bool*)
- **func_args** (*set[VirtualVariable] | None*)
- **func_addr** (*int | None*)
- **stack_arg_offsets** (*set[int] | None*)

__init__(*subject*, ***, *ail_manager*, *func_graph=None*, *only_consts=True*, *stack_pointer_tracker=None*, *func_args=None*, *func_addr=None*, *stack_arg_offsets=None*)

Parameters

- **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.**SReachingDefinitionsAnalysis**(*subject*, *func_addr=None*, *func_graph=None*, *func_args=None*, *use_callee_saved_regs_at_return=False*, *track_tmps=False*)

Bases: *Analysis*

Constant and expression propagation that only supports SSA AIL graphs.

Parameters

- **func_addr** (*int | None*)
- **func_graph** (*networkx.DiGraph[Block] | None*)
- **func_args** (*set[VirtualVariable] | None*)
- **use_callee_saved_regs_at_return** (*bool*)
- **track_tmps** (*bool*)

__init__(*subject*, *func_addr=None*, *func_graph=None*, *func_args=None*, *use_callee_saved_regs_at_return=False*, *track_tmps=False*)

class angr.analyses.**SelfModifyingCodeAnalysis**(*subject*, *max_bytes=0*, *state=None*)

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.

Parameters

- **subject** (*None | int | str | Function*)
- **max_bytes** (*int*)
- **state** (SimState | *None*)

__init__(*subject*, *max_bytes=0*, *state=None*)

Parameters

- **subject** (None | int | str | *Function*) – 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**(*func*, *reg_offsets*, *block=None*, *track_memory=True*, *cross_insn_opt=True*, *initial_reg_values=None*, *resilient=True*)

Bases: *Analysis*, *ForwardAnalysis*

Track the offset of stack pointer at the end of each basic block of a function.

Parameters

- **func** (Function | *None*)
- **reg_offsets** (*set[int]*)
- **block** (Block | *None*)

- **resilient** (*bool*)

__init__(*func*, *reg_offsets*, *block=None*, *track_memory=True*, *cross_insn_opt=True*, *initial_reg_values=None*, *resilient=True*)

Parameters

- **func** (*Function | None*)
- **reg_offsets** (*set[int]*)
- **block** (*Block | None*)
- **resilient** (*bool*)

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**(*library*, *binary=None*)

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**(*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*)

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.

Parameters

- **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*)

__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**(*func*, *max_iterations=20*, *store_live_variables=False*)

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**(*func*, *func_graph=None*, *entry_node_addr=None*, *max_iterations=2*, *low_priority=False*, *track_sp=True*, *func_args=None*, *func_ret_var=None*, *store_live_variables=False*, *unify_variables=True*, *func_arg_vvars=None*, *vvar_to_vvar=None*, *type_hints=None*)

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`.

Parameters

- **func** (Function | str | int)
- **func_graph** (networkx.DiGraph | None)
- **entry_node_addr** (int | tuple[int, int | None] | None)
- **max_iterations** (int)
- **func_args** (list[SimVariable] | None)
- **func_ret_var** (SimVariable | None)
- **func_arg_vvars** (dict[int, tuple[VirtualVariable, SimVariable]] | None)
- **vvar_to_vvar** (dict[int, int] | None)
- **type_hints** (list[tuple[atoms.VirtualVariable | atoms.MemoryLocation, str]] | None)

__init__(*func*, *func_graph=None*, *entry_node_addr=None*, *max_iterations=2*, *low_priority=False*, *track_sp=True*, *func_args=None*, *func_ret_var=None*, *store_live_variables=False*, *unify_variables=True*, *func_arg_vvars=None*, *vvar_to_vvar=None*, *type_hints=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** (GraphVisitor or None) – A graph visitor to provide successors.
- **func** (Function | str | int)
- **func_graph** (DiGraph | None)
- **entry_node_addr** (int | tuple[int, int | None] | None)
- **max_iterations** (int)
- **func_args** (list[SimVariable] | None)
- **func_ret_var** (SimVariable | None)
- **func_arg_vvars** (dict[int, tuple[VirtualVariable, SimVariable]] | None)
- **vvar_to_vvar** (dict[int, int] | None)
- **type_hints** (list[tuple[VirtualVariable | MemoryLocation, str]] | None)

Returns

None

class angr.analyses.**Veritesting**(*input_state*, *boundaries=None*, *loop_unrolling_limit=10*, *enable_function_inlining=False*, *terminator=None*, *deviation_filter=None*)

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**(*func=None*, *func_graph=None*, *block=None*, *max_iterations=1*, *replacements=None*)

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
```

__init__(*func=None*, *func_graph=None*, *block=None*, *max_iterations=1*, *replacements=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** (*GraphVisitor or None*) – A graph visitor to provide successors.

Returns

None

angr.analyses.**register_analysis**(*cls*, *name*)

class angr.analyses.analysis.**AnalysisLogEntry**(*message*, *exc_info=False*)

Bases: object

__init__(*message*, *exc_info=False*)

format()

Return type

str

class angr.analyses.analysis.**AnalysesHub**(*project*)

Bases: *PluginVendor*[Any]

This class contains functions for all the registered and runnable analyses,

__init__(*project*)

class angr.analyses.analysis.**KnownAnalysesPlugin**(**args*, ***kwargs*)

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**(*project*)

Bases: *AnalysesHub*, *KnownAnalysesPlugin*

This class has type-hinting for all built-in analyses plugin

class angr.analyses.analysis.**AnalysisFactory**(*project*, *analysis_cls*)

Bases: Generic

Parameters

- **project** (Project)
- **analysis_cls** (type[A])

__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*)

class angr.analyses.forward_analysis.**CallGraphVisitor**(*callgraph*)

Bases: GraphVisitor

Parameters
callgraph (*networkx.DiGraph*)

__init__(*callgraph*)

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

predecessors(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

Returns
A list of sorted nodes.

class angr.analyses.forward_analysis.**ForwardAnalysis**(*order_jobs=False*, *allow_merging=False*, *allow_widening=False*, *status_callback=None*, *graph_visitor=None*)

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.

Parameters

- **status_callback** (Callable[[ForwardAnalysis], Any] | None)
- **graph_visitor** (GraphVisitor[NodeType] | None)

__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**(*func*, *graph=None*)

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*)

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

predecessors(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

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).

class angr.analyses.forward_analysis.**LoopVisitor**(*loop*)

Bases: *GraphVisitor*

Parameters
loop (angr.analyses.loopfinder.Loop) – The loop to visit.

__init__(*loop*)

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

predecessors(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

Returns
A list of sorted nodes.

class angr.analyses.forward_analysis.**SingleNodeGraphVisitor**(*node*)

Bases: *GraphVisitor*

Parameters
node – The single node that should be in the graph.

__init__(*node*)

node

node_returned

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.

Returns
A node in the graph.

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

predecessors(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

Returns
A list of sorted nodes.

class angr.analyses.forward_analysis.forward_analysis.**ForwardAnalysis**(*order_jobs=False*, *allow_merging=False*, *allow_widening=False*, *status_callback=None*, *graph_visitor=None*)

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.

Parameters

- **status_callback** (Callable[[ForwardAnalysis], Any] | None)
- **graph_visitor** (GraphVisitor[NodeType] | None)

__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**(**args*, *function*, *graph*, ***kwargs*)

Bases: *ForwardAnalysis*, Generic

Parameters

- **function** (Function)
- **graph** (*networkx.DiGraph[NodeType]*)

__init__(**args*, *function*, *graph*, ***kwargs*)

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** (GraphVisitor *or* None) – A graph visitor to provide successors.

Returns
None

class angr.analyses.forward_analysis.forward_analysis.**ForwardAnalysisForClinic**(**args*, *clinic*, ***kwargs*)

Bases: *ForwardAnalysisForDummies*[AnalysisState, ailment.Block, tuple[int, int | None]], Generic

Parameters
clinic (Clinic)

__init__(**args*, *clinic*, ***kwargs*)

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** (GraphVisitor *or None*) – A graph visitor to provide successors.
- **clinic** (Clinic)

Returns
None

class angr.analyses.forward_analysis.job_info.**JobInfo**(*key*, *job*)

Bases: Generic

Stores information of each job.

Parameters

- **key** (*JobKey*)
- **job** (*JobType*)

__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.

class angr.analyses.forward_analysis.visitors.**CallGraphVisitor**(*callgraph*)

Bases: *GraphVisitor*

Parameters
callgraph (*networkx.DiGraph*)

__init__(*callgraph*)

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

predecessors(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

Returns
A list of sorted nodes.

class angr.analyses.forward_analysis.visitors.**FunctionGraphVisitor**(*func*, *graph=None*)

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*)

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

`predecessors`(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

`sort_nodes`(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

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).

class `angr.analyses.forward_analysis.visitors.`**`LoopVisitor`**(*loop*)

Bases: *`GraphVisitor`*

Parameters
loop (`angr.analyses.loopfinder.Loop`) – The loop to visit.

`__init__`(*loop*)

`successors`(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

`predecessors`(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

Returns
A list of sorted nodes.

class angr.analyses.forward_analysis.visitors.**SingleNodeGraphVisitor**(*node*)

Bases: *GraphVisitor*

Parameters
node – The single node that should be in the graph.

__init__(*node*)

node

node_returned

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.

Returns
A node in the graph.

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

predecessors(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

Returns
A list of sorted nodes.

class angr.analyses.forward_analysis.visitors.call_graph.**CallGraphVisitor**(*callgraph*)

Bases: *GraphVisitor*

Parameters
callgraph (*networkx.DiGraph*)

__init__(*callgraph*)

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

predecessors(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

Returns
A list of sorted nodes.

class angr.analyses.forward_analysis.visitors.function_graph.**FunctionGraphVisitor**(*func*, *graph=None*)

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*)

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

predecessors(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

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).

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

class angr.analyses.forward_analysis.visitors.loop.**LoopVisitor**(*loop*)

Bases: *GraphVisitor*

Parameters
loop (angr.analyses.loopfinder.Loop) – The loop to visit.

__init__(*loop*)

successors(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

`predecessors`(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

`sort_nodes`(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

Returns
A list of sorted nodes.

class `angr.analyses.forward_analysis.visitors.single_node_graph.`**`SingleNodeGraphVisitor`**(*node*)

Bases: *GraphVisitor*

Parameters
node – The single node that should be in the graph.

`__init__`(*node*)

`node`

`node_returned`

`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.

Returns
A node in the graph.

`successors`(*node*)

Get successors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of successors.

Return type
list

predecessors(*node*)

Get predecessors of a node. The node should be in the graph.

Parameters
node – The node to work with.

Returns
A list of predecessors.

sort_nodes(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

Parameters
nodes (*iterable*) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

Returns
A list of sorted nodes.

class angr.analyses.backward_slice.**BackwardSlice**(*cfg*, *cdg*, *ddg*, *targets=None*, *cfg_node=None*, *stmt_id=None*, *control_flow_slice=False*, *same_function=False*, *no_construct=False*)

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.

exception angr.analyses.bindiff.**UnmatchedStatementsException**

Bases: Exception

class angr.analyses.bindiff.**Difference**(*diff_type*, *value_a*, *value_b*)

Bases: object

__init__(*diff_type*, *value_a*, *value_b*)

class angr.analyses.bindiff.**ConstantChange**(*offset*, *value_a*, *value_b*)

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**(*block*, *function*)

Bases: object

__init__(*block*, *function*)

class angr.analyses.bindiff.**NormalizedFunction**(*function*)

Bases: object

Parameters

function (Function)

__init__(*function*)

Parameters

function (Function)

class angr.analyses.bindiff.**FunctionDiff**(*function_a*, *function_b*, *bindiff=None*)

Bases: object

This class computes the a diff between two functions.

Parameters

- **function_a** (Function)
- **function_b** (Function)

__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**(*other_project*, *cfg_a=None*, *cfg_b=None*)

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.boyscout.**BoyScout**(*cookiesize=1*)

Bases: *Analysis*

Try to determine the architecture and endieness of a binary blob

__init__(*cookiesize=1*)

class angr.analyses.calling_convention.**CallingConventionAnalysis**(*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*)

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.

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** (*networkx.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*)

__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**(*func*, *max_depth=100*, *track_arg_uses=False*, *track_arg_passthru=False*)

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.

Parameters

- **func** (Function)
- **max_depth** (*int*)
- **track_arg_uses** (*bool*)
- **track_arg_passthru** (*bool*)

__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*)

class angr.analyses.complete_calling_conventions.**CallingConventionAnalysisMode**(**values*)

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**(*mode=CallingConvention*
recover_variables=False,
low_priority=False,
force=False,
cfg=None,
analyze_callsites=False,
*skip_signature_matched_*
max_function_blocks=No
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)

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.

Parameters

- **mode** (CallingConventionAnalysisMode)
- **cfg** (CFGFast | CFGModel | *None*)
- **analyze_callsites** (*bool*)
- **skip_signature_matched_functions** (*bool*)
- **max_function_blocks** (*int* | *None*)
- **max_function_size** (*int* | *None*)

- **workers** (*int*)
- **cc_callback** (*Callable | None*)
- **prioritize_func_addrs** (*list[int] | set[int] | None*)
- **skip_other_funcs** (*bool*)
- **auto_start** (*bool*)
- **func_graphs** (*dict[int, networkx.DiGraph] | None*)
- **target_functions** (*set[int] | None*)

__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

exception angr.analyses.soot_class_hierarchy.**SootClassHierarchyError**(*msg*)

Bases: Exception

__init__(*msg*)

exception angr.analyses.soot_class_hierarchy.**NoConcreteDispatch**(*msg*)

Bases: *SootClassHierarchyError*

__init__(*msg*)

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*)

class angr.analyses.cfg.**CFG**(**kwargs)

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.

__init__(**kwargs)

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*) – 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.
- **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** – 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** – 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).
- **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** – 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.
- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.

Returns
None

class angr.analyses.cfg.**CFBlanket**(*exclude_region_types=None*, *on_object_added=None*)

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

Parameters

- **exclude_region_types** (*set[str] | None*)
- **on_object_added** (Callable*[[int, Any], None] | None*)

__init__(*exclude_region_types=None*, *on_object_added=None*)

Parameters

- **on_object_added** (Callable[[int, Any], None] | None) – Callable with parameters (addr, obj) called after an object is added to the blanket.
- **exclude_region_types** (*set[str] | None*)

property regions

Return all memory regions.

floor_addr(*addr*)

floor_item(*addr*)

floor_items(*addr=None*, *reverse=False*)

ceiling_addr(*addr*)

ceiling_item(*addr*)

ceiling_items(*addr=None*, *reverse=False*, *include_first=True*)

add_obj(*addr*, *obj*)

Adds an object *obj* to the blanket at the specified address *addr*

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**(*arch*, ***options*)

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**(*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*)

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

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**(*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*)

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`(*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*, *eh_frame=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=512*, *nodecode_threshold=0.3*, *nodecode_step=16483*, *check_funcret_max_job=500*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *retedges=False*, *drop_bad_funcs=True*, *start=None*, *end=None*, *collect_data_references=None*, *extra_cross_references=None*, *elf_eh_frame=None*, ***extra_arch_options*)

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.

Parameters

- **function_prologues** (*bool | None*)
- **force_smart_scan** (*bool | None*)
- **indirect_calls_always_return** (*bool | None*)
- **jumptable_resolver_resolves_calls** (*bool | None*)
- **retedges** (*bool*)
- **drop_bad_funcs** (*bool*)

```
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*, *eh_frame=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=512*, *nodecode_threshold=0.3*, *nodecode_step=16483*, *check_funcret_max_job=500*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *retedges=False*, *drop_bad_funcs=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.
- **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).
- **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.
- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.
- **force_smart_scan** (*bool | None*)
- **drop_bad_funcs** (*bool*)

Returns
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**(*support_jni=False*, ***kwargs*)

Bases: *CFGFast*

addr_type: Literal['int', 'block_id', 'soot'] = 'soot'

__init__(*support_jni=False*, ***kwargs*)

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*) – 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.
- **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** – 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** – 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).
- **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** – 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.
- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.

Returns
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

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.cfg.cfb.**CFBlanketView**(*cfb*)

Bases: object

A view into the control-flow blanket.

__init__(*cfb*)

class angr.analyses.cfg.cfb.**MemoryRegion**(*addr*, *size*, *type_*, *object_*, *cle_region*)

Bases: object

__init__(*addr*, *size*, *type_*, *object_*, *cle_region*)

class angr.analyses.cfg.cfb.**Unknown**(*addr*, *size*, *bytes_=None*, *object_=None*, *segment=None*, *section=None*)

Bases: object

__init__(*addr*, *size*, *bytes_=None*, *object_=None*, *segment=None*, *section=None*)

class angr.analyses.cfg.cfb.**CFBlanket**(*exclude_region_types=None*, *on_object_added=None*)

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

Parameters

- **exclude_region_types** (*set[str] | None*)
- **on_object_added** (*Callable[[int, Any], None] | None*)

__init__(*exclude_region_types=None*, *on_object_added=None*)

Parameters

- **on_object_added** (Callable[[int, Any], None] | None) – Callable with parameters (addr, obj) called after an object is added to the blanket.
- **exclude_region_types** (*set[str] | None*)

property regions

Return all memory regions.

floor_addr(*addr*)

floor_item(*addr*)

floor_items(*addr=None*, *reverse=False*)

ceiling_addr(*addr*)

ceiling_item(*addr*)

ceiling_items(*addr=None*, *reverse=False*, *include_first=True*)

add_obj(*addr*, *obj*)

Adds an object *obj* to the blanket at the specified address *addr*

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

exception angr.analyses.cfg.cfg.**OutdatedError**

Bases: Exception

class angr.analyses.cfg.cfg.**CFG**(**kwargs)

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.

__init__(**kwargs)

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*) – 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.
- **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** – 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** – 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).
- **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** – 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.

- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.

Returns
None

class angr.analyses.cfg.cfg_emulated.**CFGJob**(**args*, ***kwargs*)

Bases: *CFGJobBase*

The job class that CFGEmulated uses.

__init__(**args*, ***kwargs*)

property block_id

property is_syscall

class angr.analyses.cfg.cfg_emulated.**PendingJob**(*caller_func_addr*, *returning_source*, *state*, *src_block_id*, *src_exit_stmt_idx*, *src_exit_ins_addr*, *call_stack*)

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**(*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*)

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.cfg_base.**CFGBase**(*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*)

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

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

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**(*addr*, *size*, *mode*)

Bases: object

Describes the decoding mode (ARM/THUMB) for a given basic block identified by its address.

Parameters

- **addr** (*int*)
- **size** (*int*)
- **mode** (*int*)

__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**(*callee_func_addr*, *caller_func_addr*, *call_site_addr*, *return_to*)

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**(*kb*, *deregister_job_callback*)

Bases: object

A collection of pending jobs during CFG recovery.

__init__(*kb*, *deregister_job_callback*)

add_job(*job*)

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.**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**(*src_node*, *dst_addr*, *src_func_addr*, *to_outside=False*, *dst_func_addr=None*, *stmt_idx=None*, *ins_addr=None*, *is_exception=False*)

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**(*src_node*, *dst_addr*, *ret_addr*, *src_func_addr*, *syscall=False*, *stmt_idx=None*, *ins_addr=None*)

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**(*src_node*, *dst_addr*, *src_func_addr*, *confirmed=None*)

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**(*ret_from_addr*, *ret_to_addr*, *dst_func_addr*)

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**(**values*)

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**(*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*)

Bases: object

Defines a job to work on during the CFG recovery

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*)

__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**(*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*, *eh_frame=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=512*, *nodecode_threshold=0.3*, *nodecode_step=16483*, *check_funcret_max_job=500*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *retedges=False*, *drop_bad_funcs=True*, *start=None*, *end=None*, *collect_data_references=None*, *extra_cross_references=None*, *elf_eh_frame=None*, ***extra_arch_options*)

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.

Parameters

- **function_prologues** (*bool | None*)
- **force_smart_scan** (*bool | None*)
- **indirect_calls_always_return** (*bool | None*)
- **jumptable_resolver_resolves_calls** (*bool | None*)

- **retedges** (*bool*)
- **drop_bad_funcs** (*bool*)

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*, *eh_frame=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=512*, *nodecode_threshold=0.3*, *nodecode_step=16483*, *check_funcret_max_job=500*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *retedges=False*, *drop_bad_funcs=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.
- **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).
- **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.

- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.
- **force_smart_scan** (*bool | None*)
- **drop_bad_funcs** (*bool*)

Returns
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.cfg_arch_options.**CFGArchOptions**(*arch*, ***options*)

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.cfg_job_base.**FunctionKey**(*addr*, *callsite_tuples*)

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**(*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*)

Bases: object

Describes an entry in CFG or VFG. Only used internally by the analysis.

Parameters

- **state** (SimState)
- **jumpkind** (*str | None*)

__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

class angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got.**AMD64ElfGotResolver**(*project*)

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver that resolves GOT entries on AMD64 ELF binaries.

__init__(*project*)

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

class angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast.**ArmElfFastResolver**(*project*)

Bases: *IndirectJumpResolver*

Resolves indirect jumps in ARM ELF binaries

__init__(*project*)

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*)

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.x86_pe_iat.**X86PeIatResolver**(*project*)

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver for IAT in x86 PEs and xbes.

__init__(*project*)

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.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**(**values*)

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**(*project*)

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver for R9-based indirect function calls in MIPS ELFs.

__init__(*project*)

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*)

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.x86_elf_pic_plt.**X86ElfPicPltResolver**(*project*)

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*)

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.default_resolvers.**default_indirect_jump_resolvers**(*obj*, *project*)

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**(**values*)

Bases: int, Enum

Address transformation operations.

Assignment = 0

SignedExtension = 1

UnsignedExtension = 2

Truncation = 3

Or1 = 4

ShiftLeft = 5

ShiftRight = 6

Add = 7

Load = 8

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**AddressTransformation**(*op*, *operands*, *first_load=False*)

Bases: object

Describe and record an address transformation operation.

Parameters

- **op** (AddressTransformationTypes)
- **operands** (*list*)
- **first_load** (*bool*)

__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**(*tmp_idx*)

Bases: object

For modeling Tmp variables.

__init__(*tmp_idx*)

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**JumpTargetBaseAddr**(*stmt_loc*, *stmt*, *tmp*, *base_addr=None*, *tmp_1=None*)

Bases: object

Model for jump targets and their data origin.

Parameters

- **stmt** (*pyvex.stmt.IRStmt*)
- **tmp** (*int*)
- **base_addr** (*int | None*)
- **tmp_1** (*int | None*)

__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**(*arch*)

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**(*reg_offset*)

Bases: Annotation

Register Offset annotation.

Parameters
reg_offset (RegisterOffset)

__init__(*reg_offset*)

Parameters
reg_offset (RegisterOffset)

reg_offset

property relocatable

Returns whether this annotation can be relocated in a simplification.

Returns
True if it can be relocated, false otherwise.

property eliminatable

Returns whether this annotation can be eliminated in a simplification.

Returns
True if eliminatable, False otherwise

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**JumpTableProcessor**(*project*, *indirect_jump_node_pred_addrs*, *bp_sp_diff=256*)

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)
```

(continues on next page)

(continued from previous page)

```
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.

Parameters
indirect_jump_node_pred_addrs (*set[int]*)

__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.**RegisterInitializerHook**(*reg_offset*, *reg_bits*, *initial_value*)

Bases: object

Hook for register init.

__init__(*reg_offset*, *reg_bits*, *initial_value*)

hook(*state*)

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**BSSHook**(*project*, *bss_regions*)

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**(*gp_offset*, *gp*)

Bases: object

Hooks all reads from and writes into the gp register for MIPS32 binaries.

Parameters

- **gp_offset** (*int*)
- **gp** (*int*)

__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**(*project*, *resolve_calls=True*)

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.

Parameters

resolve_calls (*bool*)

__init__(*project*, *resolve_calls=True*)

Parameters

resolve_calls (*bool*)

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*)

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.const_resolver.**exists_in_replacements**(*replacements*, *block_loc*, *tmp_var*)

class angr.analyses.cfg.indirect_jump_resolvers.const_resolver.**ConstantResolver**(*project*, *max_func_nodes=512*)

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.

Parameters
max_func_nodes (*int*)

__init__(*project*, *max_func_nodes=512*)

Parameters
max_func_nodes (*int*)

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*)

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.resolver.**IndirectJumpResolver**(*project*, *timeless=False*, *base_state=None*)

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

class angr.analyses.cfg.indirect_jump_resolvers.**AArch64MachOGotResolver**(*project*)

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver that resolves GOT entries on AArch64 MachO binaries.

__init__(*project*)

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*)

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**(*project*)

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver that resolves GOT entries on AMD64 ELF binaries.

__init__(*project*)

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

class angr.analyses.cfg.indirect_jump_resolvers.**AMD64PeIatResolver**(*project*)

Bases: *IndirectJumpResolver*

A timeless indirect call/jump resolver for IAT in amd64 PEs.

__init__(*project*)

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

class angr.analyses.cfg.indirect_jump_resolvers.**ArmElfFastResolver**(*project*)

Bases: *IndirectJumpResolver*

Resolves indirect jumps in ARM ELF binaries

__init__(*project*)

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*)

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**(*project*, *max_func_nodes=512*)

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.

Parameters
max_func_nodes (*int*)

__init__(*project*, *max_func_nodes=512*)

Parameters
max_func_nodes (*int*)

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*)

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**(*project*, *resolve_calls=True*)

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.

Parameters
resolve_calls (*bool*)

__init__(*project*, *resolve_calls=True*)

Parameters
resolve_calls (*bool*)

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*)

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**(*project*)

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*)

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*)

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**(*project*)

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver for R9-based indirect function calls in MIPS ELFs.

__init__(*project*)

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*)

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**(*project*)

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*)

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

class angr.analyses.cfg.indirect_jump_resolvers.**SyscallResolver**(*project*)

Bases: *IndirectJumpResolver*

Resolve syscalls to SimProcedures.

__init__(*project*)

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** (*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

class angr.analyses.cfg.indirect_jump_resolvers.**X86ElfPicPltResolver**(*project*)

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*)

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

class angr.analyses.cfg.indirect_jump_resolvers.**X86PeIatResolver**(*project*)

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver for IAT in x86 PEs and xbes.

__init__(*project*)

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

class angr.analyses.cfg.cfg_fast_soot.**CFGFastSoot**(*support_jni=False*, ***kwargs*)

Bases: *CFGFast*

addr_type: Literal['int', 'block_id', 'soot'] = 'soot'

__init__(*support_jni=False*, ***kwargs*)

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*) – 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.
- **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** – 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** – 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).
- **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** – 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.
- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.

Returns

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

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.cdg.**CDG**(*cfg*, *start=None*, *no_construct=False*)

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

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.

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**(*func*)

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.angrdb.**AngrDB**(*project=None*, *nullpool=False*)

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 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*)

class angr.angrdb.db.**AngrDB**(*project=None*, *nullpool=False*)

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 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*)

class angr.angrdb.models.**DbInformation**(***kwargs*)

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**(***kwargs*)

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**(***kwargs*)

Bases: Base

Models a knowledge base.

id

name

cfgs

funcs

xrefs

comments

labels

var_collections

structured_code

__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**(***kwargs*)

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**(***kwargs*)

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.**DbVariableCollection**(***kwargs*)

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.**DbStructuredCode**(**kwargs)

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.**DbXRefs**(**kwargs)

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**(**kwargs)

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.**DbLabel**(***kwargs*)

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.

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.jar.Jar'>: 'jar', <class 'cle.backends.macho.macho.MachO'>: 'mach-o', <class 'cle.backends.minidump.Minidump'>: 'minidump', <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.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*)

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*)

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

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*)

Parameters

- **session**
- **db_kb** (*DbKnowledgeBase*)
- **kb** (*KnowledgeBase*)

Returns
A loaded function manager.

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

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

class angr.angrdb.serializers.loader.**LoadArgsJSONEncoder**(**, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: JSONEncoder

A JSON encoder that supports serializing bytes.

default(*o*)

Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return super().default(o)
```

class angr.angrdb.serializers.loader.**LoadArgsJSONDecoder**

Bases: JSONDecoder

A JSON decoder that supports unserializing into bytes.

__init__()

object_hook, if specified, will be called with the result of every JSON object decoded and its return value will be used in place of the given dict. This can be used to provide custom deserializations (e.g. to support JSON-RPC class hinting).

object_pairs_hook, if specified will be called with the result of every JSON object decoded with an ordered list of pairs. The return value of object_pairs_hook will be used instead of the dict. This feature can be used to implement custom decoders. If object_hook is also defined, the object_pairs_hook takes priority.

parse_float, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

parse_int, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

parse_constant, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

If strict is false (true is the default), then control characters will be allowed inside strings. Control characters in this context are those with character codes in the 0-31 range, including '\t' (tab), '\n', '\r' and '\0'.

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.jar.Jar'>: 'jar', <class 'cle.backends.macho.macho.MachO'>: 'mach-o', <class 'cle.backends.minidump.Minidump'>: 'minidump', <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.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*)

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

class angr.angrdb.serializers.variables.**VariableManagerSerializer**

Bases: object

Serialize/unserialize a variable manager and its variables.

static dump(*session*, *db_kb*, *var_manager*)

Parameters

- **db_kb** (DbKnowledgeBase)
- **var_manager** (VariableManager)

static dump_internal(*session*, *db_kb*, *internal_manager*, *func_addr*, *ident=None*)

Parameters

- **db_kb** (DbKnowledgeBase)
- **internal_manager** (VariableManagerInternal)
- **func_addr** (*int*)

static load(*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)

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*)

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*)

Parameters

- **session**
- **db_kb** (*DbKnowledgeBase*)
- **kb** (*KnowledgeBase*)

Return type
StructuredCodeManager

Returns
A loaded structured code manager

class angr.analyses.decompiler.structuring.recursive_structurer.**RecursiveStructurer**(*region*, *cond_proc=None*, *func=None*, *structurer_cls=None*, *, *ail_manager*, **kwargs*)

Bases: *Analysis*

Recursively structure a region and all of its subregions.

Parameters

- **func** (Function | *None*)
- **structurer_cls** (type | *None*)
- **ail_manager** (Manager)

__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.**DEFAULT_STRUCTURER**

alias of *SAILRStructurer*

class angr.analyses.decompiler.structuring.**DreamStructurer**(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, ***kwargs*)

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.

Parameters

- **func** (Function | *None*)
- **case_entry_to_switch_head** (*dict[int, int]* | *None*)

NAME: str = 'dream'

__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**(*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_C*
multistmtexpr_stmt_threshold=5, ***kwargs*)

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.

Parameters

- **func** (Function | *None*)
- **case_entry_to_switch_head** (*dict[int, int]* | *None*)
- **use_multistmtexprs** (MultiStmtExprMode)
- **multistmtexpr_stmt_threshold** (*int*)

NAME: str = 'phoenix'

__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(*graph*, *old_node_0*, *new_node*, *old_node_1=None*, *self_loop=True*, *update_node_order=False*, *drop_refinement_marks=False*)

Parameters

- **update_node_order** (bool)
- **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**(*region*, *cond_proc=None*, *func=None*, *structurer_cls=None*, ***, *ail_manager*, ***kwargs*)

Bases: Analysis

Recursively structure a region and all of its subregions.

Parameters

- **func** (Function | None)
- **structurer_cls** (type | None)
- **ail_manager** (Manager)

__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**(*region*, *improve_phoenix=True*, *postdom_max_edges=10*, *postdom_max_graph_size=50*, ***kwargs*)

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_class_from_name**(*name*)

Return type
type | None

Parameters
name (*str*)

class angr.analyses.decompiler.structuring.dream.**DreamStructurer**(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, ***kwargs*)

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.

Parameters

- **func** (Function | None)

- **case_entry_to_switch_head** (*dict[int, int] | None*)

NAME: str = 'dream'

__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.structurer_base.**StructurerBase**(*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*)

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.

Parameters

- **func** (Function | *None*)
- **case_entry_to_switch_head** (*dict[int, int] | None*)
- **jump_tables** (*dict[int, IndirectJump] | None*)
- **ail_manager** (Manager | *None*)

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*)

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**(**values*)

Bases: StrEnum

Mode of multi-statement expression creation during structuring.

NEVER = 'Never'

ALWAYS = 'Always'

MAX_ONE_CALL = 'Only when less than one call'

class angr.analyses.decompiler.structuring.phoenix.**GraphEdgeFilter**(*graph*)

Bases: object

Filters away edges in a graph that are marked as deleted (outgoing-edges) during cyclic refinement.

Parameters
graph (*networkx.DiGraph*)

__init__(*graph*)

Parameters
graph (*DiGraph*)

class angr.analyses.decompiler.structuring.phoenix.**PhoenixStructurer**(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, *improve_algorithm=False*, *use_multistmtexprs=MultiStmtExprMode.MA*, *multistmtexpr_stmt_threshold=5*, ***kwargs*)

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.

Parameters

- **func** (*Function | None*)
- **case_entry_to_switch_head** (*dict[int, int] | None*)
- **use_multistmtexprs** (*MultiStmtExprMode*)
- **multistmtexpr_stmt_threshold** (*int*)

NAME: str = 'phoenix'

__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(*graph*, *old_node_0*, *new_node*, *old_node_1=None*, *self_loop=True*, *update_node_order=False*, *drop_refinement_marks=False*)

Parameters

- **update_node_order** (*bool*)

- **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.**AILSimplifier**(*func*, *func_graph*, *ail_manager*, *remove_dead_memdefs=False*, *stack_arg_offsets=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*)

Bases: *Analysis*

Perform function-level simplifications.

Parameters

- **func** (Function)
- **func_graph** (*networkx.DiGraph[Block]*)
- **ail_manager** (Manager)
- **stack_arg_offsets** (*set[tuple[int, int]] | None*)
- **gp** (*int | None*)
- **removed_vvar_ids** (*set[int] | None*)
- **arg_vvars** (*dict[int, tuple[VirtualVariable, SimVariable]] | None*)
- **avoid_vvar_ids** (*set[int] | None*)

__init__(*func*, *func_graph*, *ail_manager*, *remove_dead_memdefs=False*, *stack_arg_offsets=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.**BlockSimplifier**(*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*)

Bases: *Analysis*

Simplify an AIL block.

Parameters

- **block** (Block | *None*)
- **ail_manager** (Manager)
- **func_addr** (*int* | *None*)
- **peephole_optimizations** (*None* | *Iterable[*type*[*PeepholeOptimizationStmtBase*]* | type*[*PeepholeOptimizationExprBase*]* | type*[*PeepholeOptimizationMultiStmtBase*]]*)
- **preserve_vvar_ids** (*set[int]* | *None*)
- **type_hints** (*list[tuple[*atoms.VirtualVariable | atoms.MemoryLocation, *str]]* | *None*)

__init__(*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*)

Parameters

- **block** (*Block* | None) – The AIL block to simplify. Setting it to None to skip calling self._analyze(), which is useful in test cases.
- **ail_manager** (Manager)
- **func_addr** (*int* | *None*)
- **peephole_optimizations** (*None* | *Iterable[*type*[*PeepholeOptimizationStmtBase*]* | type*[*PeepholeOptimizationExprBase*]* | type*[*PeepholeOptimizationMultiStmtBase*]]*)
- **preserve_vvar_ids** (*set[int]* | *None*)
- **type_hints** (*list[tuple[*VirtualVariable | MemoryLocation, *str]]* | *None*)

class angr.analyses.decompiler.**CStructuredCodeGenerator**(*func*, *sequence*, *indent=0*, *cfg=None*, *variable_kb=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*)

Bases: *BaseStructuredCodeGenerator*, *Analysis*

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*)

__init__(*func*, *sequence*, *indent=0*, *cfg=None*, *variable_kb=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*)

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*)

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
tuple[str, *PositionMapping*, *PositionMapping*, *InstructionMapping*, dict[Any, set[Any]]]

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)

class angr.analyses.decompiler.**CallSiteMaker**(*block*, **, ail_manager*, *reaching_definitions=None*, *stack_pointer_tracker=None*)

Bases: *Analysis*

Add calling convention, declaration, and args to a call site.

Parameters
ail_manager (Manager)

__init__(*block*, **, ail_manager*, *reaching_definitions=None*, *stack_pointer_tracker=None*)

Parameters
ail_manager (Manager)

class angr.analyses.decompiler.**Clinic**(*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*, *variable_kb=None*, *reset_variable_names=False*, *rewrite_ites_to_diamonds=True*, *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*, *complete_successors=False*, *typehoon_cls=<class 'angr.analyses.typehoon.typehoon.Typehoon'>*, *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'*)

Bases: *Analysis*

A Clinic deals with AILments.

Parameters

- **func** (Function)
- **peephole_optimizations** (*None | Iterable[type[PeepholeOptimizationStmtBase] | type[PeepholeOptimizationExprBase]]*)
- **must_struct** (*set[str] | None*)
- **variable_kb** (KnowledgeBase *| None*)
- **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*)
- **complete_successors** (*bool*)
- **max_type_constraints** (*int*)
- **type_constraint_set_degradation_threshold** (*int*)
- **ail_graph** (*networkx.DiGraph | None*)
- **arg_vvars** (*dict[int, tuple[ailment.Expr.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*)

__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*, *variable_kb=None*, *reset_variable_names=False*, *rewrite_ites_to_diamonds=True*, *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*, *complete_successors=False*, *typehoon_cls=<class 'angr.analyses.typehoon.typehoon.Typehoon'>*, *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'*)

Parameters

- **func** (*Function*)
- **peephole_optimizations** (*None | Iterable[type[PeepholeOptimizationStmtBase] | type[PeepholeOptimizationExprBase]]*)
- **must_struct** (*set[str] | None*)
- **variable_kb** (*KnowledgeBase | None*)
- **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*)
- **complete_successors** (*bool*)
- **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*)

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.

class angr.analyses.decompiler.**Decompiler**(*func*, *cfg=None*, *options=None*, *preset=None*, *optimization_passes=None*, *sp_tracker_track_memory=True*, *variable_kb=None*, *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=True*, *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'>*)

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

Parameters

- **func** (Function | *str* | *int*)
- **cfg** (CFGFast | CFGModel | *None*)
- **preset** (*str* | *DecompilationPreset* | *None*)
- **peephole_optimizations** (*_PEEPHOLE_OPTIMIZATIONS_TYPE*)

- **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*)

__init__(*func*, *cfg=None*, *options=None*, *preset=None*, *optimization_passes=None*, *sp_tracker_track_memory=True*, *variable_kb=None*, *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=True*, *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'>*)

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*)

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**(*func*, *ail_graph*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_kb=None*)

Bases: DephicationBase

GraphDephication removes phi expressions from an AIL graph, essentially transforms a partial-SSA form of AIL graph to a normal AIL graph.

Parameters

- **func** (*Function | str*)
- **vvar_to_vvar_mapping** (*dict[int, int] | None*)
- **rewrite** (*bool*)
- **variable_kb** (*KnowledgeBase | None*)

__init__(*func*, *ail_graph*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_kb=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_kb** (*KnowledgeBase | None*)

class angr.analyses.decompiler.**ImportSourceCode**(*function*, *flavor='source'*, *source_root=None*, *encoding='utf-8'*)

Bases: *BaseStructuredCodeGenerator*, *Analysis*

__init__(*function*, *flavor='source'*, *source_root=None*, *encoding='utf-8'*)

regenerate_text()

class angr.analyses.decompiler.**RegionIdentifier**(*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*, *complete_successors=False*, *entry_node_addr=None*)

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 GraphRegion object whose nodes are either Blocks or GraphRegions.

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.

Parameters

- **func** (*Function*)
- **cond_proc** (*ConditionProcessor | None*)
- **graph** (*networkx.DiGraph[Block] | None*)
- **ail_manager** (*Manager | None*)
- **entry_node_addr** (*tuple[int, int | None] | None*)

__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*, *complete_successors=False*, *entry_node_addr=None*)

static slice_graph(*graph*, *node*, *frontier*, *include_frontier=False*)

Generate a slice of the graph from the head node to the given frontier.

Parameters

- **graph** (*networkx.DiGraph*) – The graph to work on.
- **node** – The starting node in the graph.
- **frontier** – A list of frontier nodes.
- **include_frontier** (*bool*) – Whether the frontier nodes are included in the slice or not.

Returns
A subgraph.

Return type
networkx.DiGraph[TNode]

test_reducibility()

Return type
bool

class angr.analyses.decompiler.**RegionSimplifier**(*func*, *region*, *ail_manager*, *arg_vvars=None*, *simplify_switches=True*, *simplify_ifelse=True*, *variable_manager=None*, *loopctr_naming=True*)

Bases: *Analysis*

Simplifies a given region.

Parameters

- **ail_manager** (Manager)
- **arg_vvars** (*set[int] | None*)
- **simplify_switches** (*bool*)
- **simplify_ifelse** (*bool*)
- **variable_manager** (VariableManagerInternal | *None*)
- **loopctr_naming** (*bool*)

__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**(*func*, *seq_node*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_kb=None*)

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.

Parameters

- **func** (Function | *str*)
- **vvar_to_vvar_mapping** (*dict[int, int] | None*)
- **rewrite** (*bool*)
- **variable_kb** (KnowledgeBase | *None*)

__init__(*func*, *seq_node*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_kb=None*)

Parameters

- **func** (Function | *str*)
- **vvar_to_vvar_mapping** (*dict[int, int] | None*)
- **rewrite** (*bool*)
- **variable_kb** (KnowledgeBase | *None*)

class angr.analyses.decompiler.**Ssailification**(*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*)

Bases: *Analysis*

Ssailification (SSA-AIL-ification) transforms an AIL graph to its partial-SSA form.

Parameters

- **func** (*Function | str*)
- **ail_graph** (*networkx.DiGraph[Block]*)
- **entry** (*Block | None*)
- **canonical_size** (*int*)
- **func_addr** (*int | None*)
- **ssa_stackvars** (*bool*)
- **ssa_tmps** (*bool*)
- **func_args** (*set[VirtualVariable] | None*)
- **rewrite_vvars** (*set[int] | None*)
- **vvar_id_start** (*int*)

__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*

exception angr.analyses.decompiler.ail_simplifier.**HasCallNotification**

Bases: Exception

Notifies the existence of a call statement.

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**(***kwargs*)

Bases: *AILBlockViewer*

Collects any temporaries used in a block.

__init__(***kwargs*)

class angr.analyses.decompiler.ail_simplifier.**DefEqRelation**(**values*)

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**(*varid*, *zero_high_bits*)

Bases: *AILBlockRewriter*

Rewrites expressions whose high bits are definitely zero to constants (if possible) or mask them with masks properly.

Parameters

- **varid** (*int*)
- **zero_high_bits** (*int*)

__init__(*varid*, *zero_high_bits*)

Parameters

- **varid** (*int*)
- **zero_high_bits** (*int*)

class angr.analyses.decompiler.ail_simplifier.**AILSimplifier**(*func*, *func_graph*, *ail_manager*, *remove_dead_memdefs=False*, *stack_arg_offsets=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*)

Bases: *Analysis*

Perform function-level simplifications.

__init__(*func*, *func_graph*, *ail_manager*, *remove_dead_memdefs=False*, *stack_arg_offsets=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

exception angr.analyses.decompiler.ailgraph_walker.**RemoveNodeNotice**

Bases: Exception

class angr.analyses.decompiler.ailgraph_walker.**AILGraphWalker**(*graph*, *handler*, *replace_nodes=False*, *strict_order_start=()*)

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]*)

class angr.analyses.decompiler.block_simplifier.**HasCallExprWalker**

Bases: *AILBlockViewer*

Test if an expression contains a call expression inside.

__init__()

class angr.analyses.decompiler.block_simplifier.**BlockSimplifier**(*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*)

Bases: *Analysis*

Simplify an AIL block.

Parameters

- **block** (*Block | None*)
- **ail_manager** (*Manager*)
- **func_addr** (*int | None*)

- **peephole_optimizations** (*None | Iterable[type[PeepholeOptimizationStmtBase] | type[PeepholeOptimizationExprBase] | type[PeepholeOptimizationMultiStmtBase]]*)
- **preserve_vvar_ids** (*set[int] | None*)
- **type_hints** (*list[tuple[atoms.VirtualVariable | atoms.MemoryLocation, str]] | None*)

__init__(*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*)

Parameters

- **block** (*Block* | None) – The AIL block to simplify. Setting it to None to skip calling self._analyze(), which is useful in test cases.
- **ail_manager** (Manager)
- **func_addr** (*int | None*)
- **peephole_optimizations** (*None | Iterable[type[PeepholeOptimizationStmtBase] | type[PeepholeOptimizationExprBase] | type[PeepholeOptimizationMultiStmtBase]]*)
- **preserve_vvar_ids** (*set[int] | None*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]] | None*)

class angr.analyses.decompiler.callsite_maker.**CallSiteMaker**(*block*, ***, *ail_manager*, *reaching_definitions=None*, *stack_pointer_tracker=None*)

Bases: *Analysis*

Add calling convention, declaration, and args to a call site.

Parameters
ail_manager (Manager)

__init__(*block*, ***, *ail_manager*, *reaching_definitions=None*, *stack_pointer_tracker=None*)

Parameters
ail_manager (Manager)

class angr.analyses.decompiler.ccall_rewriters.rewriter_base.**CCallRewriterBase**(*ccall*, *project*, *ail_manager*, *rename_ccalls=False*)

Bases: object

The base class for CCall rewriters.

Parameters

- **ccall** (*ailment.Expr.VEXCCallExpression*)
- **ail_manager** (Manager)
- **rename_ccalls** (*bool*)

__init__(*ccall*, *project*, *ail_manager*, *rename_ccalls=False*)

Parameters

- **ccall** (VEXCCallExpression)
- **ail_manager** (Manager)
- **rename_ccalls** (*bool*)

project

ail_manager

result: *Expression* | None

class angr.analyses.decompiler.ccall_rewriters.amd64_ccalls.**AMD64CCallRewriter**(*ccall*, *project*, *ail_manager*, *rename_ccalls=False*)

Bases: *CCallRewriterBase*

Implements VEX ccall rewriter for AMD64.

Parameters

- **ccall** (*ailment.Expr.VEXCCallExpression*)
- **ail_manager** (Manager)
- **rename_ccalls** (*bool*)

class angr.analyses.decompiler.clinic.**BlockCache**(*rd*, *prop*)

Bases: tuple

prop

Alias for field number 1

rd

Alias for field number 0

class angr.analyses.decompiler.clinic.**ClinicMode**(**values*)

Bases: Enum

Analysis mode for Clinic.

DECOMPILE = 1

COLLECT_DATA_REFS = 2

class angr.analyses.decompiler.clinic.**DataRefDesc**(*data_addr*, *data_size*, *block_addr*, *stmt_idx*, *ins_addr*, *data_type_str*)

Bases: object

The fields of this class is compatible with items inside IRSB.data_refs.

Parameters

- **data_addr** (*int*)
- **data_size** (*int*)
- **block_addr** (*int*)

- **stmt_idx** (*int*)
- **ins_addr** (*int*)
- **data_type_str** (*str*)

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**(**values*)

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

class angr.analyses.decompiler.clinic.**ComboRegReferenceWalker**(*project*, *ail_manager*)

Bases: *AILBlockRewriter*

Rewrite references to combo registers to load from the combo register.

__init__(*project*, *ail_manager*)

class angr.analyses.decompiler.clinic.**Clinic**(*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*, *variable_kb=None*, *reset_variable_names=False*, *rewrite_ites_to_diamonds=True*, *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*, *complete_successors=False*, *typehoon_cls=<class 'angr.analyses.typehoon.typehoon.Typehoon'>*, *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'*)

Bases: *Analysis*

A Clinic deals with AILments.

Parameters

- **func** (Function)
- **peephole_optimizations** (*None | Iterable[type[PeepholeOptimizationStmtBase] | type[PeepholeOptimizationExprBase]]*)
- **must_struct** (*set[str] | None*)
- **variable_kb** (KnowledgeBase *| None*)
- **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*)
- **complete_successors** (*bool*)
- **max_type_constraints** (*int*)
- **type_constraint_set_degradation_threshold** (*int*)
- **ail_graph** (*networkx.DiGraph | None*)
- **arg_vvars** (*dict[int, tuple[ailment.Expr.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*)

__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*, *variable_kb=None*, *reset_variable_names=False*, *rewrite_ites_to_diamonds=True*, *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*, *complete_successors=False*, *typehoon_cls=<class 'angr.analyses.typehoon.typehoon.Typehoon'>*, *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'*)

Parameters

- **func** (*Function*)
- **peephole_optimizations** (*None | Iterable[type[PeepholeOptimizationStmtBase] | type[PeepholeOptimizationExprBase]]*)

- **must_struct** (*set[str] | None*)
- **variable_kb** (*KnowledgeBase | None*)
- **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*)
- **complete_successors** (*bool*)
- **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*)

cc_graph: DiGraph | None

unoptimized_graph: DiGraph | None

arg_vvars: dict[int, tuple[*VirtualVariable*, *SimVariable*]] | None

externs: set[*SimMemoryVariable*]

data_refs: dict[int, list[*DataRefDesc*]]

entry_node_addr: Address

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.

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.

property eliminatable

Returns whether this annotation can be eliminated in a simplification.

Returns
True if eliminatable, False otherwise

property relocateable

class angr.analyses.decompiler.condition_processor.**ConditionProcessor**(*arch*, *ail_manager*, *condition_mapping=None*)

Bases: object

Convert between claripy AST and AIL expressions. Also calculates reaching conditions of all nodes on a graph.

Parameters
ail_manager (Manager)

__init__(*arch*, *ail_manager*, *condition_mapping=None*)

Parameters
ail_manager (Manager)

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*)

class angr.analyses.decompiler.decompilation_options.**DecompilationOption**(*name*, *description*, *value_type*, *cls*, *param*, *value_range=None*, *category='General'*, *default_value=None*, *clears_cache=True*, *candidate_values=None*, *convert=None*)

Bases: Generic

Describes a decompilation option.

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*)

__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.**get_structurer_option**()

Return type

DecompilationOption | None

class angr.analyses.decompiler.decompilation_cache.**DecompilationCache**(*addr*)

Bases: object

Caches key data structures that can be used later for refining decompilation results, such as retyping variables.

__init__(*addr*)

parameters: dict[str, Any]

addr

type_constraints: dict[*TypeVariable*, set[*TypeConstraint*]] | None

arg_vvars: dict | None

func_typevar: *TypeVariable* | None

var_to_typevar: dict | None

stackvar_max_sizes: dict | None

stack_offset_typevars: dict | None

codegen: *BaseStructuredCodeGenerator* | None

clinic: *Clinic* | None

ite_exprs: set[tuple[int, Any]] | None

binop_operators: dict[*OpDescriptor*, str] | None

errors: list[str]

function_summary: str | None

notes: dict[str, str]

max_tv_id: int

property local_types

class angr.analyses.decompiler.decompiler.**Decompiler**(*func*, *cfg=None*, *options=None*, *preset=None*, *optimization_passes=None*, *sp_tracker_track_memory=True*, *variable_kb=None*, *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=True*, *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.CStructuredCode*

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

Parameters

- **func** (Function | *str* | *int*)

- **cfg** (CFGFast | CFGModel | None)
- **preset** (str | DecompilationPreset | None)
- **peephole_optimizations** (_PEEPHOLE_OPTIMIZATIONS_TYPE)
- **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)

__init__(*func*, *cfg=None*, *options=None*, *preset=None*, *optimization_passes=None*, *sp_tracker_track_memory=True*, *variable_kb=None*, *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=True*, *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'>*)

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)

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.empty_node_remover.**EmptyNodeRemover**(*node*, *ail_manager*, *claripy_ast_conditions=True*)

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.

Parameters

- **ail_manager** (Manager)
- **claripy_ast_conditions** (*bool*)

__init__(*node*, *ail_manager*, *claripy_ast_conditions=True*)

Parameters

- **ail_manager** (Manager)
- **claripy_ast_conditions** (*bool*)

class angr.analyses.decompiler.expression_narrower.**ExprNarrowingInfo**(*narrowable*, *to_size=None*, *use_exprs=None*, *phi_vars=None*)

Bases: object

Stores the analysis result of _narrowing_needed().

Parameters

- **narrowable** (*bool*)
- **to_size** (*int | None*)
- **use_exprs** (*list[tuple[*atoms.VirtualVariable, AILCodeLocation*]] | None*)
- **phi_vars** (*set[*VirtualVariable*] | None*)

__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**(*target_expr*, *ignore_call_args=True*)

Bases: *AILBlockWalker*[None, None, None]

Walks a statement or an expression and extracts the effective size (in bits).

For example, for target expression rax, *(rax & 0xff) + 0x1* means the effective size of rax is 8 bits, from bit 0 to bit 7. We record this information in expr_to_effective_bits as {rax: (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.

Parameters

- **target_expr** (Expression)
- **ignore_call_args** (*bool*)

__init__(*target_expr*, *ignore_call_args=True*)

Parameters

- **target_expr** (Expression)
- **ignore_call_args** (*bool*)

expr_to_effective_bits: dict[*Expression*, tuple[int, int]]

expr_used_as_call_arg_effective_bits: tuple[int, int] | None

expr_used_as_insert_base: bool

class angr.analyses.decompiler.expression_narrower.**ExpressionNarrower**(*project*, *rd*, *manager*, *narrowables*, *addr2blocks*, *new_blocks*)

Bases: *AILBlockRewriter*

Narrows an expression regardless of whether the expression is a definition or a use.

Parameters

- **manager** (Manager)
- **addr2blocks** (*dict[tuple[int, int | None], Block]*)
- **new_blocks** (*dict[Block, Block]*)

__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)

class angr.analyses.decompiler.graph_region.**GraphRegion**(*head*, *graph*, *successors*, *graph_with_successors*, *cyclic*, *full_graph*, *cyclic_ancestor=False*)

Bases: object

GraphRegion represents a region of nodes.

Variables

- **head** – The head of the region.
- **graph** – The region graph.
- **successors** – A set of successors of nodes in the graph. These successors do not belong to the current region.

- **graph_with_successors** – The region graph that includes successor nodes.

__init__(*head*, *graph*, *successors*, *graph_with_successors*, *cyclic*, *full_graph*, *cyclic_ancestor=False*)

head

graph

successors

graph_with_successors

full_graph

cyclic

cyclic_ancestor

copy()

Return type
GraphRegion

recursive_copy(*nodes_map=None*)

property addr

static dbg_get_repr(*obj*, *ident=0*)

dbg_print(*ident=0*)

replace_region(*sub_region*, *updated_sub_region*, *replace_with*, *virtualized_edges*)

Parameters

- **sub_region** (GraphRegion)
- **updated_sub_region** (GraphRegion)
- **virtualized_edges** (*set[tuple[Any, Any]]*)

replace_region_with_region(*sub_region*, *replace_with*)

Parameters

- **sub_region** (GraphRegion)
- **replace_with** (GraphRegion)

class angr.analyses.decompiler.jump_target_collector.**JumpTargetCollector**(*node*)

Bases: object

Collect all jump targets.

__init__(*node*)

class angr.analyses.decompiler.jumptable_entry_condition_rewriter.**JumpTableEntryConditionRewriter**(*jumpta*

Bases: *SequenceWalker*

Remove artificial jump table entry conditions that ConditionProcessor introduced when dealing with jump tables.

__init__(*jumptable_entry_conds*)

class angr.analyses.decompiler.optimization_passes.**BasePointerSaveSimplifier**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, *max_iters=10*, *node_idx_start=0*, ***kwargs*)

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

Parameters
node_idx_start (*int*)

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**(**args*, ***kwargs*)

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**(**args*, *region_identifier=None*, *reaching_definitions=None*, ***kwargs*)

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** (SideEffectStatement | Call)
- **call1** (SideEffectStatement | Call)

class angr.analyses.decompiler.optimization_passes.**ConstantDereferencesSimplifier**(**args*, ***kwargs*)

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**(**args*, *node_idx_start=0*, *max_opt_iters=3*, *max_call_duplications=1*, ***kwargs*)

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.

Parameters

- **node_idx_start** (*int*)
- **max_opt_iters** (*int*)
- **max_call_duplications** (*int*)

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**(**args*, *node_cutoff=200*, ***kwargs*)

Bases: *OptimizationPass*

Removes condition-unreachable blocks from the graph.

Parameters
node_cutoff (*int*)

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**(**args*, ***kwargs*)

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**(**args*, *max_guarding_conditions=4*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, *binop_operators=None*, ***kwargs*)

Bases: *SequenceOptimizationPass*

Swap operands (and the operator accordingly) in a BinOp expression.

Parameters
binop_operators (*dict[OpDescriptor, str] | None*)

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**(**args*, *flip_size=9*, ***kwargs*)

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**(**args*, *ite_exprs=None*, ***kwargs*)

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**(**args*, *max_updates=10*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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*)

static is_inlined_strcpy(*stmt*)

class angr.analyses.decompiler.optimization_passes.**InlinedStrcpySimplifierLate**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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*)

static is_inlined_wcsncpy(*stmt*)

class angr.analyses.decompiler.optimization_passes.**InlinedWcscpySimplifierLate**(**args*, ***kwargs*)

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.**LoweredSwitchSimplifier**(**args*, *min_distinct_cases=2*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**values*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, *vvar_id_start*, *max_calls_in_regions=2*, *minimize_copies_for_regions=True*, *scratch=None*, ***kwargs*)

Bases: *OptimizationPass*, ReturnDuplicatorBase

This is a light-level goto-less version of the ReturnDuplicator optimization pass. It will only duplicate return-only blocks.

Parameters

- **vvar_id_start** (*int*)
- **max_calls_in_regions** (*int*)
- **minimize_copies_for_regions** (*bool*)
- **scratch** (*dict[str, Any] | None*)

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**(**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=500*, ***kwargs*)

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.
- **vvar_id_start** (*int*)
- **scratch** (*dict[str, Any] | None*)
- **max_func_blocks** (*int*)

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=500*, ***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**(**args*, ***kwargs*)

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**(**args*, *static_buffers=None*, *static_vvars=None*, ***kwargs*)

Bases: *OptimizationPass*

Rewrite user-specified vvars as static values or fix-sized buffers. Also rewrites reads from pointers derived off of such vvars.

Parameters

- **static_buffers** (*dict[str, FixedBuffer] | None*)
- **static_vvars** (*dict[int, FixedBufferPtr | Const] | None*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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.

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.'

__init__(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**TagSlicer**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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*)

class angr.analyses.decompiler.optimization_passes.const_derefs.**BlockWalker**(*project*, *manager*)

Bases: *AILBlockRewriter*

AIL Block walker in order to perform const deref substitution

Parameters

- **project** (Project)
- **manager** (Manager)

__init__(*project*, *manager*)

Parameters

- **project** (Project)
- **manager** (Manager)

class angr.analyses.decompiler.optimization_passes.const_derefs.**ConstantDereferencesSimplifier**(**args*, ***kwargs*)

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*)

class angr.analyses.decompiler.optimization_passes.optimization_pass.**BlockCache**(*rd*, *prop*)

Bases: tuple

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**(**values*)

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**(*func*, *manager*)

Bases: object

The base class for any optimization pass.

Parameters
manager (Manager)

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**(*func*, *manager*, **, graph, blocks_by_addr=None, blocks_by_addr_and_i variable_kb=None, region_identifier=None, reaching_definitions=None, vvar_id_start=0, entry_node_addr=None, scratch=None, force_loop_single_exit refine_loops_with_single complete_successors=Fals fold_expressions=True avoid_vvar_ids=None, arg_vvars=None, peephole_optimizations=N stack_pointer_tracker= notes=None, **kwargs*)

Bases: *BaseOptimizationPass*

The base class for any function-level graph optimization pass.

Parameters

- **vvar_id_start** (*int*)
- **entry_node_addr** (*tuple[int, int | None]*)
- **scratch** (*dict[str, Any] | None*)
- **force_loop_single_exit** (*bool*)
- **refine_loops_with_single_successor** (*bool*)
- **complete_successors** (*bool*)
- **fold_expressions** (*bool*)
- **avoid_vvar_ids** (*set[int] | None*)
- **arg_vvars** (*dict[int, tuple[ailment.Expr.VirtualVariable, SimVariable]] | None*)
- **notes** (*dict | None*)

__init__(*func*, *manager*, *, *graph*, *blocks_by_addr=None*, *blocks_by_addr_and_idx=None*, *variable_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*, *complete_successors=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*)
- **complete_successors** (*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]

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**(*func*, *manager*, *seq=None*, **kwargs*)

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**(*func*, *manager*, *require_st*, *prevent_nev*, *strictly_*, *recover_st*, *must_im*, *max_op*, *simplify_ail*, *require_g*, *readd_la*, *edges_to*, ***kwarg*

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

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*)

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()

Wrapper for _analyze() that verifies the graph is structurable before and after the optimization.

class angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.**StackCanarySimplifier**(**args*, ***kwarg*

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*)

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*)

class angr.analyses.decompiler.optimization_passes.div_simplifier.**DivSimplifierAILEngine**(*project*, *manager*)

Bases: *SimplifierAILEngine*

An AIL pass for the div simplifier

__init__(*project*, *manager*)

class angr.analyses.decompiler.optimization_passes.div_simplifier.**DivSimplifier**(**args*, ***kwargs*)

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*)

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**(*block*)

Bases: *RegionWalker*

Recursively locate block in a GraphRegion 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**(*block_addr*, *target_expr*, *callback*)

Bases: *AILBlockRewriter*

Replace expressions.

__init__(*block_addr*, *target_expr*, *callback*)

class angr.analyses.decompiler.optimization_passes.ite_expr_converter.**ITEExprConverter**(**args*, *ite_exprs=None*, ***kwargs*)

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*)

class angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.**Case**(*original_node*, *node_type*, *variable_hash*, *expr*, *value*, *target*, *target_idx*, *next_addr*)

Bases: object

Describes a case in a switch-case construct.

Parameters

- **node_type** (*str | None*)
- **value** (*int | str*)
- **target_idx** (*int | None*)

__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**(*expr*)

Bases: *AILBlockViewer*

Obtain a stable hash of an AIL expression with respect to all variables and all operations applied on variables.

Parameters
expr (Expression)

__init__(*expr*)

Parameters
expr (Expression)

class angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.**LoweredSwitchSimplifier**(**a*
mi
***k*

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.mod_simplifier.**ModSimplifierAILEngine**(**args*, ***kwargs*)

Bases: *SimplifierAILEngine*

class angr.analyses.decompiler.optimization_passes.mod_simplifier.**ModSimplifier**(**args*, ***kwargs*)

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*)

class angr.analyses.decompiler.optimization_passes.engine_base.**SimplifierAILState**(*arch*, *variables=None*)

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**(**args*, ***kwargs*)

Bases: *SimEngineLightAIL*[*SimplifierAILState*, *Expression*, *Statement*, *Block*]

Essentially implements a peephole optimization engine for AIL statements (because we do not perform memory or register loads).

class angr.analyses.decompiler.optimization_passes.expr_op_swapper.**OuterWalker**(*desc*)

Bases: *SequenceWalker*

A sequence walker that finds nodes and invokes expression replacer to replace expressions.

Parameters

desc (*dict[OpDescriptor, str]*)

__init__(*desc*)

desc: **dict[*OpDescriptor*, str]**

class angr.analyses.decompiler.optimization_passes.expr_op_swapper.**ExpressionReplacer**(*block_addr*, *target_expr_predicate*, *callback*)

Bases: *AILBlockRewriter*

Replace expressions.

__init__(*block_addr*, *target_expr_predicate*, *callback*)

class angr.analyses.decompiler.optimization_passes.expr_op_swapper.**OpDescriptor**(*block_addr*, *stmt_idx*, *ins_addr*, *op*)

Bases: object

Describes a specific operator.

Parameters

- **block_addr** (*int*)
- **stmt_idx** (*int*)
- **ins_addr** (*int*)
- **op** (*str*)

__init__(*block_addr*, *stmt_idx*, *ins_addr*, *op*)

Parameters

- **block_addr** (*int*)
- **stmt_idx** (*int*)
- **ins_addr** (*int*)
- **op** (*str*)

class angr.analyses.decompiler.optimization_passes.expr_op_swapper.**ExprOpSwapper**(**args*, *binop_operators=None*, ***kwargs*)

Bases: *SequenceOptimizationPass*

Swap operands (and the operator accordingly) in a BinOp expression.

Parameters
binop_operators (*dict[OpDescriptor, str] | None*)

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*)

class angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.**RegisterSaveAreaSimpli**

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*)

class angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.**RetAddrSaveSimplifier**(**args*, ***kwa*

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*)

class angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.**X86GccGetPcSimplifier**(**args*, ***kwar*

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*)

class angr.analyses.decompiler.peephole_optimizations.base.**PeepholeOptimizationStmtBase**(*project*, *kb*, *ail_manager*, *func_addr=None*, *pre-serve_vvar_ids=Nor type_hints=None*)

Bases: object

The base class for all peephole optimizations that are applied on AIL statements.

Parameters

- **project** (*Project | None*)
- **kb** (*KnowledgeBase | None*)
- **ail_manager** (*Manager*)
- **func_addr** (*int | None*)
- **preserve_vvar_ids** (*set[int]*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]]*)

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]]

optimize(*stmt*, *stmt_idx=None*, *block=None*, ***kwargs*)

Parameters

stmt_idx (*int* | *None*)

class angr.analyses.decompiler.peephole_optimizations.base.**PeepholeOptimizationMultiStmtBase**(*project*, *kb*, *ail_manager*, *func_addr=N*, *pre-serve_vvar_id*, *type_hints=N*

Bases: object

The base class for all peephole optimizations that are applied on multiple AIL statements at once.

Parameters

- **project** (Project | *None*)
- **kb** (KnowledgeBase | *None*)
- **ail_manager** (Manager)
- **func_addr** (*int* | *None*)
- **preserve_vvar_ids** (*set[int]*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]]*)

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]]

optimize(*stmts*, *stmt_idx=None*, *block=None*, ***kwargs*)

Parameters

- **stmts** (*list[*Statement*]*)
- **stmt_idx** (*int* | *None*)

class angr.analyses.decompiler.peephole_optimizations.base.**PeepholeOptimizationExprBase**(*project*, *kb*, *ail_manager*, *func_addr=None*, *preserve_vvar_ids=Non*, *type_hints=None*)

Bases: object

The base class for all peephole optimizations that are applied on AIL expressions.

Parameters

- **project** (Project | *None*)
- **kb** (KnowledgeBase | *None*)
- **ail_manager** (Manager)
- **func_addr** (*int* | *None*)
- **preserve_vvar_ids** (*set[int]*)
- **type_hints** (*list[tuple[*VirtualVariable | MemoryLocation, *str]]*)

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]]

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*)

type angr.analyses.decompiler.region_identifier.**TNode** = *Block* | *GraphRegion* | MultiNode | ConditionNode

type angr.analyses.decompiler.region_identifier.**TGraph** = networkx.DiGraph[*TNode*]

class angr.analyses.decompiler.region_identifier.**RegionIdentifier**(*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*, *complete_successors=False*, *entry_node_addr=None*)

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 GraphRegion object whose nodes are either Blocks or GraphRegions.

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*, *complete_successors=False*, *entry_node_addr=None*)

static slice_graph(*graph*, *node*, *frontier*, *include_frontier=False*)

Generate a slice of the graph from the head node to the given frontier.

Parameters

- **graph** (*networkx.DiGraph*) – The graph to work on.
- **node** – The starting node in the graph.
- **frontier** – A list of frontier nodes.
- **include_frontier** (*bool*) – Whether the frontier nodes are included in the slice or not.

Returns
A subgraph.

Return type
networkx.DiGraph[TNode]

test_reducibility()

Return type
bool

class angr.analyses.decompiler.region_simplifiers.**RegionSimplifier**(*func*, *region*, *ail_manager*, *arg_vvars=None*, *simplify_switches=True*, *simplify_ifelse=True*, *variable_manager=None*, *loopctr_naming=True*)

Bases: *Analysis*

Simplifies a given region.

Parameters

- **ail_manager** (*Manager*)
- **arg_vvars** (*set[int] | None*)

- **simplify_switches** (*bool*)
- **simplify_ifelse** (*bool*)
- **variable_manager** (VariableManagerInternal | *None*)
- **loopctr_naming** (*bool*)

__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.region_simplifiers.cascading_cond_transformer.**CascadingConditionTransfor**

Bases: *SequenceWalker*

Identifies and transforms *if { … } else { if { … } else { … } }* to *if { … } else if { … } else if { … }*.

Parameters
manager (Manager)

__init__(*node*, *manager*)

Parameters
manager (Manager)

class angr.analyses.decompiler.region_simplifiers.cascading_ifs.**CascadingIfsRemover**(*node*, *manager*)

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*)

class angr.analyses.decompiler.region_simplifiers.expr_folding.**LocationBase**

Bases: object

class angr.analyses.decompiler.region_simplifiers.expr_folding.**StatementLocation**(*block_addr*, *block_idx*, *stmt_idx*, *phi_stmt=False*)

Bases: *LocationBase*

Describes the location of a statement.

Parameters
phi_stmt (*bool*)

__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**(*block_addr*, *block_idx*, *stmt_idx*, *expr_idx*, *phi_stmt=False*)

Bases: *LocationBase*

Describes the location of an expression.

Parameters
phi_stmt (*bool*)

__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**(*cond_node_addr*, *case_idx=None*)

Bases: *LocationBase*

Describes the location of a condition.

Parameters
case_idx (*int | None*)

__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**(*node_addr*)

Bases: *LocationBase*

Describes the location of a conditional break.

__init__(*node_addr*)

node_addr

class angr.analyses.decompiler.region_simplifiers.expr_folding.**LoopNodeFinder**(*node*)

Bases: *SequenceWalker*

Returns all loops in a given region.

Parameters
node (*SequenceNode*)

__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**(*node*)

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**(*assignments*, *uses*, *node*)

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

Parameters

- **assignments** (*dict[int, Any]*)
- **uses** (*dict[int, Any]*)

__init__(*assignments*, *uses*, *node*)

Parameters

- **assignments** (*dict[int, Any]*)
- **uses** (*dict[int, Any]*)

interfered_assignments: **set[int]**

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionReplacer**(*assignments*, *uses*)

Bases: *AILBlockRewriter*

Parameters

- **assignments** (*dict[int, Any]*)
- **uses** (*dict[int, Any]*)

__init__(*assignments*, *uses*)

Parameters

- **assignments** (*dict[int, Any]*)
- **uses** (*dict[int, Any]*)

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionFolder**(*assignments*, *uses*, *node*)

Bases: *SequenceWalker*

Parameters

- **assignments** (*dict[int, Any]*)
- **uses** (*dict[int, Any]*)

__init__(*assignments*, *uses*, *node*)

Parameters

- **assignments** (*dict[int, Any]*)
- **uses** (*dict[int, Any]*)

class angr.analyses.decompiler.region_simplifiers.expr_folding.**StoreStatementFinder**(*node*, *intervals*)

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.

Parameters

intervals (*Iterable[tuple[StatementLocation, LocationBase]]*)

__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)

class angr.analyses.decompiler.region_simplifiers.goto.**GotoSimplifier**(*node*, *function=None*, *kb=None*)

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*)

class angr.analyses.decompiler.region_simplifiers.if_.**IfSimplifier**(*node*)

Bases: *SequenceWalker*

Remove unnecessary jump or conditional jump statements if they jump to the successor right afterwards.

__init__(*node*)

class angr.analyses.decompiler.region_simplifiers.ifelse.**IfElseFlattener**(*node*, *functions*)

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*)

class angr.analyses.decompiler.region_simplifiers.loop.**LoopSimplifier**(*node*, *functions*)

Bases: *SequenceWalker*

Simplifies loops.

__init__(*node*, *functions*)

class angr.analyses.decompiler.region_simplifiers.node_address_finder.**NodeAddressFinder**(*node*)

Bases: *SequenceWalker*

Walk the entire node and collect all addresses of nodes.

__init__(*node*)

class angr.analyses.decompiler.region_simplifiers.region_simplifier.**RegionSimplifier**(*func*, *region*, *ail_manager*, *arg_vvars=None*, *simplify_switches=True*, *simplify_ifelse=True*, *variable_manager=None*, *loopctr_naming=True*)

Bases: *Analysis*

Simplifies a given region.

Parameters

- **ail_manager** (Manager)
- **arg_vvars** (*set[int] | None*)
- **simplify_switches** (*bool*)
- **simplify_ifelse** (*bool*)
- **variable_manager** (VariableManagerInternal | *None*)
- **loopctr_naming** (*bool*)

__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.region_simplifiers.switch_cluster_simplifier.**CmpOp**(**values*)

Bases: Enum

All supported comparison operators.

LT = 0

GT = 1

EQ = 2

NE = 3

class angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**ConditionalRegion**(*variable*, *op*, *value*, *node*, *parent=None*)

Bases: object

Describes a conditional region.

Parameters

- **op** (CmpOp)
- **value** (*int*)
- **node** (*ConditionNode* | ailment.Block)

__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**(*variable*, *node*, *parent=None*)

Bases: object

Describes an already-recovered switch region.

Parameters

node (*SwitchCaseNode*)

__init__(*variable*, *node*, *parent=None*)

Parameters

node (*SwitchCaseNode*)

variable

node

parent

class angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**SwitchClusterFinder**(*node*)

Bases: *SequenceWalker*

Find comparisons and switches in order to identify switch clusters.

__init__(*node*)

var2condnodes: defaultdict[Any, list[*ConditionalRegion*]]

var2switches: defaultdict[Any, list[*SwitchCaseRegion*]]

class angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**SwitchClusterReplacer**(*region*, *to_repl*, *re-place_*

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*, *tar-gets=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_i*

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*, *var2condn*
var2switch

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*
func-
tions,
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**(*node_set*)

Bases: *SequenceWalker*

Find the first node out of a set of node appearing in a SequenceNode (and its tree).

Parameters
node_set (*set[BaseNode]*)

__init__(*node_set*)

Parameters
node_set (*set[BaseNode]*)

class angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier.**SwitchExpressionSimplifier**(*node, manager*)

Bases: *SequenceWalker*

Identifies switch expressions that adds or minuses a constant, removes the constant from the switch expression, and adjust all case expressions accordingly.

Parameters
manager (Manager)

__init__(*node*, *manager*)

Parameters
manager (Manager)

class angr.analyses.decompiler.region_walker.**RegionWalker**

Bases: object

A simple traverser class that walks GraphRegion instances.

__init__()

walk(*region*)

Parameters
region (GraphRegion)

walk_node(*region*, *node*)

class angr.analyses.decompiler.redundant_label_remover.**RedundantLabelRemover**(*node*, *jump_targets*)

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.

Parameters
jump_targets (*set[tuple[int, int | None]]*)

__init__(*node*, *jump_targets*)

Parameters
jump_targets (*set[tuple[int, int | None]]*)

class angr.analyses.decompiler.sequence_walker.**SequenceWalker**(*handlers=None*, *exception_on_unsupported=False*, *update_seqnode_in_place=True*, *force_forward_scan=False*)

Bases: object

Walks a SequenceNode and all its nodes, recursively.

Parameters
force_forward_scan (*bool*)

__init__(*handlers=None*, *exception_on_unsupported=False*, *update_seqnode_in_place=True*, *force_forward_scan=False*)

Parameters
force_forward_scan (*bool*)

walk(*sequence*)

class angr.analyses.decompiler.structured_codegen.**BaseStructuredCodeGenerator**(*flavor=None*, *notes=None*, *expr_comments=None*, *stmt_comments=None*, *const_formats=None*)

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_idx(*key*)

Return type
str

Parameters
key (*str*)

reset_idx_counters()

Return type
None

class angr.analyses.decompiler.structured_codegen.**CStructuredCodeGenerator**(*func*, *sequence*, *indent=0*, *cfg=None*, *variable_kb=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*)

Bases: *BaseStructuredCodeGenerator*, *Analysis*

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*)

__init__(*func*, *sequence*, *indent=0*, *cfg=None*, *variable_kb=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*)

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*)

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
tuple[str, *PositionMapping*, *PositionMapping*, *InstructionMapping*, dict[Any, set[Any]]]

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)

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*)

class angr.analyses.decompiler.structured_codegen.**DummyStructuredCodeGenerator**(*flavor*, *expr_comments=None*, *stmt_comments=None*, *configuration=None*, *const_formats=None*)

Bases: *BaseStructuredCodeGenerator*

A dummy structured code generator that only stores user-specified information.

Parameters
flavor (*str*)

__init__(*flavor*, *expr_comments=None*, *stmt_comments=None*, *configuration=None*, *const_formats=None*)

Parameters
flavor (*str*)

class angr.analyses.decompiler.structured_codegen.**ImportSourceCode**(*function*, *flavor='source'*, *source_root=None*, *encoding='utf-8'*)

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()

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**(*ins_addr*, *posmap_pos*)

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()

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**(*start*, *length*, *obj*)

Bases: object

__init__(*start*, *length*, *obj*)

start: int

length: int

obj

class angr.analyses.decompiler.structured_codegen.**RustStructuredCodeGenerator**(*func*, *sequence*, *indent=0*, *cfg=None*, *variable_kb=None*, *func_args=None*, *binop_depth_cutoff=16*, *show_casts=True*, *braces_on_own_lines=False*, *use_compound_assignments=Tru*
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=Tru*
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*)

Bases: *BaseStructuredCodeGenerator*, *Analysis*

Parameters

- **func_args** (*list[SimVariable] | None*)
- **binop_depth_cutoff** (*int*)
- **min_data_addr** (*int*)
- **display_notes** (*bool*)

__init__(*func*, *sequence*, *indent=0*, *cfg=None*, *variable_kb=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*)

Parameters

- **func_args** (*list[SimVariable] | None*)
- **binop_depth_cutoff** (*int*)
- **min_data_addr** (*int*)
- **display_notes** (*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*)

class angr.analyses.decompiler.structured_codegen.base.**CConstantType**(**values*)

Bases: Enum

Describes the value type of a CConstant class.

INT = 0

FLOAT = 1

STRING = 2

class angr.analyses.decompiler.structured_codegen.base.**PositionMappingElement**(*start*, *length*, *obj*)

Bases: object

Parameters

- **start** (*int*)
- **length** (*int*)

__init__(*start*, *length*, *obj*)

start: int

length: int

obj

class angr.analyses.decompiler.structured_codegen.base.**PositionMapping**

Bases: object

DUPLICATION_CHECK = True

__init__()

items()

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**(*ins_addr*, *posmap_pos*)

Bases: object

Parameters

- **ins_addr** (*int*)
- **posmap_pos** (*int*)

__init__(*ins_addr*, *posmap_pos*)

ins_addr: int

posmap_pos: int

class angr.analyses.decompiler.structured_codegen.base.**InstructionMapping**

Bases: object

__init__()

items()

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**(*flavor=None*, *notes=None*, *expr_comments=None*, *stmt_comments=None*, *const_formats=None*)

Bases: object

Parameters

- **expr_comments** (*dict[int, str]*)
- **stmt_comments** (*dict[int, str]*)
- **const_formats** (*dict[tuple[int, int, str], dict[str, bool]]*)

__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]]

idx_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_idx(*key*)

Return type
str

Parameters
key (*str*)

reset_idx_counters()

Return type
None

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_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_to_c_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.c.**CConstruct**(*codegen*, *tags=None*)

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*

idx

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**(*addr*, *name*, *functy*, *arg_list*, *statements*, *variables_in_use*, *variable_manager*, *demangled_name=None*, *show_demangled_name=True*, *omit_header=False*, ***kwargs*)

Bases: *CConstruct*

Represents a function in C.

Parameters

- **functy** (SimTypeFunction)
- **arg_list** (*list[*CVariable*]*)

__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**(*tags=None*, *codegen=None*)

Bases: *CConstruct*

Represents a statement in C.

__init__(*tags=None*, *codegen=None*)

codegen: *CStructuredCodeGenerator*

idx

tags

class angr.analyses.decompiler.structured_codegen.c.**CExpression**(*collapsed=False*, *tags=None*, *codegen=None*)

Bases: *CConstruct*

Base class for C expressions.

__init__(*collapsed=False*, *tags=None*, *codegen=None*)

collapsed

property type: *SimType* | None

set_type(*v*)

class angr.analyses.decompiler.structured_codegen.c.**CStatements**(*statements*, *addr=None*, ***kwargs*)

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**(*block*, ***kwargs*)

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**(*tags=None*, *codegen=None*)

Bases: *CStatement*

Represents a loop in C.

class angr.analyses.decompiler.structured_codegen.c.**CWhileLoop**(*condition*, *body*, ***kwargs*)

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**(*condition*, *body*, ***kwargs*)

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**(*initializer*, *condition*, *iterator*, *body*, ***kwargs*)

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**(*condition_and_nodes*, *else_node=None*, *simplify_else_scope=False*, *cstyle_ifs=True*, ***kwargs*)

Bases: *CStatement*

Represents an if-else construct in C.

Parameters
condition_and_nodes (*list[tuple[CExpression, CStatement | None]]*)

__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**(*condition*, *cstyle_ifs=True*, ***kwargs*)

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**(***kwargs*)

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**(***kwargs*)

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**(*switch*, *cases*, *default*, ***kwargs*)

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**(*head*, *cases*, ***kwargs*)

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**(*lhs*, *rhs*, ***kwargs*)

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**(*expr*, *returning=True*, ***kwargs*)

Bases: *CStatement*

Wraps a CExpression so it can be used as a standalone statement.

expr;

Parameters

- **expr** (CExpression)
- **returning** (*bool*)

__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**(*callee_target*, *callee_func*, *args*, *show_demangled_name=True*, *show_disambiguated_name=True*, *tags=None*, *codegen=None*, ***kwargs*)

Bases: *CExpression*

func(arg0, arg1)

Variables
callee_func (Function) – The function getting called.

Parameters
show_disambiguated_name (*bool*)

__init__(*callee_target*, *callee_func*, *args*, *show_demangled_name=True*, *show_disambiguated_name=True*, *tags=None*, *codegen=None*, ***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**(*retval*, ***kwargs*)

Bases: *CStatement*

__init__(*retval*, ***kwargs*)

retval

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CGoto**(*target*, *target_idx*, ***kwargs*)

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**(*stmt*, ***kwargs*)

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**(*dirty*, ***kwargs*)

Bases: *CExpression*

Parameters
dirty (CDirtyExpression)

__init__(*dirty*, ***kwargs*)

Parameters
dirty (CDirtyExpression)

dirty

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CLabel**(*name*, ***kwargs*)

Bases: *CStatement*

Represents a label in C code.

Parameters
name (*str*)

__init__(*name*, ***kwargs*)

Parameters
name (*str*)

name

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CStructField**(*struct_type*, *offset*, *field*, ***kwargs*)

Bases: *CExpression*

Parameters

- **struct_type** (*SimStruct*)
- **offset** (*int*)
- **field** (*str*)

__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**(*name*, *ty*, ***kwargs*)

Bases: *CExpression*

An uninterpreted name to display in the decompilation output. Pretty much always represents an error?

Parameters

- **name** (*str*)
- **ty** (*SimType*)

__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**(*variable*, *unified_variable=None*, *variable_type=None*, *vvar_id=None*, ***kwargs*)

Bases: *CExpression*

CVariable represents access to a variable with the specified type (*variable_type*).

variable must be a SimVariable.

Parameters
variable (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**(*variable*, *index*, *variable_type=None*, ***kwargs*)

Bases: *CExpression*

Represent a variable (an array) that is indexed.

Parameters

- **variable** (CExpression)
- **index** (CExpression)

__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**(*variable*, *field*, *var_is_ptr=False*, ***kwargs*)

Bases: *CExpression*

Represent a field of a variable.

Parameters

- **variable** (CExpression)
- **field** (CStructField)
- **var_is_ptr** (*bool*)

__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**(*op*, *operand*, ***kwargs*)

Bases: *CExpression*

Unary operations.

Parameters

operand (CExpression)

__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**(*op*, *lhs*, *rhs*, ***kwargs*)

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**(*src_type*, *dst_type*, *expr*, ***kwargs*)

Bases: *CExpression*

Parameters

- **src_type** (SimType | *None*)
- **dst_type** (SimType)
- **expr** (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**(*value*, *type_*, *reference_values=None*, ***kwargs*)

Bases: *CExpression*

Parameters
type_ (SimType)

__init__(*value*, *type_*, *reference_values=None*, ***kwargs*)

Parameters
type_ (SimType)

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**(*reg*, ***kwargs*)

Bases: *CExpression*

__init__(*reg*, ***kwargs*)

reg

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CITE**(*cond*, *iftrue*, *iffalse*, ***kwargs*)

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**(*stmts*, *expr*, ***kwargs*)

Bases: *CExpression*

(stmt0, stmt1, stmt2, expr)

Parameters

- **stmts** (CStatements)
- **expr** (CExpression)

__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**(*callee*, *operands*, ***kwargs*)

Bases: *CExpression*

ccall_name(arg0, arg1, ...)

Parameters

- **callee** (*str*)
- **operands** (*list[CExpression]*)

__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.**CDirtyExpression**(*dirty*, ***kwargs*)

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

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CClosingObject**(*opening_symbol*)

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**(*text*)

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**(*name*)

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**(*func*, *sequence*, *indent=0*, *cfg=None*, *variable_kb=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*)

Bases: *BaseStructuredCodeGenerator*, *Analysis*

Parameters

- **func_args** (*list[SimVariable] | None*)
- **binop_depth_cutoff** (*int*)
- **stmt_comments** (*dict[int, str]*)
- **expr_comments** (*dict[int, str]*)
- **const_formats** (*dict[tuple[int, int, str], dict[str, bool]]*)
- **min_data_addr** (*int*)
- **display_notes** (*bool*)
- **max_str_len** (*int | None*)
- **prettify_thiscall** (*bool*)

- **cstyle_void_param** (*bool*)

__init__(*func*, *sequence*, *indent=0*, *cfg=None*, *variable_kb=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*)

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*)

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
tuple[str, *PositionMapping*, *PositionMapping*, *InstructionMapping*, dict[Any, set[Any]]]

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)

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*)

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)

class angr.analyses.decompiler.structured_codegen.dwarf_import.**ImportedLine**(*addr*)

Bases: object

__init__(*addr*)

class angr.analyses.decompiler.structured_codegen.dwarf_import.**ImportSourceCode**(*function*, *flavor='source'*, *source_root=None*, *encoding='utf-8'*)

Bases: *BaseStructuredCodeGenerator*, *Analysis*

__init__(*function*, *flavor='source'*, *source_root=None*, *encoding='utf-8'*)

regenerate_text()

class angr.analyses.decompiler.structured_codegen.dummy.**DummyStructuredCodeGenerator**(*flavor*, *expr_comments=None*, *stmt_comments=None*, *configuration=None*, *const_formats=None*)

Bases: *BaseStructuredCodeGenerator*

A dummy structured code generator that only stores user-specified information.

Parameters

- **flavor** (*str*)
- **expr_comments** (*dict[int, str]*)
- **stmt_comments** (*dict[int, str]*)
- **const_formats** (*dict[tuple[int, int, str], dict[str, bool]]*)

__init__(*flavor*, *expr_comments=None*, *stmt_comments=None*, *configuration=None*, *const_formats=None*)

Parameters
flavor (*str*)

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*)

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.**peephole_optimize_stmts**(*block*, *stmt_opts*)

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]]]*)

class angr.analyses.ddg.**AST**(*op*, **operands*)

Bases: object

A mini implementation for AST

__init__(*op*, **operands*)

class angr.analyses.ddg.**ProgramVariable**(*variable*, *location*, *initial=False*, *arch=None*)

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**(*cfg_node*, *call_depth*)

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**(*ddg*, *variable*, *simplified=False*)

Bases: object

__init__(*ddg*, *variable*, *simplified=False*)

property depends_on

property dependents

class angr.analyses.ddg.**DDGViewInstruction**(*cfg*, *ddg*, *insn_addr*, *simplified=False*)

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**(*cfg*, *ddg*, *simplified=False*)

Bases: object

A view of the data dependence graph.

__init__(*cfg*, *ddg*, *simplified=False*)

class angr.analyses.ddg.**DDG**(*cfg*, *start=None*, *call_depth=None*, *block_addrs=None*)

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.flirt.**FlirtAnalysis**(*sig=None*, *max_mismatched_bytes=0*, *dry_run=False*, *match_named_functions=False*)

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.

Parameters

- **sig** (FlirtSignature | str | None)
- **max_mismatched_bytes** (int)
- **dry_run** (bool)
- **match_named_functions** (bool)

__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**(*arch*, *platform*, *sig_name*, *sig_path*, *unique_strings=None*, *compiler=None*, *compiler_version=None*, *os_name=None*, *os_version=None*)

Bases: object

This class describes a FLIRT signature without any internal data that is only available after parsing.

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)

__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**(*version*, *arch*, *file_types*, *os_types*, *app_types*, *features*, *crc*, *ctype*, *ctypes_crc*, *nfuncs*, *pattern_size*, *libname*, *root*)

Bases: object

Describes a FLIRT signature file after parsing.

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*)

__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.

class angr.engines.light.data.**ArithmeticExpression**(*op*, *operands*)

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**(*bits*, *reg*, *offset*)

Bases: object

__init__(*bits*, *reg*, *offset*)

reg

offset

property bits

property symbolic

class angr.engines.light.data.**SpOffset**(*bits*, *offset*, *is_base=False*)

Bases: *RegisterOffset*

__init__(*bits*, *offset*, *is_base=False*)

is_base

class angr.engines.light.**ArithmeticExpression**(*op*, *operands*)

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**(*bits*, *reg*, *offset*)

Bases: object

__init__(*bits*, *reg*, *offset*)

reg

offset

property bits

property symbolic

class angr.engines.light.**SimEngineLight**(*project*, *logger=None*)

Bases: *SimEngine*, Generic

A full-featured engine base class, suitable for static analysis

Parameters
project (Project)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(*bits*, *offset*, *is_base=False*)

Bases: RegisterOffset

__init__(*bits*, *offset*, *is_base=False*)

is_base

class angr.engines.light.engine.**BlockProtocol**(**args*, ***kwargs*)

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**(*ty*)

Bases: IRExpr

A dummy IRExpr used for intra-engine communication and code-reuse.

Parameters
ty (*str*)

__init__(*ty*)

Parameters
ty (*str*)

result_type(*tyenv*)

class angr.engines.light.engine.**SimEngineLight**(*project*, *logger=None*)

Bases: *SimEngine*, Generic

A full-featured engine base class, suitable for static analysis

Parameters
project (Project)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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**(**args*, ***kwargs*)

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.analyses.propagator.**PropagatorAnalysis**(*func=None*, *block=None*, *func_graph=None*, *base_state=None*, *max_iterations=30*, *load_callback=None*, *stack_pointer_tracker=None*, *only_consts=False*, *completed_funcs=None*, *do_binops=True*, *store_tops=True*, *vex_cross_insn_opt=False*, *func_addr=None*, *gp=None*, *cache_results=False*, *key_prefix=None*, *profiling=False*)

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

Parameters

- **func_addr** (*int | None*)
- **gp** (*int | None*)
- **cache_results** (*bool*)
- **key_prefix** (*str | None*)
- **profiling** (*bool*)

__init__(*func=None*, *block=None*, *func_graph=None*, *base_state=None*, *max_iterations=30*, *load_callback=None*, *stack_pointer_tracker=None*, *only_consts=False*, *completed_funcs=None*, *do_binops=True*, *store_tops=True*, *vex_cross_insn_opt=False*, *func_addr=None*, *gp=None*, *cache_results=False*, *key_prefix=None*, *profiling=False*)

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** (*GraphVisitor or None*) – A graph visitor to provide successors.
- **func_addr** (*int | None*)
- **gp** (*int | None*)
- **cache_results** (*bool*)
- **key_prefix** (*str | None*)
- **profiling** (*bool*)

Returns

None

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.propagator.values.**Top**(*size*)

Bases: object

__init__(*size*)

size

property bits

class angr.analyses.propagator.values.**Bottom**

Bases: object

class angr.analyses.propagator.vex_vars.**VEXVariable**

Bases: object

class angr.analyses.propagator.vex_vars.**VEXMemVar**(*addr*, *size*)

Bases: object

Parameters

- **addr** (*int*)
- **size** (*int*)

__init__(*addr*, *size*)

Parameters

- **addr** (*int*)
- **size** (*int*)

addr

size

class angr.analyses.propagator.vex_vars.**VEXReg**(*offset*, *size*)

Bases: *VEXVariable*

__init__(*offset*, *size*)

offset

size

class angr.analyses.propagator.vex_vars.**VEXTmp**(*tmp*)

Bases: *VEXVariable*

__init__(*tmp*)

tmp

class angr.analyses.propagator.engine_base.**SimEnginePropagatorBaseMixin**(*project*, *stack_pointer_tracker=None*, *propagate_tmps=True*, *reaching_definitions=None*, *bp_as_gpr=False*)

Bases: *SimEngineLight*, Generic

The base class for the propagator VEX engine.

Parameters

- **project** (Project)
- **reaching_definitions** (ReachingDefinitionsModel | *None*)
- **bp_as_gpr** (*bool*)

__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*)

class angr.analyses.propagator.engine_vex.**SimEnginePropagatorVEX**(*project*, *stack_pointer_tracker=None*, *propagate_tmps=True*, *reaching_definitions=None*, *bp_as_gpr=False*)

Bases: *ClaripyDataVEXEngineMixin*[PropagatorVEXState, BV, PropagatorVEXState, None], *SimEnginePropagatorBaseMixin*[PropagatorVEXState, BV, *Block*], *SimEngineNostmtVEX*[PropagatorVEXState, BV, PropagatorVEXState]

Parameters

- **project** (Project)
- **reaching_definitions** (ReachingDefinitionsModel | *None*)
- **bp_as_gpr** (*bool*)

class angr.analyses.propagator.propagator.**PropagatorAnalysis**(*func=None*, *block=None*, *func_graph=None*, *base_state=None*, *max_iterations=30*, *load_callback=None*, *stack_pointer_tracker=None*, *only_consts=False*, *completed_funcs=None*, *do_binops=True*, *store_tops=True*, *vex_cross_insn_opt=False*, *func_addr=None*, *gp=None*, *cache_results=False*, *key_prefix=None*, *profiling=False*)

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

Parameters

- **func_addr** (*int | None*)
- **gp** (*int | None*)
- **cache_results** (*bool*)
- **key_prefix** (*str | None*)
- **profiling** (*bool*)

__init__(*func=None*, *block=None*, *func_graph=None*, *base_state=None*, *max_iterations=30*, *load_callback=None*, *stack_pointer_tracker=None*, *only_consts=False*, *completed_funcs=None*, *do_binops=True*, *store_tops=True*, *vex_cross_insn_opt=False*, *func_addr=None*, *gp=None*, *cache_results=False*, *key_prefix=None*, *profiling=False*)

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** (*GraphVisitor or None*) – A graph visitor to provide successors.
- **func_addr** (*int | None*)
- **gp** (*int | None*)
- **cache_results** (*bool*)
- **key_prefix** (*str | None*)
- **profiling** (*bool*)

Returns

None

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.propagator.top_checker_mixin.**ClaripyDataEngineMixin**(*project*, *logger=None*)

Bases: *SimEngineLight*[StateType, DataType_co | BV, BlockType, ResultType], Generic

Parameters

project (Project)

class angr.analyses.propagator.top_checker_mixin.**ClaripyDataVEXEngineMixin**(**args*, ***kwargs*)

Bases: *ClaripyDataEngineMixin*[StateType, DataType_co, *Block*, ResultType], *SimEngineLightVEX*[StateType, DataType_co | BV, ResultType, StmtDataType], Generic

class angr.analyses.reaching_definitions.**Atom**(*size*)

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(*values)`

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**(*value*, *size*)

Bases: *Atom*

Represents a constant.

Parameters

- **value** (*int*)
- **size** (*int*)

__init__(*value*, *size*)

Parameters

- **size** (int) – The size of the atom in bytes
- **value** (*int*)

value: int

class angr.analyses.reaching_definitions.**Definition**(*atom*, *codeloc*, *dummy=False*, *tags=None*)

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.

Parameters

- **atom** (*A*)
- **codeloc** (*CodeLoc*)
- **dummy** (*bool*)
- **tags** (*set[Tag] | None*)

__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**(*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*)

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.

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*)

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**(*callsite*, *target*, *args_defns*, *other_input_defns*, *ret_defns*, *other_output_defns*)

Bases: object

Produced by the function handler, provides associated callsite info and function input/output definitions.

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])

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**(*interfunction_level=0*, *extra_impls=None*)

Bases: object

A mechanism for summarizing a function call's effect on a program for ReachingDefinitionsAnalysis.

Parameters

- **interfunction_level** (*int*)
- **extra_impls** (*Iterable[type[FunctionHandler]] | None*)

__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** (*None | int | MultiValues*)
- **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**(*target*)

Bases: *Atom*

Implements a guard use.

__init__(*target*)

Parameters
size – The size of the atom in bytes

target

class angr.analyses.reaching_definitions.**LiveDefinitions**(*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*)

Bases: object

A LiveDefinitions instance contains definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

Parameters

- **arch** (*archinfo.Arch*)
- **track_tmps** (*bool*)
- **registers** (MultiValuedMemory)
- **stack** (MultiValuedMemory)
- **memory** (MultiValuedMemory)
- **heap** (MultiValuedMemory)
- **tmps** (*dict[int, set[Definition]]*)
- **others** (*dict[Atom, MultiValues]*)
- **tmp_uses** (*dict[int, set[CodeLocation]]*)
- **merge_into_tops** (*bool*)

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.analyses.reaching_definitions.**MemoryLocation**(*addr*, *size*, *endness=None*)

Bases: *Atom*

Represents a memory slice.

It is characterized by its address and its size.

Parameters

- **addr** (SpOffset | *int* | *BV*)
- **size** (*int*)
- **endness** (*Endness* | *None*)

__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**(**values*)

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

class angr.analyses.reaching_definitions.**ReachingDefinitionsAnalysis**(*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*)

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.

Parameters

- **subject** (Subject | ailment.Block | Block | Function | *str*)
- **observation_points** (*Iterable[ObservationPoint] | None*)
- **init_state** (ReachingDefinitionsState | *None*)
- **state_initializer** (*RDAStateInitializer | None*)
- **function_handler** (FunctionHandler | *None*)
- **dep_graph** (DepGraph | *bool* | *None*)
- **interfunction_level** (*int*)
- **track_liveness** (*bool*)
- **func_addr** (*int* | *None*)
- **element_limit** (*int*)
- **merge_into_tops** (*bool*)

__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*)

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**(*func_addr=None*, *track_liveness=True*)

Bases: object

Models the definitions, uses, and memory of a ReachingDefinitionState object

Parameters

- **func_addr** (*int* | *None*)
- **track_liveness** (*bool*)

__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**(*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*)

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.
- **codeloc** (CodeLocation)
- **arch** (*archinfo.Arch*)
- **track_consts** (*bool*)
- **sp_adjusted** (*bool*)
- **all_definitions** (*set[Definition[A, CodeLoc]] | None*)
- **initializer** (*RDAStateInitializer | None*)
- **element_limit** (*int*)
- **merge_into_tops** (*bool*)

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] | type[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**(*reg_offset*, *size*, *arch=None*)

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.

Parameters

- **reg_offset** (*RegisterOffset | int*)
- **size** (*int*)
- **arch** (*Arch | None*)

__init__(*reg_offset*, *size*, *arch=None*)

Parameters

- **size** (int) – The size of the atom in bytes
- **reg_offset** (*RegisterOffset | int*)
- **arch** (*Arch | None*)

reg_offset

arch

property name: str

class angr.analyses.reaching_definitions.**Tmp**(*tmp_idx*, *size*)

Bases: *Atom*

Represents a variable used by the IR to store intermediate values.

Parameters

- **tmp_idx** (*int*)
- **size** (*int*)

__init__(*tmp_idx*, *size*)

Parameters

- **size** (int) – The size of the atom in bytes
- **tmp_idx** (*int*)

tmp_idx

angr.analyses.reaching_definitions.**get_all_definitions**(*region*)

Return type
set[*Definition*]

Parameters
region (MultiValuedMemory)

class angr.analyses.reaching_definitions.call_trace.**CallSite**(*caller_func_addr*, *block_addr*, *callee_func_addr*)

Bases: object

Describes a call site on a CFG.

Parameters

- **caller_func_addr** (*int*)
- **block_addr** (*int | None*)
- **callee_func_addr** (*int*)

__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**(*target*)

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).

Parameters
target (*int*)

__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

class angr.analyses.reaching_definitions.engine_vex.**SimEngineRDVEX**(*project*, *function_handler*, *functions*)

Bases: *SimEngineNostmtVEX*[*ReachingDefinitionsState*, *MultiValues*[BV | FP], *ReachingDefinitionsState*]

Implements the VEX execution engine for reaching definition analysis.

Parameters

- **function_handler** (FunctionHandler)
- **functions** (FunctionManager)

__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*)

class angr.analyses.reaching_definitions.reaching_definitions.**ReachingDefinitionsAnalysis**(*subject*, *func_graph=Non*
max_iterations=3
track_tmps=False
track_consts=Tru
ob-
ser-
va-
tion_points=Non
init_state=None,
init_context=Non
state_initializer=
cc=None,
func-
tion_handler=No
ob-
serve_all=False,
vis-
ited_blocks=Non
dep_graph=True,
ob-
serve_callback=N
canon-
i-
cal_size=8,
stack_pointer_tra
use_callee_saved
in-
ter-
func-
tion_level=0,
track_liveness=T
func_addr=None
el-
e-
ment_limit=5,
merge_into_tops=

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.

Parameters

- **subject** (Subject | ailment.Block | Block | Function | *str*)
- **observation_points** (*Iterable[ObservationPoint] | None*)

- **init_state** (ReachingDefinitionsState | *None*)
- **state_initializer** (*RDAStateInitializer* | *None*)
- **function_handler** (FunctionHandler | *None*)
- **dep_graph** (DepGraph | *bool* | *None*)
- **interfunction_level** (*int*)
- **track_liveness** (*bool*)
- **func_addr** (*int* | *None*)
- **element_limit** (*int*)
- **merge_into_tops** (*bool*)

__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*)

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.dep_graph.**DepGraph**(*graph=None*)

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*)

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]

Filter the definitions present in the graph based on various criteria. Parameters can be any valid keyword args to *DefinitionMatchPredicate*

find_all_predecessors(*starts*, ***kwargs*)

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]]

Filter the ancestors of the given start node or nodes that match various criteria. Parameters can be any valid keyword args to *DefinitionMatchPredicate*

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])

class angr.analyses.reaching_definitions.heap_allocator.**HeapAllocator**(*canonical_size*)

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.

Parameters
canonical_size (*int*)

__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.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**(*dest*, *sources*, *value=None*, *sources_defns=None*, *apply_at_callsite=False*, *tags=None*)

Bases: object

A single effect that a function summary may apply to the state. This is largely an implementation detail; use *FunctionCallData.depends* instead.

Parameters

- **dest** (Atom | *None*)
- **sources** (*set[Atom]*)
- **value** (MultiValues | *None*)
- **sources_defns** (*set[Definition]* | *None*)
- **apply_at_callsite** (*bool*)
- **tags** (*set[Tag]* | *None*)

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**(*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*)

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.

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)

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**(*inner*)

Bases: *FunctionCallData*

A subclass of FunctionCallData which asserts that many of its members are non-None at construction time. Typechecks be gone!

Parameters
inner (FunctionCallData)

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**(*callsite*, *target*, *args_defns*, *other_input_defns*, *ret_defns*, *other_output_defns*)

Bases: object

Produced by the function handler, provides associated callsite info and function input/output definitions.

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]*)

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**(*interfunction_level=0*, *extra_impls=None*)

Bases: object

A mechanism for summarizing a function call's effect on a program for ReachingDefinitionsAnalysis.

Parameters

- **interfunction_level** (*int*)
- **extra_impls** (*Iterable[type[FunctionHandler]] | None*)

__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** (*None | int | MultiValues*)
- **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.rd_state.**ReachingDefinitionsState**(*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*)

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.
- **codeloc** (CodeLocation)
- **arch** (*Arch*)
- **track_consts** (*bool*)
- **sp_adjusted** (*bool*)
- **all_definitions** (*set[Definition[Any, Any]]*)
- **initializer** (*RDAStateInitializer | None*)
- **element_limit** (*int*)
- **merge_into_tops** (*bool*)

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] | type[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.subject.**SubjectType**(**values*)

Bases: Enum

Function = 1

Block = 2

CallTrace = 3

class angr.analyses.reaching_definitions.subject.**Subject**(*content*, *func_graph=None*, *cc=None*)

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*

class angr.analyses.reaching_definitions.engine_ail.**SimEngineRDAIL**(*project*, *function_handler*, *stack_pointer_tracker=None*, *use_callee_saved_regs_at_return=True*, *bp_as_gpr=False*)

Bases: *SimEngineNostmtAIL*[*ReachingDefinitionsState*, *MultiValues*[BV | FP], None, *ReachingDefinitionsState*]

Parameters

- **function_handler** (FunctionHandler)
- **bp_as_gpr** (*bool*)

__init__(*project*, *function_handler*, *stack_pointer_tracker=None*, *use_callee_saved_regs_at_return=True*, *bp_as_gpr=False*)

Parameters

- **function_handler** (FunctionHandler)
- **bp_as_gpr** (*bool*)

process(*state*, **, dep_graph=None*, *visited_blocks=None*, *block=None*, *fail_fast=False*, *whitelist=None*, ***kwargs*)

class angr.analyses.cfg_slice_to_sink.**CFGSliceToSink**(*target*, *transitions=None*)

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.

class angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.**CFGSliceToSink**(*target*, *transitions=None*)

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.**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.

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.

class angr.analyses.stack_pointer_tracker.**BottomType**

Bases: object

The bottom value for register values.

class angr.analyses.stack_pointer_tracker.**Constant**(*val*)

Bases: object

Represents a constant value.

__init__(*val*)

val

class angr.analyses.stack_pointer_tracker.**Register**(*offset*, *bitlen*)

Bases: object

Represent a register.

__init__(*offset*, *bitlen*)

offset

bitlen

class angr.analyses.stack_pointer_tracker.**OffsetVal**(*reg*, *offset*)

Bases: object

Represent a value with an offset added.

__init__(*reg*, *offset*)

property reg

property offset

class angr.analyses.stack_pointer_tracker.**Eq**(*val0*, *val1*)

Bases: object

Represent an equivalence condition.

__init__(*val0*, *val1*)

val0

val1

class angr.analyses.stack_pointer_tracker.**FrozenStackPointerTrackerState**(*regs*, *memory*, *is_tracking_memory*, *resilient*)

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**(*regs*, *memory*, *is_tracking_memory*, *resilient*)

Bases: object

Abstract state for StackPointerTracker analysis.

Parameters
resilient (*bool*)

__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**(*func*, *reg_offsets*, *block=None*, *track_memory=True*, *cross_insn_opt=True*, *initial_reg_values=None*, *resilient=True*)

Bases: *Analysis*, *ForwardAnalysis*

Track the offset of stack pointer at the end of each basic block of a function.

Parameters

- **func** (Function | *None*)
- **reg_offsets** (*set[int]*)
- **block** (Block | *None*)
- **resilient** (*bool*)

__init__(*func*, *reg_offsets*, *block=None*, *track_memory=True*, *cross_insn_opt=True*, *initial_reg_values=None*, *resilient=True*)

Parameters

- **func** (Function | *None*)
- **reg_offsets** (*set[int]*)
- **block** (Block | *None*)
- **resilient** (*bool*)

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*)

class angr.analyses.variable_recovery.annotations.**StackLocationAnnotation**(*offset*)

Bases: Annotation

__init__(*offset*)

property eliminatable

Returns whether this annotation can be eliminated in a simplification.

Returns
True if eliminatable, False otherwise

property relocatable

Returns whether this annotation can be relocated in a simplification.

Returns
True if it can be relocated, false otherwise.

class angr.analyses.variable_recovery.annotations.**VariableSourceAnnotation**(*block_addr*, *stmt_idx*, *ins_addr*)

Bases: Annotation

__init__(*block_addr*, *stmt_idx*, *ins_addr*)

property eliminatable

Returns whether this annotation can be eliminated in a simplification.

Returns
True if eliminatable, False otherwise

property relocatable

Returns whether this annotation can be relocated in a simplification.

Returns
True if it can be relocated, false otherwise.

static from_state(*state*)

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**(*addr_and_variables*)

Bases: Annotation

Parameters
addr_and_variables (*list[tuple[int, SimVariable]]*)

__init__(*addr_and_variables*)

Parameters
addr_and_variables (*list[tuple[int, SimVariable]]*)

addr_and_variables

property relocatable

Returns whether this annotation can be relocated in a simplification.

Returns
True if it can be relocated, false otherwise.

property eliminatable

Returns whether this annotation can be eliminated in a simplification.

Returns
True if eliminatable, False otherwise

class angr.analyses.variable_recovery.variable_recovery_base.**VariableRecoveryBase**(*func*, *max_iterations*, *store_live_variables*, *vvar_to_vvar=None*, *func_graph=None*, *entry_node_addr=None*)

Bases: *Analysis*

The base class for VariableRecovery and VariableRecoveryFast.

Parameters

- **func** (Function)
- **store_live_variables** (*bool*)

- **vvar_to_vvar** (*dict[int, int] | None*)
- **func_graph** (*networkx.DiGraph[CodeNode] | networkx.DiGraph[ailment.Block] | None*)
- **entry_node_addr** (*int | ailment.Address | None*)

__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**(*block_addr*, *analysis*, *arch*, *func*, *project*, **, tv_manager*, *stack_region=None*, *register_region=None*, *global_region=None*, *typevars=None*, *type_constraints=No*, *func_typevar=None*, *delayed_type_constrain*, *stack_offset_typevars*

Bases: object

The base abstract state for variable recovery analysis.

Parameters

- **block_addr** (*int*)
- **analysis** (VariableRecoveryBase)
- **arch** (*archinfo.Arch*)
- **func** (Function)
- **project** (Project)
- **tv_manager** (TypeVariableManager)

__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

class angr.analyses.variable_recovery.variable_recovery_fast.**VariableRecoveryFastState**(*block_addr*, *analysis*, *arch*, *func*, *project*, *stack_region=None*, *register_region=None*, *global_region=None*, *typevars=None*, *type_constraints=No*, *func_typevar=None*, *delayed_type_constrain*, *stack_offset_typevars*, *ret_val_size=None*, ***, *tv_manager*)

Bases: *VariableRecoveryStateBase*

The abstract state of variable recovery analysis.

Variables

- **stack_region** (KeyedRegion) – The stack store.
- **register_region** (KeyedRegion) – The register store.

Parameters

- **arch** (*Arch*)
- **stack_region** (MultiValuedMemory)
- **register_region** (MultiValuedMemory)
- **global_region** (MultiValuedMemory)
- **stack_offset_typevars** (*dict[int, TypeVariable]*)
- **tv_manager** (TypeVariableManager)

__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.

downsize()

Remove unnecessary members.

Return type
None

Returns
None

class angr.analyses.variable_recovery.variable_recovery_fast.**VariableRecoveryFast**(*func*, *func_graph=None*, *entry_node_addr=None*, *max_iterations=2*, *low_priority=False*, *track_sp=True*, *func_args=None*, *func_ret_var=None*, *store_live_variables=False*, *unify_variables=True*, *func_arg_vvars=None*, *vvar_to_vvar=None*, *type_hints=None*)

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`.

Parameters

- **func** (*Function | str | int*)
- **func_graph** (*networkx.DiGraph | None*)
- **entry_node_addr** (*int | tuple[int, int | None] | None*)
- **max_iterations** (*int*)
- **func_args** (*list[SimVariable] | None*)
- **func_ret_var** (*SimVariable | None*)
- **func_arg_vvars** (*dict[int, tuple[VirtualVariable, SimVariable]] | None*)
- **vvar_to_vvar** (*dict[int, int] | None*)
- **type_hints** (*list[tuple[atoms.VirtualVariable | atoms.MemoryLocation, str]] | None*)

__init__(*func*, *func_graph=None*, *entry_node_addr=None*, *max_iterations=2*, *low_priority=False*, *track_sp=True*, *func_args=None*, *func_ret_var=None*, *store_live_variables=False*, *unify_variables=True*, *func_arg_vvars=None*, *vvar_to_vvar=None*, *type_hints=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** (*GraphVisitor or None*) – A graph visitor to provide successors.
- **func** (*Function | str | int*)
- **func_graph** (*DiGraph | None*)

- **entry_node_addr** (*int | tuple[int, int | None] | None*)
- **max_iterations** (*int*)
- **func_args** (*list[SimVariable] | None*)
- **func_ret_var** (*SimVariable | None*)
- **func_arg_vvars** (*dict[int, tuple[VirtualVariable, SimVariable]] | None*)
- **vvar_to_vvar** (*dict[int, int] | None*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]] | None*)

Returns
None

class angr.analyses.variable_recovery.variable_recovery.**VariableRecoveryState**(*project*, *block_addr*, *analysis*, *arch*, *func*, *concrete_states*, ***, *tv_manager*, *stack_region=None*, *register_region=None*)

Bases: *VariableRecoveryStateBase*

The abstract state of variable recovery analysis.

Variables
variable_manager (*angr.knowledge.variable_manager.VariableManager*) – The variable manager.

Parameters

- **project** (angr.Project)
- **block_addr** (*int*)
- **arch** (*Arch*)
- **func** (Function)
- **tv_manager** (TypeVariableManager)
- **stack_region** (MultiValuedMemory)
- **register_region** (MultiValuedMemory)

__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**(*func*, *max_iterations=20*, *store_live_variables=False*)

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.engine_ail.**SimEngineVRAIL**(**args*, *type_lifter*, *call_info=None*, *vvar_to_vvar*, *vvar_type_hints=None*, *func_ret_var=None*, *tv_manager=None*, ***kwargs*)

Bases: *SimEngineNostmtAIL*[VariableRecoveryFastState, *RichR*[BV | FP], None, None], *SimEngineVRBase*[VariableRecoveryFastState, *Block*]

The engine for variable recovery on AIL.

Parameters

- **type_lifter** (TypeTranslator)
- **vvar_to_vvar** (*dict[int, int] | None*)
- **vvar_type_hints** (*dict[int, typeconsts.TypeConstant] | None*)
- **func_ret_var** (SimVariable | *None*)
- **tv_manager** (typevars.TypeVariableManager | *None*)

__init__(**args*, *type_lifter*, *call_info=None*, *vvar_to_vvar*, *vvar_type_hints=None*, *func_ret_var=None*, *tv_manager=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*)

class angr.analyses.variable_recovery.engine_vex.**SimEngineVRVEX**(**args*, *call_info=None*, ***kwargs*)

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*)

class angr.analyses.variable_recovery.engine_base.**RichR**(*data*, *variable=None*, *typevar=None*, *type_constraints=None*)

Bases: Generic

A rich representation of calculation results. The variable recovery data domain.

Parameters

- **data** (*RichRT_co*)
- **typevar** (TypeConstant | TypeVariable | *None*)
- **type_constraints** (*set[typevars.TypeConstraint] | None*)

__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**(*project*, *kb*, *vvar_type_hints=None*, *tv_manager=None*)

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.

Parameters

- **vvar_type_hints** (*dict[*int, TypeConstant*]*)
- **tv_manager** (typevars.TypeVariableManager | *None*)

__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

class angr.analyses.variable_recovery.irsb_scanner.**VEXIRSBScanner**(**args*, ***kwargs*)

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*)

class angr.analyses.variable_recovery.**VariableRecovery**(*func*, *max_iterations=20*, *store_live_variables=False*)

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**(*func*, *func_graph=None*, *entry_node_addr=None*, *max_iterations=2*, *low_priority=False*, *track_sp=True*, *func_args=None*, *func_ret_var=None*, *store_live_variables=False*, *unify_variables=True*, *func_arg_vvars=None*, *vvar_to_vvar=None*, *type_hints=None*)

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`.

Parameters

- **func** (*Function | str | int*)
- **func_graph** (*networkx.DiGraph | None*)
- **entry_node_addr** (*int | tuple[int, int | None] | None*)
- **max_iterations** (*int*)
- **func_args** (*list[SimVariable] | None*)

- **func_ret_var** (SimVariable | None)
- **func_arg_vvars** (dict[int, tuple[VirtualVariable, SimVariable]] | None)
- **vvar_to_vvar** (dict[int, int] | None)
- **type_hints** (list[tuple[atoms.VirtualVariable | atoms.MemoryLocation, str]] | None)

__init__(*func*, *func_graph=None*, *entry_node_addr=None*, *max_iterations=2*, *low_priority=False*, *track_sp=True*, *func_args=None*, *func_ret_var=None*, *store_live_variables=False*, *unify_variables=True*, *func_arg_vvars=None*, *vvar_to_vvar=None*, *type_hints=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** (GraphVisitor or None) – A graph visitor to provide successors.
- **func** (Function | str | int)
- **func_graph** (DiGraph | None)
- **entry_node_addr** (int | tuple[int, int | None] | None)
- **max_iterations** (int)
- **func_args** (list[SimVariable] | None)
- **func_ret_var** (SimVariable | None)
- **func_arg_vvars** (dict[int, tuple[VirtualVariable, SimVariable]] | None)
- **vvar_to_vvar** (dict[int, int] | None)
- **type_hints** (list[tuple[VirtualVariable | MemoryLocation, str]] | None)

Returns

None

class angr.analyses.typehoon.simple_solver.**SketchNodeBase**

Bases: object

The base class for nodes in a sketch.

class angr.analyses.typehoon.simple_solver.**SketchNode**(*typevar*)

Bases: *SketchNodeBase*

Represents a node in a sketch graph.

Parameters

typevar (TypeVariable | DerivedTypeVariable)

__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**(*target*)

Bases: *SketchNodeBase*

Represents a cycle in a sketch graph.

This is equivalent to sketches.LabelNode in the reference implementation of retypd.

Parameters
target (TypeVariable | DerivedTypeVariable)

__init__(*target*)

Parameters
target (TypeVariable | DerivedTypeVariable)

target: *TypeVariable* | *DerivedTypeVariable*

class angr.analyses.typehoon.simple_solver.**Sketch**(*solver*, *root*)

Bases: object

Describes the sketch of a type variable.

Parameters

- **solver** (SimpleSolver)
- **root** (TypeVariable)

__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**(**values*)

Bases: Enum

LEFT = 0

RIGHT = 1

UNKNOWN = 2

class angr.analyses.typehoon.simple_solver.**FORGOTTEN**(**values*)

Bases: Enum

PRE_FORGOTTEN = 0

POST_FORGOTTEN = 1

class angr.analyses.typehoon.simple_solver.**ConstraintGraphNode**(*typevar*, *variance*, *tag*, *forgotten*)

Bases: object

Parameters

- **typevar** (TypeVariable | DerivedTypeVariable | TypeConstant)
- **variance** (*Variance*)
- **tag** (ConstraintGraphTag)
- **forgotten** (FORGOTTEN)

__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

class angr.analyses.typehoon.simple_solver.**SimpleSolver**(*bits*, *constraints*, *typevars*, *constraint_set_degradation_threshold=150*, *stackvar_max_sizes=None*, *tv_manager=None*)

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.

Parameters

- **bits** (*int*)
- **typevars** (*dict[TypeVariable, set[TypeVariable]]*)
- **constraint_set_degradation_threshold** (*int*)
- **stackvar_max_sizes** (*dict[TypeVariable, int] | None*)
- **tv_manager** (TypeVariableManager | *None*)

__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*)

class angr.analyses.typehoon.translator.**SimTypeTempRef**(*typevar*)

Bases: *SimType*

Represents a temporary reference to another type. TypeVariableReference is translated to SimTypeTempRef.

__init__(*typevar*)

Parameters
label – the type label.

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**(*arch*)

Bases: object

Translate type constants to SimType and SimType to type constants.

Parameters
arch (*archinfo.Arch*)

__init__(*arch*)

Parameters
arch (*Arch*)

arch: Arch

translated: dict[*TypeConstant*, *SimType*]

translated_simtypes: dict[*SimType*, *TypeConstant*]

structs

memo

named_struct_id_counter

struct_name_to_idx

struct_name()

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.typevars.**TypeConstraint**

Bases: object

pp_str(*mapping*)

Return type
str

Parameters
mapping (*dict[TypeVariable, Any]*)

class angr.analyses.typehoon.typevars.**Equivalence**(*type_a*, *type_b*)

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**(*type_*)

Bases: *TypeConstraint*

__init__(*type_*)

type_

pp_str(*mapping*)

Return type
str

Parameters
mapping (*dict[TypeVariable, Any]*)

replace(*replacements*)

class angr.analyses.typehoon.typevars.**Subtype**(*sub_type*, *super_type*)

Bases: *TypeConstraint*

Parameters

- **sub_type** (*TypeType*)
- **super_type** (*TypeType*)

__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**(*type_0*, *type_1*, *type_r*)

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**(*type_0*, *type_1*, *type_r*)

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**(*func_addr*, *idx=0*)

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.

Parameters

- **func_addr** (*int*)
- **idx** (*int*)

__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**(*idx=None*, *name=None*)

Bases: object

Parameters

- **idx** (*tuple[int, int] | None*)
- **name** (*str | None*)

__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**(*type_var*, *label*, *labels=None*, *idx=None*)

Bases: *TypeVariable*

Parameters

- **type_var** (TypeVariable | TypeConstant)
- **label** (BaseLabel | *None*)
- **labels** (*tuple[BaseLabel, ...]*)
- **idx** (*tuple[int, int]* | *None*)

__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**(*loc*)

Bases: *BaseLabel*

__init__(*loc*)

loc

class angr.analyses.typehoon.typevars.**FuncOut**(*loc*)

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**(*n*)

Bases: *BaseLabel*

__init__(*n*)

n

class angr.analyses.typehoon.typevars.**SubN**(*n*)

Bases: *BaseLabel*

__init__(*n*)

n

class angr.analyses.typehoon.typevars.**ConvertTo**(*to_bits*)

Bases: *BaseLabel*

__init__(*to_bits*)

to_bits

class angr.analyses.typehoon.typevars.**ReinterpretAs**(*to_type*, *to_bits*)

Bases: *BaseLabel*

__init__(*to_type*, *to_bits*)

to_type

to_bits

class angr.analyses.typehoon.typevars.**HasField**(*bits*, *offset*, *elem_count=1*)

Bases: *BaseLabel*

Has a field of the given bit size * elem_count at the given offset.

Parameters
elem_count (*int*)

__init__(*bits*, *offset*, *elem_count=1*)

Parameters
elem_count (*int*)

bits

offset

elem_count

class angr.analyses.typehoon.typevars.**IsArray**

Bases: *BaseLabel*

class angr.analyses.typehoon.typehoon.**Typehoon**(*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*)

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.

Parameters

- **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*)

__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

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**(*name=None*)

Bases: object

Parameters
name (*str | None*)

SIZE = None

__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[TypeConstant, TypeConstant]*)
- **memo** (*set[TypeConstant] | None*)

class angr.analyses.typehoon.typeconsts.**TopType**(*name=None*)

Bases: *TypeConstant*

Parameters
name (*str | None*)

SIZE = 1

class angr.analyses.typehoon.typeconsts.**BottomType**(*name=None*)

Bases: *TypeConstant*

Parameters
name (*str | None*)

SIZE = 1

class angr.analyses.typehoon.typeconsts.**Int**(*name=None*)

Bases: *TypeConstant*

Parameters
name (*str | None*)

class angr.analyses.typehoon.typeconsts.**Int1**(*name=None*)

Bases: *Int*

Parameters
name (*str | None*)

SIZE = 1

class angr.analyses.typehoon.typeconsts.**Int8**(*name=None*)

Bases: *Int*

Parameters
name (*str | None*)

SIZE = 1

class angr.analyses.typehoon.typeconsts.**Int16**(*name=None*)

Bases: *Int*

Parameters
name (*str | None*)

SIZE = 2

class angr.analyses.typehoon.typeconsts.**Int32**(*name=None*)

Bases: *Int*

Parameters
name (*str | None*)

SIZE = 4

class angr.analyses.typehoon.typeconsts.**Fd**(*name=None*)

Bases: *Int*

Parameters
name (*str | None*)

SIZE = 4

class angr.analyses.typehoon.typeconsts.**Int64**(*name=None*)

Bases: *Int*

Parameters
name (*str | None*)

SIZE = 8

class angr.analyses.typehoon.typeconsts.**SInt8**(*name=None*)

Bases: *Int8*

Parameters
name (*str | None*)

class angr.analyses.typehoon.typeconsts.**UInt8**(*name=None*)

Bases: *Int8*

Parameters
name (*str | None*)

class angr.analyses.typehoon.typeconsts.**SInt16**(*name=None*)

Bases: *Int16*

Parameters
name (*str | None*)

class angr.analyses.typehoon.typeconsts.**UInt16**(*name=None*)

Bases: *Int16*

Parameters
name (*str | None*)

class angr.analyses.typehoon.typeconsts.**SInt32**(*name=None*)

Bases: *Int32*

Parameters
name (*str | None*)

class angr.analyses.typehoon.typeconsts.**UInt32**(*name=None*)

Bases: *Int32*

Parameters
name (*str | None*)

class angr.analyses.typehoon.typeconsts.**SInt64**(*name=None*)

Bases: *Int64*

Parameters
name (*str | None*)

class angr.analyses.typehoon.typeconsts.**UInt64**(*name=None*)

Bases: *Int64*

Parameters
name (*str | None*)

class angr.analyses.typehoon.typeconsts.**Int128**(*name=None*)

Bases: *Int*

Parameters
name (*str | None*)

SIZE = 16

class angr.analyses.typehoon.typeconsts.**Int256**(*name=None*)

Bases: *Int*

Parameters
name (*str | None*)

SIZE = 32

class angr.analyses.typehoon.typeconsts.**Int512**(*name=None*)

Bases: *Int*

Parameters
name (*str* | *None*)

SIZE = 32

class angr.analyses.typehoon.typeconsts.**IntVar**(*size*, *name=None*)

Bases: *Int*

Parameters
name (*str* | *None*)

__init__(*size*, *name=None*)

Parameters
name (*str* | *None*)

property size: **int**

class angr.analyses.typehoon.typeconsts.**Float**(*name=None*)

Bases: *TypeConstant*

Parameters
name (*str* | *None*)

class angr.analyses.typehoon.typeconsts.**Float32**(*name=None*)

Bases: *Float*

Parameters
name (*str* | *None*)

SIZE = 4

class angr.analyses.typehoon.typeconsts.**Float64**(*name=None*)

Bases: *Float*

Parameters
name (*str* | *None*)

SIZE = 8

class angr.analyses.typehoon.typeconsts.**Pointer**(*basetype*, *name=None*)

Bases: *TypeConstant*

Parameters

- **basetype** (TypeConstant | *None*)
- **name** (*str* | *None*)

__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[TypeConstant, TypeConstant]*)
- **memo** (*set* | *None*)

class angr.analyses.typehoon.typeconsts.**Pointer32**(*basetype=None*, *name=None*)

Bases: *Pointer*, *Int32*

32-bit pointers.

Parameters
name (*str* | *None*)

__init__(*basetype=None*, *name=None*)

Parameters
name (*str* | *None*)

class angr.analyses.typehoon.typeconsts.**Pointer64**(*basetype=None*, *name=None*)

Bases: *Pointer*, *Int64*

64-bit pointers.

Parameters
name (*str* | *None*)

__init__(*basetype=None*, *name=None*)

Parameters
name (*str* | *None*)

class angr.analyses.typehoon.typeconsts.**Array**(*element=None*, *count=None*, *name=None*)

Bases: *TypeConstant*

Parameters
name (*str* | *None*)

__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[TypeConstant, TypeConstant]*)
- **memo** (*set | None*)

class angr.analyses.typehoon.typeconsts.**Struct**(*fields=None*, *name=None*, *field_names=None*, *is_cppclass=False*, *idx=-1*)

Bases: *TypeConstant*

Parameters

- **is_cppclass** (*bool*)
- **idx** (*int*)

__init__(*fields=None*, *name=None*, *field_names=None*, *is_cppclass=False*, *idx=-1*)

Parameters

- **is_cppclass** (*bool*)
- **idx** (*int*)

property size: int

class angr.analyses.typehoon.typeconsts.**EnumVariant**(*name*, *fields*, *discriminant*, *discriminant_size*, *size*)

Bases: object

__init__(*name*, *fields*, *discriminant*, *discriminant_size*, *size*)

fields: list[tuple[*TypeConstant*, str | None]]

class angr.analyses.typehoon.typeconsts.**RustEnum**(*name=None*, *variants=None*)

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[TypeConstant, TypeConstant]*)
- **memo** (*set | None*)

class angr.analyses.typehoon.typeconsts.**Enum**(*members=None*, *base_type=None*, *name=None*, *idx=-1*)

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

Parameters

- **members** (*dict[str, int] | None*)
- **base_type** (TypeConstant | *None*)
- **name** (*str | None*)
- **idx** (*int*)

__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[TypeConstant, TypeConstant]*)
- **memo** (*set | None*)

class angr.analyses.typehoon.typeconsts.**Function**(*params*, *outputs*, *name=None*)

Bases: *TypeConstant*

Parameters

- **params** (*list*)
- **outputs** (*list*)
- **name** (*str | None*)

__init__(*params*, *outputs*, *name=None*)

Parameters

- **params** (*list*)
- **outputs** (*list*)
- **name** (*str | None*)

replace(*mapping*, *memo=None*)

Return type
TypeConstant

Parameters

- **mapping** (*dict[TypeConstant, TypeConstant]*)
- **memo** (*set | None*)

class angr.analyses.typehoon.typeconsts.**TypeVariableReference**(*typevar*, *name=None*)

Bases: *TypeConstant*

Parameters
name (*str | None*)

__init__(*typevar*, *name=None*)

Parameters
name (*str | None*)

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*)

class angr.analyses.typehoon.**Typehoon**(*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*)

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.

Parameters

- **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*)

__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.identifier.identify.**FuncInfo**

Bases: object

__init__()

class angr.analyses.identifier.identify.**Identifier**(*cfg=None*, *require_predecessors=True*, *only_find=None*)

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.loopfinder.**Loop**(*entry*, *entry_edges*, *break_edges*, *continue_edges*, *body_nodes*, *graph*, *subloops*)

Bases: object

__init__(*entry*, *entry_edges*, *break_edges*, *continue_edges*, *body_nodes*, *graph*, *subloops*)

class angr.analyses.loopfinder.**LoopFinder**(*functions=None*, *normalize=True*)

Bases: *Analysis*

Extracts all the loops from all the functions in a binary.

__init__(*functions=None*, *normalize=True*)

class angr.analyses.loop_analysis.**LoopAnalysis**(*cfunc*)

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

Parameters
cfunc (CFunction)

__init__(*cfunc*)

Parameters
cfunc (CFunction)

exception angr.analyses.veritesting.**VeritestingError**

Bases: Exception

class angr.analyses.veritesting.**CallTracingFilter**(*project*, *depth*, *blacklist=None*)

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**(*input_state*, *boundaries=None*, *loop_unrolling_limit=10*, *enable_function_inlining=False*, *terminator=None*, *deviation_filter=None*)

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.vfg.**VFGJob**(**args*, ***kwargs*)

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**(*block_id*, *state*, *call_stack*, *src_block_id*, *src_stmt_idx*, *src_ins_addr*)

Bases: object

Describes a pending job during VFG analysis.

Parameters

- **block_id** (BlockID)
- **state** (SimState)
- **call_stack** (CallStack)
- **src_block_id** (BlockID)
- **src_stmt_idx** (*int*)
- **src_ins_addr** (*int*)

__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**(*function_address*, *return_address*)

Bases: *AnalysisTask*

Analyze a function, generate fix-point states from all endpoints of that function, and then merge them to one state.

Parameters

- **function_address** (*int*)
- **return_address** (*int | None*)

__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**(*address*, *return_address*, *function_analysis_tasks=None*, *mergeable_plugins=None*)

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.

Parameters

- **address** (*int*)
- **return_address** (*None*)
- **function_analysis_tasks** (*list[Any] | None*)
- **mergeable_plugins** (*tuple[str, str] | None*)

__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**(*addr*, *key*, *state=None*)

Bases: object

A descriptor of nodes in a Value-Flow Graph

Parameters

- **addr** (*int*)
- **key** (BlockID)
- **state** (SimState | *None*)

__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**(*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*)

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}.

Parameters

- **cfg** (CFGEmulated | None)
- **context_sensitivity_level** (int)
- **start** (int | None)
- **function_start** (int | None)
- **interfunction_level** (int)
- **initial_state** (SimState | None)
- **avoid_runs** (list[int] | None)
- **remove_options** (set[str] | None)
- **timeout** (int | None)
- **max_iterations_before_widening** (int)
- **max_iterations** (int)
- **widening_interval** (int)
- **final_state_callback** (Callable[[SimState, CallStack], Any] | None)
- **status_callback** (Callable[[VFG], Any] | None)
- **record_function_final_states** (bool)

__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()

class angr.analyses.vsa_ddg.**DefUseChain**(*def_loc*, *use_loc*, *variable*)

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**(*vfg=None*, *start_addr=None*, *interfunction_level=0*, *context_sensitivity_level=2*, *keep_data=False*)

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.vtable.**Vtable**(*vaddr*, *size*, *func_addrs=None*)

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*)

class angr.analyses.find_objects_static.**PossibleObject**(*size*, *addr*, *class_name=None*)

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**(*max_addr=None*, *new_func_addr=None*, *project=None*)

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)

__init__(*max_addr=None*, *new_func_addr=None*, *project=None*)

Parameters

- **interfunction_level** – Maximum depth in to continue local function exploration
- **extra_impls** – FunctionHandler classes to implement beyond what's implemented in function_handler_library

hook(*analysis*)

Attach this instance of the function handler to an instance of RDA.

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__()

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__()

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**(*func*)

Bases: *DisassemblyPiece*

__init__(*func*)

Constructor.

Parameters
func (*angr.knowledge.Function*) – The function instance.

height(*formatting*)

class angr.analyses.disassembly.**Label**(*addr*, *name*)

Bases: *DisassemblyPiece*

__init__(*addr*, *name*)

class angr.analyses.disassembly.**IROp**(*addr*, *seq*, *obj*, *irsb*)

Bases: *DisassemblyPiece*

Parameters

- **addr** (*int*)
- **seq** (*int*)
- **obj** (*IRStmt | PcodeOp*)
- **irsb** (*IRSB | IRSB*)

__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**(*block*, *parentfunc*, *project*)

Bases: *DisassemblyPiece*

__init__(*block*, *parentfunc*, *project*)

class angr.analyses.disassembly.**Hook**(*block*)

Bases: *DisassemblyPiece*

__init__(*block*)

class angr.analyses.disassembly.**Instruction**(*insn*, *parentblock*, *project=None*)

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**(*expr*)

Bases: *DisassemblyPiece*

__init__(*expr*)

class angr.analyses.disassembly.**SootExpressionTarget**(*target_stmt_idx*)

Bases: *SootExpression*

__init__(*target_stmt_idx*)

class angr.analyses.disassembly.**SootExpressionStaticFieldRef**(*field*)

Bases: *SootExpression*

__init__(*field*)

class angr.analyses.disassembly.**SootExpressionInvoke**(*invoke_type*, *expr*)

Bases: *SootExpression*

Virtual = 'virtual'

Static = 'static'

Special = 'special'

__init__(*invoke_type*, *expr*)

class angr.analyses.disassembly.**SootStatement**(*block_addr*, *raw_stmt*)

Bases: *DisassemblyPiece*

__init__(*block_addr*, *raw_stmt*)

property stmt_idx

class angr.analyses.disassembly.**Opcode**(*parentinsn*)

Bases: *DisassemblyPiece*

__init__(*parentinsn*)

class angr.analyses.disassembly.**Operand**(*op_num*, *children*, *parentinsn*)

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**(*op_num*, *children*, *parentinsn*)

Bases: *Operand*

class angr.analyses.disassembly.**RegisterOperand**(*op_num*, *children*, *parentinsn*)

Bases: *Operand*

property register

class angr.analyses.disassembly.**MemoryOperand**(*op_num*, *children*, *parentinsn*)

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**(*reg*, *prefix=''*)

Bases: *OperandPiece*

__init__(*reg*, *prefix=''*)

class angr.analyses.disassembly.**Value**(*val*, *render_with_sign*, *render_with_pound_sign=False*)

Bases: *OperandPiece*

Parameters
render_with_pound_sign (*bool*)

__init__(*val*, *render_with_sign*, *render_with_pound_sign=False*)

Parameters
render_with_pound_sign (*bool*)

property project

class angr.analyses.disassembly.**Comment**(*addr*, *text*)

Bases: *DisassemblyPiece*

__init__(*addr*, *text*)

height(*formatting*)

class angr.analyses.disassembly.**FuncComment**(*func*)

Bases: *DisassemblyPiece*

__init__(*func*)

class angr.analyses.disassembly.**Disassembly**(*function=None*, *ranges=None*, *thumb=False*, *include_ir=False*, *block_bytes=None*)

Bases: *Analysis*

Produce formatted machine code disassembly.

Parameters

- **function** (*Function | None*)
- **ranges** (*Sequence[tuple[int, int]] | None*)
- **thumb** (*bool*)
- **include_ir** (*bool*)
- **block_bytes** (*bytes | None*)

__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*)

angr.analyses.disassembly_utils.**decode_instruction**(*arch*, *instr*)

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**(*binary*, *name*, *original_addr=None*)

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**(*binary*, *original_addr*, *name=None*)

Bases: *Label*

__init__(*binary*, *original_addr*, *name=None*)

property operand_str

class angr.analyses.reassembler.**FunctionLabel**(*binary*, *function_name*, *original_addr*, *plt=False*)

Bases: *Label*

__init__(*binary*, *function_name*, *original_addr*, *plt=False*)

property function_name

property operand_str

class angr.analyses.reassembler.**ObjectLabel**(*binary*, *symbol_name*, *original_addr*, *plt=False*)

Bases: *Label*

__init__(*binary*, *symbol_name*, *original_addr*, *plt=False*)

property symbol_name

property operand_str

class angr.analyses.reassembler.**NotypeLabel**(*binary*, *symbol_name*, *original_addr*, *plt=False*)

Bases: *Label*

__init__(*binary*, *symbol_name*, *original_addr*, *plt=False*)

property symbol_name

property operand_str

class angr.analyses.reassembler.**SymbolManager**(*binary*, *cfg*)

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**(*binary*, *insn_addr*, *insn_size*, *capstone_operand*, *operand_str*, *mnemonic*, *operand_offset*, *syntax=None*)

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**(*binary*, *addr*, *size*, *insn_bytes*, *capstone_instr*)

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**(*binary*, *addr*, *size*, *x86_getpc_retsite=False*)

Bases: object

BasicBlock represents a basic block in the binary.

Parameters
x86_getpc_retsite (*bool*)

__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**(*binary*, *function=None*, *addr=None*, *size=None*, *name=None*, *section='.text'*, *asm_code=None*)

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**(*project*, *addr*, *size*)

Bases: *Procedure*

Procedure chunk.

__init__(*project*, *addr*, *size*)

Constructor.

Parameters

- **project**
- **addr**
- **size**

Returns

class angr.analyses.reassembler.**Data**(*binary*, *memory_data=None*, *section=None*, *section_name=None*, *name=None*, *size=None*, *sort=None*, *addr=None*, *initial_content=None*)

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**(*addr*, *ref_addr*, *sort*)

Bases: object

__init__(*addr*, *ref_addr*, *sort*)

class angr.analyses.reassembler.**Reassembler**(*syntax='intel'*, *remove_cgc_attachments=True*, *log_relocations=True*)

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:

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.congruency_check.**CongruencyCheck**(*throw=False*)

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.static_hooker.**StaticHooker**(*library*, *binary=None*)

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.binary_optimizer.**ConstantPropagation**(*constant*, *constant_assignment_loc*, *constant_consuming_loc*)

Bases: object

__init__(*constant*, *constant_assignment_loc*, *constant_consuming_loc*)

class angr.analyses.binary_optimizer.**RedundantStackVariable**(*argument*, *stack_variable*, *stack_variable_consuming_locs*)

Bases: object

__init__(*argument*, *stack_variable*, *stack_variable_consuming_locs*)

class angr.analyses.binary_optimizer.**RegisterReallocation**(*stack_variable*, *register_variable*, *stack_variable_sources*, *stack_variable_consumers*, *prologue_addr*, *prologue_size*, *epilogue_addr*, *epilogue_size*)

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**(*pv*)

Bases: object

__init__(*pv*)

Constructor.

Parameters

pv (angr.analyses.ddg.ProgramVariable) – The assignment to remove.

class angr.analyses.binary_optimizer.**BinaryOptimizer**(*cfg*, *techniques*)

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*)

optimize()

class angr.analyses.callee_cleanup_finder.**CalleeCleanupFinder**(*starts=None*, *hook_all=False*)

Bases: *Analysis*

__init__(*starts=None*, *hook_all=False*)

analyze(*addr*)

class angr.analyses.dominance_frontier.**DominanceFrontier**(*func*, *func_graph=None*, *entry=None*, *exception_edges=False*)

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*)

class angr.analyses.init_finder.**SimEngineInitFinderVEX**(*project*, *replacements*, *overlay*, *pointers_only=False*)

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**(*func=None*, *func_graph=None*, *block=None*, *max_iterations=1*, *replacements=None*, *overlay=None*, *pointers_only=False*)

Bases: *ForwardAnalysis*, *Analysis*

Finds possible initializations for global data sections and generate an overlay to be used in other analyses later on.

__init__(*func=None*, *func_graph=None*, *block=None*, *max_iterations=1*, *replacements=None*, *overlay=None*, *pointers_only=False*)

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** (*GraphVisitor or None*) – A graph visitor to provide successors.

Returns
None

class angr.analyses.xrefs.**SimEngineXRefsVEX**(*xref_manager*, *project*, *replacements=None*)

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**(*func=None*, *func_graph=None*, *block=None*, *max_iterations=1*, *replacements=None*)

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
```

__init__(*func=None*, *func_graph=None*, *block=None*, *max_iterations=1*, *replacements=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** (*GraphVisitor or None*) – A graph visitor to provide successors.

Returns

None

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**(*type_*, *ref_at=None*)

Bases: object

Base class for all nodes in a proximity graph.

Parameters

- **type_** (*int*)
- **ref_at** (*set[int] | None*)

__init__(*type_*, *ref_at=None*)

Parameters

- **type_** (*int*)
- **ref_at** (*set[int] | None*)

class angr.analyses.proximity_graph.**FunctionProxiNode**(*func*, *ref_at=None*)

Bases: *BaseProxiNode*

Proximity node showing current and expanded function calls in graph.

Parameters

- **func** (FuncNode)
- **ref_at** (*set[int] | None*)

__init__(*func*, *ref_at=None*)

Parameters

- **func** (FuncNode)
- **ref_at** (*set[int] | None*)

class angr.analyses.proximity_graph.**VariableProxiNode**(*addr*, *name*, *ref_at=None*)

Bases: *BaseProxiNode*

Variable arg node

Parameters
ref_at (*set[int] | None*)

__init__(*addr*, *name*, *ref_at=None*)

Parameters
ref_at (*set[int] | None*)

class angr.analyses.proximity_graph.**StringProxiNode**(*addr*, *content*, *ref_at=None*)

Bases: *BaseProxiNode*

String arg node

Parameters
ref_at (*set[int] | None*)

__init__(*addr*, *content*, *ref_at=None*)

Parameters
ref_at (*set[int] | None*)

class angr.analyses.proximity_graph.**CallProxiNode**(*callee*, *ref_at=None*, *args=None*)

Bases: *BaseProxiNode*

Call node

Parameters

- **callee** (FuncNode)
- **ref_at** (*set[int] | None*)
- **args** (*tuple[BaseProxiNode] | None*)

__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**(*value*, *ref_at=None*)

Bases: *BaseProxiNode*

Int arg node

Parameters

- **value** (*int*)
- **ref_at** (*set[int] | None*)

__init__(*value*, *ref_at=None*)

Parameters

- **value** (*int*)
- **ref_at** (*set[int] | None*)

class angr.analyses.proximity_graph.**UnknownProxiNode**(*dummy_value*)

Bases: *BaseProxiNode*

Unknown arg node

Parameters
dummy_value (*str*)

__init__(*dummy_value*)

Parameters
dummy_value (*str*)

class angr.analyses.proximity_graph.**ProximityGraphAnalysis**(*func*, *cfg_model*, *xrefs*, *decompilation=None*, *expand_funcs=None*)

Bases: *Analysis*

Generate a proximity graph.

Parameters

- **func** (Function)
- **cfg_model** (CFGModel)
- **xrefs** (XRefManager)
- **decompilation** (Decompiler | None)
- **expand_funcs** (*set[int] | None*)

__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

Defines analysis that will generate a dynamic data-dependency graph

class angr.analyses.data_dep.data_dependency_analysis.**NodalAnnotation**(*node*)

Bases: Annotation

Allows a node to be stored as an annotation to a BV in a DefaultMemory instance

Parameters
node (BaseDepNode)

__init__(*node*)

Parameters
node (BaseDepNode)

property relocatable: bool

Can not be relocated in a simplification

property eliminatable

Can not be eliminated in a simplification

class angr.analyses.data_dep.data_dependency_analysis.**DataDependencyGraphAnalysis**(*end_state*, *start_from=None*, *end_at=None*, *block_addrs=None*)

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.

Parameters

- **end_state** (SimState)
- **start_from** (*int | None*)
- **end_at** (*int | None*)
- **block_addrs** (*list[int] | None*)

__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.sim_act_location.**SimActLocation**(*bbl_addr*, *ins_addr*, *stmt_idx*)

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

Parameters

- **bbl_addr** (*int*)
- **ins_addr** (*int*)
- **stmt_idx** (*int*)

__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**(*ins_addr*, *min_stmt_idx*, *max_stmt_idx*)

Bases: object

Used by parser to facilitate linking with recent ancestors in an efficient manner

Parameters

- **ins_addr** (*int*)
- **min_stmt_idx** (*int*)
- **max_stmt_idx** (*int*)

__init__(*ins_addr*, *min_stmt_idx*, *max_stmt_idx*)

Parameters

- **ins_addr** (*int*)
- **min_stmt_idx** (*int*)
- **max_stmt_idx** (*int*)

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**(*type_*, *sim_act*)

Bases: object

Base class for all nodes in a data-dependency graph

Parameters

- **type_** (*int*)
- **sim_act** (SimActionData)

__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**(*sim_act*, *value*)

Bases: *BaseDepNode*

Used to create a DepNode that will hold a constant, numeric value Uniquely identified by its value

Parameters

- **sim_act** (SimActionData)
- **value** (*int*)

__init__(*sim_act*, *value*)

Parameters

- **sim_act** (SimActionData)
- **value** (*int*)

class angr.analyses.data_dep.dep_nodes.**MemDepNode**(*sim_act*, *addr*)

Bases: *BaseDepNode*

Used to represent SimActions of type MEM

Parameters

- **sim_act** (SimActionData)
- **addr** (*int*)

__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**(*type_*, *sim_act*, *reg*, *arch_name=''*)

Bases: *BaseDepNode*

Abstract class for representing SimActions of TYPE reg or tmp

Parameters

- **type_** (*int*)
- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

__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**(*sim_act*, *reg*, *arch_name=''*)

Bases: *VarDepNode*

Used to represent SimActions of type TMP

Parameters

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

__init__(*sim_act*, *reg*, *arch_name=''*)

Parameters

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

class angr.analyses.data_dep.dep_nodes.**RegDepNode**(*sim_act*, *reg*, *arch_name=''*)

Bases: *VarDepNode*

Base class for representing SimActions of TYPE reg

Parameters

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

__init__(*sim_act*, *reg*, *arch_name=''*)

Parameters

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

property reg_size: int

class angr.analyses.data_dep.**BaseDepNode**(*type_*, *sim_act*)

Bases: object

Base class for all nodes in a data-dependency graph

Parameters

- **type_** (*int*)
- **sim_act** (SimActionData)

__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.**ConstantDepNode**(*sim_act*, *value*)

Bases: *BaseDepNode*

Used to create a DepNode that will hold a constant, numeric value Uniquely identified by its value

Parameters

- **sim_act** (SimActionData)
- **value** (*int | None*)

__init__(*sim_act*, *value*)

Parameters

- **sim_act** (SimActionData)
- **value** (*int*)

class angr.analyses.data_dep.**DataDependencyGraphAnalysis**(*end_state*, *start_from=None*, *end_at=None*, *block_addrs=None*)

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.

Parameters

- **end_state** (SimState)
- **start_from** (*int | None*)
- **end_at** (*int | None*)
- **block_addrs** (*list[int] | None*)

__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**(*sim_act*, *addr*)

Bases: *BaseDepNode*

Used to represent SimActions of type MEM

Parameters

- **sim_act** (SimActionData)
- **addr** (*int*)

__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**(*sim_act*, *reg*, *arch_name=''*)

Bases: *VarDepNode*

Base class for representing SimActions of TYPE reg

Parameters

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

__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**(*sim_act*, *reg*, *arch_name=''*)

Bases: *VarDepNode*

Used to represent SimActions of type TMP

Parameters

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

__init__(*sim_act*, *reg*, *arch_name=''*)

Parameters

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

class angr.analyses.data_dep.**VarDepNode**(*type_*, *sim_act*, *reg*, *arch_name=''*)

Bases: *BaseDepNode*

Abstract class for representing SimActions of TYPE reg or tmp

Parameters

- **type_** (*int*)
- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

__init__(*type_*, *sim_act*, *reg*, *arch_name=''*)

Parameters

- **type_** (*int*)
- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

property display_name: str

exception angr.blade.**BadJumpkindNotification**

Bases: Exception

Notifies the caller that the jumpkind is bad (e.g., Ijk_NoDecode)

class angr.blade.**Blade**(*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*)

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.

Parameters

- **graph** (*networkx.DiGraph*)
- **dst_run** (*int*)
- **dst_stmt_idx** (*int*)

- **direction** (*str*)
- **ignore_sp** (*bool*)
- **ignore_bp** (*bool*)
- **max_level** (*int*)
- **stop_at_calls** (*bool*)
- **max_predecessors** (*int*)
- **include_imarks** (*bool*)
- **control_dependence** (*bool*)

__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.slicer.**SimLightState**(*temps=None*, *regs=None*, *stack_offsets=None*, *options=None*)

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**(*arch*, *statements*, *target_tmps=None*, *target_regs=None*, *target_stack_offsets=None*, *inslice_callback=None*, *inslice_callback_infodict=None*, *include_imarks=True*)

Bases: object

A super lightweight intra-IRSB slicing class.

Parameters
include_imarks (*bool*)

__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*)

class angr.annocfg.**AnnotatedCFG**(*project*, *cfg=None*, *detect_loops=False*)

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.

angr.codenode.**repr_addr**(*addr*)

Return type
`str`

Parameters
addr (*K*)

class angr.codenode.**CodeNode**(*addr*, *size*, *graph=None*, *thumb=False*)

Bases: Generic

The base class of nodes in a function graph.

Parameters

- **addr** (*K*)
- **size** (*int*)

__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**(*addr*, *size*, *bytestr=None*, ***kwargs*)

Bases: *CodeNode*, Generic

Represents a block of code in a function graph.

Parameters
addr (*int*)

is_hook = False

__init__(*addr*, *size*, *bytestr=None*, ***kwargs*)

Parameters
addr (*int*)

bytestr

class angr.codenode.**SootBlockNode**(*addr*, *size*, *stmts*, ***kwargs*)

Bases: *BlockNode*[SootMethodDescriptor]

Represents a Soot block of code in a function graph.

Parameters
addr (*SootMethodDescriptor*)

__init__(*addr*, *size*, *stmts*, ***kwargs*)

Parameters
addr (*SootMethodDescriptor*)

stmts

class angr.codenode.**FuncNode**(*addr*, *func_name=None*, ***kwargs*)

Bases: *CodeNode*, Generic

Represents a function callee in a function graph.

Parameters

- **addr** (*K*)
- **func_name** (*str | None*)

__init__(*addr*, *func_name=None*, ***kwargs*)

Parameters

- **addr** (*K*)
- **func_name** (*str | None*)

func_name

property is_addr_known: bool

class angr.codenode.**HookNode**(*addr*, *size*, *sim_procedure*, ***kwargs*)

Bases: *CodeNode*, Generic

Represents a hook in a function graph.

Parameters
sim_procedure (*SimProcedure | None*)

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**(*addr*, *size*, *sim_procedure*, ***kwargs*)

Bases: *HookNode*, Generic

Represents a syscall in a function graph.

Parameters
sim_procedure (*SimProcedure | None*)

is_hook = False

sim_procedure

10.16 SimOS

Manage OS-level configuration.

class angr.simos.**SimCGC**(*project*, ***kwargs*)

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**(**args*, ***kwargs*)

Bases: *SimOS*

__init__(**args*, ***kwargs*)

state_blank(*addr=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(*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**(*project*, ***kwargs*)

Bases: *SimUserland*

OS-specific configuration for *nix-y OSes.

__init__(*project*, ***kwargs*)

configure_project()

Configure the project to set up global settings (like SimProcedures).

syscall_abi(*state*)

Optionally, override this function to determine which abi is being used for the state's current syscall.

state_blank(*fs=None*, *concrete_fs=False*, *chroot=None*, *cwd=None*, *pathsep=b'/'*, *thread_idx=None*, *init_libc=False*, ***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(*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.

initialize_segment_register_x64(*state*, *concrete_target*)

Set the fs register in the angr to the value of the fs register in the concrete process

Parameters

- **state** – state which will be modified
- **concrete_target** – concrete target that will be used to read the fs register

Returns
None

initialize_gdt_x86(*state*, *concrete_target*)

Create a GDT in the state memory and populate the segment registers. Rehook the vsyscall address using the real value in the concrete process memory

Parameters

- **state** – state which will be modified
- **concrete_target** – concrete target that will be used to read the fs register

Returns

get_segment_register_name()

class angr.simos.**SimOS**(*project*, *name=None*)

Bases: object

A class describing OS/arch-level configuration.

Parameters

- **project** (angr.Project)
- **name** (*str | None*)

__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**(*project*, *name=None*, ***kwargs*)

Bases: SimOS

This class implements the "OS" for a bare-metal firmware used at an imaginary company.

Parameters

- **project** (Project)
- **name** (*str | None*)

__init__(*project*, *name=None*, ***kwargs*)

Parameters
project (Project)

configure_project()

Configure the project to set up global settings (like SimProcedures).

class angr.simos.**SimUserland**(*project*, *syscall_library=None*, *syscall_addr_alignment=4*, ***kwargs*)

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*)

configure_project(*abi_list=None*)

Configure the project to set up global settings (like SimProcedures).

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**(*project*)

Bases: *SimOS*

Environment for the Windows Win32 subsystem. Does not support syscalls currently.

__init__(*project*)

configure_project()

Configure the project to set up global settings (like SimProcedures).

property is_dump: bool

state_entry(*args=None*, *env=None*, *argc=None*, ***kwargs*)

state_blank(*thread_idx=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

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*)

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.

`initialize_segment_register_x64`(*state*, *concrete_target*)

Set the gs register in the angr to the value of the fs register in the concrete process

Parameters

- **state** – state which will be modified
- **concrete_target** – concrete target that will be used to read the fs register

Returns
None

`initialize_gdt_x86`(*state*, *concrete_target*)

Create a GDT in the state memory and populate the segment registers.

Parameters

- **state** – state which will be modified
- **concrete_target** – concrete target that will be used to read the fs register

Returns
the created GlobalDescriptorTable object

`get_segment_register_name`()

`class` angr.simos.simos.`SimOS`(*project*, *name=None*)

Bases: `object`

A class describing OS/arch-level configuration.

Parameters

- **project** (`angr.Project`)
- **name** (*str | None*)

`__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**(*addr*, *limit*, *table*, *gdt_sel*, *cs_sel*, *ds_sel*, *es_sel*, *ss_sel*, *fs_sel*, *gs_sel*)

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*)

class angr.simos.linux.**SimLinux**(*project*, ***kwargs*)

Bases: *SimUserland*

OS-specific configuration for *nix-y OSes.

__init__(*project*, ***kwargs*)

configure_project()

Configure the project to set up global settings (like SimProcedures).

syscall_abi(*state*)

Optionally, override this function to determine which abi is being used for the state's current syscall.

state_blank(*fs=None*, *concrete_fs=False*, *chroot=None*, *cwd=None*, *pathsep=b'/'*, *thread_idx=None*, *init_libc=False*, ***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(*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.

initialize_segment_register_x64(*state*, *concrete_target*)

Set the fs register in the angr to the value of the fs register in the concrete process

Parameters

- **state** – state which will be modified
- **concrete_target** – concrete target that will be used to read the fs register

Returns
None

initialize_gdt_x86(*state*, *concrete_target*)

Create a GDT in the state memory and populate the segment registers. Rehook the vsyscall address using the real value in the concrete process memory

Parameters

- **state** – state which will be modified
- **concrete_target** – concrete target that will be used to read the fs register

Returns

get_segment_register_name()

class angr.simos.cgc.**SimCGC**(*project*, ***kwargs*)

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.userland.**SimUserland**(*project*, *syscall_library=None*, *syscall_addr_alignment=4*, ***kwargs*)

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*)

configure_project(*abi_list=None*)

Configure the project to set up global settings (like SimProcedures).

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.windows.**SecurityCookieInit**(**values*)

Bases: Enum

Security cooke initialization value initialization method.

NONE = 0

RANDOM = 1

STATIC = 2

SYMBOLIC = 3

class angr.simos.windows.**SimWindows**(*project*)

Bases: *SimOS*

Environment for the Windows Win32 subsystem. Does not support syscalls currently.

__init__(*project*)

configure_project()

Configure the project to set up global settings (like SimProcedures).

property is_dump: bool

state_entry(*args=None*, *env=None*, *argc=None*, ***kwargs*)

state_blank(*thread_idx=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

`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*)

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.

`initialize_segment_register_x64`(*state*, *concrete_target*)

Set the gs register in the angr to the value of the fs register in the concrete process

Parameters

- `state` – state which will be modified
- `concrete_target` – concrete target that will be used to read the fs register

Returns
None

`initialize_gdt_x86`(*state*, *concrete_target*)

Create a GDT in the state memory and populate the segment registers.

Parameters

- `state` – state which will be modified

- **concrete_target** – concrete target that will be used to read the fs register

Returns
the created GlobalDescriptorTable object

get_segment_register_name()

class angr.simos.javavm.**SimJavaVM**(**args*, ***kwargs*)

Bases: *SimOS*

__init__(**args*, ***kwargs*)

state_blank(*addr=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(*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.

10.17 Function Signature Matching

angr.flirt.**FS**

alias of *FlirtSignature*

class angr.flirt.**FlirtSignature**(*arch*, *platform*, *sig_name*, *sig_path*, *unique_strings=None*, *compiler=None*, *compiler_version=None*, *os_name=None*, *os_version=None*)

Bases: object

This class describes a FLIRT signature without any internal data that is only available after parsing.

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*)

__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

`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**()

10.18 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*)

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

angr.utils.constants.**is_alignment_mask**(*n*)

class angr.utils.cowdict.**ChainMapCOW**(**args*, *collapse_threshold=None*)

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.

__init__(**args*, *collapse_threshold=None*)

Initialize a ChainMap by setting *maps* to the given mappings. If no mappings are provided, a single empty dictionary is used.

copy()

New ChainMap or subclass with a new copy of maps[0] and refs to maps[1:]

Return type
`Self`

pop(*key*, **args*)

Remove *key* from maps[0] and return its value. Raise KeyError if *key* not in maps[0].

Return type
`TypeVar(V)`

Parameters
key (*K*)

get(*key*, *default=None*)

Return type
`Union[TypeVar(V), TypeVar(TD)]`

Parameters

- **key** (*K*)
- **default** (*TD*)

new_child(*m=None*)

New ChainMap with a new map followed by all previous maps. If no map is provided, an empty dict is used. Keyword arguments update the map or new empty dict.

Return type
ChainMapCOW`[TypeVar(K), TypeVar(V)]`

clean()

Return type
ChainMapCOW`[TypeVar(K), TypeVar(V)]`

class angr.utils.cowdict.**DefaultChainMapCOW**(**args*, *default_factory*, *collapse_threshold=None*)

Bases: *ChainMapCOW*, `Generic`

Implements a copy-on-write version of ChainMap with default values that supports auto-collapsing.

Parameters
default_factory (`Callable`)

__init__(**args*, *default_factory*, *collapse_threshold=None*)

Initialize a ChainMap by setting *maps* to the given mappings. If no mappings are provided, a single empty dictionary is used.

Parameters
default_factory (*Callable*)

new_child(*m=None*, ***kwargs*)

New ChainMap with a new map followed by all previous maps. If no map is provided, an empty dict is used. Keyword arguments update the map or new empty dict.

Return type
DefaultChainMapCOW`[TypeVar(K), TypeVar(V)]`

clean()

Return type
DefaultChainMapCOW[TypeVar(K), TypeVar(V)]

class angr.utils.dynamic_dictlist.**DynamicDictList**(*max_size=None*, *content=None*)

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.

Parameters

- **max_size** (*int | None*)
- **content** (*DynamicDictList | dict[int, VT] | list[VT] | None*)

__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

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*)

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).

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_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**(*label*)

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**(*obj*)

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.

Parameters
obj (*T*)

__init__(*obj*)

Parameters
obj (*T*)

index: int | None

property obj: T

class angr.utils.graph.**Dominators**(*graph*, *entry_node*, *successors_func=None*, *reverse=False*)

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**(*graph*, *entry_node*, *successors_func=None*)

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**(*scc_id*, *addr*)

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 **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 **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.

angr.utils.lazy_import.**lazy_import**(*name*)

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.

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*)

angr.utils.timing.**print_timing_total**()

angr.utils.timing.**timethis**(*func*)

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

class angr.utils.mp.**Closure**(*f: Callable[..., None], args: list[Any], kwargs: dict[str, Any]*)

Bases: NamedTuple

A pickle-able lambda; note that f, args, and kwargs must be pickleable

Parameters

- **f** (*Callable[[...], None]*)
- **args** (*list[Any]*)
- **kwargs** (*dict[str, Any]*)

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

class angr.utils.mp.**Initializer**(*, *_manual=True*)

Bases: object

A singleton class with global state used to initialize a multiprocessing.Process

Parameters

_manual (*bool*)

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**()

10.19 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**(*missing_type*)

Bases: *AngrTypeError*

Parameters

missing_type (*str*)

__init__(*missing_type*)

Parameters

missing_type (*str*)

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**(*missing_addr*, *missing_size*, **args*)

Bases: *SimMemoryError*

__init__(*missing_addr*, *missing_size*, **args*)

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**(*expr_type*, *expr*)

Bases: *SimExpressionError*

__init__(*expr_type*, *expr*)

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**(*state*)

Bases: *SimEngineError*

__init__(*state*)

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**(*addr*, *reason*, *original_addr=None*)

Bases: *SimException*, *SimMemoryError*

__init__(*addr*, *reason*, *original_addr=None*)

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.**UnsupportedNodeTypeError**

Bases: *AngrError*, NotImplementedError

exception angr.errors.**AngrAIError**

Bases: *AngrError*

10.20 Distributed analysis

angr.distributed provides a simple implementation for conducting long-running symbolic-execution-based tasks.

class angr.distributed.**Server**(*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*)

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()

class angr.distributed.server.**Server**(*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*)

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()

class angr.distributed.worker.**BadStatesDropper**(*vault*, *db*)

Bases: *ExplorationTechnique*

Dumps and drops states that are not "active".

__init__(*vault*, *db*)

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*)

class angr.distributed.worker.**ExplorationStatusNotifier**(*server_state*)

Bases: *ExplorationTechnique*

Force the exploration to stop if the server.stop is True.

Parameters
server_state (*dict*)

__init__(*server_state*)

Parameters
server_state (*dict*)

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*)

class angr.distributed.worker.**Worker**(*worker_id*, *server*, *server_state*, *recursion_limit=None*, *techniques=None*, *add_options=None*, *remove_options=None*)

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)

10.21 angr Intermediate Language

class angr.ailment.**AILBlockRewriter**(*stmt_handlers=None*, *expr_handlers=None*, *update_block=True*, *replace_phi_stmt=False*)

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.

Parameters

- **stmt_handlers** (*dict[type, Callable[[int, Any, Block | None], StmtType]]*)
- **expr_handlers** (*dict[type, Callable[[int, Any, int, Statement | None, Block | None], ExprType]]*)
- **update_block** (*bool*)
- **replace_phi_stmt** (*bool*)

__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**(*stmt_handlers=None*, *expr_handlers=None*)

Bases: *AILBlockWalker*[None, None, None]

Walks all statements and expressions of an AIL node and do nothing.

Parameters

- **stmt_handlers** (*dict[type, Callable[[int, Any, Block | None], StmtType]]*)
- **expr_handlers** (*dict[type, Callable[[int, Any, int, Statement | None, Block | None], ExprType]]*)

class angr.ailment.**AILBlockWalker**(*stmt_handlers=None*, *expr_handlers=None*)

Bases: Generic

Walks all statements and expressions of an AIL node and construct arbitrary values based on them.

Parameters

- **stmt_handlers** (*dict[type, Callable[[int, Any, Block | None], StmtType]]*)
- **expr_handlers** (*dict[type, Callable[[int, Any, int, Statement | None, Block | None], ExprType]]*)

__init__(*stmt_handlers=None*, *expr_handlers=None*)

stmt_handlers: dict[*type*, Callable[[int, Any, *Block* | None], StmtType]]

expr_handlers: dict[*type*, Callable[[int, Any, int, *Statement* | None, *Block* | None], ExprType]]

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**(*idx*, *dst*, *src*, ***kwargs*)

Bases: *Statement*

Assignment statement: expr_a = expr_b

Parameters

- **idx** (*int*)
- **dst** (Atom)
- **src** (Expression)

__init__(*idx*, *dst*, *src*, ***kwargs*)

Parameters

- **idx** (*int*)
- **dst** (Atom)
- **src** (Expression)

dst

src

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

property depth

copy()

Return type
Assignment

deep_copy(*manager*)

Return type
Assignment

class angr.ailment.**BinaryOp**(*idx*, *op*, *operands*, *signed=False*, *, *variable=None*, *variable_offset=None*, *bits=None*, *floating_point=False*, *rounding_mode=None*, *vector_count=None*, *vector_size=None*, **kwargs*)

Bases: *Op*

Parameters

- **idx** (*int*)
- **op** (*str*)
- **operands** (*Sequence[Expression]*)
- **signed** (*bool*)
- **bits** (*int*)
- **vector_count** (*int | None*)
- **vector_size** (*int | None*)

OPSTR_MAP = {'Add': '+', 'AddF': '+', 'AddV': '+', 'And': '&', 'Carry': 'CARRY', 'CmpEQ': '==', 'CmpF': 'CmpF', 'CmpGE': '>=', 'CmpGE (signed)': '>=s', 'CmpGT': '>', 'CmpGT (signed)': '>s', 'CmpLE': '<=', 'CmpLE (signed)': '<=s', 'CmpLT': '<', 'CmpLT (signed)': '<s', 'CmpNE': '!=', 'Concat': 'CONCAT', 'Div': '/', 'DivF': '/', 'LogicalAnd': '&&', 'LogicalOr': '||', 'Mod': '%', 'Mul': '*', 'MulF': '*', 'MulV': '*', 'Or': '|', 'Rol': 'ROL', 'Ror': 'ROR', 'SBorrow': 'SBORROW', 'SCarry': 'SCARRY', 'Sar': '>>a', 'Shl': '<<', 'Shr': '>>', 'Sub': '-', 'SubF': '-', 'Xor': '^'}

COMPARISON_NEGATION = {'CmpEQ': 'CmpNE', 'CmpGE': 'CmpLT', 'CmpGT': 'CmpLE', 'CmpLE': 'CmpGT', 'CmpLT': 'CmpGE', 'CmpNE': 'CmpEQ'}

__init__(*idx*, *op*, *operands*, *signed=False*, *, *variable=None*, *variable_offset=None*, *bits=None*, *floating_point=False*, *rounding_mode=None*, *vector_count=None*, *vector_size=None*, **kwargs*)

Parameters

- **idx** (*int*)
- **op** (*str*)
- **operands** (*Sequence[Expression]*)
- **signed** (*bool*)
- **vector_count** (*int | None*)
- **vector_size** (*int | None*)

operands

signed

variable

variable_offset

floating_point

rounding_mode: str | None

vector_count

vector_size

likes(*other*)

matches(*other*)

has_atom(*atom*, *identity=True*)

replace(*old_expr*, *new_expr*)

Return type
tuple[bool, *BinaryOp*]

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

property verbose_op

copy()

Return type
BinaryOp

deep_copy(*manager*)

Return type
BinaryOp

class angr.ailment.**Block**(*addr*, *original_size*, *statements=None*, *idx=None*)

Bases: object

Describes an AIL block.

__str__ should be fast because Phoenix uses networkx graph filters, and graph filters may print str(block) in some cases, e.g., https://github.com/networkx/networkx/blob/861718f8aadeed4f742a348f0c437396cacdf180/networkx/classes/coreviews.py#L305

Parameters

- **addr** (*int*)
- **statements** (*list[Statement] | None*)
- **idx** (*int | None*)

__init__(*addr*, *original_size*, *statements=None*, *idx=None*)

Parameters

- **addr** (*int*)
- **statements** (*list[Statement] | None*)
- **idx** (*int | None*)

addr

original_size

statements: **list[*Statement*]**

idx

copy(*statements=None*)

deep_copy(*manager*)

Parameters
manager (Manager)

property sort_key: **tuple[int, int, int]**

dbg_repr(*indent=0*)

pp()

Return type
None

likes(*other*)

clear_hash()

class angr.ailment.**Const**(*idx*, *variable*, *value*, *bits*, ***kwargs*)

Bases: *Atom*

Parameters

- **idx** (*int*)
- **value** (*int | float*)
- **bits** (*int*)

__init__(*idx*, *variable*, *value*, *bits*, ***kwargs*)

Parameters

- **idx** (*int*)
- **value** (*int* | *float*)
- **bits** (*int*)

value

property value_int: **int**

property value_float: **float**

likes(*other*)

matches(*other*)

property sign_bit

copy()

Return type
Const

deep_copy(*manager*)

Return type
Const

property is_int: **bool**

class angr.ailment.**Expression**(*idx*, *depth*, ***kwargs*)

Bases: *TaggedObject*, ABC

The base class of all AIL expressions.

Parameters
idx (*int*)

bits: **int**

__init__(*idx*, *depth*, ***kwargs*)

Parameters
idx (*int*)

depth

property size

has_atom(*atom*, *identity=True*)

abstractmethod likes(*other*)

abstractmethod matches(*other*)

replace(*old_expr*, *new_expr*)

Return type
tuple[bool, Self]

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

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**(*name=None*, *arch=None*)

Bases: object

Parameters
name (*str | None*)

__init__(*name=None*, *arch=None*)

Parameters
name (*str | None*)

next_atom()

reset()

property ins_addr: int | None

class angr.ailment.**PCodeIRSBConverter**(*irsb*, *manager*)

Bases: *Converter*

Converts a p-code IRSB to an AIL block

Parameters

- **irsb** (IRSB)
- **manager** (Manager)

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**(*idx*, *variable*, *reg_offset*, *bits*, ***kwargs*)

Bases: *Atom*

Parameters

- **idx** (*int*)
- **reg_offset** (*int*)
- **bits** (*int*)

__init__(*idx*, *variable*, *reg_offset*, *bits*, ***kwargs*)

Parameters

- **idx** (*int*)
- **reg_offset** (*int*)
- **bits** (*int*)

reg_offset

likes(*other*)

matches(*other*)

copy()

Return type
Register

deep_copy(*manager*)

Return type
Register

class angr.ailment.**Statement**(*idx*, ***kwargs*)

Bases: *TaggedObject*, ABC

The base class of all AIL statements.

Parameters

- **idx** (*int*)
- **kwargs** (*Unpack[TagDict]*)

abstractmethod replace(*old_expr*, *new_expr*)

Return type
tuple[bool, Self]

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

eq(*expr0*, *expr1*)

abstractmethod likes(*other*)

Return type
bool

abstractmethod matches(*other*)

Return type
bool

abstract property depth: int

class angr.ailment.**Tmp**(*idx*, *variable*, *tmp_idx*, *bits*, ***kwargs*)

Bases: *Atom*

Parameters

- **idx** (*int*)
- **tmp_idx** (*int*)
- **bits** (*int*)

__init__(*idx*, *variable*, *tmp_idx*, *bits*, ***kwargs*)

Parameters

- **idx** (*int*)
- **tmp_idx** (*int*)

tmp_idx

likes(*other*)

matches(*other*)

copy()

Return type
Tmp

deep_copy(*manager*)

Return type
Tmp

class angr.ailment.**UnaryOp**(*idx*, *op*, *operand*, *variable=None*, *variable_offset=None*, *bits=None*, ***kwargs*)

Bases: *Op*

Parameters

- **idx** (*int*)
- **op** (*str*)
- **operand** (Expression)
- **variable_offset** (*int | None*)
- **bits** (*int*)

__init__(*idx*, *op*, *operand*, *variable=None*, *variable_offset=None*, *bits=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **op** (*str*)
- **operand** (Expression)
- **variable_offset** (*int | None*)

operand

variable

variable_offset

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

property operands

copy()

Return type
UnaryOp

deep_copy(*manager*)

Return type
UnaryOp

has_atom(*atom*, *identity=True*)

class angr.ailment.**VEXIRSBConverter**

Bases: *Converter*

static convert(*irsb*, *manager*)

Parameters

- **irsb**
- **manager**

Returns

exception angr.ailment.converter_common.**SkipConversionNotice**

Bases: Exception

class angr.ailment.converter_common.**Converter**

Bases: object

static convert(*irsb*, *manager*)

class angr.ailment.converter_pcode.**PCodeIRSBConverter**(*irsb*, *manager*)

Bases: *Converter*

Converts a p-code IRSB to an AIL block

Parameters

- **irsb** (IRSB)
- **manager** (Manager)

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.converter_vex.**VEXExprConverter**

Bases: *Converter*

static simop_from_vexop(*vex_op*)

static generic_name_from_vex_op(*vex_op*)

static convert(*expr*, *manager*)

Parameters

expr

Returns

static convert_list(*exprs*, *manager*)

static register(*offset*, *bits*, *manager*)

static tmp(*tmp_idx*, *bits*, *manager*)

static RdTmp(*expr*, *manager*)

static Get(*expr*, *manager*)

static Load(*expr*, *manager*)

static Unop(*expr*, *manager*)

static Binop(*expr*, *manager*)

static Triop(*expr*, *manager*)

static Const(*expr*, *manager*)

static **const_n**(*expr*, *manager*)

static **ITE**(*expr*, *manager*)

static **CCall**(*expr*, *manager*)

Parameters
expr (*CCall*)

class angr.ailment.converter_vex.**VEXStmtConverter**

Bases: *Converter*

static **convert**(*idx*, *stmt*, *manager*)

Parameters

- **idx**
- **stmt**
- **manager**

Returns

static **WrTmp**(*idx*, *stmt*, *manager*)

static **Put**(*idx*, *stmt*, *manager*)

static **Store**(*idx*, *stmt*, *manager*)

static **Exit**(*idx*, *stmt*, *manager*)

static **LoadG**(*idx*, *stmt*, *manager*)

Parameters
stmt (*LoadG*)

static **StoreG**(*idx*, *stmt*, *manager*)

Parameters
stmt (*StoreG*)

static **CAS**(*idx*, *stmt*, *manager*)

Parameters
stmt (*CAS*)

static **Dirty**(*idx*, *stmt*, *manager*)

Parameters
stmt (*Dirty*)

class angr.ailment.converter_vex.**VEXIRSBConverter**

Bases: *Converter*

static **convert**(*irsb*, *manager*)

Parameters

- **irsb**
- **manager**

Returns

class angr.ailment.expression.**Expression**(*idx*, *depth*, ***kwargs*)

Bases: *TaggedObject*, ABC

The base class of all AIL expressions.

Parameters
idx (*int*)

bits: int

__init__(*idx*, *depth*, ***kwargs*)

Parameters
idx (*int*)

depth

property size

has_atom(*atom*, *identity=True*)

abstractmethod likes(*other*)

abstractmethod matches(*other*)

replace(*old_expr*, *new_expr*)

Return type
tuple[bool, Self]

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

class angr.ailment.expression.**Atom**(*idx*, *variable=None*, *variable_offset=0*, ***kwargs*)

Bases: *Expression*

Parameters
idx (*int*)

__init__(*idx*, *variable=None*, *variable_offset=0*, ***kwargs*)

Parameters
idx (*int*)

variable

variable_offset

class angr.ailment.expression.**Const**(*idx*, *variable*, *value*, *bits*, ***kwargs*)

Bases: *Atom*

Parameters

- **idx** (*int*)
- **value** (*int | float*)
- **bits** (*int*)

__init__(*idx*, *variable*, *value*, *bits*, ***kwargs*)

Parameters

- **idx** (*int*)
- **value** (*int* | *float*)
- **bits** (*int*)

value

property value_int: **int**

property value_float: **float**

likes(*other*)

matches(*other*)

property sign_bit

copy()

Return type
Const

deep_copy(*manager*)

Return type
Const

property is_int: **bool**

class angr.ailment.expression.**Tmp**(*idx*, *variable*, *tmp_idx*, *bits*, ***kwargs*)

Bases: *Atom*

Parameters

- **idx** (*int*)
- **tmp_idx** (*int*)
- **bits** (*int*)

__init__(*idx*, *variable*, *tmp_idx*, *bits*, ***kwargs*)

Parameters

- **idx** (*int*)
- **tmp_idx** (*int*)

tmp_idx

likes(*other*)

matches(*other*)

copy()

Return type
Tmp

deep_copy(*manager*)

Return type
Tmp

class angr.ailment.expression.**Register**(*idx*, *variable*, *reg_offset*, *bits*, ***kwargs*)

Bases: *Atom*

Parameters

- **idx** (*int*)
- **reg_offset** (*int*)
- **bits** (*int*)

__init__(*idx*, *variable*, *reg_offset*, *bits*, ***kwargs*)

Parameters

- **idx** (*int*)
- **reg_offset** (*int*)
- **bits** (*int*)

reg_offset

likes(*other*)

matches(*other*)

copy()

Return type
Register

deep_copy(*manager*)

Return type
Register

class angr.ailment.expression.**ComboRegister**(*idx*, *variable*, *registers*, ***kwargs*)

Bases: *Atom*

Parameters
registers (*list[Register | VirtualVariable]*)

__init__(*idx*, *variable*, *registers*, ***kwargs*)

Parameters
registers (*list[Register | VirtualVariable]*)

property size

likes(*other*)

matches(*other*)

copy()

Return type
ComboRegister

variable

variable_offset

class angr.ailment.expression.**VirtualVariableCategory**(**values*)

Bases: IntEnum

REGISTER = 0

STACK = 1

MEMORY = 2

PARAMETER = 3

TMP = 4

COMBO_REGISTER = 5

UNKNOWN = 6

class angr.ailment.expression.**VirtualVariable**(*idx*, *varid*, *bits*, *category*, *oident=None*, ***kwargs*)

Bases: *Atom*

Parameters

- **varid** (*int*)
- **bits** (*int*)
- **category** (VirtualVariableCategory)
- **oident** (*int | str | tuple | None*)

__init__(*idx*, *varid*, *bits*, *category*, *oident=None*, ***kwargs*)

Parameters

- **varid** (*int*)
- **category** (VirtualVariableCategory)
- **oident** (*int | str | tuple | None*)

varid

category

oident

reg_vvars

property was_reg: bool

property was_stack: bool

property was_parameter: bool

property was_tmp: bool

property was_combo_reg: bool

property reg_offset: int

property reg_offsets: tuple

property stack_offset: int

property tmp_idx: int | None

property parameter_category: *VirtualVariableCategory* | None

property parameter_reg_offset: int | None

property parameter_stack_offset: int | None

likes(*other*)

matches(*other*)

copy()

Return type
VirtualVariable

deep_copy(*manager*)

Return type
VirtualVariable

class angr.ailment.expression.**Phi**(*idx*, *bits*, *src_and_vvars*, ***kwargs*)

Bases: *Atom*

Parameters

- **bits** (*int*)
- **src_and_vvars** (*list[tuple[tuple[int, int | None], VirtualVariable | None]]*)

__init__(*idx*, *bits*, *src_and_vvars*, ***kwargs*)

Parameters
src_and_vvars (*list[tuple[tuple[int, int | None], VirtualVariable | None]]*)

src_and_vvars

property op: str

property verbose_op: str

likes(*other*)

Return type
bool

matches(*other*)

Return type
bool

copy()

Return type
Phi

deep_copy(*manager*)

Return type
Phi

replace(*old_expr*, *new_expr*)

class angr.ailment.expression.**Op**(*idx*, *depth*, *op*, ***kwargs*)

Bases: *Expression*

Parameters
op (*str*)

__init__(*idx*, *depth*, *op*, ***kwargs*)

Parameters
op (*str*)

op

property verbose_op

class angr.ailment.expression.**UnaryOp**(*idx*, *op*, *operand*, *variable=None*, *variable_offset=None*, *bits=None*, ***kwargs*)

Bases: *Op*

Parameters

- **idx** (*int*)
- **op** (*str*)
- **operand** (Expression)
- **variable_offset** (*int | None*)
- **bits** (*int*)

__init__(*idx*, *op*, *operand*, *variable=None*, *variable_offset=None*, *bits=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **op** (*str*)
- **operand** (Expression)
- **variable_offset** (*int | None*)

operand

variable

variable_offset

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

property operands

copy()

Return type
UnaryOp

deep_copy(*manager*)

Return type
UnaryOp

has_atom(*atom*, *identity=True*)

class angr.ailment.expression.**ConvertType**(**values*)

Bases: Enum

TYPE_INT = 0

TYPE_FP = 1

class angr.ailment.expression.**Convert**(*idx*, *from_bits*, *to_bits*, *is_signed*, *operand*, *from_type=ConvertType.TYPE_INT*, *to_type=ConvertType.TYPE_INT*, *rounding_mode=None*, ***kwargs*)

Bases: *UnaryOp*

Parameters

- **idx** (*int*)
- **from_bits** (*int*)
- **to_bits** (*int*)
- **is_signed** (*bool*)
- **operand** (Expression)
- **from_type** (ConvertType)
- **to_type** (ConvertType)

TYPE_INT = 0

TYPE_FP = 1

__init__(*idx*, *from_bits*, *to_bits*, *is_signed*, *operand*, *from_type=ConvertType.TYPE_INT*, *to_type=ConvertType.TYPE_INT*, *rounding_mode=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **from_bits** (*int*)
- **to_bits** (*int*)
- **is_signed** (*bool*)
- **operand** (Expression)
- **from_type** (ConvertType)
- **to_type** (ConvertType)

from_bits

to_bits

is_signed

from_type

to_type

rounding_mode

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

copy()

Return type
Convert

deep_copy(*manager*)

Return type
Convert

class angr.ailment.expression.**Reinterpret**(*idx*, *from_bits*, *from_type*, *to_bits*, *to_type*, *operand*, ***kwargs*)

Bases: *UnaryOp*

Parameters

- **from_bits** (*int*)
- **from_type** (*str*)
- **to_bits** (*int*)
- **to_type** (*str*)

__init__(*idx*, *from_bits*, *from_type*, *to_bits*, *to_type*, *operand*, ***kwargs*)

Parameters

- **from_bits** (*int*)
- **from_type** (*str*)
- **to_bits** (*int*)
- **to_type** (*str*)

from_bits

from_type

to_bits

to_type

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

copy()

Return type
Reinterpret

deep_copy(*manager*)

Return type
Reinterpret

class angr.ailment.expression.**BinaryOp**(*idx*, *op*, *operands*, *signed=False*, *, *variable=None*, *variable_offset=None*, *bits=None*, *floating_point=False*, *rounding_mode=None*, *vector_count=None*, *vector_size=None*, ***kwargs*)

Bases: *Op*

Parameters

- **idx** (*int*)
- **op** (*str*)
- **operands** (*Sequence[Expression]*)
- **signed** (*bool*)
- **bits** (*int*)
- **rounding_mode** (*str | None*)
- **vector_count** (*int | None*)
- **vector_size** (*int | None*)

OPSTR_MAP = {'Add': '+', 'AddF': '+', 'AddV': '+', 'And': '&', 'Carry': 'CARRY', 'CmpEQ': '==', 'CmpF': 'CmpF', 'CmpGE': '>=', 'CmpGE (signed)': '>=s', 'CmpGT': '>', 'CmpGT (signed)': '>s', 'CmpLE': '<=', 'CmpLE (signed)': '<=s', 'CmpLT': '<', 'CmpLT (signed)': '<s', 'CmpNE': '!=', 'Concat': 'CONCAT', 'Div': '/', 'DivF': '/', 'LogicalAnd': '&&', 'LogicalOr': '||', 'Mod': '%', 'Mul': '*', 'MulF': '*', 'MulV': '*', 'Or': '|', 'Rol': 'ROL', 'Ror': 'ROR', 'SBorrow': 'SBORROW', 'SCarry': 'SCARRY', 'Sar': '>>a', 'Shl': '<<', 'Shr': '>>', 'Sub': '-', 'SubF': '-', 'Xor': '^'}

COMPARISON_NEGATION = {'CmpEQ': 'CmpNE', 'CmpGE': 'CmpLT', 'CmpGT': 'CmpLE', 'CmpLE': 'CmpGT', 'CmpLT': 'CmpGE', 'CmpNE': 'CmpEQ'}

__init__(*idx*, *op*, *operands*, *signed=False*, *, *variable=None*, *variable_offset=None*, *bits=None*, *floating_point=False*, *rounding_mode=None*, *vector_count=None*, *vector_size=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **op** (*str*)
- **operands** (*Sequence[Expression]*)
- **signed** (*bool*)
- **vector_count** (*int | None*)

- **vector_size** (*int* | *None*)

operands

signed

variable

variable_offset

floating_point

rounding_mode: str | None

vector_count

vector_size

likes(*other*)

matches(*other*)

has_atom(*atom*, *identity=True*)

replace(*old_expr*, *new_expr*)

Return type
tuple[bool, *BinaryOp*]

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

property verbose_op

copy()

Return type
BinaryOp

deep_copy(*manager*)

Return type
BinaryOp

class angr.ailment.expression.**Load**(*idx*, *addr*, *size*, *endness*, *variable=None*, *variable_offset=None*, *guard=None*, *alt=None*, ***kwargs*)

Bases: *Expression*

Parameters

- **idx** (*int*)
- **addr** (Expression)
- **size** (*int*)
- **endness** (*str*)

__init__(*idx*, *addr*, *size*, *endness*, *variable=None*, *variable_offset=None*, *guard=None*, *alt=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **addr** (Expression)
- **size** (*int*)
- **endness** (*str*)

addr

endness

guard

alt

variable

variable_offset

has_atom(*atom*, *identity=True*)

replace(*old_expr*, *new_expr*)

likes(*other*)

matches(*other*)

copy()

Return type
Load

deep_copy(*manager*)

Return type
Load

class angr.ailment.expression.**ITE**(*idx*, *cond*, *iffalse*, *iftrue*, *variable=None*, *variable_offset=None*, ***kwargs*)

Bases: *Expression*

Parameters

- **idx** (*int*)
- **cond** (Expression)
- **iffalse** (Expression)
- **iftrue** (Expression)

__init__(*idx*, *cond*, *iffalse*, *iftrue*, *variable=None*, *variable_offset=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **cond** (Expression)

- **iffalse** (Expression)
- **iftrue** (Expression)

cond

iffalse

iftrue

variable

variable_offset

likes(*other*)

matches(*other*)

has_atom(*atom*, *identity=True*)

replace(*old_expr*, *new_expr*)

copy()

Return type
ITE

deep_copy(*manager*)

Return type
ITE

class angr.ailment.expression.**DirtyExpression**(*idx*, *callee*, *operands*, *, *guard=None*, *mfx=None*, *maddr=None*, *msize=None*, *bits*, **kwargs*)

Bases: *Expression*

Parameters

- **callee** (*str*)
- **operands** (*list[Expression]*)
- **guard** (Expression | *None*)
- **mfx** (*str* | *None*)
- **maddr** (Expression | *None*)
- **msize** (*int* | *None*)
- **bits** (*int*)

__init__(*idx*, *callee*, *operands*, *, *guard=None*, *mfx=None*, *maddr=None*, *msize=None*, *bits*, **kwargs*)

Parameters

- **callee** (*str*)
- **operands** (*list[Expression]*)
- **guard** (Expression | *None*)
- **mfx** (*str* | *None*)
- **maddr** (Expression | *None*)

- **msize** (*int | None*)
- **bits** (*int*)

callee

guard

operands

mfx

maddr

msize

property op: str

property verbose_op: str

likes(*other*)

matches(*other*)

copy()

Return type
DirtyExpression

deep_copy(*manager*)

Return type
DirtyExpression

replace(*old_expr*, *new_expr*)

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

class angr.ailment.expression.**VEXCCallExpression**(*idx*, *callee*, *operands*, *bits*, ***kwargs*)

Bases: *Expression*

Parameters

- **idx** (*int*)
- **callee** (*str*)
- **operands** (*tuple[Expression, ...]*)
- **bits** (*int*)

__init__(*idx*, *callee*, *operands*, *bits*, ***kwargs*)

Parameters

- **idx** (*int*)
- **callee** (*str*)
- **operands** (*tuple[Expression, ...]*)
- **bits** (*int*)

callee

operands

property op: str

property verbose_op: str

likes(*other*)

matches(*other*)

copy()

Return type
VEXCCallExpression

deep_copy(*manager*)

Return type
VEXCCallExpression

replace(*old_expr*, *new_expr*)

class angr.ailment.expression.**MultiStatementExpression**(*idx*, *stmts*, *expr*, ***kwargs*)

Bases: *Expression*

For representing comma-separated statements and expression in C.

Parameters

- **idx** (*int*)
- **stmts** (*list[Statement]*)
- **expr** (Expression)

__init__(*idx*, *stmts*, *expr*, ***kwargs*)

Parameters

- **idx** (*int*)
- **stmts** (*list[Statement]*)
- **expr** (Expression)

stmts

expr

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

copy()

Return type
MultiStatementExpression

deep_copy(*manager*)

Return type
MultiStatementExpression

class angr.ailment.expression.**BasePointerOffset**(*idx*, *bits*, *base*, *offset*, *variable=None*, *variable_offset=None*, ***kwargs*)

Bases: *Expression*

Parameters

- **idx** (*int*)
- **bits** (*int*)
- **base** (*Expression* | *str*)
- **offset** (*int*)

__init__(*idx*, *bits*, *base*, *offset*, *variable=None*, *variable_offset=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **bits** (*int*)
- **base** (*Expression* | *str*)
- **offset** (*int*)

base

offset

variable

variable_offset

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

copy()

Return type
BasePointerOffset

deep_copy(*manager*)

Return type
BasePointerOffset

class angr.ailment.expression.**StackBaseOffset**(*idx*, *bits*, *offset*, ***kwargs*)

Bases: *BasePointerOffset*

Parameters

- **idx** (*int*)
- **bits** (*int*)
- **offset** (*int*)

__init__(*idx*, *bits*, *offset*, ***kwargs*)

Parameters

- **idx** (*int*)
- **bits** (*int*)
- **offset** (*int*)

copy()

Return type
StackBaseOffset

deep_copy(*manager*)

Return type
StackBaseOffset

angr.ailment.expression.**negate**(*expr*, *manager*)

Return type
Expression

Parameters

- **expr** (Expression)
- **manager** (Manager)

class angr.ailment.expression.**Extract**(*idx*, *bits*, *base*, *offset*, *endness*, ***kwargs*)

Bases: *Expression*

Parameters

- **idx** (*int*)
- **bits** (*int*)
- **base** (Expression)
- **offset** (Expression)
- **endness** (*str*)

__init__(*idx*, *bits*, *base*, *offset*, *endness*, ***kwargs*)

Parameters

- **idx** (*int*)
- **bits** (*int*)
- **base** (Expression)
- **offset** (Expression)
- **endness** (*str*)

base

offset

endness

is_lsb_extract()

Return type
bool

copy()

Return type
Extract

deep_copy(*manager*)

Return type
Extract

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

class angr.ailment.expression.**Insert**(*idx*, *base*, *offset*, *value*, *endness*, ***kwargs*)

Bases: *Expression*

Parameters

- **idx** (*int*)
- **base** (Expression)
- **offset** (Expression)
- **value** (Expression)
- **endness** (*str*)

__init__(*idx*, *base*, *offset*, *value*, *endness*, ***kwargs*)

Parameters

- **idx** (*int*)
- **base** (Expression)
- **offset** (Expression)
- **value** (Expression)
- **endness** (*str*)

base

offset

value

endness

is_lsb_overwrite()

Return type
bool

copy()

Return type
Insert

deep_copy(*manager*)

Return type
Insert

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

class angr.ailment.expression.**Call**(*idx*, *target*, *calling_convention=None*, *prototype=None*, *args=None*, *bits=None*, ***kwargs*)

Bases: *Expression*

Call expression. Represents a function call that produces a value.

When used as a standalone statement (not part of an assignment), wrap it in a SideEffectStatement.

Parameters

- **idx** (*int*)
- **target** (Expression | *str*)
- **calling_convention** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)
- **args** (*Sequence[*Expression*]* | *None*)
- **bits** (*int*)

__init__(*idx*, *target*, *calling_convention=None*, *prototype=None*, *args=None*, *bits=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **target** (Expression | *str*)
- **calling_convention** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)
- **args** (*Sequence[*Expression*]* | *None*)
- **bits** (*int* | *None*)

target

calling_convention

prototype

args

likes(*other*)

matches(*other*)

property verbose_op: str

property op: str

has_atom(*atom*, *identity=True*)

replace(*old_expr*, *new_expr*)

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

copy()

deep_copy(*manager*)

class angr.ailment.expression.**StringLiteral**(*idx*, *data*, *bits*, ***kwargs*)

Bases: *Expression*

Parameters

bits (*int*)

__init__(*idx*, *data*, *bits*, ***kwargs*)

data

property size

likes(*other*)

copy()

deep_copy(*manager*)

replace(*old_expr*, *new_expr*)

Return type

tuple[bool, Self]

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

matches(*other*)

class angr.ailment.expression.**Struct**(*idx*, *name*, *fields*, *field_offsets*, *bits*, ***kwargs*)

Bases: *Expression*

Parameters

bits (*int*)

__init__(*idx*, *name*, *fields*, *field_offsets*, *bits*, ***kwargs*)

name

fields

field_offsets

field_names

get_field(*name*)

property size

likes(*other*)

copy()

deep_copy(*manager*)

replace(*old_expr*, *new_expr*)

matches(*other*)

class angr.ailment.expression.**RustEnum**(*idx*, *name*, *fields*, *bits*, ***kwargs*)

Bases: *Expression*

Parameters
bits (*int*)

__init__(*idx*, *name*, *fields*, *bits*, ***kwargs*)

bits: **int**

property size

likes(*other*)

copy()

deep_copy(*manager*)

replace(*old_expr*, *new_expr*)

matches(*other*)

depth

class angr.ailment.expression.**Array**(*idx*, *elements*, *bits*, ***kwargs*)

Bases: *Expression*

Parameters
bits (*int*)

__init__(*idx*, *elements*, *bits*, ***kwargs*)

bits: **int**

property size

property length

likes(*other*)

copy()

deep_copy(*manager*)

replace(*old_expr*, *new_expr*)

matches(*other*)

depth

class angr.ailment.expression.**Let**(*idx*, *variant*, *defs*, *src*, ***kwargs*)

Bases: *Op*

__init__(*idx*, *variant*, *defs*, *src*, ***kwargs*)

variant

defs

src

likes(*other*)

matches(*other*)

copy()

deep_copy(*manager*)

class angr.ailment.expression.**Macro**(*idx*, *name*, *delimiter='()'*, *returnty=None*, ***kwargs*)

Bases: *Call*, ABC

Parameters

name (*str*)

__init__(*idx*, *name*, *delimiter='()'*, *returnty=None*, ***kwargs*)

Parameters

name (*str*)

name

delimiter

returnty

class angr.ailment.expression.**FunctionLikeMacro**(*idx*, *name*, *args*, *bits=None*, *delimiter='()'*, *returnty=None*, ***kwargs*)

Bases: *Macro*

Parameters

- **name** (*str*)
- **bits** (*int*)

__init__(*idx*, *name*, *args*, *bits=None*, *delimiter='()'*, *returnty=None*, ***kwargs*)

Parameters

name (*str*)

property size: int

likes(*other*)

matches(*other*)

copy()

deep_copy(*manager*)

property verbose_op

property op

class angr.ailment.statement.**Statement**(*idx*, ***kwargs*)

Bases: *TaggedObject*, ABC

The base class of all AIL statements.

Parameters

- **idx** (*int*)
- **kwargs** (*Unpack[TagDict]*)

abstractmethod replace(*old_expr*, *new_expr*)

Return type
tuple[bool, Self]

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

eq(*expr0*, *expr1*)

abstractmethod likes(*other*)

Return type
bool

abstractmethod matches(*other*)

Return type
bool

abstract property depth: int

class angr.ailment.statement.**Assignment**(*idx*, *dst*, *src*, ***kwargs*)

Bases: *Statement*

Assignment statement: expr_a = expr_b

Parameters

- **idx** (*int*)
- **dst** (Atom)
- **src** (Expression)

__init__(*idx*, *dst*, *src*, ***kwargs*)

Parameters

- **idx** (*int*)
- **dst** (Atom)
- **src** (Expression)

dst

src

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

property depth

copy()

Return type
Assignment

deep_copy(*manager*)

Return type
Assignment

class angr.ailment.statement.**WeakAssignment**(*idx*, *dst*, *src*, ***kwargs*)

Bases: *Statement*

An assignment statement that does not create a new variable at its destination; It should be seen as operator=(&dst, &src) in C++-like syntax.

Parameters

- **idx** (*int*)
- **dst** (Atom)
- **src** (Expression)

__init__(*idx*, *dst*, *src*, ***kwargs*)

Parameters

- **idx** (*int*)
- **dst** (Atom)
- **src** (Expression)

dst

src

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

property depth

copy()

Return type
WeakAssignment

deep_copy(*manager*)

Return type
WeakAssignment

class angr.ailment.statement.**Store**(*idx*, *addr*, *data*, *size*, *endness*, *guard=None*, *variable=None*, *offset=None*, ***kwargs*)

Bases: *Statement*

Store statement: `*addr = data`

Parameters

- **idx** (*int*)
- **addr** (Expression)
- **data** (Expression)
- **size** (*int*)
- **endness** (*str*)
- **guard** (Expression | *None*)

__init__(*idx*, *addr*, *data*, *size*, *endness*, *guard=None*, *variable=None*, *offset=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **addr** (Expression)
- **data** (Expression)
- **size** (*int*)
- **endness** (*str*)
- **guard** (Expression | *None*)

addr

data

size

endness

variable

guard

offset

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

property depth

copy()

Return type
Store

deep_copy(*manager*)

Return type
Store

class angr.ailment.statement.**Jump**(*idx*, *target*, *target_idx=None*, ***kwargs*)

Bases: *Statement*

Jump statement: goto target

Parameters

- **idx** (*int*)
- **target** (Expression)
- **target_idx** (*int | None*)

__init__(*idx*, *target*, *target_idx=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **target** (Expression)
- **target_idx** (*int | None*)

target

target_idx

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

property depth

copy()

deep_copy(*manager*)

class angr.ailment.statement.**ConditionalJump**(*idx*, *condition*, *true_target*, *false_target*, *true_target_idx=None*, *false_target_idx=None*, ***kwargs*)

Bases: *Statement*

if (cond) {true_target} else {false_target}

Parameters

- **idx** (*int*)
- **condition** (Expression)
- **true_target** (Expression | *None*)
- **false_target** (Expression | *None*)
- **true_target_idx** (*int* | *None*)
- **false_target_idx** (*int* | *None*)

__init__(*idx*, *condition*, *true_target*, *false_target*, *true_target_idx=None*, *false_target_idx=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **condition** (Expression)
- **true_target** (Expression | *None*)
- **false_target** (Expression | *None*)
- **true_target_idx** (*int* | *None*)
- **false_target_idx** (*int* | *None*)

condition

true_target

false_target

true_target_idx

false_target_idx

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

property depth

copy()

Return type
ConditionalJump

deep_copy(*manager*)

Return type
ConditionalJump

class angr.ailment.statement.**SideEffectStatement**(*idx*, *expr*, *ret_expr=None*, *fp_ret_expr=None*, ***kwargs*)

Bases: *Statement*

A statement wrapping an expression that has side effects (e.g., a function call).

When wrapping a Call expression, ret_expr and fp_ret_expr hold the return value destinations.

Parameters

- **idx** (*int*)
- **expr** (ailment.expression.Call)
- **ret_expr** (Expression | *None*)
- **fp_ret_expr** (Expression | *None*)

__init__(*idx*, *expr*, *ret_expr=None*, *fp_ret_expr=None*, ***kwargs*)

Parameters

- **idx** (*int*)
- **expr** (Call)
- **ret_expr** (Expression | *None*)
- **fp_ret_expr** (Expression | *None*)

expr

ret_expr

fp_ret_expr

property bits

property size

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

property depth

copy()

Return type
SideEffectStatement

deep_copy(*manager*)

Return type
SideEffectStatement

class angr.ailment.statement.**Return**(*idx*, *ret_exprs*, ***kwargs*)

Bases: *Statement*

Return statement: (return expr_a), (return)

Parameters

- **idx** (*int*)
- **ret_exprs** (*Iterable[Expression]*)

__init__(*idx*, *ret_exprs*, ***kwargs*)

Parameters

- **idx** (*int*)
- **ret_exprs** (*Iterable[Expression]*)

ret_exprs

likes(*other*)

matches(*other*)

replace(*old_expr*, *new_expr*)

property depth

copy()

deep_copy(*manager*)

class angr.ailment.statement.**CAS**(*idx*, *addr*, *data_lo*, *data_hi*, *expd_lo*, *expd_hi*, *old_lo*, *old_hi*, *endness*, ***kwargs*)

Bases: *Statement*

Atomic compare-and-swap.

`*_lo` and `*_hi` are used to represent the low and high parts of a 128-bit CAS operation; `*_hi` is None if the CAS operation works on values that are less than or equal to 64 bits.

addr: The address to be compared and swapped. data: The value to be written if the comparison is successful. expd: The expected value to be compared against. old: The value that is currently stored at addr before compare-and-swap; it will be returned after compare-and-swap.

Parameters

- **idx** (*int*)
- **addr** (*Expression*)
- **data_lo** (*Expression*)
- **data_hi** (*Expression | None*)
- **expd_lo** (*Expression*)
- **expd_hi** (*Expression | None*)
- **old_lo** (*Atom*)
- **old_hi** (*Atom | None*)
- **endness** (*str*)

__init__(*idx*, *addr*, *data_lo*, *data_hi*, *expd_lo*, *expd_hi*, *old_lo*, *old_hi*, *endness*, ***kwargs*)

Parameters

- **idx** (*int*)
- **addr** (Expression)
- **data_lo** (Expression)
- **data_hi** (Expression | *None*)
- **expd_lo** (Expression)
- **expd_hi** (Expression | *None*)
- **old_lo** (Atom)
- **old_hi** (Atom | *None*)
- **endness** (*str*)

addr

data_lo

data_hi

expd_lo

expd_hi

old_lo

old_hi

endness

replace(*old_expr*, *new_expr*)

Return type
tuple[bool, *CAS*]

Parameters

- **old_expr** (Expression)
- **new_expr** (Expression)

property depth: int

copy()

Return type
CAS

deep_copy(*manager*)

Return type
CAS

likes(*other*)

Return type
bool

matches(*other*)

Return type
bool

property bits: int

property size: int

class angr.ailment.statement.**DirtyStatement**(*idx*, *dirty*, ***kwargs*)

Bases: *Statement*

Wrapper around the original statement, which is usually not convertible (temporarily).

Parameters

- **idx** (*int*)
- **dirty** (DirtyExpression)

__init__(*idx*, *dirty*, ***kwargs*)

Parameters

- **idx** (*int*)
- **dirty** (DirtyExpression)

dirty

replace(*old_expr*, *new_expr*)

property depth: int

copy()

Return type
DirtyStatement

deep_copy(*manager*)

Return type
DirtyStatement

likes(*other*)

matches(*other*)

class angr.ailment.statement.**Label**(*idx*, *name*, ***kwargs*)

Bases: *Statement*

A dummy statement that indicates a label with a name.

Parameters

- **idx** (*int*)
- **name** (*str*)

__init__(*idx*, *name*, ***kwargs*)

Parameters

- **idx** (*int*)
- **name** (*str*)

name

likes(*other*)

Parameters
other (Label)

replace(*old_expr*, *new_expr*)

property depth: int

matches(*other*)

Parameters
other (Label)

copy()

Return type
Label

deep_copy(*manager*)

Return type
Label

class angr.ailment.block.**Block**(*addr*, *original_size*, *statements=None*, *idx=None*)

Bases: object

Describes an AIL block.

__str__ should be fast because Phoenix uses networkx graph filters, and graph filters may print str(block) in some cases, e.g., https://github.com/networkx/networkx/blob/861718f8aadeed4f742a348f0c437396cacdf180/networkx/classes/coreviews.py#L305

Parameters

- **addr** (*int*)
- **statements** (*list[Statement]*)
- **idx** (*int | None*)

__init__(*addr*, *original_size*, *statements=None*, *idx=None*)

Parameters

- **addr** (*int*)
- **statements** (*list[Statement] | None*)
- **idx** (*int | None*)

addr

original_size

statements: list[*Statement*]

idx

copy(*statements=None*)

deep_copy(*manager*)

Parameters
manager (Manager)

property sort_key: tuple[int, int, int]

dbg_repr(*indent=0*)

pp()

Return type
None

likes(*other*)

clear_hash()

class angr.ailment.manager.**Manager**(*name=None*, *arch=None*)

Bases: object

Parameters
name (*str* | *None*)

__init__(*name=None*, *arch=None*)

Parameters
name (*str* | *None*)

next_atom()

reset()

property ins_addr: int | None

class angr.ailment.tagged_object.**TagDict**

Bases: TypedDict

Typed dict of tags for TaggedObject.

always_propagate: bool

block_idx: int

custom_string: bool

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

reference_values: dict[*SimType*, Any]

reference_variable_offset: int

reference_variable: *SimVariable*

reg_name: str

type: dict[str, *SimType*]

uninitialized: bool

vex_block_addr: int

vex_stmt_idx: int

write_size: int

class angr.ailment.tagged_object.**TaggedObject**(*idx*, ***kwargs*)

Bases: object

A class that takes tags.

Parameters

- **idx** (*int*)
- **kwargs** (*Unpack[TagDict]*)

__init__(*idx*, ***kwargs*)

Parameters

- **idx** (*int*)
- **kwargs** (*Unpack[TagDict]*)

tags: *TagDict*

idx

copy()

Return type
Self

deep_copy(*manager*)

Return type
Self

Parameters
manager (Manager)

type angr.ailment.utils.**GetBitsTypeParams** = *ailment.expression.Expression*

angr.ailment.utils.**get_bits**(*expr*)

Return type
int

Parameters
expr (GetBitsTypeParams)

angr.ailment.utils.**stable_hash**(*t*)

Return type
int

Parameters
t (*tuple*)

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

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

angr, 155
angr.ailment, 1364
angr.ailment.block, 1406
angr.ailment.converter_common, 1373
angr.ailment.converter_pcode, 1373
angr.ailment.converter_vex, 1374
angr.ailment.expression, 1375
angr.ailment.manager, 1407
angr.ailment.statement, 1397
angr.ailment.tagged_object, 1407
angr.ailment.utils, 1408
angr.analyses, 873
angr.analyses.analysis, 928
angr.analyses.backward_slice, 946
angr.analyses.binary_optimizer, 1306
angr.analyses.bindiff, 947
angr.analyses.boyscout, 950
angr.analyses.callee_cleanup_finder, 1307
angr.analyses.calling_convention, 950
angr.analyses.cdg, 1017
angr.analyses.cfg, 956
angr.analyses.cfg.cfb, 972
angr.analyses.cfg.cfg, 973
angr.analyses.cfg.cfg_arch_options, 992
angr.analyses.cfg.cfg_base, 981
angr.analyses.cfg.cfg_emulated, 976
angr.analyses.cfg.cfg_fast, 983
angr.analyses.cfg.cfg_fast_soot, 1015
angr.analyses.cfg.cfg_job_base, 993
angr.analyses.cfg.indirect_jump_resolvers, 1005
angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got, 994
angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast, 994
angr.analyses.cfg.indirect_jump_resolvers.const_resolver, 1003
angr.analyses.cfg.indirect_jump_resolvers.default_resolvers, 998
angr.analyses.cfg.indirect_jump_resolvers.jumptable, 998
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast, 996
angr.analyses.cfg.indirect_jump_resolvers.resolver, 1004
angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt, 997
angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat, 995
angr.analyses.cfg_slice_to_sink, 1238
angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink, 1241
angr.analyses.cfg_slice_to_sink.graph, 1242
angr.analyses.cfg_slice_to_sink.transitions, 1243
angr.analyses.class_identifier, 1292
angr.analyses.code_tagging, 1018
angr.analyses.complete_calling_conventions, 953
angr.analyses.congruency_check, 1305
angr.analyses.data_dep, 1316
angr.analyses.data_dep.data_dependency_analysis, 1311
angr.analyses.data_dep.dep_nodes, 1313
angr.analyses.data_dep.sim_act_location, 1312
angr.analyses.datagraph_meta, 1017
angr.analyses.ddg, 1156
angr.analyses.decompiler, 1039
angr.analyses.decompiler.ail_simplifier, 1052
angr.analyses.decompiler.ailgraph_walker, 1054
angr.analyses.decompiler.block_simplifier, 1054
angr.analyses.decompiler.callsite_maker, 1055
angr.analyses.decompiler.ccall_rewriters, 1055
angr.analyses.decompiler.ccall_rewriters.amd64_ccalls, 1056
angr.analyses.decompiler.ccall_rewriters.rewriter_base, 1055
angr.analyses.decompiler.clinic, 1056
angr.analyses.decompiler.condition_processor, 1061

angr.analyses.decompiler.decompilation_cache, 1064
angr.analyses.decompiler.decompilation_options, 1063
angr.analyses.decompiler.decompiler, 1065
angr.analyses.decompiler.empty_node_remover, 1068
angr.analyses.decompiler.expression_narrower, 1069
angr.analyses.decompiler.graph_region, 1070
angr.analyses.decompiler.jump_target_collector, 1071
angr.analyses.decompiler.jumptable_entry_condition_analyzer, 1071
angr.analyses.decompiler.optimization_passes, 1071
angr.analyses.decompiler.optimization_passes.base_optimization_pass, 1097
angr.analyses.decompiler.optimization_passes.constant_derefs, 1091
angr.analyses.decompiler.optimization_passes.div_simplifier, 1098
angr.analyses.decompiler.optimization_passes.engine_base, 1101
angr.analyses.decompiler.optimization_passes.expr_op_swapper, 1102
angr.analyses.decompiler.optimization_passes.ite_expr_converter, 1098
angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier, 1099
angr.analyses.decompiler.optimization_passes.mod_simplifier, 1101
angr.analyses.decompiler.optimization_passes.optimization_pass, 1092
angr.analyses.decompiler.optimization_passes.ret_addr_simplifier, 1103
angr.analyses.decompiler.optimization_passes.ret_addr_simplifier, 1103
angr.analyses.decompiler.optimization_passes.stack_analyses_simplifier, 1097
angr.analyses.decompiler.optimization_passes.x86_cc_simplifier, 1104
angr.analyses.decompiler.peephole_optimizations, 1104
angr.analyses.decompiler.peephole_optimizations.base, 1104
angr.analyses.decompiler.redundant_label_remover, 1119
angr.analyses.decompiler.region_identifier, 1107
angr.analyses.decompiler.region_simplifiers, 1108
angr.analyses.decompiler.region_simplifiers.cascading_ifs, 1109
angr.analyses.decompiler.region_simplifiers.cascading_ifs, 1109
angr.analyses.decompiler.region_simplifiers.expr_folding, 1109
angr.analyses.decompiler.region_simplifiers.goto, 1114
angr.analyses.decompiler.region_simplifiers.if_, 1114
angr.analyses.decompiler.region_simplifiers.ifelse, 1114
angr.analyses.decompiler.region_simplifiers.loop, 1114
angr.analyses.decompiler.region_simplifiers.node_address_f 1114
angr.analyses.decompiler.region_simplifiers.region_simplif 1114
angr.analyses.decompiler.region_simplifiers.switch_cluster 1115
angr.analyses.decompiler.region_simplifiers.switch_expr_si 1119
angr.analyses.decompiler.region_walker, 1119
angr.analyses.decompiler.sequence_walker, 1119
angr.analyses.decompiler.structured_codegen, 1130
angr.analyses.decompiler.structured_codegen.base, 1127
angr.analyses.decompiler.structured_codegen.c, 1130
angr.analyses.decompiler.structured_codegen.dummy, 1149
angr.analyses.decompiler.structured_codegen.dwarf_import, 1149
angr.analyses.decompiler.structuring, 1032
angr.analyses.decompiler.structuring.dream, 1035
angr.analyses.decompiler.structuring.phoenix, 1037
angr.analyses.decompiler.structuring.recursive_structurer, 1032
angr.analyses.decompiler.structuring.structurer_base, 1036
angr.analyses.decompiler.utils, 1150
angr.analyses.disassembly, 1293
angr.analyses.disassembly_utils, 1297
angr.analyses.dominance_frontier, 1307
angr.analyses.find_objects_static, 1292
angr.analyses.flirt, 1161
angr.analyses.forward_analysis, 931
angr.analyses.forward_analysis.forward_analysis, 935
angr.analyses.forward_analysis.job_info, 937
angr.analyses.forward_analysis.visitors, 937
angr.analyses.forward_analysis.visitors.call_graph,

941
angr.analyses.forward_analysis.visitors.function_graph, 941
angr.analyses.forward_analysis.visitors.graph, 942
angr.analyses.forward_analysis.visitors.loop, 944
angr.analyses.forward_analysis.visitors.single_node_graph, 945
angr.analyses.identifier.identify, 1283
angr.analyses.init_finder, 1307
angr.analyses.loop_analysis, 1284
angr.analyses.loopfinder, 1283
angr.analyses.propagator, 1174
angr.analyses.propagator.engine_base, 1176
angr.analyses.propagator.engine_vex, 1177
angr.analyses.propagator.propagator, 1177
angr.analyses.propagator.top_checker_mixin, 1178
angr.analyses.propagator.values, 1175
angr.analyses.propagator.vex_vars, 1175
angr.analyses.proximity_graph, 1309
angr.analyses.reaching_definitions, 1178
angr.analyses.reaching_definitions.call_trace, 1212
angr.analyses.reaching_definitions.dep_graph, 1217
angr.analyses.reaching_definitions.engine_ail, 1238
angr.analyses.reaching_definitions.engine_vex, 1213
angr.analyses.reaching_definitions.function_handler, 1221
angr.analyses.reaching_definitions.heap_allocator, 1220
angr.analyses.reaching_definitions.rd_state, 1229
angr.analyses.reaching_definitions.reaching_definitions, 1213
angr.analyses.reaching_definitions.subject, 1237
angr.analyses.reassembler, 1297
angr.analyses.soot_class_hierarchy, 955
angr.analyses.stack_pointer_tracker, 1243
angr.analyses.static_hooker, 1305
angr.analyses.typehoon, 1281
angr.analyses.typehoon.simple_solver, 1259
angr.analyses.typehoon.translator, 1265
angr.analyses.typehoon.typeconsts, 1274
angr.analyses.typehoon.typehoon, 1272
angr.analyses.typehoon.typevars, 1266
angr.analyses.variable_recovery, 1257
angr.analyses.variable_recovery.annotations, 1246
angr.analyses.variable_recovery.engine_ail, 1255
angr.analyses.variable_recovery.engine_base, 1256
angr.analyses.variable_recovery.engine_vex, 1256
angr.analyses.variable_recovery.irsb_scanner, 1257
angr.analyses.variable_recovery.variable_recovery, 1254
angr.analyses.variable_recovery.variable_recovery_base, 1247
angr.analyses.variable_recovery.variable_recovery_fast, 1251
angr.analyses.veritesting, 1284
angr.analyses.vfg, 1285
angr.analyses.vsa_ddg, 1290
angr.analyses.vtable, 1291
angr.analyses.xrefs, 1308
angr.angrdb, 1018
angr.angrdb.db, 1020
angr.angrdb.models, 1021
angr.angrdb.serializers, 1025
angr.angrdb.serializers.cfg_model, 1026
angr.angrdb.serializers.comments, 1027
angr.angrdb.serializers.funcs, 1027
angr.angrdb.serializers.kb, 1027
angr.angrdb.serializers.labels, 1028
angr.angrdb.serializers.loader, 1028
angr.angrdb.serializers.structured_code, 1031
angr.angrdb.serializers.variables, 1030
angr.angrdb.serializers.xrefs, 1030
angr.annocfg, 1321
angr.blade, 1319
angr.block, 232
angr.callable, 710
angr.calling_conventions, 658
angr.code_location, 865
angr.codenode, 1322
angr.concretization_strategies, 452
angr.concretization_strategies.any, 543
angr.concretization_strategies.controlled_data, 543
angr.concretization_strategies.eval, 541
angr.concretization_strategies.max, 542
angr.concretization_strategies.nonzero, 543
angr.concretization_strategies.nonzero_range, 542
angr.concretization_strategies.norepeats, 541
angr.concretization_strategies.norepeats_range, 543
angr.concretization_strategies.range, 542
angr.concretization_strategies.single, 541
angr.concretization_strategies.solutions, 541

angr.concretization_strategies.unlimited_range, 544
angr.distributed, 1361
angr.distributed.server, 1362
angr.distributed.worker, 1363
angr.engines, 587
angr.engines.engine, 594
angr.engines.failure, 598
angr.engines.hook, 597
angr.engines.light, 1167
angr.engines.light.data, 1166
angr.engines.light.engine, 1170
angr.engines.pcode, 603
angr.engines.pcode.behavior, 616
angr.engines.pcode.cc, 636
angr.engines.pcode.emulate, 615
angr.engines.pcode.engine, 604
angr.engines.pcode.lifter, 605
angr.engines.procedure, 597
angr.engines.soot, 601
angr.engines.soot.engine, 602
angr.engines.successors, 594
angr.engines.syscall, 598
angr.engines.unicorn, 603
angr.engines.vex, 599
angr.errors, 1355
angr.exploration_techniques, 552
angr.exploration_techniques.bucketizer, 586
angr.exploration_techniques.common, 585
angr.exploration_techniques.dfs, 569
angr.exploration_techniques.director, 579
angr.exploration_techniques.driller_core, 577
angr.exploration_techniques.explorer, 569
angr.exploration_techniques.lengthlimiter, 571
angr.exploration_techniques.local_loop_seer, 583
angr.exploration_techniques.loop_seer, 582
angr.exploration_techniques.manual_mergepoint, 571
angr.exploration_techniques.memory_watcher, 585
angr.exploration_techniques.oppologist, 581
angr.exploration_techniques.slicecutor, 578
angr.exploration_techniques.spiller, 571
angr.exploration_techniques.spiller_db, 574
angr.exploration_techniques.stochastic, 584
angr.exploration_techniques.suggestions, 586
angr.exploration_techniques.tech_builder, 585
angr.exploration_techniques.threading, 574
angr.exploration_techniques.timeout, 569
angr.exploration_techniques.tracer, 575
angr.exploration_techniques.unique, 584
angr.exploration_techniques.veritesting, 574
angr.factory, 227
angr.flirt, 1342
angr.flirt.build_sig, 1343
angr.keyed_region, 867
angr.knowledge_base, 711
angr.knowledge_plugins, 712
angr.knowledge_plugins.callsite_prototypes, 739
angr.knowledge_plugins.cfg, 740
angr.knowledge_plugins.cfg.cfg_manager, 766
angr.knowledge_plugins.cfg.cfg_model, 756
angr.knowledge_plugins.cfg.cfg_node, 766
angr.knowledge_plugins.cfg.indirect_jump, 769
angr.knowledge_plugins.cfg.memory_data, 764
angr.knowledge_plugins.comments, 775
angr.knowledge_plugins.data, 775
angr.knowledge_plugins.debug_variables, 820
angr.knowledge_plugins.functions, 776
angr.knowledge_plugins.functions.function, 799
angr.knowledge_plugins.functions.function_manager, 790
angr.knowledge_plugins.functions.function_parser, 809
angr.knowledge_plugins.functions.soot_function, 810
angr.knowledge_plugins.indirect_jumps, 775
angr.knowledge_plugins.key_definitions, 822
angr.knowledge_plugins.key_definitions.atoms, 836
angr.knowledge_plugins.key_definitions.constants, 841
angr.knowledge_plugins.key_definitions.definition, 841
angr.knowledge_plugins.key_definitions.environment, 844
angr.knowledge_plugins.key_definitions.heap_address, 845
angr.knowledge_plugins.key_definitions.key_definition_mana 845
angr.knowledge_plugins.key_definitions.live_definitions, 846
angr.knowledge_plugins.key_definitions.rd_model, 855
angr.knowledge_plugins.key_definitions.tag, 857
angr.knowledge_plugins.key_definitions.undefined, 858
angr.knowledge_plugins.key_definitions.unknown_size, 859
angr.knowledge_plugins.key_definitions.uses, 859
angr.knowledge_plugins.labels, 775
angr.knowledge_plugins.patches, 737

angr.knowledge_plugins.plugin, 738
angr.knowledge_plugins.propagations, 773
angr.knowledge_plugins.structured_code, 822
angr.knowledge_plugins.types, 772
angr.knowledge_plugins.variables, 811
angr.knowledge_plugins.variables.variable_access, 812
angr.knowledge_plugins.variables.variable_manager, 812
angr.knowledge_plugins.xrefs, 861
angr.knowledge_plugins.xrefs.xref, 863
angr.knowledge_plugins.xrefs.xref_manager, 864
angr.knowledge_plugins.xrefs.xref_types, 864
angr.misc.plugins, 235
angr.procedures, 646
angr.procedures.definitions, 649
angr.procedures.stubs.format_parser, 646
angr.project, 223
angr.protos, 871
angr.serializable, 870
angr.sim_manager, 544
angr.sim_options, 242
angr.sim_procedure, 642
angr.sim_state, 238
angr.sim_state_options, 242
angr.sim_type, 686
angr.sim_variable, 680
angr.simos, 1325
angr.simos.cgc, 1337
angr.simos.javavm, 1340
angr.simos.linux, 1335
angr.simos.simos, 1333
angr.simos.userland, 1337
angr.simos.windows, 1338
angr.slicer, 1320
angr.state_hierarchy, 551
angr.state_plugins, 246
angr.state_plugins.callstack, 371
angr.state_plugins.cgc, 379
angr.state_plugins.debug_variables, 421
angr.state_plugins.filesystem, 354
angr.state_plugins.gdb, 378
angr.state_plugins.globals, 386
angr.state_plugins.heap, 404
angr.state_plugins.heap.heap_base, 411
angr.state_plugins.heap.heap_brk, 412
angr.state_plugins.heap.heap_freelist, 414
angr.state_plugins.heap.heap_libc, 415
angr.state_plugins.heap.heap_ptmalloc, 416
angr.state_plugins.heap.utils, 420
angr.state_plugins.history, 375
angr.state_plugins.inspect, 331
angr.state_plugins.javavm_classloader, 400
angr.state_plugins.jni_references, 402
angr.state_plugins.libc, 343
angr.state_plugins.light_registers, 374
angr.state_plugins.log, 370
angr.state_plugins.loop_data, 399
angr.state_plugins.plugin, 329
angr.state_plugins.posix, 347
angr.state_plugins.preconstrainer, 390
angr.state_plugins.scratch, 388
angr.state_plugins.sim_action, 638
angr.state_plugins.sim_action_object, 640
angr.state_plugins.sim_event, 642
angr.state_plugins.solver, 361
angr.state_plugins.symbolizer, 420
angr.state_plugins.trace_additions, 381
angr.state_plugins.uc_manager, 387
angr.state_plugins.unicorn_engine, 392
angr.state_plugins.view, 426
angr.storage, 423
angr.storage.file, 431
angr.storage.memory_mixins, 456
angr.storage.memory_mixins.actions_mixin, 493
angr.storage.memory_mixins.address_concretization_mixin, 495
angr.storage.memory_mixins.bvv_conversion_mixin, 491
angr.storage.memory_mixins.conditional_store_mixin, 497
angr.storage.memory_mixins.convenient_mappings_mixin, 499
angr.storage.memory_mixins.default_filler_mixin, 490
angr.storage.memory_mixins.dirty_addrs_mixin, 495
angr.storage.memory_mixins.hex_dumper_mixin, 491
angr.storage.memory_mixins.label_merger_mixin, 497
angr.storage.memory_mixins.multi_value_merger_mixin, 503
angr.storage.memory_mixins.name_resolution_mixin, 490
angr.storage.memory_mixins.paged_memory, 504
angr.storage.memory_mixins.paged_memory.page_backer_mixins 508
angr.storage.memory_mixins.paged_memory.paged_memory_mixin 504
angr.storage.memory_mixins.paged_memory.pages, 510
angr.storage.memory_mixins.paged_memory.pages.cooperation, 520
angr.storage.memory_mixins.paged_memory.pages.history_trac 519
angr.storage.memory_mixins.paged_memory.pages.ispo_mixin,

520
angr.storage.memory_mixins.paged_memory.pages.list_pages,
521
angr.storage.memory_mixins.paged_memory.pages.multi_values,
501
angr.storage.memory_mixins.paged_memory.pages.page_8,
499
angr.storage.memory_mixins.paged_memory.pages.page_mixin,
519
angr.storage.memory_mixins.paged_memory.pages.refcount_mixin,
518
angr.storage.memory_mixins.paged_memory.pages.updates,
522
angr.storage.memory_mixins.paged_memory.privileged_mixin,
510
angr.storage.memory_mixins.paged_memory.stack_allocation_mixin,
509
angr.storage.memory_mixins.regioned_memory,
524
angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor,
535
angr.storage.memory_mixins.regioned_memory.abstract_merger_mixin,
538
angr.storage.memory_mixins.regioned_memory.region_category_mixin,
535
angr.storage.memory_mixins.regioned_memory.region_data,
533
angr.storage.memory_mixins.regioned_memory.region_meta_mixin,
536
angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin,
538
angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin,
530
angr.storage.memory_mixins.regioned_memory.static_find_mixin,
535
angr.storage.memory_mixins.simple_interface_mixin,
493
angr.storage.memory_mixins.simplification_mixin,
498
angr.storage.memory_mixins.size_resolution_mixin,
494
angr.storage.memory_mixins.slotted_memory,
539
angr.storage.memory_mixins.smart_find_mixin,
490
angr.storage.memory_mixins.symbolic_merger_mixin,
493
angr.storage.memory_mixins.top_merger_mixin,
503
angr.storage.memory_mixins.underconstrained_mixin,
492
angr.storage.memory_mixins.unwrapper_mixin,
498
angr.storage.memory_object, 451
angr.utils, 1344
angr.utils.algo, 1344
angr.utils.constants, 1344
angr.utils.crowdict, 1344
angr.utils.dynamic_dictlist, 1346
angr.utils.enums_conv, 1346
angr.utils.env, 1346
angr.utils.formatting, 1353
angr.utils.graph, 1346
angr.utils.kazy_import, 1351
angr.utils.library, 1351
angr.utils.loader, 1351
angr.utils.mp, 1354
angr.utils.timing, 1353
angr.vaults, 871

INDEX

Symbols

__init__() (*angr.AngrMissingTypeError method*), 158
__init__() (*angr.BP method*), 155
__init__() (*angr.Blade method*), 159
__init__() (*angr.Block method*), 160
__init__() (*angr.Emulator method*), 161
__init__() (*angr.ExplorationTechnique method*), 162
__init__() (*angr.KnowledgeBase method*), 164
__init__() (*angr.LLMClient method*), 165
__init__() (*angr.PTChunk method*), 167
__init__() (*angr.PointerWrapper method*), 168
__init__() (*angr.Project method*), 169
__init__() (*angr.Server method*), 173
__init__() (*angr.SimCC method*), 174
__init__() (*angr.SimCC.ArgSession method*), 175
__init__() (*angr.SimFile method*), 179
__init__() (*angr.SimFileBase method*), 181
__init__() (*angr.SimFileDescriptor method*), 182
__init__() (*angr.SimFileDescriptorDuplex method*), 185
__init__() (*angr.SimFileStream method*), 187
__init__() (*angr.SimHeapBrk method*), 189
__init__() (*angr.SimHeapPTMalloc method*), 191
__init__() (*angr.SimHostFilesystem method*), 193
__init__() (*angr.SimMemoryMissingError method*), 194
__init__() (*angr.SimOS method*), 195
__init__() (*angr.SimPackets method*), 198
__init__() (*angr.SimPacketsStream method*), 200
__init__() (*angr.SimProcedure method*), 204
__init__() (*angr.SimReliftException method*), 206
__init__() (*angr.SimSegfaultException method*), 206
__init__() (*angr.SimState method*), 208
__init__() (*angr.SimStatePlugin method*), 211
__init__() (*angr.SimUninitializedAccessError method*), 213
__init__() (*angr.SimulationManager method*), 215
__init__() (*angr.StateHierarchy method*), 221
__init__() (*angr.ailment.AILBlockRewriter method*), 1364
__init__() (*angr.ailment.AILBlockWalker method*), 1364
__init__() (*angr.ailment.Assignment method*), 1365
__init__() (*angr.ailment.BinaryOp method*), 1366
__init__() (*angr.ailment.Block method*), 1368
__init__() (*angr.ailment.Const method*), 1368
__init__() (*angr.ailment.Expression method*), 1369
__init__() (*angr.ailment.Manager method*), 1370
__init__() (*angr.ailment.PCodeIRSBConverter method*), 1370
__init__() (*angr.ailment.Register method*), 1371
__init__() (*angr.ailment.Tmp method*), 1372
__init__() (*angr.ailment.UnaryOp method*), 1372
__init__() (*angr.ailment.block.Block method*), 1406
__init__() (*angr.ailment.converter_pcode.PCodeIRSBConverter method*), 1374
__init__() (*angr.ailment.expression.Array method*), 1395
__init__() (*angr.ailment.expression.Atom method*), 1376
__init__() (*angr.ailment.expression.BasePointerOffset method*), 1390
__init__() (*angr.ailment.expression.BinaryOp method*), 1384
__init__() (*angr.ailment.expression.Call method*), 1393
__init__() (*angr.ailment.expression.ComboRegister method*), 1378
__init__() (*angr.ailment.expression.Const method*), 1376
__init__() (*angr.ailment.expression.Convert method*), 1382
__init__() (*angr.ailment.expression.DirtyExpression method*), 1387
__init__() (*angr.ailment.expression.Expression method*), 1376
__init__() (*angr.ailment.expression.Extract method*), 1391
__init__() (*angr.ailment.expression.FunctionLikeMacro method*), 1396
__init__() (*angr.ailment.expression.ITE method*), 1386
__init__() (*angr.ailment.expression.Insert method*), 1392
__init__() (*angr.ailment.expression.Let method*), 1396

__init__() (*angr.ailment.expression.Load method*), 1385
__init__() (*angr.ailment.expression.Macro method*), 1396
__init__() (*angr.ailment.expression.MultiStatementExpression method*), 1389
__init__() (*angr.ailment.expression.Op method*), 1381
__init__() (*angr.ailment.expression.Phi method*), 1380
__init__() (*angr.ailment.expression.Register method*), 1378
__init__() (*angr.ailment.expression.Reinterpret method*), 1383
__init__() (*angr.ailment.expression.RustEnum method*), 1395
__init__() (*angr.ailment.expression.StackBaseOffset method*), 1390
__init__() (*angr.ailment.expression.StringLiteral method*), 1394
__init__() (*angr.ailment.expression.Struct method*), 1394
__init__() (*angr.ailment.expression.Tmp method*), 1377
__init__() (*angr.ailment.expression.UnaryOp method*), 1381
__init__() (*angr.ailment.expression.VEXCCallExpression method*), 1388
__init__() (*angr.ailment.expression.VirtualVariable method*), 1379
__init__() (*angr.ailment.manager.Manager method*), 1407
__init__() (*angr.ailment.statement.Assignment method*), 1397
__init__() (*angr.ailment.statement.CAS method*), 1403
__init__() (*angr.ailment.statement.ConditionalJump method*), 1401
__init__() (*angr.ailment.statement.DirtyStatement method*), 1405
__init__() (*angr.ailment.statement.Jump method*), 1400
__init__() (*angr.ailment.statement.Label method*), 1405
__init__() (*angr.ailment.statement.Return method*), 1403
__init__() (*angr.ailment.statement.SideEffectStatement method*), 1402
__init__() (*angr.ailment.statement.Store method*), 1399
__init__() (*angr.ailment.statement.WeakAssignment method*), 1398
__init__() (*angr.ailment.tagged_object.TaggedObject method*), 1408
__init__() (*angr.analyses.AnalysesHub method*), 881
__init__() (*angr.analyses.BackwardSlice method*), 881
__init__() (*angr.analyses.BinDiff method*), 883
__init__() (*angr.analyses.BinaryOptimizer method*), 884
__init__() (*angr.analyses.BoyScout method*), 884
__init__() (*angr.analyses.CDG method*), 873
__init__() (*angr.analyses.CFG method*), 874
__init__() (*angr.analyses.CFGArchOptions method*), 884
__init__() (*angr.analyses.CFGEmulated method*), 885
__init__() (*angr.analyses.CFGFast method*), 890
__init__() (*angr.analyses.CFGFastSoot method*), 892
__init__() (*angr.analyses.CalleeCleanupFinder method*), 894
__init__() (*angr.analyses.CallingConventionAnalysis method*), 895
__init__() (*angr.analyses.ClassIdentifier method*), 896
__init__() (*angr.analyses.CodeCaveAnalysis method*), 896
__init__() (*angr.analyses.CodeTagging method*), 897
__init__() (*angr.analyses.CompleteCallingConventionsAnalysis method*), 898
__init__() (*angr.analyses.CongruencyCheck method*), 899
__init__() (*angr.analyses.DDG method*), 876
__init__() (*angr.analyses.DataDependencyGraphAnalysis method*), 900
__init__() (*angr.analyses.Decompiler method*), 901
__init__() (*angr.analyses.Disassembly method*), 903
__init__() (*angr.analyses.DominanceFrontier method*), 904
__init__() (*angr.analyses.FactCollector method*), 905
__init__() (*angr.analyses.FastConstantPropagation method*), 905
__init__() (*angr.analyses.FlirtAnalysis method*), 906
__init__() (*angr.analyses.ForwardAnalysis method*), 906
__init__() (*angr.analyses.Identifier method*), 907
__init__() (*angr.analyses.InitializationFinder method*), 908
__init__() (*angr.analyses.LanguageDetector method*), 908
__init__() (*angr.analyses.LoopAnalysis method*), 909
__init__() (*angr.analyses.LoopFinder method*), 909
__init__() (*angr.analyses.LoopUnroller method*), 909
__init__() (*angr.analyses.PackingDetector method*), 910
__init__() (*angr.analyses.PatchFinderAnalysis method*), 910
__init__() (*angr.analyses.Pathfinder method*), 910
__init__() (*angr.analyses.PropagatorAnalysis method*), 912
__init__() (*angr.analyses.ProximityGraphAnalysis method*), 912
__init__() (*angr.analyses.ReachingDefinitionsAnalysis method*), 913

__init__() (*angr.analyses.Reassembler method*), 916
__init__() (*angr.analyses.SLivenessAnalysis method*), 919
__init__() (*angr.analyses.SPropagatorAnalysis method*), 920
__init__() (*angr.analyses.SReachingDefinitionsAnalysis method*), 921
__init__() (*angr.analyses.SelfModifyingCodeAnalysis method*), 921
__init__() (*angr.analyses.SootClassHierarchy method*), 922
__init__() (*angr.analyses.StackPointerTracker method*), 923
__init__() (*angr.analyses.StaticHooker method*), 923
__init__() (*angr.analyses.StaticObjectFinder method*), 923
__init__() (*angr.analyses.Typehoon method*), 924
__init__() (*angr.analyses.VFG method*), 879
__init__() (*angr.analyses.VSA_DDG method*), 880
__init__() (*angr.analyses.VariableRecovery method*), 925
__init__() (*angr.analyses.VariableRecoveryFast method*), 926
__init__() (*angr.analyses.Veritesting method*), 927
__init__() (*angr.analyses.VtableFinder method*), 927
__init__() (*angr.analyses.XRefsAnalysis method*), 928
__init__() (*angr.analyses.analysis.AnalysesHub method*), 928
__init__() (*angr.analyses.analysis.AnalysisFactory method*), 930
__init__() (*angr.analyses.analysis.AnalysisLogEntry method*), 928
__init__() (*angr.analyses.analysis.KnownAnalysesPlugin method*), 929
__init__() (*angr.analyses.backward_slice.BackwardSlice method*), 946
__init__() (*angr.analyses.binary_optimizer.BinaryOptimizer method*), 1307
__init__() (*angr.analyses.binary_optimizer.ConstantPropagation method*), 1306
__init__() (*angr.analyses.binary_optimizer.DeadAssignment method*), 1306
__init__() (*angr.analyses.binary_optimizer.RedundantStackVariable method*), 1306
__init__() (*angr.analyses.binary_optimizer.RegisterReallocation method*), 1306
__init__() (*angr.analyses.bindiff.BinDiff method*), 949
__init__() (*angr.analyses.bindiff.ConstantChange method*), 947
__init__() (*angr.analyses.bindiff.Difference method*), 947
__init__() (*angr.analyses.bindiff.FunctionDiff method*), 948
__init__() (*angr.analyses.bindiff.NormalizedBlock method*), 948
__init__() (*angr.analyses.bindiff.NormalizedFunction method*), 948
__init__() (*angr.analyses.boyscout.BoyScout method*), 950
__init__() (*angr.analyses.callee_cleanup_finder.CalleeCleanupFinder method*), 1307
__init__() (*angr.analyses.calling_convention.CallingConventionAnalysis method*), 952
__init__() (*angr.analyses.calling_convention.FactCollector method*), 952
__init__() (*angr.analyses.cdg.CDG method*), 1017
__init__() (*angr.analyses.cfg.CFBlanket method*), 958
__init__() (*angr.analyses.cfg.CFG method*), 956
__init__() (*angr.analyses.cfg.CFGArchOptions method*), 959
__init__() (*angr.analyses.cfg.CFGBase method*), 960
__init__() (*angr.analyses.cfg.CFGEmulated method*), 962
__init__() (*angr.analyses.cfg.CFGFast method*), 967
__init__() (*angr.analyses.cfg.CFGFastSoot method*), 970
__init__() (*angr.analyses.cfg.cfb.CFBlanket method*), 973
__init__() (*angr.analyses.cfg.cfb.CFBlanketView method*), 972
__init__() (*angr.analyses.cfg.cfb.MemoryRegion method*), 972
__init__() (*angr.analyses.cfg.cfb.Unknown method*), 973
__init__() (*angr.analyses.cfg.cfg.CFG method*), 974
__init__() (*angr.analyses.cfg.cfg_arch_options.CFGArchOptions method*), 993
__init__() (*angr.analyses.cfg.cfg_base.CFGBase method*), 981
__init__() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 977
__init__() (*angr.analyses.cfg.cfg_emulated.CFGJob method*), 976
__init__() (*angr.analyses.cfg.cfg_emulated.PendingJob method*), 976
__init__() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 990
__init__() (*angr.analyses.cfg.cfg_fast.CFGJob method*), 987
__init__() (*angr.analyses.cfg.cfg_fast.DecodingAssumption method*), 984
__init__() (*angr.analyses.cfg.cfg_fast.FunctionCallEdge method*), 986
__init__() (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge method*), 986
__init__() (*angr.analyses.cfg.cfg_fast.FunctionReturn method*), 984
__init__() (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge

method), 987
__init__() (angr.analyses.cfg.cfg_fast.FunctionTransitionEdge
method), 986
__init__() (angr.analyses.cfg.cfg_fast.PendingJobs
method), 984
__init__() (angr.analyses.cfg.cfg_fast_soot.CFGFastSoot
method), 1015
__init__() (angr.analyses.cfg.cfg_job_base.CFGJobBase
method), 993
__init__() (angr.analyses.cfg.cfg_job_base.FunctionKey
method), 993
__init__() (angr.analyses.cfg.indirect_jump_resolvers.AArch64MachineCodeResolver
method), 1005
__init__() (angr.analyses.cfg.indirect_jump_resolvers.AMD64ElfGotPlt
method), 1005
__init__() (angr.analyses.cfg.indirect_jump_resolvers.AMD64PeIat
method), 1006
__init__() (angr.analyses.cfg.indirect_jump_resolvers.ArmElfFast
method), 1007
__init__() (angr.analyses.cfg.indirect_jump_resolvers.ConstantResolver
method), 1008
__init__() (angr.analyses.cfg.indirect_jump_resolvers.JumpTableResolver
method), 1009
__init__() (angr.analyses.cfg.indirect_jump_resolvers.MemoryLoadResolver
method), 1010
__init__() (angr.analyses.cfg.indirect_jump_resolvers.MipsElfFast
method), 1011
__init__() (angr.analyses.cfg.indirect_jump_resolvers.MipsElfGot
method), 1011
__init__() (angr.analyses.cfg.indirect_jump_resolvers.SyscallHeuristic
method), 1012
__init__() (angr.analyses.cfg.indirect_jump_resolvers.X86ElfPicPlt
method), 1013
__init__() (angr.analyses.cfg.indirect_jump_resolvers.X86PeIat
method), 1014
__init__() (angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got
method), 994
__init__() (angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast
method), 994
__init__() (angr.analyses.cfg.indirect_jump_resolvers.constant_resolver
method), 1003
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransformation
method), 999
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.BSHeuristic
method), 1001
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessor
method), 1001
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessorState
method), 1000
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver
method), 1002
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.TangerineAddr
method), 999
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.LoadHook
method), 1001
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.MIPSGP
method), 1002
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffs
method), 1000
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.Register
method), 1001
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.Tmp
method), 999
__init__() (angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.Mips
method), 996
__init__() (angr.analyses.cfg.indirect_jump_resolvers.resolver.IndirectJu
method), 1004
__init__() (angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt.X8
method), 997
__init__() (angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat.X86Pe
method), 995
__init__() (angr.analyses.cfg_slice_to_sink.CFGSliceToSink
method), 1238
__init__() (angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceT
method), 1241
__init__() (angr.analyses.class_identifier.ClassIdentifier
method), 1292
__init__() (angr.analyses.code_tagging.CodeTagging
method), 1018
__init__() (angr.analyses.complete_calling_conventions.CompleteCallin
method), 954
__init__() (angr.analyses.congruency_check.CongruencyCheck
method), 1305
__init__() (angr.analyses.data_dep.BaseDepNode
method), 1316
__init__() (angr.analyses.data_dep.ConstantDepNode
method), 1316
__init__() (angr.analyses.data_dep.DataDependencyGraphAnalysis
method), 1317
__init__() (angr.analyses.data_dep.MemDepNode
method), 1318
__init__() (angr.analyses.data_dep.RegDepNode
method), 1318
__init__() (angr.analyses.data_dep.TmpDepNode
method), 1319
__init__() (angr.analyses.data_dep.VarDepNode
method), 1319
__init__() (angr.analyses.data_dep.data_dependency_analysis.DataDep
method), 1312
__init__() (angr.analyses.data_dep.data_dependency_analysis.NodalAn
method), 1311
__init__() (angr.analyses.data_dep.dep_nodes.BaseDepNode
method), 1314
__init__() (angr.analyses.data_dep.dep_nodes.ConstantDepNode
method), 1314
__init__() (angr.analyses.data_dep.dep_nodes.MemDepNode
method), 1314
__init__() (angr.analyses.data_dep.dep_nodes.RegDepNode

method), 1316
__init__() (*angr.analyses.data_dep.dep_nodes.TmpDepNode method*), 1315
__init__() (*angr.analyses.data_dep.dep_nodes.VarDepNode method*), 1315
__init__() (*angr.analyses.data_dep.sim_act_location.ParsedInstruction method*), 1313
__init__() (*angr.analyses.data_dep.sim_act_location.SimActLocation method*), 1313
__init__() (*angr.analyses.datagraph_meta.DataGraphMeta method*), 1018
__init__() (*angr.analyses.ddg.AST method*), 1156
__init__() (*angr.analyses.ddg.DDG method*), 1159
__init__() (*angr.analyses.ddg.DDGJob method*), 1157
__init__() (*angr.analyses.ddg.DDGView method*), 1159
__init__() (*angr.analyses.ddg.DDGViewInstruction method*), 1159
__init__() (*angr.analyses.ddg.DDGViewItem method*), 1158
__init__() (*angr.analyses.ddg.LiveDefinitions method*), 1157
__init__() (*angr.analyses.ddg.ProgramVariable method*), 1157
__init__() (*angr.analyses.decompiler.AILSimplifier method*), 1039
__init__() (*angr.analyses.decompiler.BlockSimplifier method*), 1040
__init__() (*angr.analyses.decompiler.CStructuredCodeGenerator method*), 1041
__init__() (*angr.analyses.decompiler.CallSiteMaker method*), 1043
__init__() (*angr.analyses.decompiler.Clinic method*), 1044
__init__() (*angr.analyses.decompiler.Decompiler method*), 1047
__init__() (*angr.analyses.decompiler.GraphDephication method*), 1049
__init__() (*angr.analyses.decompiler.ImportSourceCode method*), 1050
__init__() (*angr.analyses.decompiler.RegionIdentifier method*), 1050
__init__() (*angr.analyses.decompiler.RegionSimplifier method*), 1051
__init__() (*angr.analyses.decompiler.SeqNodeDephication method*), 1051
__init__() (*angr.analyses.decompiler.Ssailification method*), 1052
__init__() (*angr.analyses.decompiler.ail_simplifier.AILBlockTempCollector method*), 1053
__init__() (*angr.analyses.decompiler.ail_simplifier.AILSimplifier method*), 1053
__init__() (*angr.analyses.decompiler.ail_simplifier.PartialConstantPropagation method*), 1053
__init__() (*angr.analyses.decompiler.ailgraph_walker.AILGraphWalker method*), 1054
__init__() (*angr.analyses.decompiler.block_simplifier.BlockSimplifier method*), 1055
__init__() (*angr.analyses.decompiler.block_simplifier.HasCallExprWalker method*), 1054
__init__() (*angr.analyses.decompiler.callsite_maker.CallSiteMaker method*), 1055
__init__() (*angr.analyses.decompiler.ccall_rewriters.rewriter_base.CCall method*), 1055
__init__() (*angr.analyses.decompiler.clinic.Clinic method*), 1059
__init__() (*angr.analyses.decompiler.clinic.ComboRegReferenceWalker method*), 1058
__init__() (*angr.analyses.decompiler.clinic.DataRefDesc method*), 1057
__init__() (*angr.analyses.decompiler.condition_processor.ConditionProc method*), 1062
__init__() (*angr.analyses.decompiler.decompilation_cache.DecompilationCache method*), 1064
__init__() (*angr.analyses.decompiler.decompilation_options.DecompilationOptions method*), 1064
__init__() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1066
__init__() (*angr.analyses.decompiler.empty_node_remover.EmptyNodeRemoverBase method*), 1068
__init__() (*angr.analyses.decompiler.expression_narrower.EffectiveSizeExpr method*), 1069
__init__() (*angr.analyses.decompiler.expression_narrower.ExprNarrowing method*), 1069
__init__() (*angr.analyses.decompiler.expression_narrower.ExpressionNarrower method*), 1070
__init__() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 1071
__init__() (*angr.analyses.decompiler.jump_target_collector.JumpTargetCollector method*), 1071
__init__() (*angr.analyses.decompiler.jumptable_entry_condition_rewriter method*), 1071
__init__() (*angr.analyses.decompiler.optimization_passes.BasePointerSaveSimplifier method*), 1072
__init__() (*angr.analyses.decompiler.optimization_passes.CallStatementRewriter method*), 1072
__init__() (*angr.analyses.decompiler.optimization_passes.CodeMotionOptimization method*), 1073
__init__() (*angr.analyses.decompiler.optimization_passes.ConditionConstantPropagation method*), 1074
__init__() (*angr.analyses.decompiler.optimization_passes.ConstPropOpt method*), 1074
__init__() (*angr.analyses.decompiler.optimization_passes.ConstantDereference method*), 1075
__init__() (*angr.analyses.decompiler.optimization_passes.CrossJumpReverter method*), 1076
__init__() (*angr.analyses.decompiler.optimization_passes.DeadblockRemoverBase method*), 1076

__init__() (angr.analyses.decompiler.optimization_passes.DivSimplifier method), 1076
__init__() (angr.analyses.decompiler.optimization_passes.DuplicateReturn method), 1077
__init__() (angr.analyses.decompiler.optimization_passes.EmptyStatementNoOp method), 1078
__init__() (angr.analyses.decompiler.optimization_passes.EmptyOpSwap method), 1079
__init__() (angr.analyses.decompiler.optimization_passes.FlipBoolCmp method), 1079
__init__() (angr.analyses.decompiler.optimization_passes.ITEExprConverter method), 1079
__init__() (angr.analyses.decompiler.optimization_passes.ITERegionConverter method), 1080
__init__() (angr.analyses.decompiler.optimization_passes.InlinedMemcpySimplifier method), 1080
__init__() (angr.analyses.decompiler.optimization_passes.InlinedMemset8Simplifier method), 1081
__init__() (angr.analyses.decompiler.optimization_passes.InlinedStrcpyConsolidation method), 1081
__init__() (angr.analyses.decompiler.optimization_passes.InlinedStrcpyTransformation method), 1082
__init__() (angr.analyses.decompiler.optimization_passes.InlinedStrcpyWordSimplifier method), 1082
__init__() (angr.analyses.decompiler.optimization_passes.InlinedWcscpySimplifier method), 1082
__init__() (angr.analyses.decompiler.optimization_passes.LoweredSwitchSimplifier method), 1083
__init__() (angr.analyses.decompiler.optimization_passes.MipsGpSetupSimplifier method), 1083
__init__() (angr.analyses.decompiler.optimization_passes.ModSimplifier method), 1084
__init__() (angr.analyses.decompiler.optimization_passes.RegisterSetAngrSimplifier method), 1085
__init__() (angr.analyses.decompiler.optimization_passes.RegisterSetAngrSimplifierAcrossCompiler method), 1085
__init__() (angr.analyses.decompiler.optimization_passes.RetAddrSetSimplifier method), 1085
__init__() (angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseDuplicator method), 1086
__init__() (angr.analyses.decompiler.optimization_passes.SwitchHighPriorityCaseDuplicator method), 1086
__init__() (angr.analyses.decompiler.optimization_passes.SwitchLowPriorityCaseDuplicator method), 1088
__init__() (angr.analyses.decompiler.optimization_passes.StackCanarySimplifier method), 1088
__init__() (angr.analyses.decompiler.optimization_passes.StackVariableRenaming method), 1089
__init__() (angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseDuplicator method), 1089
__init__() (angr.analyses.decompiler.optimization_passes.SwitchReadelfJumpTableRewriter method), 1090
__init__() (angr.analyses.decompiler.optimization_passes.TagSlicer method), 1090
__init__() (angr.analyses.decompiler.optimization_passes.WinStackCanaryInstrumentation method), 1091
__init__() (angr.analyses.decompiler.optimization_passes.X86GccGetPcThunkToMovPop method), 1091
__init__() (angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePtrSaveSimplifier method), 1098
__init__() (angr.analyses.decompiler.optimization_passes.const_derefs.BinopConstDeref method), 1091
__init__() (angr.analyses.decompiler.optimization_passes.const_derefs.ConstantDeref method), 1092
__init__() (angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifierAngr method), 1098
__init__() (angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifierBase method), 1098
__init__() (angr.analyses.decompiler.optimization_passes.engine_base.SimplifyingOptimizationPassMixin method), 1101
__init__() (angr.analyses.decompiler.optimization_passes.expr_op_swap.BinaryOpSwap method), 1103
__init__() (angr.analyses.decompiler.optimization_passes.expr_op_swap.CmpOpSwap method), 1102
__init__() (angr.analyses.decompiler.optimization_passes.expr_op_swap.OpTransformationSimplifier method), 1102
__init__() (angr.analyses.decompiler.optimization_passes.expr_op_swap.UnaryOpSwap method), 1102
__init__() (angr.analyses.decompiler.optimization_passes.ite_expr_converters.ITEEarlyConverter method), 1098
__init__() (angr.analyses.decompiler.optimization_passes.ite_expr_converters.ITEEarlyCustomizedCondition method), 1098
__init__() (angr.analyses.decompiler.optimization_passes.ite_expr_converters.ITEEarlyRegionConverter method), 1099
__init__() (angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier method), 1099
__init__() (angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifierAngr method), 1100
__init__() (angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifierAngrAcrossCompiler method), 1100
__init__() (angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier method), 1101
__init__() (angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage1Base method), 1093
__init__() (angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage1High method), 1094
__init__() (angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage1Low method), 1096
__init__() (angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage2 method), 1097
__init__() (angr.analyses.decompiler.optimization_passes.register_save_restore.RegisterSaveRestoreSimplifier method), 1103
__init__() (angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier method), 1104
__init__() (angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier method), 1097
__init__() (angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_thunk_simplifier.X86GccGetPcThunkSimplifier method), 1104

__init__() (angr.analyses.decompiler.peephole_optimizations.ConstantExpressionSimplifier method), 1106
__init__() (angr.analyses.decompiler.peephole_optimizations.ConstantMultiplication method), 1105
__init__() (angr.analyses.decompiler.peephole_optimizations.ConstantStmtBase method), 1104
__init__() (angr.analyses.decompiler.redundant_label_remover.RedundantLabelRemover method), 1119
__init__() (angr.analyses.decompiler.region_identifier.RegionIdentifier method), 1108
__init__() (angr.analyses.decompiler.region_simplifiers.RegionSimplifier method), 1109
__init__() (angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer.CascadingConditionTransformer method), 1109
__init__() (angr.analyses.decompiler.region_simplifiers.cascading_ifs_remover.CascadingIfsRemover method), 1109
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ConditionalExpressionFolder method), 1111
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ConditionalBreakExpressionFolder method), 1111
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter method), 1112
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionFolder method), 1113
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionLocator method), 1110
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionReplacer method), 1113
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionSpotter method), 1112
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionUseFinder method), 1112
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.InterferenceChecker method), 1112
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.LongNodeFinder method), 1111
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.MultiStatementExpressionAssignmentFinder method), 1111
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.StatementLocator method), 1110
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.StatementReplacer method), 1113
__init__() (angr.analyses.decompiler.region_simplifiers.goto_simplifier.GotoSimplifier method), 1114
__init__() (angr.analyses.decompiler.region_simplifiers.if_simplifier.IfSimplifier method), 1114
__init__() (angr.analyses.decompiler.region_simplifiers.ifelse_simplifier.IfElseFlattener method), 1114
__init__() (angr.analyses.decompiler.region_simplifiers.loop_simplifier.LoopSimplifier method), 1114
__init__() (angr.analyses.decompiler.region_simplifiers.node_address_finder.NodeAddressFinder method), 1114
__init__() (angr.analyses.decompiler.region_simplifiers.region_simplifier.RegionSimplifier method), 1115
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterBase method), 1116
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterMultiStmtBase method), 1118
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterSimpleBase method), 1116
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterSimplifier method), 1117
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchCluster method), 1117
__init__() (angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier.SwitchExpressionSimplifier method), 1119
__init__() (angr.analyses.decompiler.structurer_base.SequenceOrderingVisitorWalker method), 1119
__init__() (angr.analyses.decompiler.sequence_walker.SequenceWalker method), 1120
__init__() (angr.analyses.decompiler.structured_codegen.BaseStructuredCodeGenerator method), 1120
__init__() (angr.analyses.decompiler.structured_codegen.CStructuredCodeGenerator method), 1121
__init__() (angr.analyses.decompiler.structured_codegen.DummyStructuredCodeGenerator method), 1124
__init__() (angr.analyses.decompiler.structured_codegen.ImportSourceCollector method), 1124
__init__() (angr.analyses.decompiler.structured_codegen.InstructionMappingCollector method), 1124
__init__() (angr.analyses.decompiler.structured_codegen.InstructionMappingStmtAccessor method), 1124
__init__() (angr.analyses.decompiler.structured_codegen.PositionMappingCollector method), 1125
__init__() (angr.analyses.decompiler.structured_codegen.PositionMappingStmtAccessor method), 1125
__init__() (angr.analyses.decompiler.structured_codegen.RustStructuredCodeGenerator method), 1126
__init__() (angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method), 1129
__init__() (angr.analyses.decompiler.structured_codegen.base.InstructionMappingStmtAccessor method), 1129
__init__() (angr.analyses.decompiler.structured_codegen.base.InstructionMappingStatementVisitor method), 1128
__init__() (angr.analyses.decompiler.structured_codegen.base.PositionMappingStmtAccessor method), 1128
__init__() (angr.analyses.decompiler.structured_codegen.base.PositionMappingStatementVisitor method), 1128
__init__() (angr.analyses.decompiler.structured_codegen.c.CAILBlock method), 1133
__init__() (angr.analyses.decompiler.structured_codegen.c.CArrayTypeLiteral method), 1144
__init__() (angr.analyses.decompiler.structured_codegen.c.CAssignment method), 1135
__init__() (angr.analyses.decompiler.structured_codegen.c.CBinaryOp method), 1140
__init__() (angr.analyses.decompiler.structured_codegen.c.CBreak method), 1135

__init__() (angr.analyses.decompiler.structured_codegen.c.CClosingObject method), 1144
__init__() (angr.analyses.decompiler.structured_codegen.c.CConstant method), 1141
__init__() (angr.analyses.decompiler.structured_codegen.c.CConstructor method), 1131
__init__() (angr.analyses.decompiler.structured_codegen.c.CContinue method), 1135
__init__() (angr.analyses.decompiler.structured_codegen.c.CDirtyExpression method), 1144
__init__() (angr.analyses.decompiler.structured_codegen.c.CDirtyStatement method), 1137
__init__() (angr.analyses.decompiler.structured_codegen.c.CDoWhileLoop method), 1133
__init__() (angr.analyses.decompiler.structured_codegen.c.CExpression method), 1133
__init__() (angr.analyses.decompiler.structured_codegen.c.CExpressionStatement method), 1136
__init__() (angr.analyses.decompiler.structured_codegen.c.CFakeVariable method), 1138
__init__() (angr.analyses.decompiler.structured_codegen.c.CFirstLoop method), 1134
__init__() (angr.analyses.decompiler.structured_codegen.c.CFunction method), 1131
__init__() (angr.analyses.decompiler.structured_codegen.c.CFunctionCall method), 1136
__init__() (angr.analyses.decompiler.structured_codegen.c.CGoto method), 1137
__init__() (angr.analyses.decompiler.structured_codegen.c.CIfE__() (angr.analyses.decompiler.structuring.RecursiveStructurer method), 1034
__init__() (angr.analyses.decompiler.structured_codegen.c.CIfBreak() (angr.analyses.decompiler.structuring.SAILRStructurer method), 1035
__init__() (angr.analyses.decompiler.structured_codegen.c.CIfElse() (angr.analyses.decompiler.structuring.dream.DreamStructure method), 1036
__init__() (angr.analyses.decompiler.structured_codegen.c.CIncompleteSwitchCase() (angr.analyses.decompiler.structuring.phoenix.GraphEdgeFilt method), 1037
__init__() (angr.analyses.decompiler.structured_codegen.c.CIndexedVariable() (angr.analyses.decompiler.structuring.phoenix.PhoenixStructu method), 1038
__init__() (angr.analyses.decompiler.structured_codegen.c.CLabel() (angr.analyses.decompiler.structuring.recursive_structurer.Re method), 1032
__init__() (angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression() (angr.analyses.decompiler.structuring.structurer_base.Structu method), 1036
__init__() (angr.analyses.decompiler.structured_codegen.c.CRegister() (angr.analyses.disassembly.BlockStart method), 1294
__init__() (angr.analyses.decompiler.structured_codegen.c.CReturn() (angr.analyses.disassembly.Comment method), 1296
__init__() (angr.analyses.decompiler.structured_codegen.c.CStatement() (angr.analyses.disassembly.Disassembly method), 1296
__init__() (angr.analyses.decompiler.structured_codegen.c.CStatements() (angr.analyses.disassembly.FuncComment method), 1296
__init__() (angr.analyses.decompiler.structured_codegen.c.CStructField() (angr.analyses.disassembly.FunctionStart method), 1293
__init__() (angr.analyses.decompiler.structured_codegen.c.CStructFieldNameDef() (angr.analyses.disassembly.Hook method), 1294
__init__() (angr.analyses.decompiler.structured_codegen.c.CStructuredCode() (angr.analyses.decompiler.structured_codegen.c.CSwitchCase method), 1146
__init__() (angr.analyses.decompiler.structured_codegen.c.CTypeCast method), 1135
__init__() (angr.analyses.decompiler.structured_codegen.c.CUnaryOp method), 1141
__init__() (angr.analyses.decompiler.structured_codegen.c.CUnsupported() (angr.analyses.decompiler.structured_codegen.c.CUnsupported method), 1140
__init__() (angr.analyses.decompiler.structured_codegen.c.CVEXCallExpr() (angr.analyses.decompiler.structured_codegen.c.CVEXCallExpr method), 1137
__init__() (angr.analyses.decompiler.structured_codegen.c.CVariable() (angr.analyses.decompiler.structured_codegen.c.CVariable method), 1143
__init__() (angr.analyses.decompiler.structured_codegen.c.CVariableField() (angr.analyses.decompiler.structured_codegen.c.CVariableField method), 1139
__init__() (angr.analyses.decompiler.structured_codegen.c.CWhileLoop() (angr.analyses.decompiler.structured_codegen.c.CWhileLoop method), 1140
__init__() (angr.analyses.decompiler.structured_codegen.dummy.Dummy() (angr.analyses.decompiler.structured_codegen.dummy.Dummy method), 1133
__init__() (angr.analyses.decompiler.structured_codegen.dwarf_import.I() (angr.analyses.decompiler.structured_codegen.dwarf_import.I method), 1149
__init__() (angr.analyses.decompiler.structured_codegen.dwarf_import.I() (angr.analyses.decompiler.structured_codegen.dwarf_import.I method), 1149
__init__() (angr.analyses.decompiler.structuring.DreamStructurer method), 1033
__init__() (angr.analyses.decompiler.structuring.PhoenixStructurer method), 1033
__init__() (angr.analyses.decompiler.structuring.RecursiveStructurer method), 1034
__init__() (angr.analyses.decompiler.structuring.SAILRStructurer method), 1035
__init__() (angr.analyses.decompiler.structuring.dream.DreamStructure method), 1036
__init__() (angr.analyses.decompiler.structuring.phoenix.GraphEdgeFilt method), 1037
__init__() (angr.analyses.decompiler.structuring.phoenix.PhoenixStructu method), 1038
__init__() (angr.analyses.decompiler.structuring.recursive_structurer.Re method), 1032
__init__() (angr.analyses.decompiler.structuring.structurer_base.Structu method), 1036
__init__() (angr.analyses.disassembly.BlockStart method), 1294
__init__() (angr.analyses.disassembly.Comment method), 1296
__init__() (angr.analyses.disassembly.Disassembly method), 1296
__init__() (angr.analyses.disassembly.FuncComment method), 1296
__init__() (angr.analyses.disassembly.FunctionStart method), 1293
__init__() (angr.analyses.disassembly.Hook method), 1294

__init__() (*angr.analyses.disassembly.IROp method*), 1293
__init__() (*angr.analyses.disassembly.Instruction method*), 1294
__init__() (*angr.analyses.disassembly.Label method*), 1293
__init__() (*angr.analyses.disassembly.MemoryOperand method*), 1295
__init__() (*angr.analyses.disassembly.Opcode method*), 1295
__init__() (*angr.analyses.disassembly.Operand method*), 1295
__init__() (*angr.analyses.disassembly.Register method*), 1295
__init__() (*angr.analyses.disassembly.SootExpression method*), 1294
__init__() (*angr.analyses.disassembly.SootExpressionInvoke method*), 1295
__init__() (*angr.analyses.disassembly.SootExpressionStaticFieldRef method*), 1294
__init__() (*angr.analyses.disassembly.SootExpressionTarget method*), 1294
__init__() (*angr.analyses.disassembly.SootStatement method*), 1295
__init__() (*angr.analyses.disassembly.Value method*), 1296
__init__() (*angr.analyses.dominance_frontier.DominanceFrontier method*), 1307
__init__() (*angr.analyses.find_objects_static.NewFunctionFinder method*), 1292
__init__() (*angr.analyses.find_objects_static.PossibleObject method*), 1292
__init__() (*angr.analyses.find_objects_static.StaticObjectFinder method*), 1292
__init__() (*angr.analyses.flirt.FlirtAnalysis method*), 1162
__init__() (*angr.analyses.flirt.FlirtSignature method*), 1162
__init__() (*angr.analyses.flirt.FlirtSignatureParsed method*), 1163
__init__() (*angr.analyses.forward_analysis.CallGraphVisitor method*), 931
__init__() (*angr.analyses.forward_analysis.ForwardAnalysis method*), 932
__init__() (*angr.analyses.forward_analysis.FunctionGraphVisitor method*), 932
__init__() (*angr.analyses.forward_analysis.LoopVisitor method*), 933
__init__() (*angr.analyses.forward_analysis.SingleNodeGraphVisitor method*), 934
__init__() (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method*), 935
__init__() (*angr.analyses.forward_analysis.forward_analysis.FunctionGraphVisitor method*), 937
__init__() (*angr.analyses.forward_analysis.forward_analysis.ForwardAn method*), 936
__init__() (*angr.analyses.forward_analysis.job_info.JobInfo method*), 937
__init__() (*angr.analyses.forward_analysis.visitors.CallGraphVisitor method*), 937
__init__() (*angr.analyses.forward_analysis.visitors.FunctionGraphVisito method*), 938
__init__() (*angr.analyses.forward_analysis.visitors.LoopVisitor method*), 939
__init__() (*angr.analyses.forward_analysis.visitors.SingleNodeGraphVis method*), 940
__init__() (*angr.analyses.forward_analysis.visitors.call_graph.CallGrap method*), 941
__init__() (*angr.analyses.forward_analysis.visitors.function_graph.Func method*), 941
__init__() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 942
__init__() (*angr.analyses.forward_analysis.visitors.loop.LoopVisitor method*), 944
__init__() (*angr.analyses.forward_analysis.visitors.single_node_graph.S method*), 945
__init__() (*angr.analyses.identifier.identify.FuncInfo method*), 1283
__init__() (*angr.analyses.identifier.identify.Identifier method*), 1283
__init__() (*angr.analyses.init_finder.InitializationFinder method*), 1307
__init__() (*angr.analyses.init_finder.SimEngineInitFinderVEX method*), 1307
__init__() (*angr.analyses.loop_analysis.LoopAnalysis method*), 1284
__init__() (*angr.analyses.loopfinder.Loop method*), 1283
__init__() (*angr.analyses.loopfinder.LoopFinder method*), 1284
__init__() (*angr.analyses.propagator.PropagatorAnalysis method*), 1175
__init__() (*angr.analyses.propagator.engine_base.SimEnginePropagator method*), 1176
__init__() (*angr.analyses.propagator.propagator.PropagatorAnalysis method*), 1178
__init__() (*angr.analyses.propagator.values.Top method*), 1175
__init__() (*angr.analyses.propagator.vex_vars.VEXMemVar method*), 1176
__init__() (*angr.analyses.propagator.vex_vars.VEXReg method*), 1176
__init__() (*angr.analyses.propagator.vex_vars.VEXTmp method*), 1176
__init__() (*angr.analyses.proximity_graph.BaseProxiNode method*), 1309
__init__() (*angr.analyses.proximity_graph.CallProxiNode method*), 1310

__init__() (angr.analyses.proximity_graph.FunctionProxiNode method), 1309
__init__() (angr.analyses.proximity_graph.IntegerProxiNode method), 1311
__init__() (angr.analyses.proximity_graph.ProximityGraphAnalysis method), 1311
__init__() (angr.analyses.proximity_graph.StringProxiNode method), 1310
__init__() (angr.analyses.proximity_graph.UnknownProxiNode method), 1311
__init__() (angr.analyses.proximity_graph.VariableProxiNode method), 1310
__init__() (angr.analyses.reaching_definitions.Atom method), 1179
__init__() (angr.analyses.reaching_definitions.ConstantSrc method), 1181
__init__() (angr.analyses.reaching_definitions.Definition method), 1181
__init__() (angr.analyses.reaching_definitions.FunctionCallData method), 1184
__init__() (angr.analyses.reaching_definitions.FunctionCallRelationships method), 1186
__init__() (angr.analyses.reaching_definitions.FunctionHandler method), 1186
__init__() (angr.analyses.reaching_definitions.GuardUse method), 1189
__init__() (angr.analyses.reaching_definitions.LiveDefinitions method), 1189
__init__() (angr.analyses.reaching_definitions.MemoryLocation method), 1197
__init__() (angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method), 1199
__init__() (angr.analyses.reaching_definitions.ReachingDefinitionsModel method), 1201
__init__() (angr.analyses.reaching_definitions.ReachingDefinitionsState method), 1204
__init__() (angr.analyses.reaching_definitions.Register method), 1211
__init__() (angr.analyses.reaching_definitions.Tmp method), 1212
__init__() (angr.analyses.reaching_definitions.call_trace.CallSite method), 1212
__init__() (angr.analyses.reaching_definitions.call_trace.CallTrace method), 1212
__init__() (angr.analyses.reaching_definitions.dep_graph.DepGraph method), 1217
__init__() (angr.analyses.reaching_definitions.engine_ail.SimEngineRDAIL method), 1238
__init__() (angr.analyses.reaching_definitions.engine_vex.SimEngineRDVEX method), 1213
__init__() (angr.analyses.reaching_definitions.function_handler.FunctionCallData method), 1225
__init__() (angr.analyses.reaching_definitions.function_handler.FunctionCallDataUnwrapped method), 1226
__init__() (angr.analyses.reaching_definitions.function_handler.Function method), 1227
__init__() (angr.analyses.reaching_definitions.function_handler.Function method), 1222
__init__() (angr.analyses.reaching_definitions.function_handler.Function method), 1227
__init__() (angr.analyses.reaching_definitions.heap_allocator.HeapAlloc method), 1220
__init__() (angr.analyses.reaching_definitions.rd_state.ReachingDefiniti method), 1231
__init__() (angr.analyses.reaching_definitions.reaching_definitions.Reac method), 1215
__init__() (angr.analyses.reaching_definitions.subject.Subject method), 1238
__init__() (angr.analyses.reassembler.BasicBlock method), 1300
__init__() (angr.analyses.reassembler.Data method), 1301
__init__() (angr.analyses.reassembler.DataLabel method), 1297
__init__() (angr.analyses.reassembler.FunctionLabel method), 1298
__init__() (angr.analyses.reassembler.Instruction method), 1299
__init__() (angr.analyses.reassembler.Label method), 1297
__init__() (angr.analyses.reassembler.NotypeLabel method), 1298
__init__() (angr.analyses.reassembler.ObjectLabel method), 1298
__init__() (angr.analyses.reassembler.Operand method), 1299
__init__() (angr.analyses.reassembler.Procedure method), 1300
__init__() (angr.analyses.reassembler.ProcedureChunk method), 1301
__init__() (angr.analyses.reassembler.Reassembler method), 1302
__init__() (angr.analyses.reassembler.Relocation method), 1302
__init__() (angr.analyses.reassembler.SymbolManager method), 1298
__init__() (angr.analyses.soot_class_hierarchy.NoConcreteDispatch method), 955
__init__() (angr.analyses.soot_class_hierarchy.SootClassHierarchy method), 955
__init__() (angr.analyses.soot_class_hierarchy.SootClassHierarchyError method), 955
__init__() (angr.analyses.stack_pointer_tracker.Constant method), 1243
__init__() (angr.analyses.stack_pointer_tracker.Eq method), 1244
__init__() (angr.analyses.stack_pointer_tracker.FrozenStackPointerTrack method), 1244

__init__() (angr.analyses.stack_pointer_tracker.OffsetVal method), 1243
__init__() (angr.analyses.stack_pointer_tracker.Register method), 1243
__init__() (angr.analyses.stack_pointer_tracker.StackPointerTracker method), 1245
__init__() (angr.analyses.stack_pointer_tracker.StackPointerTracker.State method), 1244
__init__() (angr.analyses.static_hooker.StaticHooker method), 1305
__init__() (angr.analyses.typehoon.Typehoon method), 1282
__init__() (angr.analyses.typehoon.simple_solver.ConstraintGraphNode method), 1261
__init__() (angr.analyses.typehoon.simple_solver.RecursiveRefNode method), 1260
__init__() (angr.analyses.typehoon.simple_solver.SimpleSolver method), 1262
__init__() (angr.analyses.typehoon.simple_solver.Sketch method), 1260
__init__() (angr.analyses.typehoon.simple_solver.SketchNodeInfo method), 1259
__init__() (angr.analyses.typehoon.translator.SimTypeTemplate method), 1265
__init__() (angr.analyses.typehoon.translator.TypeTranslator method), 1265
__init__() (angr.analyses.typehoon.typeconsts.Array method), 1278
__init__() (angr.analyses.typehoon.typeconsts.Enum method), 1280
__init__() (angr.analyses.typehoon.typeconsts.EnumVariant method), 1279
__init__() (angr.analyses.typehoon.typeconsts.Function method), 1280
__init__() (angr.analyses.typehoon.typeconsts.IntVar method), 1277
__init__() (angr.analyses.typehoon.typeconsts.Pointer method), 1277
__init__() (angr.analyses.typehoon.typeconsts.Pointer32 method), 1278
__init__() (angr.analyses.typehoon.typeconsts.Pointer64 method), 1278
__init__() (angr.analyses.typehoon.typeconsts.RustEnum method), 1279
__init__() (angr.analyses.typehoon.typeconsts.Struct method), 1279
__init__() (angr.analyses.typehoon.typeconsts.TypeConstant method), 1274
__init__() (angr.analyses.typehoon.typeconsts.TypeVariableReference method), 1281
__init__() (angr.analyses.typehoon.typehoon.Typehoon method), 1273
__init__() (angr.analyses.typehoon.typevars.Add method), 1267
__init__() (angr.analyses.typehoon.typevars.AddN method), 1272
__init__() (angr.analyses.typehoon.typevars.BaseLabel method), 1271
__init__() (angr.analyses.typehoon.typevars.ConvertTo method), 1272
__init__() (angr.analyses.typehoon.typevars.DerivedTypeVariable method), 1270
__init__() (angr.analyses.typehoon.typevars.Equivalence method), 1266
__init__() (angr.analyses.typehoon.typevars.Existence method), 1267
__init__() (angr.analyses.typehoon.typevars.FuncIn method), 1271
__init__() (angr.analyses.typehoon.typevars.FuncOut method), 1271
__init__() (angr.analyses.typehoon.typevars.HasField method), 1272
__init__() (angr.analyses.typehoon.typevars.ReinterpretAs method), 1272
__init__() (angr.analyses.typehoon.typevars.Sub method), 1268
__init__() (angr.analyses.typehoon.typevars.SubN method), 1272
__init__() (angr.analyses.typehoon.typevars.Subtype method), 1267
__init__() (angr.analyses.typehoon.typevars.TypeVariable method), 1269
__init__() (angr.analyses.typehoon.typevars.TypeVariableManager method), 1268
__init__() (angr.analyses.typehoon.typevars.TypeVariables method), 1271
__init__() (angr.analyses.variable_recovery.VariableRecovery method), 1258
__init__() (angr.analyses.variable_recovery.VariableRecoveryFast method), 1259
__init__() (angr.analyses.variable_recovery.annotations.StackLocationA method), 1246
__init__() (angr.analyses.variable_recovery.annotations.VariableSource method), 1246
__init__() (angr.analyses.variable_recovery.engine_ail.SimEngineVRAIL method), 1256
__init__() (angr.analyses.variable_recovery.engine_base.RichR method), 1256
__init__() (angr.analyses.variable_recovery.engine_base.SimEngineVRB method), 1257
__init__() (angr.analyses.variable_recovery.engine_vex.SimEngineVRVE method), 1256
__init__() (angr.analyses.variable_recovery.irsb_scanner.VEXIRSBScan method), 1257
__init__() (angr.analyses.variable_recovery.variable_recovery.VariableR method), 1255
__init__() (angr.analyses.variable_recovery.variable_recovery.VariableR method), 1254

__init__() (angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBaseState method), 1247
__init__() (angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase method), 1248
__init__() (angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method), 1248
__init__() (angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFastBase method), 1253
__init__() (angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFastState method), 1252
__init__() (angr.analyses.veritesting.CallTracingFilter method), 1284
__init__() (angr.analyses.veritesting.Veritesting method), 1285
__init__() (angr.analyses.vfg.AnalysisTask method), 1286
__init__() (angr.analyses.vfg.CallAnalysis method), 1287
__init__() (angr.analyses.vfg.FunctionAnalysis method), 1287
__init__() (angr.analyses.vfg.PendingJob method), 1286
__init__() (angr.analyses.vfg.VFG method), 1289
__init__() (angr.analyses.vfg.VFGJob method), 1285
__init__() (angr.analyses.vfg.VFGNode method), 1288
__init__() (angr.analyses.vsa_ddg.DefUseChain method), 1290
__init__() (angr.analyses.vsa_ddg.VSA_DDG method), 1291
__init__() (angr.analyses.vtable.Vtable method), 1291
__init__() (angr.analyses.vtable.VtableFinder method), 1291
__init__() (angr.analyses.xrefs.SimEngineXRefsVEX method), 1308
__init__() (angr.analyses.xrefs.XRefsAnalysis method), 1308
__init__() (angr.angrdb.AngrDB method), 1018
__init__() (angr.angrdb.db.AngrDB method), 1020
__init__() (angr.angrdb.models.DbCFGModel method), 1023
__init__() (angr.angrdb.models.DbComment method), 1025
__init__() (angr.angrdb.models.DbFunction method), 1023
__init__() (angr.angrdb.models.DbInformation method), 1021
__init__() (angr.angrdb.models.DbKnowledgeBase method), 1022
__init__() (angr.angrdb.models.DbLabel method), 1025
__init__() (angr.angrdb.models.DbObject method), 1022
__init__() (angr.angrdb.models.DbStructuredCode method), 1024
__init__() (angr.angrdb.models.DbVariableCollection method), 1023
__init__() (angr.angrdb.models.DbXRefs method), 1024
__init__() (angr.angrdb.serialize.JSONLoader.LoadArgsJSONDecoder method), 1029
__init__() (angr.analyses.cfg.CFGFastAnnotatedCFG method), 1321
__init__() (angr.analyses.cfg.CFGFast method), 1320
__init__() (angr.block.Block method), 233
__init__() (angr.block.CapstoneInsn method), 232
__init__() (angr.block.DisassemblerBlock method), 232
__init__() (angr.block.PCodeInsn method), 233
__init__() (angr.block.SootBlock method), 234
__init__() (angr.callable.Callable method), 710
__init__() (angr.calling_conventions.AllocHelper method), 658
__init__() (angr.calling_conventions.ArgSession method), 663
__init__() (angr.calling_conventions.PointerWrapper method), 658
__init__() (angr.calling_conventions.SerializableCounter method), 659
__init__() (angr.calling_conventions.SerializableListIterator method), 659
__init__() (angr.calling_conventions.SimArrayArg method), 662
__init__() (angr.calling_conventions.SimCC method), 663
__init__() (angr.calling_conventions.SimCC.ArgSession method), 664
__init__() (angr.calling_conventions.SimCCUsercall method), 667
__init__() (angr.calling_conventions.SimComboArg method), 661
__init__() (angr.calling_conventions.SimFunctionArgument method), 659
__init__() (angr.calling_conventions.SimLyingRegArg method), 667
__init__() (angr.calling_conventions.SimReferenceArgument method), 662
__init__() (angr.calling_conventions.SimRegArg method), 660
__init__() (angr.calling_conventions.SimStackArg method), 661
__init__() (angr.calling_conventions.SimStructArg method), 662
__init__() (angr.calling_conventions.UsercallArgSession method), 663
__init__() (angr.code_location.AILCodeLocation method), 867
__init__() (angr.code_location.CodeLocation method), 865

__init__() (angr.code_location.ExternalCodeLocation method), 866
__init__() (angr.codenode.BlockNode method), 1323
__init__() (angr.codenode.CodeNode method), 1323
__init__() (angr.codenode.FuncNode method), 1324
__init__() (angr.codenode.HookNode method), 1324
__init__() (angr.codenode.SootBlockNode method), 1323
__init__() (angr.concretization_strategies.SimConcretizationStrategy method), 452
__init__() (angr.concretization_strategies.SimConcretizationStrategyControlledData method), 453
__init__() (angr.concretization_strategies.SimConcretizationStrategyMax method), 453
__init__() (angr.concretization_strategies.SimConcretizationStrategyNonzero method), 453
__init__() (angr.concretization_strategies.SimConcretizationStrategyNonzeroRange method), 454
__init__() (angr.concretization_strategies.SimConcretizationStrategyNoreplacements method), 454
__init__() (angr.concretization_strategies.SimConcretizationStrategyNoreplacementsRange method), 454
__init__() (angr.concretization_strategies.SimConcretizationStrategyRange method), 455
__init__() (angr.concretization_strategies.SimConcretizationStrategySolutions method), 455
__init__() (angr.concretization_strategies.SimConcretizationStrategyUnlimitedRange method), 455
__init__() (angr.concretization_strategies.controlled_data.SimConcretizationStrategyControlledData method), 543
__init__() (angr.concretization_strategies.eval.SimConcretizationStrategyEval method), 541
__init__() (angr.concretization_strategies.max.SimConcretizationStrategyMax method), 542
__init__() (angr.concretization_strategies.nonzero_range.SimConcretizationStrategyNonzeroRange method), 542
__init__() (angr.concretization_strategies.norepeats.SimConcretizationStrategyNoreplacements method), 541
__init__() (angr.concretization_strategies.norepeats_range.SimConcretizationStrategyNoreplacementsRange method), 543
__init__() (angr.concretization_strategies.range.SimConcretizationStrategyRange method), 542
__init__() (angr.concretization_strategies.solutions.SimConcretizationStrategySolutions method), 542
__init__() (angr.concretization_strategies.unlimited_range.SimConcretizationStrategyUnlimitedRange method), 544
__init__() (angr.distributed.Server method), 1362
__init__() (angr.distributed.server.Server method), 1362
__init__() (angr.distributed.worker.BadStatesDropper method), 1363
__init__() (angr.distributed.worker.ExplorationStatusNotifier method), 1363
__init__() (angr.distributed.worker.Worker method), 1363
__init__() (angr.engines.SimEngine method), 589
__init__() (angr.engines.SimEngineUnicorn method), 590
__init__() (angr.engines.SimSuccessors method), 591
__init__() (angr.engines.SuccessorsEngine method), 593
__init__() (angr.engines.TrackActionsMixin method), 594
__init__() (angr.engines.engine.SimEngine method), 597
__init__() (angr.engines.light.ArithmeticExpression method), 1167
__init__() (angr.engines.light.RegisterOffset method), 1167
__init__() (angr.engines.light.SimEngineLight method), 1168
__init__() (angr.engines.light.SimEngineLightAIL method), 1169
__init__() (angr.engines.light.SimEngineLightVEX method), 1170
__init__() (angr.engines.light.SpOffset method), 1170
__init__() (angr.engines.light.data.ArithmeticExpression method), 1166
__init__() (angr.engines.light.data.RegisterOffset method), 1167
__init__() (angr.engines.light.data.SpOffset method), 1167
__init__() (angr.engines.light.engine.CallingConvention method), 1171
__init__() (angr.engines.light.engine.IRTop method), 1171
__init__() (angr.engines.light.engine.SimEngineLight method), 1171
__init__() (angr.engines.light.engine.SimEngineLightAIL method), 1174
__init__() (angr.engines.light.engine.SimEngineLightVEX method), 1173
__init__() (angr.engines.pcode.PcodeMixin method), 604
__init__() (angr.engines.pcode.behavior.BehaviorFactory method), 635
__init__() (angr.engines.pcode.behavior.OpBehavior method), 616
__init__() (angr.engines.pcode.behavior.OpBehaviorBoolAnd method), 629
__init__() (angr.engines.pcode.behavior.OpBehaviorBoolNegate method), 629
__init__() (angr.engines.pcode.behavior.OpBehaviorBoolOr method), 630
__init__() (angr.engines.pcode.behavior.OpBehaviorBoolXor method), 629
__init__() (angr.engines.pcode.behavior.OpBehaviorCopy method), 617

__init__() (*angr.engines.pcode.behavior.OpBehaviorEqual method*), 617
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatAbs method*), 632
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatAdd method*), 631
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatCeil method*), 633
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatDiv method*), 632
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatEqual method*), 630
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatFloat2Float method*), 633
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatFloor method*), 634
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatInt2Float method*), 633
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatLess method*), 631
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatLessEqual method*), 631
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatMult method*), 632
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatNan method*), 631
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatNeg method*), 632
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatNotEqual method*), 631
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatRound method*), 634
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatSqrt method*), 633
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatSub method*), 632
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatTrunc method*), 633
__init__() (*angr.engines.pcode.behavior.OpBehaviorInt2Comp method*), 623
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntAdd method*), 621
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntAnd method*), 624
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntCarry method*), 622
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntDiv method*), 627
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntLeft method*), 625
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntLess method*), 619
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual method*), 620
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntMult method*), 626
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntNegate method*), 623
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntOr method*), 625
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntRem method*), 628
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntRight method*), 626
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSborrow method*), 623
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntScarry method*), 622
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSdiv method*), 627
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSext method*), 620
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSless method*), 618
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual method*), 619
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSrem method*), 628
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSright method*), 626
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSub method*), 621
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntXor method*), 624
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntZext method*), 620
__init__() (*angr.engines.pcode.behavior.OpBehaviorLzcount method*), 635
__init__() (*angr.engines.pcode.behavior.OpBehaviorNotEqual method*), 618
__init__() (*angr.engines.pcode.behavior.OpBehaviorPiece method*), 634
__init__() (*angr.engines.pcode.behavior.OpBehaviorPopcount method*), 635
__init__() (*angr.engines.pcode.behavior.OpBehaviorSubpiece method*), 634
__init__() (*angr.engines.pcode.emulate.PcodeEmulatorMixin method*), 615
__init__() (*angr.engines.pcode.engine.HeavyPcodeMixin method*), 605
__init__() (*angr.engines.pcode.lifter.ExitStatement method*), 605
__init__() (*angr.engines.pcode.lifter.IRSB method*), 607
__init__() (*angr.engines.pcode.lifter.Lifter method*), 610
__init__() (*angr.engines.pcode.lifter.PcodeBasicBlockLifter method*), 612

__init__() (angr.engines.pcode.lifter.PcodeDisassemblerInsn method), 606
__init__() (angr.engines.pcode.lifter.PcodeLifterEngineMixin method), 614
__init__() (angr.engines.successors.SimSuccessors method), 595
__init__() (angr.engines.successors.SuccessorsEngine method), 596
__init__() (angr.engines.unicorn.SimEngineUnicorn method), 603
__init__() (angr.engines.vex.TrackActionsMixin method), 600
__init__() (angr.engines.vex.VEXLifter method), 600
__init__() (angr.engines.vex.VEXMixin method), 601
__init__() (angr.engines.vex.VEXSlicingMixin method), 601
__init__() (angr.errors.AngrMissingTypeError method), 1356
__init__() (angr.errors.SimMemoryMissingError method), 1358
__init__() (angr.errors.SimReliftException method), 1360
__init__() (angr.errors.SimSegfaultException method), 1361
__init__() (angr.errors.SimUninitializedAccessError method), 1359
__init__() (angr.exploration_techniques.CallFunctionGoal method), 553
__init__() (angr.exploration_techniques.DFS method), 552
__init__() (angr.exploration_techniques.Director method), 554
__init__() (angr.exploration_techniques.DrillerCore method), 554
__init__() (angr.exploration_techniques.ExecuteAddressGoal method), 555
__init__() (angr.exploration_techniques.ExplorationTechnique method), 555
__init__() (angr.exploration_techniques.Explorer method), 557
__init__() (angr.exploration_techniques.LengthLimiter method), 558
__init__() (angr.exploration_techniques.LocalLoopSeer method), 558
__init__() (angr.exploration_techniques.LoopSeer method), 559
__init__() (angr.exploration_techniques.ManualMergepoint method), 560
__init__() (angr.exploration_techniques.MemoryWatcher method), 561
__init__() (angr.exploration_techniques.Oppologist method), 561
__init__() (angr.exploration_techniques.Slicecutor method), 562
__init__() (angr.exploration_techniques.Spiller method), 563
__init__() (angr.exploration_techniques.StochasticSearch method), 564
__init__() (angr.exploration_techniques.Suggestions method), 564
__init__() (angr.exploration_techniques.TechniqueBuilder method), 565
__init__() (angr.exploration_techniques.Threading method), 565
__init__() (angr.exploration_techniques.Timeout method), 565
__init__() (angr.exploration_techniques.Tracer method), 566
__init__() (angr.exploration_techniques.UniqueSearch method), 568
__init__() (angr.exploration_techniques.Veritesting method), 568
__init__() (angr.exploration_techniques.dfs.DFS method), 569
__init__() (angr.exploration_techniques.director.BaseGoal method), 579
__init__() (angr.exploration_techniques.director.CallFunctionGoal method), 580
__init__() (angr.exploration_techniques.director.Director method), 581
__init__() (angr.exploration_techniques.director.ExecuteAddressGoal method), 579
__init__() (angr.exploration_techniques.driller_core.DrillerCore method), 577
__init__() (angr.exploration_techniques.explorer.Explorer method), 570
__init__() (angr.exploration_techniques.lengthlimiter.LengthLimiter method), 571
__init__() (angr.exploration_techniques.local_loop_seer.LocalLoopSeer method), 583
__init__() (angr.exploration_techniques.loop_seer.LoopSeer method), 582
__init__() (angr.exploration_techniques.manual_mergepoint.ManualMer method), 571
__init__() (angr.exploration_techniques.memory_watcher.MemoryWatch method), 586
__init__() (angr.exploration_techniques.oppologist.Oppologist method), 581
__init__() (angr.exploration_techniques.slicecutor.Slicecutor method), 578
__init__() (angr.exploration_techniques.spiller.PickledStatesDb method), 572
__init__() (angr.exploration_techniques.spiller.PickledStatesList method), 572
__init__() (angr.exploration_techniques.spiller.Spiller method), 573
__init__() (angr.exploration_techniques.spiller_db.PickledState method), 574

__init__() (*angr.exploration_techniques.stochastic.StochasticSearch method*), 584
__init__() (*angr.exploration_techniques.suggestions.Suggestions method*), 586
__init__() (*angr.exploration_techniques.tech_builder.TechniqueBuilder method*), 585
__init__() (*angr.exploration_techniques.threading.Threading method*), 574
__init__() (*angr.exploration_techniques.timeout.Timeout method*), 569
__init__() (*angr.exploration_techniques.tracer.RepHook method*), 575
__init__() (*angr.exploration_techniques.tracer.Tracer method*), 576
__init__() (*angr.exploration_techniques.tracer.TracerDesyncError method*), 575
__init__() (*angr.exploration_techniques.unique.UniqueSearch method*), 584
__init__() (*angr.exploration_techniques.veritesting.Veritesting method*), 574
__init__() (*angr.factory.AngrObjectFactory method*), 228
__init__() (*angr.flirt.FlirtSignature method*), 1342
__init__() (*angr.keyed_region.KeyedRegion method*), 868
__init__() (*angr.keyed_region.RegionObject method*), 868
__init__() (*angr.keyed_region.StoredObject method*), 868
__init__() (*angr.knowledge_base.KnowledgeBase method*), 711
__init__() (*angr.knowledge_plugins.CFGManager method*), 712
__init__() (*angr.knowledge_plugins.CallsitePrototypes method*), 713
__init__() (*angr.knowledge_plugins.CustomStrings method*), 714
__init__() (*angr.knowledge_plugins.DebugVariableManager method*), 715
__init__() (*angr.knowledge_plugins.Function method*), 716
__init__() (*angr.knowledge_plugins.FunctionManager method*), 725
__init__() (*angr.knowledge_plugins.IndirectJumps method*), 730
__init__() (*angr.knowledge_plugins.KeyDefinitionManager method*), 730
__init__() (*angr.knowledge_plugins.KnowledgeBasePlugin method*), 730
__init__() (*angr.knowledge_plugins.Labels method*), 731
__init__() (*angr.knowledge_plugins.Obfuscations method*), 731
__init__() (*angr.knowledge_plugins.PatchManager method*), 731
__init__() (*angr.knowledge_plugins.PropagationManager method*), 733
__init__() (*angr.knowledge_plugins.RuntimeDb method*), 734
__init__() (*angr.knowledge_plugins.StructuredCodeManager method*), 734
__init__() (*angr.knowledge_plugins.TypesStore method*), 735
__init__() (*angr.knowledge_plugins.VariableManager method*), 735
__init__() (*angr.knowledge_plugins.XRefManager method*), 736
__init__() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototype method*), 739
__init__() (*angr.knowledge_plugins.cfg.BlockID method*), 740
__init__() (*angr.knowledge_plugins.cfg.CFGENode method*), 741
__init__() (*angr.knowledge_plugins.cfg.CFGManager method*), 742
__init__() (*angr.knowledge_plugins.cfg.CFGModel method*), 742
__init__() (*angr.knowledge_plugins.cfg.CFGNode method*), 750
__init__() (*angr.knowledge_plugins.cfg.IndirectJump method*), 752
__init__() (*angr.knowledge_plugins.cfg.MemoryData method*), 754
__init__() (*angr.knowledge_plugins.cfg.cfg_manager.CFGManager method*), 766
__init__() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 756
__init__() (*angr.knowledge_plugins.cfg.cfg_node.CFGENode method*), 768
__init__() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 767
__init__() (*angr.knowledge_plugins.cfg.cfg_node.CFGNodeCreationFail method*), 766
__init__() (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump method*), 771
__init__() (*angr.knowledge_plugins.cfg.indirect_jump.JumptableInfo method*), 770
__init__() (*angr.knowledge_plugins.cfg.memory_data.MemoryData method*), 765
__init__() (*angr.knowledge_plugins.debug_variables.DebugVariable method*), 820
__init__() (*angr.knowledge_plugins.debug_variables.DebugVariableCon method*), 820
__init__() (*angr.knowledge_plugins.debug_variables.DebugVariableMan method*), 821
__init__() (*angr.knowledge_plugins.functions.Function method*), 776
__init__() (*angr.knowledge_plugins.functions.FunctionManager*

method), 785
__init__() (*angr.knowledge_plugins.functions.function.Function*
method), 801
__init__() (*angr.knowledge_plugins.functions.function.FunctionInfo*
method), 800
__init__() (*angr.knowledge_plugins.functions.function_manager.FunctionDict*
method), 791
__init__() (*angr.knowledge_plugins.functions.function_manager.FunctionDictBase*
method), 790
__init__() (*angr.knowledge_plugins.functions.function_manager.FunctionManager*
method), 794
__init__() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict*
method), 793
__init__() (*angr.knowledge_plugins.functions.soot_function.SootFunction*
method), 810
__init__() (*angr.knowledge_plugins.indirect_jumps.IndirectJumps*
method), 775
__init__() (*angr.knowledge_plugins.key_definitions.Definition*
method), 823
__init__() (*angr.knowledge_plugins.key_definitions.KeyDefinitionManager*
method), 823
__init__() (*angr.knowledge_plugins.key_definitions.LiveDefinitions*
method), 824
__init__() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel*
method), 832
__init__() (*angr.knowledge_plugins.key_definitions.Uses*
method), 834
__init__() (*angr.knowledge_plugins.key_definitions.atoms.Atom*
method), 836
__init__() (*angr.knowledge_plugins.key_definitions.atoms.Constant*
method), 838
__init__() (*angr.knowledge_plugins.key_definitions.atoms.GuardUse*
method), 838
__init__() (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation*
method), 840
__init__() (*angr.knowledge_plugins.key_definitions.atoms.Register*
method), 839
__init__() (*angr.knowledge_plugins.key_definitions.atoms.Tmp*
method), 839
__init__() (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable*
method), 840
__init__() (*angr.knowledge_plugins.key_definitions.definitions.Definition*
method), 843
__init__() (*angr.knowledge_plugins.key_definitions.definitions.DefinitionPlaceholder*
method), 842
__init__() (*angr.knowledge_plugins.key_definitions.environment.Environment*
method), 844
__init__() (*angr.knowledge_plugins.key_definitions.heap_address.HeapAddress*
method), 845
__init__() (*angr.knowledge_plugins.key_definitions.key_definition_manager.KeyDefinitionManager*
method), 845
__init__() (*angr.knowledge_plugins.key_definitions.key_definition_manager.RSAObserverControl*
method), 845
__init__() (*angr.knowledge_plugins.key_definitions.live_definitions.DefinitionAnnotation*
method), 846
__init__() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveD*
method), 847
__init__() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingD*
method), 855
__init__() (*angr.knowledge_plugins.key_definitions.tag.FunctionTag*
method), 857
__init__() (*angr.knowledge_plugins.key_definitions.tag.Tag*
method), 857
__init__() (*angr.knowledge_plugins.key_definitions.uses.Uses*
method), 859
__init__() (*angr.knowledge_plugins.labels.Labels*
method), 776
__init__() (*angr.knowledge_plugins.patches.Patch*
method), 737
__init__() (*angr.knowledge_plugins.patches.PatchManager*
method), 737
__init__() (*angr.knowledge_plugins.plugin.KnowledgeBasePlugin*
method), 738
__init__() (*angr.knowledge_plugins.propagations.PropagationManager*
method), 773
__init__() (*angr.knowledge_plugins.propagations.PropagationModel*
method), 774
__init__() (*angr.knowledge_plugins.structured_code.StructuredCodeMan*
method), 822
__init__() (*angr.knowledge_plugins.types.TypesStore*
method), 772
__init__() (*angr.knowledge_plugins.variables.VariableManager*
method), 811
__init__() (*angr.knowledge_plugins.variables.variable_access.VariableA*
method), 812
__init__() (*angr.knowledge_plugins.variables.variable_manager.LiveVar*
method), 813
__init__() (*angr.knowledge_plugins.variables.variable_manager.Variabl*
method), 819
__init__() (*angr.knowledge_plugins.variables.variable_manager.Variabl*
method), 813
__init__() (*angr.knowledge_plugins.xrefs.XRef*
method), 861
__init__() (*angr.knowledge_plugins.xrefs.XRefManager*
method), 862
__init__() (*angr.knowledge_plugins.xrefs.xref.XRef*
method), 863
__init__() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager*
method), 864
__init__() (*angr.misc.plugins.PluginHub method*), 235
__init__() (*angr.misc.plugins.PluginPreset method*), 236
__init__() (*angr.procedures.definitions.SimLibrary method*), 650
__init__() (*angr.procedures.definitions.SimSyscallLibrary method*), 655
__init__() (*angr.procedures.definitions.SimTypeCollection method*), 649

__init__() (*angr.procedures.stubs.format_parser.FormatSpecifier method*), 647
__init__() (*angr.procedures.stubs.format_parser.FormatString method*), 646
__init__() (*angr.project.Project method*), 224
__init__() (*angr.sim_manager.ErrorRecord method*), 551
__init__() (*angr.sim_manager.SimulationManager method*), 545
__init__() (*angr.sim_procedure.SimProcedure method*), 644
__init__() (*angr.sim_state.SimState method*), 240
__init__() (*angr.sim_state_options.SimStateOptions method*), 245
__init__() (*angr.sim_state_options.StateOption method*), 242
__init__() (*angr.sim_type.NamedTypeMixin method*), 688
__init__() (*angr.sim_type.SimCppClass method*), 706
__init__() (*angr.sim_type.SimCppClassValue method*), 707
__init__() (*angr.sim_type.SimStruct method*), 701
__init__() (*angr.sim_type.SimStructValue method*), 702
__init__() (*angr.sim_type.SimType method*), 686
__init__() (*angr.sim_type.SimTypeArray method*), 696
__init__() (*angr.sim_type.SimTypeBitfield method*), 704
__init__() (*angr.sim_type.SimTypeBool method*), 693
__init__() (*angr.sim_type.SimTypeChar method*), 692
__init__() (*angr.sim_type.SimTypeCppFunction method*), 699
__init__() (*angr.sim_type.SimTypeDouble method*), 700
__init__() (*angr.sim_type.SimTypeEnum method*), 703
__init__() (*angr.sim_type.SimTypeFd method*), 693
__init__() (*angr.sim_type.SimTypeFloat method*), 700
__init__() (*angr.sim_type.SimTypeFunction method*), 698
__init__() (*angr.sim_type.SimTypeInt method*), 690
__init__() (*angr.sim_type.SimTypeLength method*), 700
__init__() (*angr.sim_type.SimTypeNum method*), 690
__init__() (*angr.sim_type.SimTypeNumOffset method*), 707
__init__() (*angr.sim_type.SimTypePointer method*), 694
__init__() (*angr.sim_type.SimTypeRef method*), 708
__init__() (*angr.sim_type.SimTypeReference method*), 695
__init__() (*angr.sim_type.SimTypeReg method*), 689
__init__() (*angr.sim_type.SimTypeString method*), 697
__init__() (*angr.sim_type.SimTypeTop method*), 689
__init__() (*angr.sim_type.SimTypeWString method*), 697
__init__() (*angr.sim_type.SimTypeWideChar method*), 693
__init__() (*angr.sim_type.SimUnion method*), 702
__init__() (*angr.sim_type.SimUnionValue method*), 703
__init__() (*angr.sim_type.TypeRef method*), 688
__init__() (*angr.sim_variable.SimComboRegisterVariable method*), 684
__init__() (*angr.sim_variable.SimConstantVariable method*), 681
__init__() (*angr.sim_variable.SimMemoryVariable method*), 685
__init__() (*angr.sim_variable.SimRegisterVariable method*), 683
__init__() (*angr.sim_variable.SimStackVariable method*), 685
__init__() (*angr.sim_variable.SimTemporaryVariable method*), 682
__init__() (*angr.sim_variable.SimVariable method*), 680
__init__() (*angr.simos.SimCGC method*), 1325
__init__() (*angr.simos.SimJavaVM method*), 1325
__init__() (*angr.simos.SimLinux method*), 1327
__init__() (*angr.simos.SimOS method*), 1328
__init__() (*angr.simos.SimSnimmucNxp method*), 1330
__init__() (*angr.simos.SimUserland method*), 1330
__init__() (*angr.simos.SimWindows method*), 1331
__init__() (*angr.simos.cgc.SimCGC method*), 1337
__init__() (*angr.simos.javavm.SimJavaVM method*), 1340
__init__() (*angr.simos.linux.SimLinux method*), 1335
__init__() (*angr.simos.simos.GlobalDescriptorTable method*), 1335
__init__() (*angr.simos.simos.SimOS method*), 1333
__init__() (*angr.simos.userland.SimUserland method*), 1337
__init__() (*angr.simos.windows.SimWindows method*), 1338
__init__() (*angr.slicer.SimLightState method*), 1320
__init__() (*angr.slicer.SimSlicer method*), 1321
__init__() (*angr.state_hierarchy.StateHierarchy method*), 551
__init__() (*angr.state_plugins.CallStack method*), 248
__init__() (*angr.state_plugins.GDB method*), 247
__init__() (*angr.state_plugins.InspectAttrs method*), 255
__init__() (*angr.state_plugins.PTChunk method*), 257
__init__() (*angr.state_plugins.PTChunkIterator method*), 259
__init__() (*angr.state_plugins.SimAction method*), 263
__init__() (*angr.state_plugins.SimActionConstraint method*), 263

__init__() (*angr.state_plugins.SimActionData method*), 263
__init__() (*angr.state_plugins.SimActionExit method*), 264
__init__() (*angr.state_plugins.SimActionObject method*), 264
__init__() (*angr.state_plugins.SimActionOperation method*), 266
__init__() (*angr.state_plugins.SimDebugVariable method*), 266
__init__() (*angr.state_plugins.SimEvent method*), 268
__init__() (*angr.state_plugins.SimFilesystem method*), 268
__init__() (*angr.state_plugins.SimHeapBase method*), 270
__init__() (*angr.state_plugins.SimHeapBrk method*), 271
__init__() (*angr.state_plugins.SimHeapPTMalloc method*), 273
__init__() (*angr.state_plugins.SimHostFilesystem method*), 276
__init__() (*angr.state_plugins.SimInspector method*), 276
__init__() (*angr.state_plugins.SimJavaVmClassloader method*), 279
__init__() (*angr.state_plugins.SimLightRegisters method*), 281
__init__() (*angr.state_plugins.SimMemView method*), 282
__init__() (*angr.state_plugins.SimSolver method*), 288
__init__() (*angr.state_plugins.SimStateCGC method*), 297
__init__() (*angr.state_plugins.SimStateEdgeHitmap method*), 298
__init__() (*angr.state_plugins.SimStateGlobals method*), 299
__init__() (*angr.state_plugins.SimStateHistory method*), 301
__init__() (*angr.state_plugins.SimStateIcicle method*), 304
__init__() (*angr.state_plugins.SimStateJNIReferences method*), 305
__init__() (*angr.state_plugins.SimStateLibc method*), 309
__init__() (*angr.state_plugins.SimStateLog method*), 311
__init__() (*angr.state_plugins.SimStateLoopData method*), 312
__init__() (*angr.state_plugins.SimStatePlugin method*), 314
__init__() (*angr.state_plugins.SimStatePreconstrainer method*), 316
__init__() (*angr.state_plugins.SimStateScratch method*), 318
__init__() (*angr.state_plugins.SimSymbolizer method*), 320
__init__() (*angr.state_plugins.SimSystemPosix method*), 322
__init__() (*angr.state_plugins.SimUCManager method*), 325
__init__() (*angr.state_plugins.StructMode method*), 326
__init__() (*angr.state_plugins.Unicorn method*), 327
__init__() (*angr.state_plugins.callstack.CallStack method*), 371
__init__() (*angr.state_plugins.callstack.CallStackAction method*), 374
__init__() (*angr.state_plugins.cgc.SimStateCGC method*), 380
__init__() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 421
__init__() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 358
__init__() (*angr.state_plugins.filesystem.SimFilesystem method*), 356
__init__() (*angr.state_plugins.filesystem.SimHostFilesystem method*), 360
__init__() (*angr.state_plugins.gdb.GDB method*), 379
__init__() (*angr.state_plugins.globals.SimStateGlobals method*), 386
__init__() (*angr.state_plugins.heap.PTChunk method*), 404
__init__() (*angr.state_plugins.heap.PTChunkIterator method*), 406
__init__() (*angr.state_plugins.heap.SimHeapBase method*), 406
__init__() (*angr.state_plugins.heap.SimHeapBrk method*), 407
__init__() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 409
__init__() (*angr.state_plugins.heap.heap_base.SimHeapBase method*), 411
__init__() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 412
__init__() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 414
__init__() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 416
__init__() (*angr.state_plugins.heap.heap_ptmalloc.PTChunkIterator method*), 417
__init__() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 418
__init__() (*angr.state_plugins.history.LambdaAttrIter method*), 378
__init__() (*angr.state_plugins.history.LambdaIterIter method*), 378
__init__() (*angr.state_plugins.history.SimStateHistory method*), 375

__init__() (angr.state_plugins.history.TreeIter method), 378
__init__() (angr.state_plugins.inspect.BP method), 339
__init__() (angr.state_plugins.inspect.InspectAttrs method), 336
__init__() (angr.state_plugins.inspect.SimInspector method), 340
__init__() (angr.state_plugins.javavm_classloader.SimJavaVmClassloader method), 401
__init__() (angr.state_plugins.jni_references.SimStateJNIReferences method), 402
__init__() (angr.state_plugins.libc.SimStateLibc method), 345
__init__() (angr.state_plugins.light_registers.SimLightRegisters method), 375
__init__() (angr.state_plugins.log.SimStateLog method), 370
__init__() (angr.state_plugins.loop_data.SimStateLoopData method), 399
__init__() (angr.state_plugins.plugin.SimStatePlugin method), 329
__init__() (angr.state_plugins.posix.SimSystemPosix method), 352
__init__() (angr.state_plugins.preconstrainer.SimStatePreconstrainer method), 390
__init__() (angr.state_plugins.scratch.SimStateScratch method), 388
__init__() (angr.state_plugins.sim_action.SimAction method), 638
__init__() (angr.state_plugins.sim_action.SimActionConstraint method), 639
__init__() (angr.state_plugins.sim_action.SimActionData method), 639
__init__() (angr.state_plugins.sim_action.SimActionExit method), 639
__init__() (angr.state_plugins.sim_action.SimActionOperation method), 639
__init__() (angr.state_plugins.sim_action_object.SimActionObject method), 640
__init__() (angr.state_plugins.sim_event.SimEvent method), 642
__init__() (angr.state_plugins.solver.SimSolver method), 361
__init__() (angr.state_plugins.symbolizer.SimSymbolizer method), 420
__init__() (angr.state_plugins.trace_additions.ChallRespInfo method), 382
__init__() (angr.state_plugins.trace_additions.FormatInfoDynamicConstraint method), 382
__init__() (angr.state_plugins.trace_additions.FormatInfoInitStr method), 382
__init__() (angr.state_plugins.trace_additions.FormatInfoSymbolic method), 382
__init__() (angr.state_plugins.trace_additions.ZenPlugin method), 384
__init__() (angr.state_plugins.uc_manager.SimUCManager method), 387
__init__() (angr.state_plugins.unicorn_engine.AggressiveConcretization method), 396
__init__() (angr.state_plugins.unicorn_engine.Unicorn method), 397
__init__() (angr.state_plugins.unicorn_engine.Uniwrapper method), 396
__init__() (angr.state_plugins.view.SimMemView method), 428
__init__() (angr.state_plugins.view.StructMode method), 431
__init__() (angr.storage.SimFile method), 423
__init__() (angr.storage.SimMemoryObject method), 425
__init__() (angr.storage.file.SimFile method), 434
__init__() (angr.storage.file.SimFileBase method), 433
__init__() (angr.storage.file.SimFileDescriptor method), 444
__init__() (angr.storage.file.SimFileDescriptorDuplex method), 446
__init__() (angr.storage.file.SimFileStream method), 436
__init__() (angr.storage.file.SimPackets method), 438
__init__() (angr.storage.file.SimPacketsSlots method), 449
__init__() (angr.storage.file.SimPacketsStream method), 440
__init__() (angr.storage.memory_mixins.AddressConcretizationMixin method), 456
__init__() (angr.storage.memory_mixins.ClemoryBackerMixin method), 458
__init__() (angr.storage.memory_mixins.ConvenientMappingsMixin method), 459
__init__() (angr.storage.memory_mixins.DictBackerMixin method), 460
__init__() (angr.storage.memory_mixins.ExplicitFillerMixin method), 461
__init__() (angr.storage.memory_mixins.HistoryTrackingMixin method), 462
__init__() (angr.storage.memory_mixins.JavaVmMemoryMixin method), 464
__init__() (angr.storage.memory_mixins.KeyValueMemoryMixin method), 466
__init__() (angr.storage.memory_mixins.LabelMergerMixin method), 467
__init__() (angr.storage.memory_mixins.ListPage method), 467
__init__() (angr.storage.memory_mixins.MVListPage method), 469
__init__() (angr.storage.memory_mixins.MVListPagesMixin method), 471

__init__() (angr.storage.memory_mixins.MemoryRegionMetaMixin method), 471
__init__() (angr.storage.memory_mixins.MultiValueMergerMixin method), 473
__init__() (angr.storage.memory_mixins.PagedMemoryMixin method), 474
__init__() (angr.storage.memory_mixins.PermissionsMixin method), 477
__init__() (angr.storage.memory_mixins.RefcountMixin method), 478
__init__() (angr.storage.memory_mixins.RegionedAddressConcretizationMixin method), 479
__init__() (angr.storage.memory_mixins.RegionedMemoryMixin method), 480
__init__() (angr.storage.memory_mixins.SizeConcretizationMixin method), 483
__init__() (angr.storage.memory_mixins.SlottedMemoryMixin method), 484
__init__() (angr.storage.memory_mixins.SpecialFillerMixin method), 486
__init__() (angr.storage.memory_mixins.StackAllocationMixin method), 486
__init__() (angr.storage.memory_mixins.TopMergerMixin method), 487
__init__() (angr.storage.memory_mixins.UltraPage method), 487
__init__() (angr.storage.memory_mixins.UnderconstrainedMixin method), 489
__init__() (angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method), 495
__init__() (angr.storage.memory_mixins.convenient_mapping_mixin.ConvenientMappingMixin method), 499
__init__() (angr.storage.memory_mixins.default_filler_mixin.ExplicitFillerMixin method), 491
__init__() (angr.storage.memory_mixins.default_filler_mixin.SpecialFillerMixin method), 491
__init__() (angr.storage.memory_mixins.label_merger_mixin.LabelMergerMixin method), 498
__init__() (angr.storage.memory_mixins.multi_value_merger_mixin.MultiValueMergerMixin method), 504
__init__() (angr.storage.memory_mixins.paged_memory.page_backer_mixins.ClaripyBackerMixin method), 508
__init__() (angr.storage.memory_mixins.paged_memory.page_backer_mixins.DictBackerMixin method), 509
__init__() (angr.storage.memory_mixins.paged_memory.page_backer_mixins.NullBackerMixin method), 508
__init__() (angr.storage.memory_mixins.paged_memory.page_management_mixins.MVListPagesMixin method), 507
__init__() (angr.storage.memory_mixins.paged_memory.page_management_mixins.PageSizeMixin method), 504
__init__() (angr.storage.memory_mixins.paged_memory.page_tracking_mixins.HistoryTrackingMixin method), 510
__init__() (angr.storage.memory_mixins.paged_memory.page_tracking_mixins.ListPageMixin method), 511
__init__() (angr.storage.memory_mixins.paged_memory.pages.MVListPa method), 513
__init__() (angr.storage.memory_mixins.paged_memory.pages.Permissio method), 515
__init__() (angr.storage.memory_mixins.paged_memory.pages.Refcount method), 516
__init__() (angr.storage.memory_mixins.paged_memory.pages.UltraPag method), 516
__init__() (angr.storage.memory_mixins.paged_memory.pages.history_t method), 519
__init__() (angr.storage.memory_mixins.paged_memory.pages.list_page. method), 521
__init__() (angr.storage.memory_mixins.paged_memory.pages.multi_val method), 501
__init__() (angr.storage.memory_mixins.paged_memory.pages.mv_list_p method), 499
__init__() (angr.storage.memory_mixins.paged_memory.pages.permissio method), 519
__init__() (angr.storage.memory_mixins.paged_memory.pages.refcount_ method), 518
__init__() (angr.storage.memory_mixins.paged_memory.pages.ultra_pag method), 522
__init__() (angr.storage.memory_mixins.paged_memory.stack_allocatio method), 509
__init__() (angr.storage.memory_mixins.regioned_memory.MemoryRegi method), 524
__init__() (angr.storage.memory_mixins.regioned_memory.RegionedAd method), 526
__init__() (angr.storage.memory_mixins.regioned_memory.RegionedMe method), 528
__init__() (angr.storage.memory_mixins.regioned_memory.abstract_ad method), 536
__init__() (angr.storage.memory_mixins.regioned_memory.region_data. method), 533
__init__() (angr.storage.memory_mixins.regioned_memory.region_data. method), 534
__init__() (angr.storage.memory_mixins.regioned_memory.region_data. method), 534
__init__() (angr.storage.memory_mixins.regioned_memory.region_meta method), 536
__init__() (angr.storage.memory_mixins.regioned_memory.region_meta method), 536
__init__() (angr.storage.memory_mixins.regioned_memory.region_meta method), 536
__init__() (angr.storage.memory_mixins.regioned_memory.regioned_ad method), 538
__init__() (angr.storage.memory_mixins.regioned_memory.regioned_me method), 531
__init__() (angr.storage.memory_mixins.size_resolution_mixin.SizeConc method), 494
__init__() (angr.storage.memory_mixins.slotted_memory.SlottedMemory method), 539
__init__() (angr.storage.memory_mixins.top_merger_mixin.TopMergerM method), 503

__init__() (*angr.storage.memory_mixins.underconstrained_memory.UnderconstrainedMixin method*), 492
__init__() (*angr.storage.memory_object.SimLabeledMemoryObject method*), 451
__init__() (*angr.storage.memory_object.SimMemoryObject method*), 451
__init__() (*angr.utils.cowdict.ChainMapCOW method*), 1344
__init__() (*angr.utils.cowdict.DefaultChainMapCOW method*), 1345
__init__() (*angr.utils.dynamic_dictlist.DynamicDictList method*), 1346
__init__() (*angr.utils.graph.ContainerNode method*), 1348
__init__() (*angr.utils.graph.Dominators method*), 1348
__init__() (*angr.utils.graph.PostDominators method*), 1348
__init__() (*angr.utils.graph.SCCPlaceholder method*), 1349
__init__() (*angr.utils.graph.TemporaryNode method*), 1348
__init__() (*angr.utils.mp.Initializer method*), 1354
__init__() (*angr.vaults.Vault method*), 872
__init__() (*angr.vaults.VaultDict method*), 872
__init__() (*angr.vaults.VaultDir method*), 872
__init__() (*angr.vaults.VaultDirShelf method*), 873
__init__() (*angr.vaults.VaultPickler method*), 871
__init__() (*angr.vaults.VaultShelf method*), 872
__init__() (*angr.vaults.VaultUnpickler method*), 871

A

AArch64MachOGotResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1005
abort() (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method*), 936
abort() (*angr.analyses.forward_analysis.ForwardAnalysis method*), 932
abort() (*angr.analyses.ForwardAnalysis method*), 907
absolutize() (*angr.storage.memory_mixins.regioned_memory.region_data.RegionMap method*), 534
absorb() (*angr.sim_manager.SimulationManager method*), 548
absorb() (*angr.SimulationManager method*), 218
abstract() (*angr.analyses.typehoon.simple_solver.SimpleSolver static method*), 1264
AbstractAddressDescriptor (*class in angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor*), 535
AbstractLocation (*class in angr.storage.memory_mixins.regioned_memory.region_meta_mixin*), 536
AbstractMemoryUnderconstrainedMixin (*class in angr.storage.memory_mixins*), 456
AbstractMergerMixin (*class in angr.storage.memory_mixins*), 456
AbstractMergerMixin (*class in angr.storage.memory_mixins.regioned_memory*), 524
AbstractMergerMixin (*class in angr.storage.memory_mixins.regioned_memory.abstract_merger_*, 538
access_type (*angr.knowledge_plugins.variables.variable_access.VariableAccess attribute*), 812
accessed_data_references (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
accessed_data_references (*angr.knowledge_plugins.cfg.CFGNode property*), 751
AccessingZeroPageError, 396
acquire_shared() (*angr.storage.memory_mixins.paged_memory.pages.re method*), 518
acquire_shared() (*angr.storage.memory_mixins.paged_memory.pages.R method*), 516
acquire_shared() (*angr.storage.memory_mixins.RefcountMixin method*), 478
acquire_unique() (*angr.storage.memory_mixins.HistoryTrackingMixin method*), 462
acquire_unique() (*angr.storage.memory_mixins.paged_memory.pages.hi method*), 520
acquire_unique() (*angr.storage.memory_mixins.paged_memory.pages.H method*), 510
acquire_unique() (*angr.storage.memory_mixins.paged_memory.pages.re method*), 518
acquire_unique() (*angr.storage.memory_mixins.paged_memory.pages.R method*), 516
acquire_unique() (*angr.storage.memory_mixins.RefcountMixin method*), 478
Action (*in module angr.state_plugins.inspect*), 339
action() (*angr.state_plugins.inspect.SimInspector method*), 340
action() (*angr.state_plugins.SimInspector method*), 276
action_id (*angr.analyses.data_dep.BaseDepNode attribute*), 1316
action_id (*angr.analyses.data_dep.dep_nodes.BaseDepNode attribute*), 1314
actions (*angr.analyses.vfg.VFGNode attribute*), 1288
actions (*angr.state_plugins.history.SimStateHistory property*), 377
actions (*angr.state_plugins.log.SimStateLog property*), 370
actions (*angr.state_plugins.SimStateHistory property*), 303
actions (*angr.state_plugins.SimStateLog property*), 311
actions_of_type() (*angr.state_plugins.log.SimStateLog*

method), 370
actions_of_type() (*angr.state_plugins.SimStateLog method*), 311
ActionsMixinHigh (*class in angr.storage.memory_mixins*), 456
ActionsMixinHigh (*class in angr.storage.memory_mixins.actions_mixin*), 493
ActionsMixinLow (*class in angr.storage.memory_mixins*), 456
ActionsMixinLow (*class in angr.storage.memory_mixins.actions_mixin*), 493
activate() (*angr.misc.plugins.PluginPreset method*), 237
active (*angr.sim_manager.SimulationManager attribute*), 545
active (*angr.SimulationManager attribute*), 215
active_workers (*angr.distributed.Server property*), 1362
active_workers (*angr.distributed.server.Server property*), 1362
active_workers (*angr.Server property*), 173
Add (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransformation attribute*), 998
Add (*angr.engines.light.ArithmeticExpression attribute*), 1167
Add (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
Add (*class in angr.analyses.typehoon.typevars*), 1267
add() (*angr.exploration_techniques.spiller.PickledStatesBase method*), 571
add() (*angr.exploration_techniques.spiller.PickledStatesDb method*), 572
add() (*angr.exploration_techniques.spiller.PickledStatesList method*), 572
add() (*angr.procedures.definitions.SimLibrary method*), 651
add() (*angr.procedures.definitions.SimTypeCollection method*), 649
add() (*angr.sim_state_options.SimStateOptions method*), 245
add() (*angr.state_plugins.SimSolver method*), 292
add() (*angr.state_plugins.solver.SimSolver method*), 366
add_action() (*angr.state_plugins.history.SimStateHistory method*), 377
add_action() (*angr.state_plugins.log.SimStateLog method*), 370
add_action() (*angr.state_plugins.SimStateHistory method*), 303
add_action() (*angr.state_plugins.SimStateLog method*), 311
add_alias() (*angr.procedures.definitions.SimLibrary method*), 651
add_alias() (*angr.procedures.definitions.SimSyscallLibrary method*), 656
add_all_from_dict() (*angr.procedures.definitions.SimLibrary method*), 651
add_block_to_whitelist() (*angr.annocfg.AnnotatedCFG method*), 1321
add_breakpoint() (*angr.Emulator method*), 162
add_breakpoint() (*angr.state_plugins.inspect.SimInspector method*), 341
add_breakpoint() (*angr.state_plugins.SimInspector method*), 277
add_constraint() (*angr.analyses.typehoon.simple_solver.Sketch method*), 1261
add_constraints() (*angr.sim_state.SimState method*), 241
add_constraints() (*angr.SimState method*), 209
add_data_seg() (*angr.analyses.cfg.cfg_fast.DecodingAssumption method*), 984
add_def() (*angr.analyses.ddg.LiveDefinitions method*), 1157
add_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 1201
add_def() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDe method*), 855
add_def() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsM method*), 832
add_default_plugin() (*angr.misc.plugins.PluginPreset method*), 237
add_defs() (*angr.analyses.ddg.LiveDefinitions method*), 1157
add_dependencies_for_concrete_pointers_of() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 1218
add_edge() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 1218
add_edge() (*angr.analyses.typehoon.simple_solver.Sketch method*), 1260
add_edge_to_buffer() (*in module angr.utils.formatting*), 1354
add_event() (*angr.state_plugins.history.SimStateHistory method*), 377
add_event() (*angr.state_plugins.log.SimStateLog method*), 370
add_event() (*angr.state_plugins.SimStateHistory method*), 303
add_event() (*angr.state_plugins.SimStateLog method*), 311
add_exit_to_whitelist() (*angr.annocfg.AnnotatedCFG method*), 1321
add_final_job() (*angr.analyses.vfg.CallAnalysis method*), 1287
add_function() (*angr.analyses.cfg.cfb.CFBlanket

method), 973
add_function() (*angr.analyses.cfg.CFBlanket method*), 958
add_function_edge() (*angr.analyses.cfg.cfg_fast.CFGJob method*), 988
add_goal() (*angr.exploration_techniques.Director method*), 554
add_goal() (*angr.exploration_techniques.director.Director method*), 581
add_heap_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1195
add_heap_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1236
add_heap_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1209
add_heap_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 853
add_heap_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 830
add_heap_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1195
add_heap_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 853
add_heap_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 830
add_heap_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1236
add_heap_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1209
add_history() (*angr.state_hierarchy.StateHierarchy method*), 552
add_history() (*angr.StateHierarchy method*), 221
add_job() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 985
add_job() (*angr.analyses.forward_analysis.job_info.JobInfo method*), 937
add_jumpout_site() (*angr.knowledge_plugins.Function method*), 720
add_jumpout_site() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 781
add_jumpout_site() (*angr.knowledge_plugins.functions.function.Function method*), 805
add_jumptable() (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump method*), 771
add_jumptable() (*angr.knowledge_plugins.cfg.IndirectJump method*), 753
add_key_func_addr() (*angr.knowledge_plugins.FunctionManager method*), 729
add_key_func_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 799
add_key_func_addr() (*angr.knowledge_plugins.functions.FunctionManager method*), 790
add_label() (*angr.analyses.Reassembler method*), 917
add_label() (*angr.analyses.reassembler.Reassembler method*), 1303
add_labels() (*in module angr.analyses.decompiler.utils*), 1153
add_loop() (*angr.analyses.cfg.AnnotatedCFG method*), 1321
add_mapping() (*angr.analyses.decompiler.structured_codegen.base.InstructionMapping method*), 1129
add_mapping() (*angr.analyses.decompiler.structured_codegen.base.PositionMapping method*), 1128
add_mapping() (*angr.analyses.decompiler.structured_codegen.InstructionMapping method*), 1124
add_mapping() (*angr.analyses.decompiler.structured_codegen.PositionMapping method*), 1125
add_memory_data() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 762
add_memory_data() (*angr.knowledge_plugins.cfg.CFGModel method*), 748
add_memory_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1195
add_memory_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 853
add_memory_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 831
add_memory_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1196
add_memory_use_by_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1236
add_memory_use_by_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1209
add_memory_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 854
add_memory_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 831
add_memory_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1236
add_memory_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1209
add_node() (*angr.analyses.reaching_definitions.dep_graph.DepGraph*

method), 1217
add_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 758
add_node() (*angr.knowledge_plugins.cfg.CFGModel method*), 744
add_nonreturning_function() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 985
add_number_mapping() (*angr.procedures.definitions.SimSyscallLibrary method*), 655
add_number_mapping_from_dict() (*angr.procedures.definitions.SimSyscallLibrary method*), 655
add_obj() (*angr.analyses.cfg.cfb.CFBlanket method*), 973
add_obj() (*angr.analyses.cfg.CFBlanket method*), 958
add_object() (*angr.keyed_region.KeyedRegion method*), 869
add_object() (*angr.keyed_region.RegionObject method*), 868
add_patch() (*angr.knowledge_plugins.patches.PatchManager method*), 737
add_patch() (*angr.knowledge_plugins.PatchManager method*), 731
add_patch_obj() (*angr.knowledge_plugins.patches.PatchManager method*), 737
add_patch_obj() (*angr.knowledge_plugins.PatchManager method*), 732
add_regioned_address() (*angr.storage.memory_mixins.regioned_memory.abstract_addresses.AbstractAddressDescriptor method*), 536
add_register_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1194
add_register_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1235
add_register_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1208
add_register_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 852
add_register_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 829
add_register_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1194
add_register_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 852
add_register_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 830
add_register_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1235
add_register_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1209
add_retout_site() (*angr.knowledge_plugins.Function method*), 721
add_retout_site() (*angr.knowledge_plugins.functions.Function method*), 781
add_retout_site() (*angr.knowledge_plugins.functions.function.Function method*), 805
add_returning_function() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 985
add_sinkhole() (*angr.state_plugins.cgc.SimStateCGC method*), 381
add_sinkhole() (*angr.state_plugins.SimStateCGC method*), 298
add_stack_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1195
add_stack_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1235
add_stack_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1209
add_stack_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 853
add_stack_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 830
add_stack_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1195
add_stack_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 853
add_stack_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 830
add_stack_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1236
add_stack_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1209
add_state() (*angr.state_hierarchy.StateHierarchy method*), 552
add_state() (*angr.StateHierarchy method*), 221
add_statements_to_whitelist() (*angr.annocfg.AnnotatedCFG method*), 1321
add_successors() (*angr.engines.SimSuccessors method*), 592
add_successor() (*angr.engines.successors.SimSuccessors method*), 595
add_tmp_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1196
add_tmp_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1235

add_tmp_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState* *method*), 1208
add_tmp_use() (*angr.knowledge_plugins.key_definitions.live_definitions* *method*), 854
add_tmp_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions* *method*), 831
add_tmp_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions* *method*), 1196
add_tmp_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions* *method*), 854
add_tmp_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions* *method*), 831
add_tmp_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState* *method*), 1235
add_tmp_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState* *method*), 1208
add_transitions() (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink* *method*), 1241
add_transitions() (*angr.analyses.cfg_slice_to_sink.CFGSliceToSink* *method*), 1239
add_type_constraint() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase* *method*), 1250
add_type_constraint_for_function() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase* *method*), 1251
add_type_variable() (*angr.analyses.typehoon.typevars.TypeVariables* *method*), 1271
add_use() (*angr.analyses.reaching_definitions.LiveDefinitions* *method*), 1193
add_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState* *method*), 1234
add_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState* *method*), 1208
add_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions* *method*), 851
add_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions* *method*), 827
add_use() (*angr.knowledge_plugins.key_definitions.Uses* *method*), 834
add_use() (*angr.knowledge_plugins.key_definitions.uses.Uses* *method*), 859
add_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions* *method*), 1193
add_use_by_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState* *method*), 1235
add_use_by_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsState* *method*), 1208
add_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions* *method*), 851
add_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions* *method*), 827
add_variable() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValuedMemory* *method*), 502
add_variable() (*angr.keyed_region.KeyedRegion* *method*), 869
add_variable() (*angr.knowledge_plugins.debug_variables.DebugVariableManagerPlugin* *method*), 821
add_variable() (*angr.knowledge_plugins.DebugVariableManager* *method*), 715
add_variable() (*angr.knowledge_plugins.variables.variable_manager.VariableManager* *method*), 814
add_variable_list() (*angr.knowledge_plugins.debug_variables.DebugVariableManagerPlugin* *method*), 822
add_variable_list() (*angr.knowledge_plugins.DebugVariableManager* *method*), 715
add_xref() (*angr.analyses.xrefs.SimEngineXRefsVEX* *method*), 1308
add_xref() (*angr.knowledge_plugins.XRefManager* *method*), 736
add_xref() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager* *method*), 864
add_xref() (*angr.knowledge_plugins.xrefs.XRefManager* *method*), 862
add_xrefs() (*angr.knowledge_plugins.XRefManager* *method*), 736
add_xrefs() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager* *method*), 864
add_xrefs() (*angr.knowledge_plugins.xrefs.XRefManager* *method*), 862
added_constraints (*angr.state_plugins.inspect.InspectAttrs* *attribute*), 335
added_constraints (*angr.state_plugins.InspectAttrs* *attribute*), 254
AddNodeStmt (*in angr.analyses.typehoon.typevars*), 1271
addr (*angr.ailment.Block attribute*), 1368
addr (*angr.ailment.manager.Block attribute*), 1406
addr (*angr.ailment.expression.Load attribute*), 1386
addr (*angr.ailment.statement.CAS attribute*), 1404
addr (*angr.ailment.statement.Store attribute*), 1399
addr (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
addr (*angr.analyses.decompiler.decompilation_cache.DecompilationCache* *attribute*), 1064
addr (*angr.analyses.decompiler.graph_region.GraphRegion* *property*), 1071
addr (*angr.analyses.decompiler.structured_codegen.c.CFunction* *attribute*), 1132
addr (*angr.analyses.decompiler.structured_codegen.c.CStatements* *attribute*), 1133
addr (*angr.analyses.disassembly.DisassemblyPiece* *at-*

tribute), 1293
addr (*angr.analyses.disassembly.IROp attribute*), 1293
addr (*angr.analyses.disassembly.OperandPiece attribute*), 1295
addr (*angr.analyses.propagator.vex_vars.VEXMemVar attribute*), 1176
addr (*angr.analyses.reaching_definitions.MemoryLocation attribute*), 1197
addr (*angr.angrdb.models.DbComment attribute*), 1025
addr (*angr.angrdb.models.DbFunction attribute*), 1023
addr (*angr.angrdb.models.DbLabel attribute*), 1025
addr (*angr.Block attribute*), 160
addr (*angr.block.Block attribute*), 233
addr (*angr.block.DisassemblerBlock attribute*), 232
addr (*angr.code_location.AILCodeLocation attribute*), 867
addr (*angr.codenode.CodeNode attribute*), 1323
addr (*angr.engines.light.engine.BlockProtocol attribute*), 1171
addr (*angr.engines.pcode.lifter.IRSB attribute*), 608
addr (*angr.engines.pcode.lifter.Lifter attribute*), 610
addr (*angr.engines.pcode.lifter.PcodeDisassemblerBlock attribute*), 606
addr (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
addr (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
addr (*angr.knowledge_plugins.cfg.CFGNode property*), 750
addr (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 771
addr (*angr.knowledge_plugins.cfg.indirect_jump.JumptableInfo attribute*), 770
addr (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 752
addr (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 765
addr (*angr.knowledge_plugins.cfg.MemoryData attribute*), 754
addr (*angr.knowledge_plugins.Function attribute*), 717
addr (*angr.knowledge_plugins.functions.Function attribute*), 777
addr (*angr.knowledge_plugins.functions.function.Function attribute*), 802
addr (*angr.knowledge_plugins.functions.soot_function.SootAddress attribute*), 810
addr (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation attribute*), 841
addr (*angr.sim_state.SimState property*), 240
addr (*angr.sim_variable.SimMemoryVariable attribute*), 685
addr (*angr.SimState property*), 209
addr (*angr.state_plugins.history.SimStateHistory property*), 375
addr (*angr.state_plugins.SimStateHistory property*), 301
addr (*angr.utils.graph.SCCPlaceholder attribute*), 1349
addr_and_variables (*angr.analyses.variable_recovery.variable_recovery attribute*), 1247
addr_invalid() (*angr.state_plugins.cgc.SimStateCGC method*), 380
addr_invalid() (*angr.state_plugins.SimStateCGC method*), 297
addr_to_instruction_addr() (*angr.knowledge_plugins.Function method*), 723
addr_to_instruction_addr() (*angr.knowledge_plugins.functions.Function method*), 783
addr_to_instruction_addr() (*angr.knowledge_plugins.functions.function.Function method*), 807
addr_type (*angr.analyses.cfg.cfg_base.CFGBase attribute*), 981
addr_type (*angr.analyses.cfg.cfg_emulated.CFGEmulated attribute*), 977
addr_type (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 990
addr_type (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot attribute*), 1015
addr_type (*angr.analyses.cfg.CFGBase attribute*), 959
addr_type (*angr.analyses.cfg.CFGEmulated attribute*), 962
addr_type (*angr.analyses.cfg.CFGFast attribute*), 967
addr_type (*angr.analyses.cfg.CFGFastSoot attribute*), 970
addr_type (*angr.analyses.CFGEmulated attribute*), 885
addr_type (*angr.analyses.CFGFast attribute*), 890
addr_type (*angr.analyses.CFGFastSoot attribute*), 892
addr_type (*angr.knowledge_plugins.cfg.cfg_model.CFGModel property*), 757
addr_type (*angr.knowledge_plugins.cfg.CFGModel property*), 743
address (*angr.analyses.reaching_definitions.function_handler.FunctionCall attribute*), 1224
address (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
address (*angr.block.CapstoneInsn property*), 232
address (*angr.block.DisassemblerInsn property*), 232
address (*angr.block.PCodeInsn property*), 233
address (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 606
address (*angr.knowledge_plugins.cfg.memory_data.MemoryData property*), 765
address (*angr.knowledge_plugins.cfg.MemoryData property*), 754
address (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
address (*angr.state_plugins.InspectAttrs attribute*), 254

address (*angr.state_plugins.unicorn_engine.MEM_PATCH attribute*), 392
address (*angr.state_plugins.unicorn_engine.MemoryValue attribute*), 392
address (*angr.storage.memory_mixins.regioned_memory.region_data.AddressWrapper attribute*), 533
ADDRESS_CONCRETIZATION (*angr.state_plugins.inspect.EventType attribute*), 331
address_concretization_action (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
address_concretization_action (*angr.state_plugins.InspectAttrs attribute*), 254
address_concretization_add_constraints (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
address_concretization_add_constraints (*angr.state_plugins.InspectAttrs attribute*), 255
address_concretization_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
address_concretization_expr (*angr.state_plugins.InspectAttrs attribute*), 254
address_concretization_memory (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
address_concretization_memory (*angr.state_plugins.InspectAttrs attribute*), 254
address_concretization_result (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
address_concretization_result (*angr.state_plugins.InspectAttrs attribute*), 255
address_concretization_strategy (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
address_concretization_strategy (*angr.state_plugins.InspectAttrs attribute*), 254
address_multi (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 1223
address_multi (*angr.analyses.reaching_definitions.function_handler.FunctionCallDataUnwrapped attribute*), 1226
address_multi (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
AddressConcretizationMixin (*class in angr.storage.memory_mixins*), 456
AddressConcretizationMixin (*class in angr.storage.memory_mixins.address_concretization_mixin*), 495
AddressOperand (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 998
AddressTransformation (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 998
AddressTransformationTypes (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 998
AddressWrapper (*class in angr.storage.memory_mixins.regioned_memory.region_data*), 533
addrs_for_hash() (*angr.storage.memory_mixins.convenient_mappings_m method*), 499
addrs_for_hash() (*angr.storage.memory_mixins.ConvenientMappingsMi method*), 459
addrs_for_name() (*angr.storage.memory_mixins.convenient_mappings_m method*), 499
addrs_for_name() (*angr.storage.memory_mixins.ConvenientMappingsMi method*), 459
ADDS_EXITS (*angr.sim_procedure.SimProcedure attribute*), 645
ADDS_EXITS (*angr.SimProcedure attribute*), 204
adjust_mapping_positions() (*angr.analyses.decompiler.structured_codegen.base.BaseStructur static method*), 1129
adjust_mapping_positions() (*angr.analyses.decompiler.structured_codegen.BaseStructuredCo static method*), 1120
AFTER (*angr.state_plugins.inspect.When attribute*), 339
AFTER_AIL_GRAPH_CREATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 1092
AFTER_AIL_GRAPH_CREATION (*angr.analyses.decompiler.optimization_passes.OptimizationPass. attribute*), 1084
AFTER_GLOBAL_SIMPLIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 1092
AFTER_GLOBAL_SIMPLIFICATION (*angr.analyses.decompiler.optimization_passes.OptimizationPass. attribute*), 1084
AFTER_MAKING_CALLSITES (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 1092
AFTER_MAKING_CALLSITES (*angr.analyses.decompiler.optimization_passes.OptimizationPass. attribute*), 1084
AFTER_SINGLE_BLOCK_SIMPLIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 1092
AFTER_SINGLE_BLOCK_SIMPLIFICATION

(*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 1084
AFTER_SSA_LEVEL1_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 1092
AFTER_SSA_LEVEL1_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 1084
AFTER_STRUCTURING (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 1093
AFTER_STRUCTURING (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 1084
AFTER_VARIABLE_RECOVERY (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 1092
AFTER_VARIABLE_RECOVERY (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 1084
AggressiveConcretizationAnnotation (*class in angr.state_plugins.unicorn_engine*), 396
ail_block_idx (*angr.state_plugins.scratch.SimStateScratch property*), 388
ail_block_idx (*angr.state_plugins.SimStateScratch property*), 318
AIL_GRAPH_CONVERSION (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1057
ail_manager (*angr.analyses.decompiler.ccall_rewriters.rewriterBase attribute*), 1056
AILBlockRewriter (*class in angr.ailment*), 1364
AILBlockTempCollector (*class in angr.analyses.decompiler.ail_simplifier*), 1052
AILBlockViewer (*class in angr.ailment*), 1364
AILBlockWalker (*class in angr.ailment*), 1364
AILCodeLocation (*class in angr.code_location*), 866
ailexpr2cnode (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator attribute*), 1146
AILExprIdAnnotation (*class in angr.analyses.decompiler.condition_processor*), 1061
AILGraphWalker (*class in angr.analyses.decompiler.ailgraph_walker*), 1054
AILSimplifier (*class in angr.analyses.decompiler*), 1039
AILSimplifier (*class in angr.analyses.decompiler.ail_simplifier*), 1053
align (*angr.sim_type.SimStruct property*), 701
Alignment (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
Alignment (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
alignment (*angr.sim_type.SimStruct property*), 702
alignment (*angr.sim_type.SimType property*), 686
alignment (*angr.sim_type.SimTypeArray property*), 696
alignment (*angr.sim_type.SimTypeBitfield property*), 704
alignment (*angr.sim_type.SimTypeDouble property*), 701
alignment (*angr.sim_type.SimTypeEnum property*), 703
alignment (*angr.sim_type.SimTypeString property*), 697
alignment (*angr.sim_type.SimTypeWString property*), 698
alignment (*angr.sim_type.SimUnion property*), 702
alignment (*angr.sim_type.TypeRef property*), 688
ALL (*angr.SimulationManager attribute*), 215
ALL (*angr.simos.OptimizationPassStage attribute*), 545
all_bytes_changed_in_history() (*angr.storage.memory_mixins.HistoryTrackingMixin method*), 463
all_bytes_changed_in_history() (*angr.storage.memory_mixins.paged_memory.pages.history_track method*), 520
all_bytes_changed_in_history() (*angr.storage.memory_mixins.paged_memory.pages.HistoryTrack method*), 511
all_constants (*angr.engines.pcode.lifter.IRSB property*), 609
all_definitions (*angr.analyses.reaching_definitions.rd_state.ReachingD attribute*), 1231
all_definitions (*angr.analyses.reaching_definitions.reaching_definition property*), 1216
all_definitions (*angr.analyses.reaching_definitions.ReachingDefinition property*), 1200
all_definitions (*angr.analyses.reaching_definitions.ReachingDefinition attribute*), 1205
all_definitions (*angr.analyses.ReachingDefinitionsAnalysis property*), 914
all_flavors() (*angr.knowledge_plugins.structured_code.StructuredCode method*), 822
all_flavors() (*angr.knowledge_plugins.StructuredCodeManager method*), 734
all_objects (*angr.state_plugins.sim_action.SimAction property*), 638
all_objects (*angr.state_plugins.sim_action.SimActionConstraint property*), 639
all_objects (*angr.state_plugins.sim_action.SimActionData property*), 640
all_objects (*angr.state_plugins.sim_action.SimActionExit property*), 639
all_objects (*angr.state_plugins.sim_action.SimActionOperation property*), 639
all_objects (*angr.state_plugins.SimAction property*), 263
all_objects (*angr.state_plugins.SimActionConstraint

property), 263
all_objects (*angr.state_plugins.SimActionData property*), 264
all_objects (*angr.state_plugins.SimActionExit property*), 264
all_objects (*angr.state_plugins.SimActionOperation property*), 266
all_stashes (*angr.analyses.Veritesting attribute*), 926
all_stashes (*angr.analyses.veritesting.Veritesting attribute*), 1285
all_states (*angr.analyses.vfg.VFGNode attribute*), 1288
all_successors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 943
all_successors() (*angr.state_hierarchy.StateHierarchy method*), 552
all_successors() (*angr.StateHierarchy method*), 221
ALL_TABLES (*angr.angrdb.AngrDB attribute*), 1018
ALL_TABLES (*angr.angrdb.db.AngrDB attribute*), 1020
all_uses (*angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter attribute*), 1112
all_uses (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 1216
all_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 1200
all_uses (*angr.analyses.ReachingDefinitionsAnalysis property*), 914
alloc() (*angr.calling_conventions.AllocHelper method*), 658
allocate() (*angr.analyses.reaching_definitions.heap_allocator.HeapAllocator method*), 1221
allocate() (*angr.knowledge_plugins.CustomStrings method*), 714
allocate() (*angr.SimHeapBrk method*), 190
allocate() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 412
allocate() (*angr.state_plugins.heap.SimHeapBrk method*), 407
allocate() (*angr.state_plugins.SimHeapBrk method*), 271
allocate_stack_pages() (*angr.storage.memory_mixins.paged_memory.stack_allocation_mixin.StackAllocationMixin method*), 509
allocate_stack_pages() (*angr.storage.memory_mixins.StackAllocationMixin method*), 486
allocated_addresses (*angr.analyses.reaching_definitions.heap_allocator.HeapAllocator property*), 1221
allocated_chunks() (*angr.SimHeapPTMalloc method*), 191
allocated_chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 415
allocated_chunks() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 418
allocated_chunks() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 409
allocated_chunks() (*angr.state_plugins.SimHeapPTMalloc method*), 274
AllocHelper (*class in angr.calling_conventions*), 658
allow_arch_optimizations (*angr.engines.pcode.lifter.Lifter attribute*), 610
allow_arch_optimizations (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
alt (*angr.ailment.expression.Load attribute*), 1386
ALT_NAMES (*angr.sim_procedure.SimProcedure attribute*), 645
ALT_NAMES (*angr.SimProcedure attribute*), 204
ALWAYS (*angr.analyses.decompiler.structuring.phoenix.MultiStmtExprMode attribute*), 1037
always_propagate (*angr.ailment.tagged_object.TagDict property*), 1407
AMD64CCallRewriter (*class in angr.analyses.decompiler.ccall_rewriters.amd64_ccalls*), 1056
AMD64ElfGotResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1005
AMD64ElfGotResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got*), 994
AMD64PeIatResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1006
analyses (*angr.Project property*), 170
analyses (*angr.project.Project property*), 225
AnalysesHub (*class in angr.analyses*), 881
AnalysesHub (*class in angr.analyses.analysis*), 928
AnalysesHubWithDefault (*class in angr.analyses.analysis*), 929
analysis (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitions attribute*), 1231
analysis (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 1205
Analysis (*class in angr*), 156
Analysis (*class in angr.analyses*), 881
Analysis (*class in angr.analyses.analysis*), 930
AnalysisFactory (*class in angr.analyses.analysis*), 929
AnalysisLogEntry (*class in angr.analyses.analysis*), 928
AnalysisTask (*class in angr.analyses.vfg*), 1286
analyze() (*angr.analyses.callee_cleanup_finder.CalleeCleanupFinder method*), 1307
analyze() (*angr.analyses.CalleeCleanupFinder method*), 894
analyze() (*angr.analyses.code_tagging.CodeTagging*

method), 1018
analyze() (*angr.analyses.CodeTagging method*), 897
analyze() (*angr.analyses.decompiler.optimization_passes.optimization_pass.BaseOptimizationPass method*), 1093
analyze() (*angr.analyses.decompiler.optimization_passes.optimization_pass.StructuringOptimizationPass method*), 1097
analyze() (*angr.analyses.PackingDetector method*), 910
analyze() (*angr.analyses.Pathfinder method*), 910
analyze() (*angr.analyses.vtable.VtableFinder method*), 1291
analyze() (*angr.analyses.VtableFinder method*), 927
analyze_transmit() (*angr.state_plugins.trace_additions.ZeroPlugin method*), 385
And (*angr.engines.light.ArithmeticExpression attribute*), 1167
And (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
angr
module, 155
angr.ailment
module, 1364
angr.ailment.block
module, 1406
angr.ailment.converter_common
module, 1373
angr.ailment.converter_pcode
module, 1373
angr.ailment.converter_vex
module, 1374
angr.ailment.expression
module, 1375
angr.ailment.manager
module, 1407
angr.ailment.statement
module, 1397
angr.ailment.tagged_object
module, 1407
angr.ailment.utils
module, 1408
angr.analyses
module, 873
angr.analyses.analysis
module, 928
angr.analyses.backward_slice
module, 946
angr.analyses.binary_optimizer
module, 1306
angr.analyses.bindiff
module, 947
angr.analyses.boyscout
module, 950
angr.analyses.callee_cleanup_finder
module, 1307
angr.analyses.calling_convention
module, 950
angr.analyses.cdg
module, 1017
angr.analyses.cfg
module, 956
angr.analyses.cfg.cfb
module, 972
angr.analyses.cfg.cfg
module, 973
angr.analyses.cfg.cfg_arch_options
module, 992
angr.analyses.cfg.cfg_base
module, 981
angr.analyses.cfg.cfg_emulated
module, 976
angr.analyses.cfg.cfg_fast
module, 983
angr.analyses.cfg.cfg_fast_soot
module, 1015
angr.analyses.cfg.cfg_job_base
module, 993
angr.analyses.cfg.indirect_jump_resolvers
module, 1005
angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got
module, 994
angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast
module, 994
angr.analyses.cfg.indirect_jump_resolvers.const_resolver
module, 1003
angr.analyses.cfg.indirect_jump_resolvers.default_resolver
module, 998
angr.analyses.cfg.indirect_jump_resolvers.jumptable
module, 998
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast
module, 996
angr.analyses.cfg.indirect_jump_resolvers.resolver
module, 1004
angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt
module, 997
angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat
module, 995
angr.analyses.cfg_slice_to_sink
module, 1238
angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink
module, 1241
angr.analyses.cfg_slice_to_sink.graph
module, 1242
angr.analyses.cfg_slice_to_sink.transitions
module, 1243
angr.analyses.class_identifier
module, 1292
angr.analyses.code_tagging
module, 1018
angr.analyses.complete_calling_conventions

module, 953
angr.analyses.congruency_check
module, 1305
angr.analyses.data_dep
module, 1316
angr.analyses.data_dep.data_dependency_analysis
module, 1311
angr.analyses.data_dep.dep_nodes
module, 1313
angr.analyses.data_dep.sim_act_location
module, 1312
angr.analyses.datagraph_meta
module, 1017
angr.analyses.ddg
module, 1156
angr.analyses.decompiler
module, 1039
angr.analyses.decompiler.ail_simplifier
module, 1052
angr.analyses.decompiler.ailgraph_walker
module, 1054
angr.analyses.decompiler.block_simplifier
module, 1054
angr.analyses.decompiler.callsite_maker
module, 1055
angr.analyses.decompiler.ccall_rewriters
module, 1055
angr.analyses.decompiler.ccall_rewriters.amd64_rewriters
module, 1056
angr.analyses.decompiler.ccall_rewriters.rewriter_base
module, 1055
angr.analyses.decompiler.clinic
module, 1056
angr.analyses.decompiler.condition_processor
module, 1061
angr.analyses.decompiler.decompilation_cache
module, 1064
angr.analyses.decompiler.decompilation_options
module, 1063
angr.analyses.decompiler.decompiler
module, 1065
angr.analyses.decompiler.empty_node_remover
module, 1068
angr.analyses.decompiler.expression_narrower
module, 1069
angr.analyses.decompiler.graph_region
module, 1070
angr.analyses.decompiler.jump_target_collector
module, 1071
angr.analyses.decompiler.jumptable_entry_condition_rewriter
module, 1071
angr.analyses.decompiler.optimization_passes
module, 1071
angr.analyses.decompiler.optimization_passes.base_optimization_pass
module, 1097
angr.analyses.decompiler.optimization_passes.const_derefs
module, 1091
angr.analyses.decompiler.optimization_passes.div_simplifie
module, 1098
angr.analyses.decompiler.optimization_passes.engine_base
module, 1101
angr.analyses.decompiler.optimization_passes.expr_op_swapp
module, 1102
angr.analyses.decompiler.optimization_passes.ite_expr_conv
module, 1098
angr.analyses.decompiler.optimization_passes.lowered_switc
module, 1099
angr.analyses.decompiler.optimization_passes.mod_simplifie
module, 1101
angr.analyses.decompiler.optimization_passes.optimization_
module, 1092
angr.analyses.decompiler.optimization_passes.register_save
module, 1103
angr.analyses.decompiler.optimization_passes.ret_addr_save
module, 1103
angr.analyses.decompiler.optimization_passes.stack_canary_
module, 1097
angr.analyses.decompiler.optimization_passes.x86_gcc_getpc
module, 1104
angr.analyses.decompiler.peephole_optimizations
module, 1104
angr.analyses.decompiler.peephole_optimizations.base
module, 1104
angr.analyses.decompiler.redundant_label_remover
module, 1119
angr.analyses.decompiler.region_identifier
module, 1107
angr.analyses.decompiler.region_simplifiers
module, 1108
angr.analyses.decompiler.region_simplifiers.cascading_cond
module, 1109
angr.analyses.decompiler.region_simplifiers.cascading_ifs
module, 1109
angr.analyses.decompiler.region_simplifiers.expr_folding
module, 1109
angr.analyses.decompiler.region_simplifiers.goto
module, 1114
angr.analyses.decompiler.region_simplifiers.if_
module, 1114
angr.analyses.decompiler.region_simplifiers.ifelse
module, 1114
angr.analyses.decompiler.region_simplifiers.loop
module, 1114
angr.analyses.decompiler.region_simplifiers.node_address_f
module, 1114
angr.analyses.decompiler.region_simplifiers.region_simplif
module, 1114
angr.analyses.decompiler.region_simplifiers.switch_cluster

module, 1115
angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier
module, 1119
angr.analyses.decompiler.region_walker
module, 1119
angr.analyses.decompiler.sequence_walker
module, 1119
angr.analyses.decompiler.structured_codegen
module, 1120
angr.analyses.decompiler.structured_codegen.base
module, 1127
angr.analyses.decompiler.structured_codegen.c
module, 1130
angr.analyses.decompiler.structured_codegen.dummy
module, 1149
angr.analyses.decompiler.structured_codegen.dwarf_import
module, 1149
angr.analyses.decompiler.structuring
module, 1032
angr.analyses.decompiler.structuring.dream
module, 1035
angr.analyses.decompiler.structuring.phoenix
module, 1037
angr.analyses.decompiler.structuring.recursive_structurer
module, 1032
angr.analyses.decompiler.structuring.structurer_base
module, 1036
angr.analyses.decompiler.utils
module, 1150
angr.analyses.disassembly
module, 1293
angr.analyses.disassembly_utils
module, 1297
angr.analyses.dominance_frontier
module, 1307
angr.analyses.find_objects_static
module, 1292
angr.analyses.flirt
module, 1161
angr.analyses.forward_analysis
module, 931
angr.analyses.forward_analysis.forward_analysis
module, 935
angr.analyses.forward_analysis.job_info
module, 937
angr.analyses.forward_analysis.visitors
module, 937
angr.analyses.forward_analysis.visitors.call_graph
module, 941
angr.analyses.forward_analysis.visitors.function_graph
module, 941
angr.analyses.forward_analysis.visitors.graph
module, 942
angr.analyses.forward_analysis.visitors.loop
module, 944
angr.analyses.forward_analysis.visitors.single_node_graph
module, 945
angr.analyses.identifier.identify
module, 1283
angr.analyses.init_finder
module, 1307
angr.analyses.loop_analysis
module, 1284
angr.analyses.loopfinder
module, 1283
angr.analyses.propagator
module, 1174
angr.analyses.propagator.engine_base
module, 1176
angr.analyses.propagator.engine_vex
module, 1177
angr.analyses.propagator.propagator
module, 1177
angr.analyses.propagator.top_checker_mixin
module, 1178
angr.analyses.propagator.values
module, 1175
angr.analyses.propagator.vex_vars
module, 1175
angr.analyses.proximity_graph
module, 1309
angr.analyses.reaching_definitions
module, 1178
angr.analyses.reaching_definitions.call_trace
module, 1212
angr.analyses.reaching_definitions.dep_graph
module, 1217
angr.analyses.reaching_definitions.engine_ail
module, 1238
angr.analyses.reaching_definitions.engine_vex
module, 1213
angr.analyses.reaching_definitions.function_handler
module, 1221
angr.analyses.reaching_definitions.heap_allocator
module, 1220
angr.analyses.reaching_definitions.rd_state
module, 1229
angr.analyses.reaching_definitions.reaching_definitions
module, 1213
angr.analyses.reaching_definitions.subject
module, 1237
angr.analyses.reassembler
module, 1297
angr.analyses.soot_class_hierarchy
module, 955
angr.analyses.stack_pointer_tracker
module, 1243
angr.analyses.static_hooker

module, 1305
angr.analyses.typehoon
module, 1281
angr.analyses.typehoon.simple_solver
module, 1259
angr.analyses.typehoon.translator
module, 1265
angr.analyses.typehoon.typeconsts
module, 1274
angr.analyses.typehoon.typehoon
module, 1272
angr.analyses.typehoon.typevars
module, 1266
angr.analyses.variable_recovery
module, 1257
angr.analyses.variable_recovery.annotations
module, 1246
angr.analyses.variable_recovery.engine_ail
module, 1255
angr.analyses.variable_recovery.engine_base
module, 1256
angr.analyses.variable_recovery.engine_vex
module, 1256
angr.analyses.variable_recovery.irsb_scanner
module, 1257
angr.analyses.variable_recovery.variable_recovery
module, 1254
angr.analyses.variable_recovery.variable_recovery_base
module, 1247
angr.analyses.variable_recovery.variable_recovery_fast
module, 1251
angr.analyses.veritesting
module, 1284
angr.analyses.vfg
module, 1285
angr.analyses.vsa_ddg
module, 1290
angr.analyses.vtable
module, 1291
angr.analyses.xrefs
module, 1308
angr.angrdb
module, 1018
angr.angrdb.db
module, 1020
angr.angrdb.models
module, 1021
angr.angrdb.serializers
module, 1025
angr.angrdb.serializers.cfg_model
module, 1026
angr.angrdb.serializers.comments
module, 1027
angr.angrdb.serializers.funcs
module, 1027
angr.angrdb.serializers.kb
module, 1027
angr.angrdb.serializers.labels
module, 1028
angr.angrdb.serializers.loader
module, 1028
angr.angrdb.serializers.structured_code
module, 1031
angr.angrdb.serializers.variables
module, 1030
angr.angrdb.serializers.xrefs
module, 1030
angr.annocfg
module, 1321
angr.blade
module, 1319
angr.block
module, 232
angr.callable
module, 710
angr.calling_conventions
module, 658
angr.code_location
module, 865
angr.codenode
module, 1322
angr.concretization_strategies
module, 452
angr.concretization_strategies.any
module, 543
angr.concretization_strategies.controlled_data
module, 543
angr.concretization_strategies.eval
module, 541
angr.concretization_strategies.max
module, 542
angr.concretization_strategies.nonzero
module, 543
angr.concretization_strategies.nonzero_range
module, 542
angr.concretization_strategies.norepeats
module, 541
angr.concretization_strategies.norepeats_range
module, 543
angr.concretization_strategies.range
module, 542
angr.concretization_strategies.single
module, 541
angr.concretization_strategies.solutions
module, 541
angr.concretization_strategies.unlimited_range
module, 544
angr.distributed

module, 1361
angr.distributed.server
module, 1362
angr.distributed.worker
module, 1363
angr.engines
module, 587
angr.engines.engine
module, 594
angr.engines.failure
module, 598
angr.engines.hook
module, 597
angr.engines.light
module, 1167
angr.engines.light.data
module, 1166
angr.engines.light.engine
module, 1170
angr.engines.pcode
module, 603
angr.engines.pcode.behavior
module, 616
angr.engines.pcode.cc
module, 636
angr.engines.pcode.emulate
module, 615
angr.engines.pcode.engine
module, 604
angr.engines.pcode.lifter
module, 605
angr.engines.procedure
module, 597
angr.engines.soot
module, 601
angr.engines.soot.engine
module, 602
angr.engines.successors
module, 594
angr.engines.syscall
module, 598
angr.engines.unicorn
module, 603
angr.engines.vex
module, 599
angr.errors
module, 1355
angr.exploration_techniques
module, 552
angr.exploration_techniques.bucketizer
module, 586
angr.exploration_techniques.common
module, 585
angr.exploration_techniques.dfs
module, 569
angr.exploration_techniques.director
module, 579
angr.exploration_techniques.driller_core
module, 577
angr.exploration_techniques.explorer
module, 569
angr.exploration_techniques.lengthlimiter
module, 571
angr.exploration_techniques.local_loop_seer
module, 583
angr.exploration_techniques.loop_seer
module, 582
angr.exploration_techniques.manual_mergepoint
module, 571
angr.exploration_techniques.memory_watcher
module, 585
angr.exploration_techniques.oppologist
module, 581
angr.exploration_techniques.slicecutor
module, 578
angr.exploration_techniques.spiller
module, 571
angr.exploration_techniques.spiller_db
module, 574
angr.exploration_techniques.stochastic
module, 584
angr.exploration_techniques.suggestions
module, 586
angr.exploration_techniques.tech_builder
module, 585
angr.exploration_techniques.threading
module, 574
angr.exploration_techniques.timeout
module, 569
angr.exploration_techniques.tracer
module, 575
angr.exploration_techniques.unique
module, 584
angr.exploration_techniques.veritesting
module, 574
angr.factory
module, 227
angr.flirt
module, 1342
angr.flirt.build_sig
module, 1343
angr.keyed_region
module, 867
angr.knowledge_base
module, 711
angr.knowledge_plugins
module, 712
angr.knowledge_plugins.callsite_prototypes

module, 739
angr.knowledge_plugins.cfg
module, 740
angr.knowledge_plugins.cfg.cfg_manager
module, 766
angr.knowledge_plugins.cfg.cfg_model
module, 756
angr.knowledge_plugins.cfg.cfg_node
module, 766
angr.knowledge_plugins.cfg.indirect_jump
module, 769
angr.knowledge_plugins.cfg.memory_data
module, 764
angr.knowledge_plugins.comments
module, 775
angr.knowledge_plugins.data
module, 775
angr.knowledge_plugins.debug_variables
module, 820
angr.knowledge_plugins.functions
module, 776
angr.knowledge_plugins.functions.function
module, 799
angr.knowledge_plugins.functions.function_manager
module, 790
angr.knowledge_plugins.functions.function_parser
module, 809
angr.knowledge_plugins.functions.soot_function
module, 810
angr.knowledge_plugins.indirect_jumps
module, 775
angr.knowledge_plugins.key_definitions
module, 822
angr.knowledge_plugins.key_definitions.atoms
module, 836
angr.knowledge_plugins.key_definitions.constants
module, 841
angr.knowledge_plugins.key_definitions.definitions
module, 841
angr.knowledge_plugins.key_definitions.environment
module, 844
angr.knowledge_plugins.key_definitions.heap_address
module, 845
angr.knowledge_plugins.key_definitions.key_definition_manager
module, 845
angr.knowledge_plugins.key_definitions.live_definitions
module, 846
angr.knowledge_plugins.key_definitions.rd_model
module, 855
angr.knowledge_plugins.key_definitions.tag
module, 857
angr.knowledge_plugins.key_definitions.undefined
module, 858
angr.knowledge_plugins.key_definitions.unknown_size
module, 859
angr.knowledge_plugins.key_definitions.uses
module, 859
angr.knowledge_plugins.labels
module, 775
angr.knowledge_plugins.patches
module, 737
angr.knowledge_plugins.plugin
module, 738
angr.knowledge_plugins.propagations
module, 773
angr.knowledge_plugins.structured_code
module, 822
angr.knowledge_plugins.types
module, 772
angr.knowledge_plugins.variables
module, 811
angr.knowledge_plugins.variables.variable_access
module, 812
angr.knowledge_plugins.variables.variable_manager
module, 812
angr.knowledge_plugins.xrefs
module, 861
angr.knowledge_plugins.xrefs.xref
module, 863
angr.knowledge_plugins.xrefs.xref_manager
module, 864
angr.knowledge_plugins.xrefs.xref_types
module, 864
angr.misc.plugins
module, 235
angr.procedures
module, 646
angr.procedures.definitions
module, 649
angr.procedures.stubs.format_parser
module, 646
angr.project
module, 223
angr.protos
module, 871
angr.serializable
module, 870
angr.sim_manager
module, 544
angr.sim_options
module, 242
angr.sim_procedure
module, 642
angr.sim_state
module, 238
angr.sim_state_options
module, 242
angr.sim_type

module, 686
angr.sim_variable
module, 680
angr.simos
module, 1325
angr.simos.cgc
module, 1337
angr.simos.javavm
module, 1340
angr.simos.linux
module, 1335
angr.simos.simos
module, 1333
angr.simos.userland
module, 1337
angr.simos.windows
module, 1338
angr.slicer
module, 1320
angr.state_hierarchy
module, 551
angr.state_plugins
module, 246
angr.state_plugins.callstack
module, 371
angr.state_plugins.cgc
module, 379
angr.state_plugins.debug_variables
module, 421
angr.state_plugins.filesystem
module, 354
angr.state_plugins.gdb
module, 378
angr.state_plugins.globals
module, 386
angr.state_plugins.heap
module, 404
angr.state_plugins.heap.heap_base
module, 411
angr.state_plugins.heap.heap_brk
module, 412
angr.state_plugins.heap.heap_freelist
module, 414
angr.state_plugins.heap.heap_libc
module, 415
angr.state_plugins.heap.heap_ptmalloc
module, 416
angr.state_plugins.heap.utils
module, 420
angr.state_plugins.history
module, 375
angr.state_plugins.inspect
module, 331
angr.state_plugins.javavm_classloader
module, 400
angr.state_plugins.jni_references
module, 402
angr.state_plugins.libc
module, 343
angr.state_plugins.light_registers
module, 374
angr.state_plugins.log
module, 370
angr.state_plugins.loop_data
module, 399
angr.state_plugins.plugin
module, 329
angr.state_plugins.posix
module, 347
angr.state_plugins.preconstrainer
module, 390
angr.state_plugins.scratch
module, 388
angr.state_plugins.sim_action
module, 638
angr.state_plugins.sim_action_object
module, 640
angr.state_plugins.sim_event
module, 642
angr.state_plugins.solver
module, 361
angr.state_plugins.symbolizer
module, 420
angr.state_plugins.trace_additions
module, 381
angr.state_plugins.uc_manager
module, 387
angr.state_plugins.unicorn_engine
module, 392
angr.state_plugins.view
module, 426
angr.storage
module, 423
angr.storage.file
module, 431
angr.storage.memory_mixins
module, 456
angr.storage.memory_mixins.actions_mixin
module, 493
angr.storage.memory_mixins.address_concretization_mixin
module, 495
angr.storage.memory_mixins.bvv_conversion_mixin
module, 491
angr.storage.memory_mixins.conditional_store_mixin
module, 497
angr.storage.memory_mixins.convenient_mappings_mixin
module, 499
angr.storage.memory_mixins.default_filler_mixin

module, 490
angr.storage.memory_mixins.dirty_addrs_mixin
module, 495
angr.storage.memory_mixins.hex_dumper_mixin
module, 491
angr.storage.memory_mixins.label_merger_mixin
module, 497
angr.storage.memory_mixins.multi_value_merger_mixin
module, 503
angr.storage.memory_mixins.name_resolution_mixin
module, 490
angr.storage.memory_mixins.paged_memory
module, 504
angr.storage.memory_mixins.paged_memory.page_backers_mixin
module, 508
angr.storage.memory_mixins.paged_memory.paged_memory_mixin
module, 504
angr.storage.memory_mixins.paged_memory.pages
module, 510
angr.storage.memory_mixins.paged_memory.pages.cooperative
module, 520
angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin
module, 519
angr.storage.memory_mixins.paged_memory.pages.iterators_mixin
module, 520
angr.storage.memory_mixins.paged_memory.pages.list_pages
module, 521
angr.storage.memory_mixins.paged_memory.pages.multi_values
module, 501
angr.storage.memory_mixins.paged_memory.pages.page_8
module, 499
angr.storage.memory_mixins.paged_memory.pages.permissions_mixin
module, 519
angr.storage.memory_mixins.paged_memory.pages.refcounts_mixin
module, 518
angr.storage.memory_mixins.paged_memory.pages.utils
module, 522
angr.storage.memory_mixins.paged_memory.privileged_mixin
module, 510
angr.storage.memory_mixins.paged_memory.stack_allocations_mixin
module, 509
angr.storage.memory_mixins.regioned_memory
module, 524
angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor
module, 535
angr.storage.memory_mixins.regioned_memory.abstract_merge_mixin
module, 538
angr.storage.memory_mixins.regioned_memory.region_category_mixin
module, 535
angr.storage.memory_mixins.regioned_memory.region_data
module, 533
angr.storage.memory_mixins.regioned_memory.region_meta_mixin
module, 536
angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin
module, 538
angr.storage.memory_mixins.regioned_memory.regioned_memory
module, 530
angr.storage.memory_mixins.regioned_memory.static_find_mixin
module, 535
angr.storage.memory_mixins.simple_interface_mixin
module, 493
angr.storage.memory_mixins.simplification_mixin
module, 498
angr.storage.memory_mixins.size_resolution_mixin
module, 494
angr.storage.memory_mixins.slotted_memory
module, 539
angr.storage.memory_mixins.smart_find_mixin
module, 490
angr.storage.memory_mixins.symbolic_merger_mixin
module, 493
angr.storage.memory_mixins.top_merger_mixin
module, 503
angr.storage.memory_mixins.underconstrained_mixin
module, 492
angr.storage.memory_mixins.unwrapper_mixin
module, 498
angr.storage.memory_object
module, 451
angr.tails
module, 1344
angr.utils.algo
module, 1344
angr.utils.constants
module, 1344
angr.utils.dict
module, 1344
angr.utils.dynamic_dictlist
module, 1346
angr.utils.enums_conv
module, 1346
angr.utils.env
module, 1346
angr.utils.formatting
module, 1353
angr.utils.graph
module, 1346
angr.utils.lazy_import
module, 1351
angr.utils.mixins
module, 1351
angr.utils.orderedset
module, 1351
angr.utils.mp
module, 1354
angr.utils.timing
module, 1353

module, 871
AngrAIError, 156, 1361
AngrAnalysisError, 156, 1356
AngrAnnotatedCFGError, 156, 1356
AngrAssemblyError, 156, 1356
AngrBackwardSlicingError, 156, 1356
AngrBladeError, 157, 1356
AngrBladeSimProcError, 157, 1356
AngrCallableError, 157, 1356
AngrCallableMultistateError, 157, 1356
AngrCFGError, 157, 1357
AngrCorruptDBError, 157, 1357
AngrDataGraphError, 157, 1357
AngrDB (*class in angr.angrdb*), 1018
AngrDB (*class in angr.angrdb.db*), 1020
AngrDBError, 157, 1357
AngrDDGError, 157, 1357
AngrDecompilationError, 157, 1361
AngrDelayJobNotice, 157, 1356
AngrDirectorError, 157, 1357
AngrError, 157, 1355
AngrExitError, 157, 1355
AngrExplorationTechniqueError, 157, 1357
AngrExplorerError, 157, 1357
AngrForwardAnalysisError, 157, 1356
AngrIncompatibleDBError, 157, 1357
AngrIncongruencyError, 157, 1356
AngrInvalidArgumentError, 158, 1355
AngrJobMergingFailureNotice, 158, 1357
AngrJobWideningFailureNotice, 158, 1357
AngrLifterError, 158, 1355
AngrLoopAnalysisError, 158, 1357
AngrMissingTypeError, 158, 1356
AngrNoPluginError, 158, 1361
AngrObjectFactory (*class in angr.factory*), 227
AngrPathError, 158, 1355
AngrRuntimeDbError, 1357
AngrRuntimeError, 158, 1355
AngrSimOSError, 158, 1356
AngrSkipJobNotice, 158, 1356
AngrSurveyorError, 158, 1356
AngrSyscallError, 158, 1356
AngrTracerError, 158, 1357
AngrTypeError, 158, 1356
AngrUnsupportedSyscallError, 158, 1360
AngrValueError, 159, 1355
AngrVariableRecoveryError, 1357
AngrVaultError, 159, 1355
AngrVFGError, 158, 1357
AngrVFGRestartAnalysisNotice, 159, 1357
annotate_mv_with_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1232
annotate_mv_with_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1206
annotate_with_def() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 1191
annotate_with_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsS method*), 1232
annotate_with_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1206
annotate_with_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi static method*), 849
annotate_with_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 826
annotate_with_variables() (*angr.analyses.variable_recovery.variable_recovery_base.Variabl static method*), 1249
annotated_cfg() (*angr.analyses.backward_slice.BackwardSlice method*), 947
annotated_cfg() (*angr.analyses.BackwardSlice method*), 882
AnnotatedCFG (*class in angr.annocfg*), 1321
annotations (*angr.state_plugins.sim_action_object.SimActionObject property*), 641
annotations (*angr.state_plugins.SimActionObject property*), 265
ansi_color() (*in module angr.utils.formatting*), 1353
app_types (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
append_data() (*angr.analyses.Reassembler method*), 918
append_data() (*angr.analyses.reassembler.Reassembler method*), 1304
append_procedure() (*angr.analyses.Reassembler method*), 918
append_procedure() (*angr.analyses.reassembler.Reassembler method*), 1304
append_state() (*angr.analyses.vfg.VFGNode method*), 1288
append_statement() (*in module angr.analyses.decompiler.utils*), 1150
apply() (*angr.analyses.cfg.cfg_fast.FunctionCallEdge method*), 986
apply() (*angr.analyses.cfg.cfg_fast.FunctionEdge method*), 985
apply() (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge method*), 986
apply() (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge method*), 987
apply() (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge

method), 986
apply() (*angr.calling_conventions.AllocHelper method*), 658
apply() (*angr.sim_manager.SimulationManager method*), 550
apply() (*angr.SimulationManager method*), 219
apply_at_callsite (*angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute*), 1222
apply_definition() (*angr.knowledge_plugins.Function method*), 724
apply_definition() (*angr.knowledge_plugins.functions.Function method*), 784
apply_definition() (*angr.knowledge_plugins.functions.function.Function method*), 809
apply_function_edges() (*angr.analyses.cfg.cfg_fast.CFGJob method*), 988
apply_patches_to_binary() (*angr.knowledge_plugins.patches.PatchManager method*), 738
apply_patches_to_binary() (*angr.knowledge_plugins.PatchManager method*), 732
apply_patches_to_state() (*angr.knowledge_plugins.patches.PatchManager method*), 738
apply_patches_to_state() (*angr.knowledge_plugins.PatchManager method*), 732
arch (*angr.analyses.cfg.cfg_arch_options.CFGArchOptions attribute*), 993
arch (*angr.analyses.cfg.CFGArchOptions attribute*), 959
arch (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessor attribute*), 1000
arch (*angr.analyses.CFGArchOptions attribute*), 885
arch (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
arch (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
arch (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState attribute*), 1231
arch (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis attribute*), 1205
arch (*angr.analyses.reaching_definitions.Register attribute*), 1211
arch (*angr.analyses.typehoon.translator.TypeTranslator attribute*), 1265
arch (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase attribute*), 1249
arch (*angr.Block attribute*), 160
arch (*angr.block.Block attribute*), 233
arch (*angr.block.DisassemblerBlock attribute*), 232
ARCH (*angr.calling_conventions.SimCC attribute*), 664
ARCH (*angr.calling_conventions.SimCCAArch64 attribute*), 674
ARCH (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 674
ARCH (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 672
ARCH (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 672
ARCH (*angr.calling_conventions.SimCCARM attribute*), 672
ARCH (*angr.calling_conventions.SimCCARMHF attribute*), 673
ARCH (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 673
ARCH (*angr.calling_conventions.SimCCCdecl attribute*), 668
ARCH (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 670
ARCH (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 669
ARCH (*angr.calling_conventions.SimCCN64 attribute*), 676
ARCH (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 676
ARCH (*angr.calling_conventions.SimCCO32 attribute*), 675
ARCH (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 675
ARCH (*angr.calling_conventions.SimCCPowerPC attribute*), 677
ARCH (*angr.calling_conventions.SimCCPowerPC64 attribute*), 677
ARCH (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 678
ARCH (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 677
ARCH (*angr.calling_conventions.SimCCRISCV64 attribute*), 674
ARCH (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 675
ARCH (*angr.calling_conventions.SimCCS390X attribute*), 679
ARCH (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 679
ARCH (*angr.calling_conventions.SimCCSoot attribute*), 678
ARCH (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 671
ARCH (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 670
ARCH (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 671
arch (*angr.engines.pcode.lifter.IRSB attribute*), 608
arch (*angr.engines.pcode.lifter.Lifter attribute*), 610
arch (*angr.engines.pcode.lifter.PcodeDisassemblerBlock

attribute), 606
arch (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
arch (*angr.knowledge_plugins.key_definitions.atoms.Register attribute*), 839
arch (*angr.knowledge_plugins.key_definitions.live_definitions.Definitions attribute*), 848
arch (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 824
arch (*angr.Project attribute*), 170
arch (*angr.project.Project attribute*), 224
arch (*angr.sim_state.SimState property*), 240
ARCH (*angr.SimCC attribute*), 174
arch (*angr.SimState property*), 209
ARCH() (*angr.engines.pcode.cc.SimCCPCodeBase class method*), 636
arch_overridable() (*in module angr.sim_state*), 238
ARCHES (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePtrSaveSimplifier attribute*), 1097
ARCHES (*angr.analyses.decompiler.optimization_passes.BaseOptimizationPass attribute*), 1072
ARCHES (*angr.analyses.decompiler.optimization_passes.CallSiteMaker attribute*), 1072
ARCHES (*angr.analyses.decompiler.optimization_passes.CodeMotion attribute*), 1073
ARCHES (*angr.analyses.decompiler.optimization_passes.ConstantDerivationPropagation attribute*), 1074
ARCHES (*angr.analyses.decompiler.optimization_passes.constant_dereferences.ConstantDereferencesSimplifier attribute*), 1091
ARCHES (*angr.analyses.decompiler.optimization_passes.ConstantDerivationPropagation attribute*), 1075
ARCHES (*angr.analyses.decompiler.optimization_passes.ConstantMultiplication attribute*), 1074
ARCHES (*angr.analyses.decompiler.optimization_passes.DeadassignmentRemover attribute*), 1076
ARCHES (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifier attribute*), 1098
ARCHES (*angr.analyses.decompiler.optimization_passes.DivSimplifier attribute*), 1076
ARCHES (*angr.analyses.decompiler.optimization_passes.EagerReturnsConstructor attribute*), 1078
ARCHES (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.ExprOpSwapper attribute*), 1103
ARCHES (*angr.analyses.decompiler.optimization_passes.ExpressionPropagator attribute*), 1078
ARCHES (*angr.analyses.decompiler.optimization_passes.FlipBooleanCmp attribute*), 1079
ARCHES (*angr.analyses.decompiler.optimization_passes.InlinedAssertion attribute*), 1080
ARCHES (*angr.analyses.decompiler.optimization_passes.InlinedCallocSimplifier attribute*), 1080
ARCHES (*angr.analyses.decompiler.optimization_passes.InlinedMemcpy attribute*), 1081
ARCHES (*angr.analyses.decompiler.optimization_passes.InlinedMemset attribute*), 1081
ARCHES (*angr.analyses.decompiler.optimization_passes.InlinedStrlenSimplifier attribute*), 1082
ARCHES (*angr.analyses.decompiler.optimization_passes.InlinedWcscpySimplifier attribute*), 1082
ARCHES (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 1099
ARCHES (*angr.analyses.decompiler.optimization_passes.ITEExprConverter attribute*), 1079
ARCHES (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 1080
ARCHES (*angr.analyses.decompiler.optimization_passes.MipsGpSettingSimplifier attribute*), 1083
ARCHES (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 1101
ARCHES (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 1084
ARCHES (*angr.analyses.decompiler.optimization_passes.optimization_pass.BinopSimplifier attribute*), 1093
ARCHES (*angr.analyses.decompiler.optimization_passes.optimization_pass.SaveSimplifier attribute*), 1096
ARCHES (*angr.analyses.decompiler.optimization_passes.register_save_area_rewriter.RegisterSaveAreaRewriter attribute*), 1103
ARCHES (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifier attribute*), 1084
ARCHES (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifierAdvanced attribute*), 1085
ARCHES (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 1103
ARCHES (*angr.analyses.decompiler.optimization_passes.RetAddrSaveSimplifier attribute*), 1085
ARCHES (*angr.analyses.decompiler.optimization_passes.ReturnDeduplicator attribute*), 1086
ARCHES (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorHigh attribute*), 1086
ARCHES (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorLow attribute*), 1087
ARCHES (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 1097
ARCHES (*angr.analyses.decompiler.optimization_passes.StackCanarySimplifier attribute*), 1088
ARCHES (*angr.analyses.decompiler.optimization_passes.StaticVVarRewriter attribute*), 1089
ARCHES (*angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseDuplicator attribute*), 1089
ARCHES (*angr.analyses.decompiler.optimization_passes.SwitchReusedEntryRewriter attribute*), 1090
ARCHES (*angr.analyses.decompiler.optimization_passes.TagSlicer attribute*), 1090
ARCHES (*angr.analyses.decompiler.optimization_passes.WinStackCanarySimplifier attribute*), 1090
ARCHES (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier attribute*), 1104
ARCHES (*angr.analyses.decompiler.optimization_passes.X86GccGetPcSimplifier attribute*), 1105

attribute), 1091
arches() (*angr.calling_conventions.SimCC class method*), 667
arches() (*angr.SimCC class method*), 177
arg_list (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 1132
arg_locs() (*angr.calling_conventions.SimCC method*), 665
arg_locs() (*angr.calling_conventions.SimCCMicrosoftThiscall method*), 668
arg_locs() (*angr.SimCC method*), 176
arg_names (*angr.sim_type.SimTypeFunction attribute*), 698
ARG_REGS (*angr.calling_conventions.SimCC attribute*), 663
ARG_REGS (*angr.calling_conventions.SimCCAArch64 attribute*), 673
ARG_REGS (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 674
ARG_REGS (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 672
ARG_REGS (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 672
ARG_REGS (*angr.calling_conventions.SimCCARM attribute*), 672
ARG_REGS (*angr.calling_conventions.SimCCARMHF attribute*), 673
ARG_REGS (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 673
ARG_REGS (*angr.calling_conventions.SimCCCdecl attribute*), 668
ARG_REGS (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 669
ARG_REGS (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 669
ARG_REGS (*angr.calling_conventions.SimCCMicrosoftThiscall attribute*), 668
ARG_REGS (*angr.calling_conventions.SimCCN64 attribute*), 676
ARG_REGS (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 676
ARG_REGS (*angr.calling_conventions.SimCCO32 attribute*), 675
ARG_REGS (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 675
ARG_REGS (*angr.calling_conventions.SimCCPowerPC attribute*), 676
ARG_REGS (*angr.calling_conventions.SimCCPowerPC64 attribute*), 677
ARG_REGS (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 677
ARG_REGS (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 677
ARG_REGS (*angr.calling_conventions.SimCCRISCV64 attribute*), 674
ARG_REGS (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 674
ARG_REGS (*angr.calling_conventions.SimCCS390X attribute*), 679
ARG_REGS (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 679
ARG_REGS (*angr.calling_conventions.SimCCSoot attribute*), 678
ARG_REGS (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 671
ARG_REGS (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 670
ARG_REGS (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 671
ARG_REGS (*angr.engines.pcode.cc.SimCCM68k attribute*), 636
ARG_REGS (*angr.engines.pcode.cc.SimCCPARISC attribute*), 637
ARG_REGS (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 637
ARG_REGS (*angr.engines.pcode.cc.SimCCRISCV attribute*), 636
ARG_REGS (*angr.engines.pcode.cc.SimCCSH4 attribute*), 637
ARG_REGS (*angr.engines.pcode.cc.SimCCSPARC attribute*), 637
ARG_REGS (*angr.engines.pcode.cc.SimCCXtensa attribute*), 638
ARG_REGS (*angr.SimCC attribute*), 174
arg_session (*angr.sim_procedure.SimProcedure attribute*), 644
arg_session() (*angr.calling_conventions.SimCC method*), 665
arg_session() (*angr.SimCC method*), 175
arg_vvars (*angr.analyses.decompiler.clinic.Clinic attribute*), 1060
arg_vvars (*angr.analyses.decompiler.decompilation_cache.Decompilation attribute*), 1064
args (*angr.ailment.expression.Call attribute*), 1393
args (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 1136
args (*angr.sim_type.SimTypeFunction attribute*), 698
args (*angr.state_plugins.sim_action_object.SimActionObject property*), 641
args (*angr.state_plugins.SimActionObject property*), 265
args (*angr.utils.mp.Closure attribute*), 1354
args_atoms (*angr.analyses.reaching_definitions.function_handler.Function attribute*), 1224
args_atoms (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
args_defns (*angr.analyses.reaching_definitions.function_handler.Function attribute*), 1226
args_defns (*angr.analyses.reaching_definitions.FunctionCallRelationship

attribute), 1186
ARGS_MISMATCH (*angr.procedures.stubs.format_parser.FormatParser attribute*), 648
ARGS_MISMATCH (*angr.sim_procedure.SimProcedure attribute*), 645
ARGS_MISMATCH (*angr.SimProcedure attribute*), 204
args_values (*angr.analyses.reaching_definitions.function_handler.FunctionCallDefs attribute*), 1224
args_values (*angr.analyses.reaching_definitions.FunctionCallDefs attribute*), 1183
ArgSession (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 670
ArgSession (*angr.calling_conventions.SimCCUsercall attribute*), 667
ArgSession (*class in angr.calling_conventions*), 663
argument_types (*angr.sim_procedure.SimProcedure property*), 646
argument_types (*angr.SimProcedure property*), 206
arguments (*angr.knowledge_plugins.Function property*), 723
arguments (*angr.knowledge_plugins.functions.Function property*), 783
arguments (*angr.knowledge_plugins.functions.function.Function property*), 808
ArithmeticExpression (*class in angr.engines.light*), 1167
ArithmeticExpression (*class in angr.engines.light.data*), 1166
ARM (*angr.analyses.cfg.cfg_fast.ARMDecodingMode attribute*), 984
ARMDecodingMode (*class in angr.analyses.cfg.cfg_fast*), 983
ArmElfFastResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1007
ArmElfFastResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast*), 994
Array (*class in angr.ailment.expression*), 1395
Array (*class in angr.analyses.typehoon.typeconsts*), 1278
array() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 422
array() (*angr.state_plugins.SimDebugVariable method*), 267
array() (*angr.state_plugins.SimMemView method*), 285
array() (*angr.state_plugins.view.SimMemView method*), 431
assembly() (*angr.analyses.Reassembler method*), 918
assembly() (*angr.analyses.reassembler.BasicBlock method*), 1300
assembly() (*angr.analyses.reassembler.Data method*), 1302
assembly() (*angr.analyses.reassembler.Instruction method*), 1299
assembly() (*angr.analyses.reassembler.Operand method*), 1299
assembly() (*angr.analyses.reassembler.Procedure method*), 1300
assembly() (*angr.analyses.reassembler.Reassembler method*), 1304
assign() (*angr.state_plugins.uc_manager.SimUCManager method*), 325
assign() (*angr.state_plugins.uc_manager.SimUCManager method*), 387
assign_labels() (*angr.analyses.reassembler.BasicBlock method*), 1300
assign_labels() (*angr.analyses.reassembler.Data method*), 1302
assign_labels() (*angr.analyses.reassembler.Instruction method*), 1299
assign_labels() (*angr.analyses.reassembler.Procedure method*), 1300
assign_unified_variable_names() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 817
assign_variable_names() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 817
Assignment (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.Address attribute*), 998
Assignment (*class in angr.ailment*), 1365
Assignment (*class in angr.ailment.statement*), 1397
assignments (*angr.analyses.decompiler.region_simplifiers.expr_folding.E attribute*), 1112
ast (*angr.analyses.data_dep.BaseDepNode property*), 1316
ast (*angr.analyses.data_dep.dep_nodes.BaseDepNode property*), 1314
ast (*angr.state_plugins.sim_action_object.SimActionObject attribute*), 640
ast (*angr.state_plugins.SimActionObject attribute*), 265
AST (*class in angr.analyses.ddg*), 1156
ast_graph (*angr.analyses.DDG property*), 876
ast_graph (*angr.analyses.ddg.DDG property*), 1160
ast_preserving_op() (*in module angr.state_plugins.sim_action_object*), 640
ast_stripping_decorator() (*in module angr.state_plugins.sim_action_object*), 640
ast_weight() (*in module angr.exploration_techniques.suggestions*), 586
at_new_block() (*angr.analyses.reaching_definitions.ReachingDefinitions method*), 1202
at_new_block() (*angr.knowledge_plugins.key_definitions.rd_model.Reach method*), 855
at_new_block() (*angr.knowledge_plugins.key_definitions.ReachingDefini method*), 833
at_new_stmt() (*angr.analyses.reaching_definitions.ReachingDefinitionsM*

method), 1201
at_new_stmt() (angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method), 855
at_new_stmt() (angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method), 832
atoi_dumps() (angr.state_plugins.trace_additions.ChallRespInfo static method), 384
atom (angr.analyses.reaching_definitions.Definition attribute), 1182
atom (angr.knowledge_plugins.key_definitions.Definition attribute), 823
atom (angr.knowledge_plugins.key_definitions.definition.Definition attribute), 843
Atom (class in angr.ailment.expression), 1376
Atom (class in angr.analyses.reaching_definitions), 1178
Atom (class in angr.knowledge_plugins.key_definitions.atoms), 836
atom_hash (angr.knowledge_plugins.variables.variable_access.VariableAccess attribute), 812
AtomKind (class in angr.analyses.reaching_definitions), 1180
AtomKind (class in angr.knowledge_plugins.key_definitions.atoms), 836
atypical_alignments (angr.analyses.PatchFinderAnalysis attribute), 910
available_flavors() (angr.knowledge_plugins.structured_code.StructuredCodeManager method), 822
available_flavors() (angr.knowledge_plugins.StructuredCodeManager method), 734

B

b() (angr.state_plugins.inspect.SimInspector method), 341
b() (angr.state_plugins.SimInspector method), 277
back_edges() (angr.analyses.forward_analysis.FunctionGraphVisitor method), 933
back_edges() (angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor method), 942
back_edges() (angr.analyses.forward_analysis.visitors.FunctionGraphVisitor method), 939
back_edges() (angr.analyses.forward_analysis.visitors.graph.GraphVisitor method), 943
backend (angr.angrdb.models.DbObject attribute), 1022
backend2name (angr.angrdb.serializers.loader.LoaderSerializer attribute), 1029
backend2name (angr.angrdb.serializers.LoaderSerializer attribute), 1026
backend_args (angr.angrdb.models.DbObject attribute), 1022
backpatch() (angr.analyses.typehoon.translator.TypeTranslator method), 1266
backtrace (angr.state_plugins.inspect.InspectAttrs attribute), 326
backtrace (angr.state_plugins.InspectAttrs attribute), 254
BackwardSlice (angr.analyses.analysis.KnownAnalysesPlugin attribute), 929
BackwardSlice (class in angr.analyses), 881
BackwardSlice (class in angr.analyses.backward_slice), 946
BadJumpkindNotification, 1319
BadStatesDropper (class in angr.distributed.worker), 1363
base (angr.ailment.expression.BasePointerOffset attribute), 1390
base (angr.ailment.expression.Extract attribute), 1391
base (angr.ailment.expression.Insert attribute), 1392
base (angr.sim_type.SimType attribute), 686
base (angr.sim_type.SimTypeFunction attribute), 698
base (angr.sim_variable.SimStackVariable attribute), 686
base (angr.storage.memory_object.SimMemoryObject attribute), 451
base (angr.storage.SimMemoryObject attribute), 425
base_addr (angr.sim_variable.SimStackVariable attribute), 686
base_addr_available (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTarget property), 1000
base_address (angr.storage.memory_mixins.regioned_memory.region_da attribute), 534
base_type (angr.sim_type.SimTypeBitfield property), 704
base_type (angr.sim_type.SimTypeEnum property), 703
BaseDepNode (class in angr.analyses.data_dep), 1316
BaseDepNode (class in angr.analyses.data_dep.dep_nodes), 1313
BaseGoal (class in angr.exploration_techniques.director), 579
BaseLabel (class in angr.analyses.typehoon.typevars), 1271
BaseOptimizationPass (class in angr.analyses.decompiler.optimization_passes.optimization_pass), 1093
BasePointerOffset (class in angr.ailment.expression), 1390
BasePointerSaveSimplifier (class in angr.analyses.decompiler.optimization_passes), 1071
BasePointerSaveSimplifier (class in angr.analyses.decompiler.optimization_passes.base_ptr_save_sim 1097
BaseProxiNode (class in angr.analyses.proximity_graph), 1309
BaseStructuredCodeGenerator (class in angr.analyses.decompiler.structured_codegen),

1120
BaseStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.base*), 1129
basetype (*angr.analyses.typehoon.typeconsts.Pointer attribute*), 1277
basic_spec (*angr.procedures.stubs.format_parser.FormatParser attribute*), 648
basic_spec (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 648
BasicBlock (*class in angr.analyses.reassembler*), 1299
basicblock_key (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.AbstractLocation property*), 536
BasicClaripyCooperation (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 521
bbl_addr (*angr.code_location.AILCodeLocation property*), 867
bbl_addr (*angr.errors.SimError attribute*), 1358
bbl_addr (*angr.knowledge_plugins.key_definitions.definition.Definition property*), 842
bbl_addr (*angr.SimError attribute*), 178
bbl_addrs (*angr.state_plugins.history.SimStateHistory property*), 378
bbl_addrs (*angr.state_plugins.SimStateHistory property*), 303
bck_chunk() (*angr.PTChunk method*), 168
bck_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 414
bck_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 417
bck_chunk() (*angr.state_plugins.heap.PTChunk method*), 405
bck_chunk() (*angr.state_plugins.PTChunk method*), 258
BEFORE (*angr.state_plugins.inspect.When attribute*), 339
BEFORE_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 1092
BEFORE_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.OptimizationPass attribute*), 1084
BEFORE_SSA_LEVEL0_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 1092
BEFORE_SSA_LEVEL0_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.OptimizationPass attribute*), 1084
BEFORE_SSA_LEVEL1_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 1092
BEFORE_SSA_LEVEL1_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.OptimizationPass attribute*), 1084
BEFORE_VARIABLE_RECOVERY (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 1092
BEFORE_VARIABLE_RECOVERY (*angr.analyses.decompiler.optimization_passes.OptimizationPass attribute*), 1084
begin_txn() (*angr.knowledge_plugins.RuntimeDb method*), 734
BehaviorFactory (*class in angr.engines.pcode.behavior*), 635
behaviors (*angr.engines.pcode.lifter.IRSB attribute*), 608
behaviors (*angr.engines.pcode.lifter.PcodeBasicBlockLifter attribute*), 612
bfs_nodes(n) (*angr.analyses.decompiler.optimization_passes.optimization_ method*), 1095
binary (*angr.knowledge_plugins.Function property*), 720
binary (*angr.knowledge_plugins.functions.Function property*), 780
binary (*angr.knowledge_plugins.functions.function.Function property*), 805
binary_insert() (*in module angr.utils.algo*), 1344
binary_name (*angr.knowledge_plugins.Function attribute*), 717
binary_name (*angr.knowledge_plugins.functions.Function attribute*), 777
binary_name (*angr.knowledge_plugins.functions.function.Function attribute*), 802
binary_name (*angr.knowledge_plugins.functions.soot_function.SootFunctio attribute*), 810
BinaryError, 1297
BinaryOp (*class in angr.ailment*), 1366
BinaryOp (*class in angr.ailment.expression*), 1384
BinaryOptimizer (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
BinaryOptimizer (*class in angr.analyses*), 884
BinaryOptimizerOptimizationPassStage (*class in angr.analyses.binary_optimizer*), 1306
BinDiff (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
BinDiff (*class in angr.analyses*), 883
BinDiff (*class in angr.analyses.bindiff*), 949
binop() (*angr.engines.vex.irop.IROpEnumerator static method*), 1374
binop_handler() (*angr.engines.light.engine.SimEngineLightVEXProcessor static method*), 1172
binop_handler() (*angr.engines.light.SimEngineLightVEX static method*), 1169
binop_handler() (*angr.decompiler.decompilation_cache.DecompilationCache attribute*), 1065
binopv_handler() (*angr.engines.light.engine.SimEngineLightVEXProcessor static method*), 1172
binopv_handler() (*angr.engines.light.SimEngineLightVEX

static method), 1169
bitlen (angr.analyses.stack_pointer_tracker.Register attribute), 1243
bits (angr.ailment.Expression attribute), 1369
bits (angr.ailment.expression.Array attribute), 1395
bits (angr.ailment.expression.Expression attribute), 1376
bits (angr.ailment.expression.RustEnum attribute), 1395
bits (angr.ailment.statement.CAS property), 1405
bits (angr.ailment.statement.SideEffectStatement property), 1402
bits (angr.analyses.propagator.values.Top property), 1175
bits (angr.analyses.reaching_definitions.Atom property), 1179
bits (angr.analyses.typehoon.typevars.HasField attribute), 1272
bits (angr.analyses.variable_recovery.engine_base.RichR property), 1257
bits (angr.engines.light.data.RegisterOffset property), 1167
bits (angr.engines.light.RegisterOffset property), 1167
bits (angr.knowledge_plugins.key_definitions.atoms.Atom property), 836
bits (angr.sim_variable.SimComboRegisterVariable property), 684
bits (angr.sim_variable.SimVariable property), 681
Blade (class in angr), 159
Blade (class in angr.blade), 1319
blank_state() (angr.factory.AngrObjectFactory method), 228
blob (angr.angrdb.models.DbCFGModel attribute), 1023
blob (angr.angrdb.models.DbFunction attribute), 1023
blob (angr.angrdb.models.DbVariableCollection attribute), 1023
blob (angr.angrdb.models.DbXRefs attribute), 1024
block (angr.analyses.decompiler.structured_codegen.c.CAILBlock attribute), 1133
Block (angr.analyses.reaching_definitions.subject.SubjectType attribute), 1238
block (angr.engines.light.engine.SimEngineLight attribute), 1171
block (angr.engines.light.SimEngineLight attribute), 1168
block (angr.knowledge_plugins.cfg.cfg_node.CFGNode property), 768
block (angr.knowledge_plugins.cfg.CFGNode property), 751
Block (class in angr), 160
Block (class in angr.ailment), 1367
Block (class in angr.ailment.block), 1406
Block (class in angr.block), 233
block() (angr.analyses.decompiler.Clinic method), 1045
block() (angr.analyses.decompiler.clinic.Clinic method), 1061
block() (angr.factory.AngrObjectFactory method), 231
block() (angr.sim_state.SimState method), 241
block() (angr.SimState method), 210
block_addr (angr.analyses.decompiler.clinic.DataRefDesc attribute), 1057
block_addr (angr.analyses.decompiler.region_simplifiers.expr_folding.Exp attribute), 1110
block_addr (angr.analyses.decompiler.region_simplifiers.expr_folding.Sta attribute), 1110
block_addr (angr.analyses.reaching_definitions.call_trace.CallSite attribute), 1212
block_addr (angr.code_location.AILCodeLocation property), 867
block_addr (angr.code_location.CodeLocation attribute), 866
block_addr (angr.knowledge_plugins.xrefs.XRef attribute), 861
block_addr (angr.knowledge_plugins.xrefs.xref.XRef attribute), 863
block_addr (angr.state_plugins.unicorn_engine.BlockDetails attribute), 393
block_addr (angr.state_plugins.unicorn_engine.StopDetails attribute), 395
block_addrs (angr.knowledge_plugins.Function property), 718
block_addrs (angr.knowledge_plugins.functions.Function property), 778
block_addrs (angr.knowledge_plugins.functions.function.Function property), 803
block_addrs_set (angr.knowledge_plugins.Function property), 718
block_addrs_set (angr.knowledge_plugins.functions.Function property), 779
block_addrs_set (angr.knowledge_plugins.functions.function.Function property), 803
block_beginning_state() (angr.knowledge_plugins.propagations.PropagationModel method), 775
block_count (angr.state_plugins.history.SimStateHistory property), 377
block_count (angr.state_plugins.SimStateHistory property), 303
block_id (angr.analyses.cfg.cfg_emulated.CFGJob property), 976
block_id (angr.analyses.vfg.PendingJob attribute), 1286
block_id (angr.analyses.vfg.VFGJob property), 1285
block_id (angr.knowledge_plugins.cfg.cfg_node.CFGNode property), 767
block_id (angr.knowledge_plugins.cfg.CFGNode property), 750
block_idx (angr.ailment.tagged_object.TagDict attribute), 1407
block_idx (angr.analyses.decompiler.region_simplifiers.expr_folding.Expr

attribute), 1110
block_idx (*angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter attribute*), 1110
block_idx (*angr.code_location.AILCodeLocation attribute*), 867
block_idx (*angr.code_location.CodeLocation attribute*), 866
block_initial_reg_values (*angr.knowledge_plugins.propagations.PropagationModel attribute*), 775
block_matches (*angr.analyses.bindiff.FunctionDiff property*), 949
BLOCK_MAX_SIZE (*angr.Block attribute*), 160
BLOCK_MAX_SIZE (*angr.block.Block attribute*), 233
block_similarity() (*angr.analyses.bindiff.FunctionDiff method*), 949
block_size (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 393
block_size (*angr.state_plugins.unicorn_engine.StopDetails attribute*), 395
block_trace_ind (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 393
BlockCache (*class in angr.analyses.decompiler.clinic*), 1056
BlockCache (*class in angr.analyses.decompiler.optimization_passes.const_derefs*), 1092
BlockDetails (*class in angr.state_plugins.unicorn_engine*), 393
BlockID (*class in angr.knowledge_plugins.cfg*), 740
BlockLocator (*class in angr.analyses.decompiler.optimization_passes.ite_to_cmov*), 1098
BlockNode (*class in angr.codenode*), 1323
BlockProtocol (*class in angr.engines.light.engine*), 1170
blocks (*angr.analyses.decompiler.ail_simplifier.AILSimplifier attribute*), 1054
blocks (*angr.knowledge_plugins.Function property*), 718
blocks (*angr.knowledge_plugins.functions.Function property*), 778
blocks (*angr.knowledge_plugins.functions.function.Function property*), 802
blocks_by_addr (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass property*), 1095
blocks_by_addr_and_idx (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass property*), 1095
blocks_probably_identical() (*angr.analyses.bindiff.FunctionDiff method*), 949
BLOCKS_THRESHOLD (*angr.analyses.binary_optimizer.BinaryOptimizer attribute*), 1306
BLOCKS_THRESHOLD (*angr.analyses.BinaryOptimizer attribute*), 884
blocks_with_differing_constants (*angr.analyses.BinDiff property*), 883
blocks_with_differing_constants (*angr.analyses.bindiff.BinDiff property*), 950
blocks_with_differing_constants (*angr.analyses.bindiff.FunctionDiff property*), 949
BlockSimplifier (*class in angr.analyses.decompiler*), 1039
BlockSimplifier (*class in angr.analyses.decompiler.block_simplifier*), 1054
BlockStart (*class in angr.analyses.disassembly*), 1294
BlockWalker (*class in angr.analyses.decompiler.optimization_passes.const_derefs*), 1091
body (*angr.analyses.decompiler.structured_codegen.c.CDoWhileLoop attribute*), 1134
body (*angr.analyses.decompiler.structured_codegen.c.CForLoop attribute*), 1134
body (*angr.analyses.decompiler.structured_codegen.c.CWhileLoop attribute*), 1133
boolean_operations_pin_condition() (*angr.analyses.decompiler.optimization_passes.DuplicationReverter static method*), 1077
booleanize() (*angr.engines.pcode.behavior.OpBehavior class method*), 617
BOTH (*angr.state_plugins.inspect.When attribute*), 339
both_iter (*angr.calling_conventions.ArgSession attribute*), 663
both_iter (*angr.calling_conventions.MicrosoftAMD64ArgSession attribute*), 669
both_iter (*angr.calling_conventions.SimCC.ArgSession attribute*), 664
both_iter (*angr.SimCC.ArgSession attribute*), 175
Bottom (*class in angr.analyses.propagator.values*), 1175
BottomType (*class in angr.analyses.stack_pointer_tracker*), 1243
BottomType (*class in angr.analyses.typehoon.typeconsts*), 1274
BoyScout (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
BoyScout (*class in angr.analyses*), 884
BoyScout (*class in angr.analyses.boyscout*), 950
BP (*class in angr.OptimizationPass*), 155
BP (*class in angr.state_plugins.inspect*), 339
BP_AFTER (*angr.state_plugins.inspect.SimInspector attribute*), 340
BP_AFTER (*angr.state_plugins.SimInspector attribute*), 276
BP_BEFORE (*angr.state_plugins.inspect.SimInspector attribute*), 340

BP_BEFORE (*angr.state_plugins.SimInspector attribute*), 276
BP_BOTH (*angr.state_plugins.inspect.SimInspector attribute*), 340
BP_BOTH (*angr.state_plugins.SimInspector attribute*), 276
BP_IPDB() (*in module angr*), 155
BP_IPDB() (*in module angr.state_plugins*), 246
BP_IPDB() (*in module angr.state_plugins.inspect*), 339
BP_IPYTHON() (*in module angr*), 156
BP_IPYTHON() (*in module angr.state_plugins*), 246
BP_IPYTHON() (*in module angr.state_plugins.inspect*), 339
bp_on_stack (*angr.knowledge_plugins.Function attribute*), 717
bp_on_stack (*angr.knowledge_plugins.functions.Function attribute*), 777
bp_on_stack (*angr.knowledge_plugins.functions.function.Function attribute*), 802
bp_on_stack (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 810
branch() (*angr.analyses.ddg.LiveDefinitions method*), 1157
BREAKPOINT (*angr.EmulatorStopReason attribute*), 162
breakpoints (*angr.Emulator property*), 162
bss_memory_read_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.BSSHook method*), 1001
bss_memory_write_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.BSSHook method*), 1002
BSSHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 1001
Bucketizer (*class in angr.exploration_techniques*), 553
Bucketizer (*class in angr.exploration_techniques.bucketizer*), 586
build() (*angr.analyses.disassembly.Operand static method*), 1295
bv_slice() (*in module angr.storage.memory_object*), 451
bv_to_fp() (*angr.state_plugins.sim_action_object.SimActionObject method*), 641
bv_to_fp() (*angr.state_plugins.SimActionObject method*), 266
BVS() (*angr.state_plugins.SimSolver method*), 289
BVS() (*angr.state_plugins.solver.SimSolver method*), 362
byte_string (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
byte_string (*angr.knowledge_plugins.cfg.CFGNode property*), 750
bytes (*angr.Block property*), 161
bytes (*angr.block.Block property*), 234
bytes_at() (*angr.storage.memory_object.SimMemoryObject method*), 451
bytes_at() (*angr.storage.SimMemoryObject method*), 426
bytes_offset (*angr.engines.pcode.lifter.Lifter attribute*), 610
bytes_offset (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
bytestr (*angr.codenode.BlockNode attribute*), 1323

C

c_args_as_atoms() (*angr.analyses.reaching_definitions.function_handler static method*), 1229
c_args_as_atoms() (*angr.analyses.reaching_definitions.FunctionHandler static method*), 1188
c_repr() (*angr.analyses.decompiler.structured_codegen.c.CConstruct method*), 1131
c_repr() (*angr.analyses.typehoon.translator.SimTypeTempRef method*), 1265
c_repr() (*angr.sim_type.SimStruct method*), 702
c_repr() (*angr.sim_type.SimType method*), 687
c_repr() (*angr.sim_type.SimTypeArray method*), 696
c_repr() (*angr.sim_type.SimTypeBitfield method*), 705
c_repr() (*angr.sim_type.SimTypeBottom method*), 689
c_repr() (*angr.sim_type.SimTypeEnum method*), 704
c_repr() (*angr.sim_type.SimTypeFixedSizeInt method*), 691
c_repr() (*angr.sim_type.SimTypeFunction method*), 699
c_repr() (*angr.sim_type.SimTypeInt method*), 691
c_repr() (*angr.sim_type.SimTypePointer method*), 695
c_repr() (*angr.sim_type.SimTypeRef method*), 708
c_repr() (*angr.sim_type.SimTypeReference method*), 695
c_repr() (*angr.sim_type.SimTypeString method*), 697
c_repr() (*angr.sim_type.SimTypeWString method*), 698
c_repr() (*angr.sim_type.SimUnion method*), 703
c_repr() (*angr.sim_type.TypeRef method*), 688
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CAILB method*), 1133
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CAssig method*), 1136
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CBina method*), 1141
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CBrea method*), 1135
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CCons method*), 1142
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CCons method*), 1131
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CCont method*), 1135
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CDirty method*), 1144
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CDirty method*), 1137
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CDoW method*), 1134

c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CExpressionStatement method*), 1136
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFor method*), 1139
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CGoto method*), 1134
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIfBreak method*), 1132
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIfElse method*), 1137
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIfThenElse method*), 1137
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIntrinsic method*), 1135
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CLabel method*), 1134
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression method*), 1135
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CReturn method*), 1139
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CTE method*), 1143
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp method*), 1138
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CVariable method*), 1143
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CWhileLoop method*), 1142
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CStatements method*), 1137
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CStructField method*), 1133
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CTypeCast method*), 1138
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CUnsupportedStatement method*), 1135
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CVariableDeclaration method*), 1141
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CWhileLoop method*), 1140
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 1137
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall method*), 1139
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CVariableField method*), 1140
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp method*), 1144
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunctionCallExpression method*), 1133
c_return_as_atoms() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler static method*), 1229
c_return_as_atoms() (*angr.analyses.reaching_definitions.FunctionHandler static method*), 1188
cache (*angr.knowledge_plugins.FunctionManager property*), 729
cache (*angr.knowledge_plugins.functions.function_manager.FunctionManager property*), 799
cache (*angr.knowledge_plugins.functions.function_manager.SpillingCache property*), 794
cache (*angr.knowledge_plugins.functions.FunctionManager property*), 790
cache_size() (*angr.analyses.Pathfinder method*), 910
cached_count (*angr.knowledge_plugins.functions.function_manager.SpillingCache property*), 794
cached_function_count (*angr.knowledge_plugins.FunctionManager property*), 729
cached_function_count (*angr.knowledge_plugins.functions.function_manager.FunctionManager property*), 799
cached_function_count (*angr.knowledge_plugins.functions.FunctionManager property*), 790
cached_keys (*angr.knowledge_plugins.functions.function_manager.SpillingCache property*), 794
CallBlockMultiStatementExpression (*angr.analyses.decompiler.structured_codegen.c*), 1133
calc_size() (*angr.calling_conventions.AllocHelper class method*), 658
calculate_iterated_dominace_frontier_set() (*in module angr.analyses.dominance_frontier*), 1807
calculate_stack_depth() (*angr.analyses.decompiler.Clinic method*), 1045
calculate_stack_depth() (*angr.analyses.decompiler.clinic.Clinic method*), 1061
CALL (*angr.state_plugins.inspect.EventType attribute*), 372
Call (*class in angr.ailment.expression*), 1393
call() (*angr.procedures.SimProcedure method*), 646
call() (*angr.SimProcedure method*), 205
call() (*angr.state_plugins.CallStack method*), 250
call() (*angr.state_plugins.callstack.CallStack method*), 374
call_analysis (*angr.analyses.vfg.FunctionAnalysis attribute*), 1185
call_c() (*angr.callable.Callable method*), 711
call_graph() (*in module angr.analyses.decompiler.utils*), 1155
call_site_addr (*angr.analyses.cfg.cfg_fast.FunctionReturn attribute*), 984
call_stack (*angr.analyses.cfg.cfg_job_base.CFGJobBase property*), 993

call_stack (*angr.analyses.vfg.PendingJob attribute*), 1286
call_stack_copy() (*angr.analyses.cfg.cfg_job_base.CFGJobBase method*), 993
call_state() (*angr.factory.AngrObjectFactory method*), 229
call_string (*angr.code_location.ExternalCodeLocation attribute*), 866
callable (*angr.knowledge_plugins.Function property*), 723
callable (*angr.knowledge_plugins.functions.Function property*), 783
callable (*angr.knowledge_plugins.functions.function.Function property*), 808
Callable (*class in angr.callable*), 710
callable() (*angr.factory.AngrObjectFactory method*), 230
CallAnalysis (*class in angr.analyses.vfg*), 1287
callee (*angr.ailment.expression.DirtyExpression attribute*), 1388
callee (*angr.ailment.expression.VEXCCallExpression attribute*), 1388
callee (*angr.analyses.decompiler.structured_codegen.c.CVEXCCallExpression attribute*), 1143
CALLEE_CLEANUP (*angr.calling_conventions.SimCC attribute*), 664
CALLEE_CLEANUP (*angr.calling_conventions.SimCCMicrosoftThiscall attribute*), 668
CALLEE_CLEANUP (*angr.calling_conventions.SimCCStdcall attribute*), 669
CALLEE_CLEANUP (*angr.SimCC attribute*), 174
callee_func (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 1136
callee_func_addr (*angr.analyses.cfg.cfg_fast.FunctionReturn attribute*), 984
callee_func_addr (*angr.analyses.reaching_definitions.call_trace.CallSite attribute*), 1212
callee_target (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 1136
CalleeCleanupFinder (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 928
CalleeCleanupFinder (*class in angr.analyses*), 894
CalleeCleanupFinder (*class in angr.analyses.callee_cleanup_finder*), 1307
caller_func_addr (*angr.analyses.cfg.cfg_fast.FunctionReturn attribute*), 984
caller_func_addr (*angr.analyses.reaching_definitions.call_trace.CallSite attribute*), 1212
CALLER_SAVED_REGS (*angr.calling_conventions.SimCC attribute*), 664
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 672
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCARM attribute*), 672
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCARMHF attribute*), 673
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCCdecl attribute*), 668
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 669
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCMicrosoftThiscall attribute*), 668
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCN64 attribute*), 676
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCO32 attribute*), 675
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCRISCV64 attribute*), 674
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 671
CALLER_SAVED_REGS (*angr.SimCC attribute*), 174
caller_saved_regs_as_atoms() (*angr.analyses.reaching_definitions.function_handler.FunctionHa static method*), 1229
caller_saved_regs_as_atoms() (*angr.analyses.reaching_definitions.FunctionHandler static method*), 1188
caller_will_handle_single_ret (*angr.analyses.reaching_definitions.function_handler.FunctionCa attribute*), 1224
caller_will_handle_single_ret (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1184
CallFunctionGoal (*class in angr.exploration_techniques*), 553
CallFunctionGoal (*class in angr.exploration_techniques.director*), 580
callgraph (*angr.knowledge_base.KnowledgeBase property*), 712
callgraph (*angr.KnowledgeBase property*), 164
CallGraphVisitor (*class in angr.analyses.forward_analysis*), 931
CallGraphVisitor (*class in angr.analyses.forward_analysis.visitors*), 937
CallGraphVisitor (*class in angr.analyses.forward_analysis.visitors.call_graph*), 941
calling_convention (*angr.ailment.expression.Call attribute*), 1393
calling_convention (*angr.knowledge_plugins.Function property*), 717
calling_convention (*angr.knowledge_plugins.functions.Function property*), 778
calling_convention (*angr.knowledge_plugins.functions.function.Functio property*), 802
CallingConvention (*angr.analyses.analysis.KnownAnalysesPlugin*

attribute), 929
CallingConventionAnalysis (class in angr.analyses), 895
CallingConventionAnalysis (class in angr.analyses.calling_convention), 950
CallingConventionAnalysisMode (class in angr.analyses.complete_calling_conventions), 953
CallingConventionSerializer (class in angr.knowledge_plugins.functions.function_parser), 809
calloc() (angr.SimHeapPTMalloc method), 192
calloc() (angr.state_plugins.heap.heap_libc.SimHeapLibc method), 415
calloc() (angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method), 419
calloc() (angr.state_plugins.heap.SimHeapLibc method), 408
calloc() (angr.state_plugins.heap.SimHeapPTMalloc method), 410
calloc() (angr.state_plugins.SimHeapLibc method), 273
calloc() (angr.state_plugins.SimHeapPTMalloc method), 274
callout_sites (angr.knowledge_plugins.Function property), 720
callout_sites (angr.knowledge_plugins.functions.Function property), 780
callout_sites (angr.knowledge_plugins.functions.function_manager.FunctionManager property), 805
CallProxiNode (class in angr.analyses.proximity_graph), 1310
calls_in_graph() (in module angr.analyses.decompiler.utils), 1155
callsite (angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute), 1226
callsite (angr.analyses.reaching_definitions.FunctionCallData attribute), 1185
CallSite (class in angr.analyses.reaching_definitions.call_trace), 1212
callsite_codeloc (angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute), 1223
callsite_codeloc (angr.analyses.reaching_definitions.FunctionCallData attribute), 1183
CALLSITE_DECOMPILER (angr.knowledge_plugins.functions.function.Prototype attribute), 799
CALLSITE_LOW (angr.knowledge_plugins.functions.function.Prototype attribute), 799
callsite_repr() (angr.analyses.cfg.cfg_job_base.FunctionKey method), 993
callsite_repr() (angr.knowledge_plugins.cfg.BlockID method), 740
CallSiteMaker (class in angr.analyses.decompiler), 1042
CallSiteMaker (class in angr.analyses.decompiler.callsite_maker), 1055
CallsitePrototypeKind (class in angr.knowledge_plugins.callsite_prototypes), 739
CallsitePrototypes (class in angr.knowledge_plugins), 713
CallsitePrototypes (class in angr.knowledge_plugins.callsite_prototypes), 739
callsites (angr.analyses.reaching_definitions.call_trace.CallTrace attribute), 1213
callsites_to() (angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method), 1217
callsites_to() (angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method), 1201
callsites_to() (angr.analyses.ReachingDefinitionsAnalysis method), 916
callstack (angr.sim_state.SimState attribute), 239
callstack (angr.SimState attribute), 208
CallStack (class in angr.state_plugins), 248
CallStack (class in angr.state_plugins.callstack), 371
callstack_key (angr.knowledge_plugins.cfg.cfg_node.CFGENode property), 769
callstack_key (angr.knowledge_plugins.cfg.cfg_node.CFGNode property), 768
callstack_key (angr.knowledge_plugins.cfg.CFGENode property), 741
callstack_key (angr.knowledge_plugins.cfg.CFGNode property), 751
callstack_repr() (angr.analyses.vfg.VFGJob method), 1285
CallStackAndCallRelationships (class in angr.state_plugins.callstack), 374
CallStatementRewriter (class in angr.analyses.decompiler.optimization_passes), 1072
CallTrace (angr.analyses.reaching_definitions.subject.SubjectType attribute), 1236
CallTrace (class in angr.analyses.reaching_definitions.call_trace), 1212
CallTracingFilter (class in angr.analyses.veritesting), 1284
can_call_same_name() (angr.analyses.Identifier method), 907
can_call_same_name() (angr.analyses.identifier.identify.Identifier method), 1283
candidate_names (angr.sim_variable.SimVariable attribute), 681
capstone (angr.Block property), 161
capstone (angr.block.Block property), 233

CapstoneBlock (*class in angr.block*), 232
CapstoneInsn (*class in angr.block*), 232
cardinality (*angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor.AbstractAddressDescriptor property*), 536
CArrayTypeLength (*class in angr.analyses.decompiler.structured_codegen.c*), 1144
CAS (*class in angr.ailment.statement*), 1403
CAS() (*angr.ailment.converter_vex.VEXStmtConverter static method*), 1375
CascadingConditionTransformer (*class in angr.analyses.decompiler.region_simplifiers.cascading_condition_transformer*), 1109
CascadingIfsRemover (*class in angr.analyses.decompiler.region_simplifiers.cascading_ifs*), 1109
Case (*class in angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier*), 1099
Case2Result (*class in angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 996
case_idx (*angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter attribute*), 1111
cases (*angr.analyses.decompiler.structured_codegen.c.CIncompleteSwitchCase attribute*), 1135
cases (*angr.analyses.decompiler.structured_codegen.c.CSwitchCase attribute*), 1135
cases_issubset() (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier static method*), 1100
cases_issubset() (*angr.analyses.decompiler.optimization_passes.switch_simplifier.SwitchSimplifier static method*), 1083
CAssignment (*class in angr.analyses.decompiler.structured_codegen.c*), 1135
cast_primitive() (*angr.simos.javavm.SimJavaVM static method*), 1340
cast_primitive() (*angr.simos.SimJavaVM static method*), 1326
cast_to_mem() (*angr.analyses.data_dep.dep_nodes.MemDepNode class method*), 1315
cast_to_mem() (*angr.analyses.data_dep.MemDepNode class method*), 1318
CastType (*angr.state_plugins.SimSolver attribute*), 292
CastType (*angr.state_plugins.solver.SimSolver attribute*), 366
CatchDesync (*angr.exploration_techniques.tracer.TracingMixin attribute*), 575
category (*angr.ailment.expression.VirtualVariable attribute*), 1379
category (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable attribute*), 840
category (*angr.sim_variable.SimVariable attribute*), 680
category (*angr.SimFile property*), 179
category (*angr.storage.file.SimFile property*), 434
category (*angr.storage.memory_mixins.RegionCategoryMixin property*), 472
category (*angr.storage.memory_mixins.regioned_memory.region_categor property*), 535
category (*angr.storage.memory_mixins.regioned_memory.RegionCategor property*), 526
category (*angr.storage.SimFile property*), 423
CBinaryOp (*class in angr.analyses.decompiler.structured_codegen.c*), 1140
CBreak (*class in angr.analyses.decompiler.structured_codegen.c*), 1135
cc (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 1224
cc (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
cc (*angr.analyses.reaching_definitions.subject.Subject property*), 1238
cc (*angr.calling_conventions.ArgSession attribute*), 663
cc (*angr.calling_conventions.MicrosoftAMD64ArgSession attribute*), 669
cc (*angr.calling_conventions.SimCC.ArgSession attribute*), 664
cc (*angr.calling_conventions.UsercallArgSession attribute*), 663
cc (*angr.SimCC.ArgSession attribute*), 175
cc() (*angr.factory.AngrObjectFactory method*), 231
cc_graph (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier attribute*), 1060
CCA_DECOMPILER (*angr.knowledge_plugins.functions.function.PrototypeSo attribute*), 800
CCA_LOW (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 799
CCall() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1375
ccall_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1173
ccall_handler() (*angr.engines.light.SimEngineLightVEX static method*), 1170
CCallRewriterBase (*class in angr.analyses.decompiler.ccall_rewriters.rewriter_base*), 1055
CClosingObject (*class in angr.analyses.decompiler.structured_codegen.c*), 1144
CConstant (*class in angr.analyses.decompiler.structured_codegen.c*), 1141
CConstantType (*class in angr.analyses.decompiler.structured_codegen.base*), 1127
CConstruct (*class in angr.analyses.decompiler.structured_codegen.c*), 1131
CContinue (*class in angr.analyses.decompiler.structured_codegen.c*), 1135

CDG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 928
CDG (*class in angr.analyses*), 873
CDG (*class in angr.analyses.cdg*), 1017
CDirtyExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 1144
CDirtyStatement (*class in angr.analyses.decompiler.structured_codegen.c*), 1137
CDoWhileLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 1133
ceil_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 759
ceil_addr() (*angr.knowledge_plugins.cfg.CFGModel method*), 745
ceiling_addr() (*angr.analyses.cfg.cfb.CFBlanket method*), 973
ceiling_addr() (*angr.analyses.cfg.CFBlanket method*), 958
ceiling_addr() (*angr.knowledge_plugins.FunctionManager method*), 726
ceiling_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 792
ceiling_addr() (*angr.knowledge_plugins.functions.function_dictionary.FunctionDict method*), 790
ceiling_addr() (*angr.knowledge_plugins.functions.function.Function method*), 796
ceiling_addr() (*angr.knowledge_plugins.functions.FunctionManager method*), 787
ceiling_func() (*angr.knowledge_plugins.FunctionManager method*), 727
ceiling_func() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 796
ceiling_func() (*angr.knowledge_plugins.functions.FunctionManager method*), 787
ceiling_item() (*angr.analyses.cfg.cfb.CFBlanket method*), 973
ceiling_item() (*angr.analyses.cfg.CFBlanket method*), 958
ceiling_items() (*angr.analyses.cfg.cfb.CFBlanket method*), 973
ceiling_items() (*angr.analyses.cfg.CFBlanket method*), 958
CExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 1132
CExpressionStatement (*class in angr.analyses.decompiler.structured_codegen.c*), 1136
cexterns (*angr.analyses.decompiler.structured_codegen.c.CFunctionDecl attribute*), 1146
CFakeVariable (*class in angr.analyses.decompiler.structured_codegen.c*), 1138
CFB (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
CFBlanket (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
CFBlanket (*class in angr.analyses.cfg*), 958
CFBlanket (*class in angr.analyses.cfg.cfb*), 973
CFBlanketView (*class in angr.analyses.cfg.cfb*), 972
CFG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
CFG (*class in angr.analyses*), 873
CFG (*class in angr.analyses.cfg*), 956
CFG (*class in angr.analyses.cfg.cfg*), 973
cfg_cache (*angr.analyses.Veritesting attribute*), 926
cfg_cache (*angr.analyses.veritesting.CallTracingFilter attribute*), 1284
cfg_cache (*angr.analyses.veritesting.Veritesting attribute*), 1284
CFG_HANDLE_JOB (*angr.state_plugins.inspect.EventType attribute*), 332
cfg_jumpkind_from_pb() (*in module angr.utils.enums_conv*), 1346
cfg_jumpkind_to_pb() (*in module angr.utils.enums_conv*), 1346
CFGArchOptions (*class in angr.analyses*), 884
CFGArchOptions (*class in angr.analyses.cfg*), 959
CFGArchOptionsFunctionManager (*class in angr.analyses.cfg.cfg_arch_options*), 992
CFGBase (*class in angr.analyses.cfg*), 959
CFGBase (*class in angr.analyses.cfg.cfg_base*), 981
CFGEmulated (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
CFGEmulated (*class in angr.analyses*), 885
CFGEmulated (*class in angr.analyses.cfg*), 962
CFGEmulated (*class in angr.analyses.cfg.cfg_emulated*), 976
CFGENode (*class in angr.knowledge_plugins.cfg*), 740
CFGENode (*class in angr.knowledge_plugins.cfg.cfg_node*), 768
CFGFast (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
CFGFast (*class in angr.analyses*), 889
CFGFast (*class in angr.analyses.cfg*), 966
CFGFast (*class in angr.analyses.cfg.cfg_fast*), 988
CFGFastSoot (*class in angr.analyses*), 892
CFGFastSoot (*class in angr.analyses.cfg*), 970
CFGFastSoot (*class in angr.analyses.cfg.cfg_fast_soot*), 1015
CFGJob (*class in angr.analyses.cfg.cfg_emulated*), 976
CFGJob (*class in angr.analyses.cfg.cfg_fast*), 987
CFGJobBase (*class in angr.analyses.cfg.cfg_job_base*), 993
CFGJobType (*class in angr.analyses.cfg.cfg_fast*), 987

CFGManager (*class in angr.knowledge_plugins*), 712
CFGManager (*class in angr.knowledge_plugins.cfg*), 742
CFGManager (*class in angr.knowledge_plugins.cfg.cfg_manager*), 766
CFGModel (*class in angr.knowledge_plugins.cfg*), 742
CFGModel (*class in angr.knowledge_plugins.cfg.cfg_model*), 756
CFGModelSerializer (*class in angr.angrdb.serializers.cfg_model*), 1026
CFGNode (*class in angr.knowledge_plugins.cfg*), 750
CFGNode (*class in angr.knowledge_plugins.cfg.cfg_node*), 766
CFGNodeCreationFailure (*class in angr.knowledge_plugins.cfg.cfg_node*), 766
cfgs (*angr.angrdb.models.DbKnowledgeBase attribute*), 1022
cfgs (*angr.knowledge_base.KnowledgeBase attribute*), 711
cfgs (*angr.KnowledgeBase attribute*), 164
CFGSliceToSink (*class in angr.analyses.cfg_slice_to_sink*), 1238
CFGSliceToSink (*class in angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink*), 1241
CForLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 1134
cfunc (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator attribute*), 1146
CFunction (*class in angr.analyses.decompiler.structured_codegen.c*), 1131
CFunctionCall (*class in angr.analyses.decompiler.structured_codegen.c*), 1136
CGoto (*class in angr.analyses.decompiler.structured_codegen.c*), 1137
ChainMapCOW (*class in angr.utils.cowdict*), 1344
ChallRespInfo (*class in angr.state_plugins.trace_additions*), 382
changed_bytes() (*angr.storage.memory_mixins.HistoryTrackingMixin method*), 463
changed_bytes() (*angr.storage.memory_mixins.ListPage method*), 468
changed_bytes() (*angr.storage.memory_mixins.MVListPage method*), 470
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.mv_mixin.PagedMemoryMixin method*), 506
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.HistoryTrackingMixin method*), 520
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPageMixin method*), 511
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.list_page.ListPageMixin method*), 522
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPageMixin method*), 512
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.mv method*), 501
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.MV method*), 514
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.ult method*), 524
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.Ult method*), 518
changed_bytes() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 476
changed_bytes() (*angr.storage.memory_mixins.slotted_memory.SlottedM method*), 541
changed_bytes() (*angr.storage.memory_mixins.SlottedMemoryMixin method*), 485
changed_bytes() (*angr.storage.memory_mixins.UltraPage method*), 489
changed_pages() (*angr.storage.memory_mixins.paged_memory.paged_m method*), 506
changed_pages() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 476
chdir() (*angr.state_plugins.filesystem.SimFilesystem method*), 357
chdir() (*angr.state_plugins.SimFilesystem method*), 269
check() (*angr.BP method*), 155
check() (*angr.exploration_techniques.CallFunctionGoal method*), 553
check() (*angr.exploration_techniques.director.BaseGoal method*), 579
check() (*angr.exploration_techniques.director.CallFunctionGoal method*), 580
check() (*angr.exploration_techniques.director.ExecuteAddressGoal method*), 580
check() (*angr.exploration_techniques.ExecuteAddressGoal method*), 555
check() (*angr.state_plugins.inspect.BP method*), 339
check_offset() (*angr.calling_conventions.SimRegArg method*), 660
check_state() (*angr.exploration_techniques.CallFunctionGoal method*), 553
check_state() (*angr.exploration_techniques.director.BaseGoal method*), 579
check_state() (*angr.exploration_techniques.director.CallFunctionGoal method*), 580
check_state() (*angr.exploration_techniques.director.ExecuteAddressGoal method*), 580
check_state() (*angr.exploration_techniques.ExecuteAddressGoal method*), 555
check_tests() (*angr.analyses.Identifier method*), 907
check_tests() (*angr.analyses.identifier.identify.Identifier method*), 1283
check_value() (*angr.calling_conventions.SimFunctionArgument method*), 659
check_value_list() (*angr.calling_conventions.SimFunctionArgument method*), 659

children (*angr.analyses.disassembly.Operand attribute*), 1295
Chunk (*class in angr.state_plugins.heap.heap_freelist*), 414
chunk_from_mem() (*angr.SimHeapPTMalloc method*), 192
chunk_from_mem() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 415
chunk_from_mem() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 418
chunk_from_mem() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 409
chunk_from_mem() (*angr.state_plugins.SimHeapPTMalloc method*), 274
chunks() (*angr.SimHeapPTMalloc method*), 191
chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 415
chunks() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 418
chunks() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 409
chunks() (*angr.state_plugins.SimHeapPTMalloc method*), 274
CIfBreak (*class in angr.analyses.decompiler.structured_codegen.c*), 1134
CIfElse (*class in angr.analyses.decompiler.structured_codegen.c*), 1134
CIncompleteSwitchCase (*class in angr.analyses.decompiler.structured_codegen.c*), 1135
CIndexedVariable (*class in angr.analyses.decompiler.structured_codegen.c*), 1139
CITE (*class in angr.analyses.decompiler.structured_codegen.c*), 1142
CLabel (*class in angr.analyses.decompiler.structured_codegen.c*), 1137
claripy_ast_from_ail_condition() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 1063
claripy_ast_to_sympy_expr() (*angr.analyses.decompiler.condition_processor.ConditionProcessor static method*), 1063
ClaripyDataEngineMixin (*class in angr.analyses.propagator.top_checker_mixin*), 1178
ClaripyDataMixin (*class in angr.engines.vex*), 599
ClaripyDataVEXEngineMixin (*class in angr.analyses.propagator.top_checker_mixin*), 1178
ClassIdentifier (*class in angr.analyses*), 896
ClassIdentifier (*class in angr.analyses.class_identifier*), 1292
clean() (*angr.utils.cowdict.ChainMapCOW method*), 1345
clean() (*angr.utils.cowdict.DefaultChainMapCOW method*), 1346
cleanup() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 985
cleanup() (*angr.analyses.decompiler.CStructuredCodeGenerator method*), 1042
cleanup() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 1146
cleanup() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeGenerator method*), 1122
cleanup() (*angr.analyses.decompiler.structured_codegen.RustStructuredCodeGenerator method*), 1127
cleanup() (*angr.knowledge_plugins.RuntimeDb method*), 734
clear() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 1062
clear() (*angr.knowledge_plugins.FunctionManager method*), 725
clear() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 795
clear() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionManager method*), 793
clear() (*angr.knowledge_plugins.functions.FunctionManager method*), 785
clear() (*angr.knowledge_plugins.XRefManager method*), 736
clear() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 864
clear() (*angr.knowledge_plugins.xrefs.XRefManager method*), 862
clear() (*angr.state_plugins.log.SimStateLog method*), 371
clear() (*angr.state_plugins.scratch.SimStateScratch method*), 390
clear() (*angr.state_plugins.SimStateLog method*), 312
clear() (*angr.state_plugins.SimStateScratch method*), 320
clear() (*angr.storage.memory_mixins.regioned_memory.abstract_address.AddressWrapper method*), 536
clear_cache() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 614
clear_cache() (*angr.engines.vex.VEXLifter method*), 600
clear_hash() (*angr.ailment.Block method*), 1368
clear_hash() (*angr.ailment.block.Block method*), 1407
clear_hash() (*angr.sim_variable.SimVariable method*), 681
clear_local_references() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 403
clear_local_references() (*angr.state_plugins.SimStateJNIReferences method*), 306

clear_page_cache() (*angr.state_plugins.Unicorn method*), 328
clear_page_cache() (*angr.state_plugins.unicorn_engine.Unicorn method*), 398
clear_region_for_reflow() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 763
clear_region_for_reflow() (*angr.knowledge_plugins.cfg.CFGModel method*), 749
clear_updated_functions() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 985
ClemoryBackerMixin (*class in angr.storage.memory_mixins*), 458
ClemoryBackerMixin (*class in angr.storage.memory_mixins.paged_memory.page_mixins*), 508
Clinic (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
clinic (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 1065
Clinic (*class in angr.analyses.decompiler*), 1043
Clinic (*class in angr.analyses.decompiler.clinic*), 1058
ClinicMode (*class in angr.analyses.decompiler.clinic*), 1056
ClinicStage (*class in angr.analyses.decompiler.clinic*), 1057
CLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 1133
close() (*angr.state_plugins.posix.SimSystemPosix method*), 353
close() (*angr.state_plugins.SimSystemPosix method*), 323
close() (*angr.vaults.Vault static method*), 872
close() (*angr.vaults.VaultShelf method*), 872
closed_fds (*angr.state_plugins.posix.SimSystemPosix property*), 352
closed_fds (*angr.state_plugins.SimSystemPosix property*), 322
closest_common_ancestor() (*angr.state_plugins.history.SimStateHistory method*), 378
closest_common_ancestor() (*angr.state_plugins.SimStateHistory method*), 303
Closure (*class in angr.utils.mp*), 1354
cls_ (*in module angr.calling_conventions*), 680
CmpOp (*class in angr.analyses.decompiler.region_simplifiers.cmpflag*), 1115
CMultiStatementExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 1143
cnode2ailexpr (*angr.analyses.decompiler.structured_codegen.base.CStructuredCodeGenerator attribute*), 1146
code_constants (*angr.knowledge_plugins.Function property*), 719
code_constants (*angr.knowledge_plugins.functions.Function property*), 779
code_constants (*angr.knowledge_plugins.functions.function.Function property*), 804
code_nodes (*angr.knowledge_plugins.Function property*), 718
code_nodes (*angr.knowledge_plugins.functions.Function property*), 778
code_nodes (*angr.knowledge_plugins.functions.function.Function property*), 802
CodeCaveAnalysis (*class in angr.analyses*), 896
codecaves (*angr.analyses.CodeCaveAnalysis attribute*), 896
codegen (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 1065
codegen (*angr.analyses.decompiler.structured_codegen.c.CConstruct attribute*), 1131
codegen (*angr.analyses.decompiler.structured_codegen.c.CStatement attribute*), 1132
codeloc (*angr.analyses.reaching_definitions.Definition attribute*), 1182
codeloc (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsS attribute*), 1231
codeloc (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 1205
codeloc (*angr.knowledge_plugins.key_definitions.Definition attribute*), 823
codeloc (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 843
codeloc_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefin attribute*), 1231
codeloc_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsSta attribute*), 1205
CodeLocation (*class in angr.code_location*), 865
CodeMotionOptimization (*class in angr.analyses.decompiler.optimization_passes*), 1072
codenode (*angr.Block property*), 161
codenode (*angr.block.Block property*), 234
codenode (*angr.block.SootBlock property*), 234
CodeNode (*class in angr.codenode*), 1323
CodeReference (*angr.knowledge_plugins.cfg.memory_data.MemoryDataS attribute*), 764
CodeReference (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 755
CodeTagging (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
CodeTagging (*class in angr.analyses*), 896
CodeTagging (*class in angr.analyses.code_tagging*), 1018
CodeTags (*class in angr.analyses.code_tagging*), 1018

collapse() (*angr.analyses.decompiler.structured_codegen.c.CBreakStatement class method*), 1148
collapsed (*angr.analyses.decompiler.structured_codegen.c.CFor attribute*), 1133
collapsed (*angr.analyses.decompiler.structured_codegen.c.CIfElse attribute*), 1140
collapsed (*angr.analyses.decompiler.structured_codegen.c.CWhileLoop attribute*), 1140
collect_conditions_between_nodes() (*angr.analyses.decompiler.optimization_passes.DuplicateReturnEdge method*), 1078
COLLECT_DATA_REFS (*angr.analyses.decompiler.clinic.ClinicMode attribute*), 1056
collect_data_refs (*angr.engines.pcode.lifter.Lifter attribute*), 610
collect_data_refs (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
COLLECT_EXTERNS (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1058
color() (*angr.analyses.disassembly.DisassemblyPiece static method*), 1293
COMBO_REGISTER (*angr.ailment.expression.VirtualVariableCategory attribute*), 1379
ComboRegister (*class in angr.ailment.expression*), 1378
ComboRegReferenceWalker (*class in angr.analyses.decompiler.clinic*), 1058
comment (*angr.angrdb.models.DbComment attribute*), 1025
Comment (*class in angr.analyses.disassembly*), 1296
comments (*angr.angrdb.models.DbKnowledgeBase attribute*), 1022
Comments (*class in angr.knowledge_plugins*), 714
Comments (*class in angr.knowledge_plugins.comments*), 775
CommentsSerializer (*class in angr.angrdb.serializers.comments*), 1027
common_type (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp attribute*), 1141
compare() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1192
compare() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1234
compare() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 1207
compare() (*angr.knowledge_plugins.key_definitions.environment.Environment method*), 844
compare() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 850
compare() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 827
compare() (*angr.storage.memory_mixins.MVListPage method*), 470
compare() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 506
compare() (*angr.storage.memory_mixins.paged_memory.pages.MVListPage method*), 501
compare() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 514
compare() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 476
compare_path_group() (*angr.analyses.congruency_check.CongruencyCheck method*), 1305
compare_path_group() (*angr.analyses.CongruencyCheck method*), 899
compare_paths() (*angr.analyses.congruency_check.CongruencyCheck method*), 1305
compare_paths() (*angr.analyses.CongruencyCheck method*), 899
compare_statement_dict() (*in module angr.analyses.bindiff*), 948
compare_states() (*angr.analyses.congruency_check.CongruencyCheck method*), 1305
compare_states() (*angr.analyses.CongruencyCheck method*), 899
COMPARISON_NEGATION (*angr.ailment.BinaryOp attribute*), 1366
COMPARISON_NEGATION (*angr.ailment.expression.BinaryOp attribute*), 1384
compiler (*angr.analyses.LanguageDetector property*), 908
compiler_version (*angr.analyses.LanguageDetector property*), 908
complete() (*angr.exploration_techniques.ExplorationTechnique method*), 557
complete() (*angr.exploration_techniques.Explorer method*), 558
complete() (*angr.exploration_techniques.explorer.Explorer method*), 570
complete() (*angr.exploration_techniques.Tracer method*), 567
complete() (*angr.exploration_techniques.tracer.Tracer method*), 576
complete() (*angr.ExplorationTechnique method*), 164
complete() (*angr.sim_manager.SimulationManager method*), 547
complete() (*angr.SimulationManager method*), 216
COMPLETE_SCANNING (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 987
CompleteCallingConventions (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
CompleteCallingConventionsAnalysis (*class in angr.analyses*), 897
CompleteCallingConventionsAnalysis (*class in angr.analyses.complete_calling_conventions*),

953
completion() (*angr.LLMClient method*), 166
completion_json() (*angr.LLMClient method*), 166
completion_structured() (*angr.LLMClient method*), 166
compute() (*angr.state_plugins.trace_additions.FormatInfo method*), 382
compute() (*angr.state_plugins.trace_additions.FormatInfoConcreteBacker method*), 382
compute() (*angr.state_plugins.trace_additions.FormatInfoConcreteBacker method*), 382
compute() (*angr.state_plugins.trace_additions.FormatInfoStrToInt method*), 382
compute_common_type() (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp static method*), 1141
compute_dominance_frontier() (*in module angr.utils.graph*), 1348
compute_quotient_graph() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 1263
concat() (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.MultiValueBacker method*), 503
concrete (*angr.state_plugins.debug_variables.SimDebugVariable property*), 422
concrete (*angr.state_plugins.SimDebugVariable property*), 267
concrete (*angr.state_plugins.SimMemView property*), 285
concrete (*angr.state_plugins.view.SimMemView property*), 431
concrete_bytes() (*angr.storage.memory_object.SimMemoryObject method*), 451
concrete_bytes() (*angr.storage.SimMemoryObject method*), 426
concrete_load() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 497
concrete_load() (*angr.storage.memory_mixins.AddressConcretizationMixin method*), 458
concrete_load() (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.JavaVmMemoryMixin method*), 506
concrete_load() (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.MultiValueBacker method*), 524
concrete_load() (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.UltraPage method*), 518
concrete_load() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 476
concrete_load() (*angr.storage.memory_mixins.UltraPage method*), 488
concrete_path_bool() (*in module angr.state_plugins.solver*), 361
concrete_path_list() (*in module angr.state_plugins.solver*), 361
concrete_path_not_bool() (*in module angr.state_plugins.solver*), 361
concrete_path_scalar() (*in module angr.state_plugins.solver*), 361
concrete_path_tuple() (*in module angr.state_plugins.solver*), 361
concrete_states (*angr.analyses.variable_recovery.variable_recovery.VariableRecoveryFast property*), 1254
ConcreteBackerMixin (*class in angr.storage.memory_mixins*), 459
ConcreteBackerMixin (*class in angr.storage.memory_mixins.paged_memory.page_backer_mixins*), 509
concretize() (*angr.analyses.typehoon.translator.TypeTranslator method*), 1266
concretize() (*angr.concretization_strategies.SimConcretizationStrategy method*), 452
concretize() (*angr.SimFile method*), 179
concretize() (*angr.SimFileBase method*), 181
concretize() (*angr.SimFileDescriptor method*), 183
concretize() (*angr.SimFileDescriptorDuplex method*), 186
concretize() (*angr.SimPackets method*), 198
concretize() (*angr.storage.file.SimFile method*), 434
concretize() (*angr.storage.file.SimFileBase method*), 433
concretize() (*angr.storage.file.SimFileDescriptor method*), 445
concretize() (*angr.storage.file.SimFileDescriptorBase method*), 443
concretize() (*angr.storage.file.SimFileDescriptorDuplex method*), 447
concretize() (*angr.storage.file.SimPackets method*), 438
concretize() (*angr.storage.file.SimPacketsSlots method*), 449
concretize() (*angr.storage.SimFile method*), 424
concretize() (*in module angr.state_plugins.heap.utils*), 420
concretize_load_idx() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 465
concretize_read_addr() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 497
concretize_read_addr() (*angr.storage.memory_mixins.AddressConcretizationMixin method*), 458
concretize_store_idx() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 464
concretize_write_addr() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 496
concretize_write_addr()

(*angr.storage.memory_mixins.AddressConcretizationMixin method*), 457
cond (*angr.ailment.expression.ITE attribute*), 1387
cond (*angr.analyses.decompiler.structured_codegen.c.CITE attribute*), 1143
condition (*angr.ailment.statement.ConditionalJump attribute*), 1401
condition (*angr.analyses.decompiler.structured_codegen.c.CDoWhileLoop attribute*), 1134
condition (*angr.analyses.decompiler.structured_codegen.c.CFailure attribute*), 1134
condition (*angr.analyses.decompiler.structured_codegen.c.CIfBreak attribute*), 1134
condition (*angr.analyses.decompiler.structured_codegen.c.CWhileLoop attribute*), 1133
condition_and_nodes (*angr.analyses.decompiler.structured_codegen.c.CIfElse attribute*), 1134
condition_to_lambda() (*in module angr.exploration_techniques.common*), 585
CONDITIONAL (*angr.state_plugins.sim_action.SimActionExit attribute*), 639
CONDITIONAL (*angr.state_plugins.SimActionExit attribute*), 264
ConditionalBreakLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1111
ConditionalJump (*class in angr.ailment.statement*), 1400
ConditionalMixin (*class in angr.storage.memory_mixins*), 459
ConditionalMixin (*class in angr.storage.memory_mixins.conditional_store_mixin*), 497
ConditionalRegion (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier*), 1115
ConditionConstantPropagation (*class in angr.analyses.decompiler.optimization_passes*), 1073
ConditionLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1110
ConditionProcessor (*class in angr.analyses.decompiler.condition_processor*), 1062
confidence (*angr.analyses.LanguageDetector property*), 908
configuration (*angr.angrdb.models.DbStructuredCode attribute*), 1024
configure_project() (*angr.SimOS method*), 195
configure_project() (*angr.simos.linux.SimLinux method*), 1335
configure_project() (*angr.simos.SimLinux method*), 1327
configure_project() (*angr.simos.SimOS method*), 1328
configure_project() (*angr.simos.simos.SimOS method*), 1333
configure_project() (*angr.simos.SimSnimmucNxp method*), 1330
configure_project() (*angr.simos.SimUserland method*), 1330
configure_project() (*angr.simos.SimWindows method*), 1331
configure_project() (*angr.simos.userland.SimUserland method*), 1337
configure_project() (*angr.simos.windows.SimWindows method*), 1338
confirmed (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge attribute*), 986
CongruencyCheck (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
CongruencyCheck (*class in angr.analyses*), 899
CongruencyCheck (*class in angr.analyses.congruency_check*), 1305
Const (*class in angr.ailment*), 1368
Const (*class in angr.ailment.expression*), 1376
Const() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
const_formats (*angr.analyses.decompiler.structured_codegen.base.BaseS attribute*), 1129
const_formats (*angr.angrdb.models.DbStructuredCode attribute*), 1024
const_n() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
CONST_TYPES (*angr.engines.light.ArithmeticExpression attribute*), 1167
CONST_TYPES (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
const_val (*angr.knowledge_plugins.key_definitions.definition.DefinitionM attribute*), 842
const_vals (*angr.engines.pcode.lifter.IRSB attribute*), 608
Constant (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 1313
Constant (*angr.analyses.data_dep.DepNodeTypes attribute*), 1318
CONSTANT (*angr.analyses.reaching_definitions.AtomKind attribute*), 1181
CONSTANT (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 836
Constant (*class in angr.analyses.stack_pointer_tracker*), 1243
constant_after() (*angr.analyses.stack_pointer_tracker.StackPointerTrac method*), 1246

constant_after() (*angr.analyses.StackPointerTracker method*), 923
constant_after_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1246
constant_after_block() (*angr.analyses.StackPointerTracker method*), 923
constant_before() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1246
constant_before() (*angr.analyses.StackPointerTracker method*), 923
constant_before_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1246
constant_before_block() (*angr.analyses.StackPointerTracker method*), 923
constant_jump_targets (*angr.engines.pcode.lifter.IRSB property*), 609
constant_jump_targets_and_jumpkinds (*angr.engines.pcode.lifter.IRSB property*), 609
CONSTANT_PROPAGATION (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1057
ConstantChange (*class in angr.analyses.bindiff*), 947
ConstantDepNode (*class in angr.analyses.data_dep*), 1316
ConstantDepNode (*class in angr.analyses.data_dep.dep_nodes*), 1314
ConstantDereferencesSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1075
ConstantDereferencesSimplifier (*class in angr.analyses.decompiler.optimization_passes.constant_derefs*), 1091
ConstantOperand (*class in angr.analyses.disassembly*), 1295
ConstantPropagation (*class in angr.analyses.binary_optimizer*), 1306
ConstantResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1008
ConstantResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.const_resolver*), 1003
constants (*angr.engines.pcode.lifter.IRSB property*), 609
ConstantSrc (*class in angr.analyses.reaching_definitions*), 1181
ConstantSrc (*class in angr.knowledge_plugins.key_definitions.atoms*), 838
ConstFormatsSerializer (*class in angr.angrdb.serializers.structured_code*), 1031
ConstPropOptReverter (*class in angr.analyses.decompiler.optimization_passes*), 1074
constrain_all_zero() (*angr.analyses.Identifier static method*), 907
constrain_all_zero() (*angr.analyses.identifier.identify.Identifier static method*), 1283
constrain_callee_prototypes() (*angr.analyses.decompiler.Clinic method*), 1046
constrain_callee_prototypes() (*angr.analyses.decompiler.clinic.Clinic method*), 1061
constraint_hook() (*in module angr.state_plugins.trace_additions*), 382
ConstraintGraphNode (*class in angr.analyses.typehoon.simple_solver*), 1261
ConstraintGraphTag (*class in angr.analyses.typehoon.simple_solver*), 1261
CONSTRAINTS (*angr.state_plugins.inspect.EventType attribute*), 331
constraints (*angr.state_plugins.SimSolver property*), 291
constraints (*angr.state_plugins.solver.SimSolver property*), 364
constraints_since() (*angr.state_plugins.history.SimStateHistory method*), 378
constraints_since() (*angr.state_plugins.SimStateHistory method*), 303
construct() (*angr.knowledge_plugins.key_definitions.definition.Definition static method*), 842
ContainerNode (*class in angr.utils.graph*), 1348
contains() (*angr.knowledge_plugins.debug_variables.DebugVariable method*), 820
contains_addr() (*angr.knowledge_plugins.FunctionManager method*), 726
contains_addr() (*angr.knowledge_plugins.functions.function_manager.F method*), 796
contains_addr() (*angr.knowledge_plugins.functions.FunctionManager method*), 787
contains_atom() (*angr.analyses.reaching_definitions.dep_graph.DepGra method*), 1218
content (*angr.analyses.reaching_definitions.subject.Subject property*), 1238
content (*angr.analyses.reassembler.Data property*), 1301
content (*angr.angrdb.models.DbObject attribute*), 1022

content (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 765
content (*angr.knowledge_plugins.cfg.MemoryData attribute*), 754
content_gen() (*angr.storage.memory_mixins.MVListPage method*), 471
content_gen() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 501
content_gen() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 515
context (*angr.code_location.CodeLocation attribute*), 866
context (*angr.engines.pcode.lifter.PcodeBasicBlockLifter attribute*), 612
context_sensitivity_level (*angr.analyses.cfg.cfg_base.CFGBase property*), 982
context_sensitivity_level (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 980
context_sensitivity_level (*angr.analyses.cfg.CFGBase property*), 961
context_sensitivity_level (*angr.analyses.cfg.CFGEmulated property*), 966
context_sensitivity_level (*angr.analyses.CFGEmulated property*), 888
ContinueScanningNotification, 983
ConvenientMappingsMixin (*class in angr.storage.memory_mixins*), 459
ConvenientMappingsMixin (*class in angr.storage.memory_mixins.convenient_mappings_mixin*), 499
Convert (*class in angr.ailment.expression*), 1382
convert() (*angr.ailment.converter_common.Converter static method*), 1373
convert() (*angr.ailment.converter_pcode.PCodeIRSBConverter static method*), 1374
convert() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
convert() (*angr.ailment.converter_vex.VEXIRSBConverter static method*), 1375
convert() (*angr.ailment.converter_vex.VEXStmtConverter static method*), 1375
convert() (*angr.ailment.IRSBConverter static method*), 1370
convert() (*angr.ailment.PCodeIRSBConverter static method*), 1370
convert() (*angr.ailment.VEXIRSBConverter static method*), 1373
convert_claripy_bool_ast() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 1063
convert_claripy_bool_ast_core() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 1063
convert_cppproto_to_py() (*in module angr.utils.library*), 1352
convert_cproto_to_py() (*in module angr.utils.library*), 1352
convert_list() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
convert_variable_list() (*angr.knowledge_plugins.VariableManager static method*), 736
convert_variable_list() (*angr.knowledge_plugins.variables.variable_manager.VariableManager static method*), 820
convert_variable_list() (*angr.knowledge_plugins.variables.VariableManager static method*), 811
Converter (*class in angr.ailment.converter_common*), 1373
ConvertTo (*class in angr.analyses.typehoon.typevars*), 1272
ConvertType (*class in angr.ailment.expression*), 1382
CooperationBase (*class in angr.storage.memory_mixins*), 460
CooperationBase (*class in angr.storage.memory_mixins.paged_memory.pages*), 510
CooperationBase (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 520
copied_var_ids (*angr.analyses.decompiler.clinic.Clinic attribute*), 1061
copy() (*angr.ailment.Assignment method*), 1366
copy() (*angr.ailment.BinaryOp method*), 1367
copy() (*angr.ailment.Block method*), 1368
copy() (*angr.ailment.block.Block method*), 1406
copy() (*angr.ailment.Const method*), 1369
copy() (*angr.ailment.expression.Array method*), 1395
copy() (*angr.ailment.expression.BasePointerOffset method*), 1390
copy() (*angr.ailment.expression.BinaryOp method*), 1385
copy() (*angr.ailment.expression.Call method*), 1394
copy() (*angr.ailment.expression.ComboRegister method*), 1378
copy() (*angr.ailment.expression.Const method*), 1377
copy() (*angr.ailment.expression.Convert method*), 1383
copy() (*angr.ailment.expression.DirtyExpression method*), 1388
copy() (*angr.ailment.expression.Extract method*), 1392
copy() (*angr.ailment.expression.FunctionLikeMacro method*), 1396
copy() (*angr.ailment.expression.Insert method*), 1392

copy() (*angr.ailment.expression.ITE method*), 1387
copy() (*angr.ailment.expression.Let method*), 1396
copy() (*angr.ailment.expression.Load method*), 1386
copy() (*angr.ailment.expression.MultiStatementExpression method*), 1389
copy() (*angr.ailment.expression.Phi method*), 1380
copy() (*angr.ailment.expression.Register method*), 1378
copy() (*angr.ailment.expression.Reinterpret method*), 1384
copy() (*angr.ailment.expression.RustEnum method*), 1395
copy() (*angr.ailment.expression.StackBaseOffset method*), 1391
copy() (*angr.ailment.expression.StringLiteral method*), 1394
copy() (*angr.ailment.expression.Struct method*), 1395
copy() (*angr.ailment.expression.Tmp method*), 1377
copy() (*angr.ailment.expression.UnaryOp method*), 1381
copy() (*angr.ailment.expression.VEXCCallExpression method*), 1389
copy() (*angr.ailment.expression.VirtualVariable method*), 1380
copy() (*angr.ailment.Register method*), 1371
copy() (*angr.ailment.statement.Assignment method*), 1398
copy() (*angr.ailment.statement.CAS method*), 1404
copy() (*angr.ailment.statement.ConditionalJump method*), 1401
copy() (*angr.ailment.statement.DirtyStatement method*), 1405
copy() (*angr.ailment.statement.Jump method*), 1400
copy() (*angr.ailment.statement.Label method*), 1406
copy() (*angr.ailment.statement.Return method*), 1403
copy() (*angr.ailment.statement.SideEffectStatement method*), 1402
copy() (*angr.ailment.statement.Store method*), 1400
copy() (*angr.ailment.statement.WeakAssignment method*), 1399
copy() (*angr.ailment.tagged_object.TaggedObject method*), 1408
copy() (*angr.ailment.Tmp method*), 1372
copy() (*angr.ailment.UnaryOp method*), 1373
copy() (*angr.analyses.cfg.cfg_base.CFGBase method*), 982
copy() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 978
copy() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 992
copy() (*angr.analyses.cfg.CFGBase method*), 961
copy() (*angr.analyses.cfg.CFGEmulated method*), 964
copy() (*angr.analyses.cfg.CFGFast method*), 970
copy() (*angr.analyses.CFGEmulated method*), 886
copy() (*angr.analyses.CFGFast method*), 892
copy() (*angr.analyses.ddg.LiveDefinitions method*), 1157
copy() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 1071
copy() (*angr.analyses.decompiler.optimization_passes.engine_base.Simpli method*), 1101
copy() (*angr.analyses.decompiler.region_simplifiers.expr_folding.Statemen method*), 1110
copy() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 1213
copy() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1190
copy() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsSt method*), 1233
copy() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 1202
copy() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1207
copy() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 1245
copy() (*angr.analyses.typehoon.typevars.TypeVariables method*), 1271
copy() (*angr.analyses.variable_recovery.variable_recovery.VariableRecov method*), 1255
copy() (*angr.analyses.variable_recovery.variable_recovery_fast.VariableR method*), 1252
copy() (*angr.analyses.VFG method*), 880
copy() (*angr.analyses.vfg.VFG method*), 1290
copy() (*angr.concretization_strategies.norepeats.SimConcretizationStrateg method*), 541
copy() (*angr.concretization_strategies.norepeats_range.SimConcretization method*), 543
copy() (*angr.concretization_strategies.SimConcretizationStrategy method*), 452
copy() (*angr.concretization_strategies.SimConcretizationStrategyNorepeat method*), 454
copy() (*angr.concretization_strategies.SimConcretizationStrategyNorepeat method*), 454
copy() (*angr.engines.pcode.lifter.IRSB method*), 608
copy() (*angr.keyed_region.KeyedRegion method*), 868
copy() (*angr.keyed_region.RegionObject method*), 868
copy() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 740
copy() (*angr.knowledge_plugins.CallsitePrototypes method*), 714
copy() (*angr.knowledge_plugins.cfg.cfg_manager.CFGManager method*), 766
copy() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 758
copy() (*angr.knowledge_plugins.cfg.cfg_node.CFGENode method*), 769
copy() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 768
copy() (*angr.knowledge_plugins.cfg.CFGENode*

method), 741
copy() (*angr.knowledge_plugins.cfg.CFGManager method*), 742
copy() (*angr.knowledge_plugins.cfg.CFGModel method*), 744
copy() (*angr.knowledge_plugins.cfg.CFGNode method*), 751
copy() (*angr.knowledge_plugins.cfg.memory_data.MemoryData method*), 765
copy() (*angr.knowledge_plugins.cfg.MemoryData method*), 755
copy() (*angr.knowledge_plugins.CFGManager method*), 712
copy() (*angr.knowledge_plugins.Comments method*), 714
copy() (*angr.knowledge_plugins.comments.Comments method*), 775
copy() (*angr.knowledge_plugins.CustomStrings method*), 714
copy() (*angr.knowledge_plugins.Data method*), 715
copy() (*angr.knowledge_plugins.data.Data method*), 775
copy() (*angr.knowledge_plugins.Function method*), 724
copy() (*angr.knowledge_plugins.FunctionManager method*), 725
copy() (*angr.knowledge_plugins.functions.Function method*), 785
copy() (*angr.knowledge_plugins.functions.function.Function method*), 809
copy() (*angr.knowledge_plugins.functions.function.FunctionInfo method*), 800
copy() (*angr.knowledge_plugins.functions.function_manager.FunctionDict method*), 792
copy() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 795
copy() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict method*), 793
copy() (*angr.knowledge_plugins.functions.FunctionManager method*), 785
copy() (*angr.knowledge_plugins.indirect_jumps.IndirectJumps method*), 775
copy() (*angr.knowledge_plugins.IndirectJumps method*), 730
copy() (*angr.knowledge_plugins.key_definitions.key_definition_manager.KeyDefinitionManager method*), 846
copy() (*angr.knowledge_plugins.key_definitions.KeyDefinitionManager method*), 824
copy() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 848
copy() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 825
copy() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 856
copy() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 833
copy() (*angr.knowledge_plugins.key_definitions.Uses method*), 836
copy() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 860
copy() (*angr.knowledge_plugins.KeyDefinitionManager method*), 730
copy() (*angr.knowledge_plugins.KnowledgeBasePlugin method*), 731
copy() (*angr.knowledge_plugins.Labels method*), 731
copy() (*angr.knowledge_plugins.labels.Labels method*), 776
copy() (*angr.knowledge_plugins.Obfuscations method*), 731
copy() (*angr.knowledge_plugins.patches.PatchManager method*), 738
copy() (*angr.knowledge_plugins.PatchManager method*), 732
copy() (*angr.knowledge_plugins.plugin.KnowledgeBasePlugin method*), 738
copy() (*angr.knowledge_plugins.PropagationManager method*), 733
copy() (*angr.knowledge_plugins.propagations.PropagationManager method*), 773
copy() (*angr.knowledge_plugins.structured_code.StructuredCodeManager method*), 822
copy() (*angr.knowledge_plugins.StructuredCodeManager method*), 735
copy() (*angr.knowledge_plugins.types.TypesStore method*), 772
copy() (*angr.knowledge_plugins.TypesStore method*), 735
copy() (*angr.knowledge_plugins.VariableManager method*), 736
copy() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 819
copy() (*angr.knowledge_plugins.variables.VariableManager method*), 811
copy() (*angr.knowledge_plugins.XRefManager method*), 736
copy() (*angr.knowledge_plugins.xrefs.XRef method*), 862
copy() (*angr.knowledge_plugins.xrefs.xref.XRef method*), 864
copy() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 864
copy() (*angr.knowledge_plugins.xrefs.XRefManager method*), 862
copy() (*angr.misc.plugins.PluginPreset method*), 237
copy() (*angr.procedures.definitions.SimLibrary method*), 650
copy() (*angr.procedures.definitions.SimSyscallLibrary method*), 655
copy() (*angr.sim_manager.SimulationManager method*),

546
copy() (*angr.sim_state.SimState method*), 241
copy() (*angr.sim_state_options.SimStateOptions method*), 246
copy() (*angr.sim_type.SimCppClass method*), 707
copy() (*angr.sim_type.SimCppClassValue method*), 707
copy() (*angr.sim_type.SimStruct method*), 702
copy() (*angr.sim_type.SimStructValue method*), 702
copy() (*angr.sim_type.SimType method*), 687
copy() (*angr.sim_type.SimTypeArray method*), 696
copy() (*angr.sim_type.SimTypeBitfield method*), 705
copy() (*angr.sim_type.SimTypeBool method*), 693
copy() (*angr.sim_type.SimTypeBottom method*), 689
copy() (*angr.sim_type.SimTypeChar method*), 692
copy() (*angr.sim_type.SimTypeCppFunction method*), 699
copy() (*angr.sim_type.SimTypeDouble method*), 701
copy() (*angr.sim_type.SimTypeEnum method*), 704
copy() (*angr.sim_type.SimTypeFd method*), 694
copy() (*angr.sim_type.SimTypeFloat method*), 700
copy() (*angr.sim_type.SimTypeFunction method*), 699
copy() (*angr.sim_type.SimTypeInt method*), 691
copy() (*angr.sim_type.SimTypeLength method*), 700
copy() (*angr.sim_type.SimTypeNum method*), 690
copy() (*angr.sim_type.SimTypeNumOffset method*), 707
copy() (*angr.sim_type.SimTypePointer method*), 695
copy() (*angr.sim_type.SimTypeReference method*), 695
copy() (*angr.sim_type.SimTypeReg method*), 690
copy() (*angr.sim_type.SimTypeString method*), 697
copy() (*angr.sim_type.SimTypeTop method*), 689
copy() (*angr.sim_type.SimTypeWideChar method*), 693
copy() (*angr.sim_type.SimTypeWString method*), 698
copy() (*angr.sim_type.SimUnion method*), 703
copy() (*angr.sim_type.SimUnionValue method*), 703
copy() (*angr.sim_type.TypeRef method*), 688
copy() (*angr.sim_variable.SimComboRegisterVariable method*), 684
copy() (*angr.sim_variable.SimConstantVariable method*), 681
copy() (*angr.sim_variable.SimMemoryVariable method*), 685
copy() (*angr.sim_variable.SimRegisterVariable method*), 683
copy() (*angr.sim_variable.SimStackVariable method*), 686
copy() (*angr.sim_variable.SimTemporaryVariable method*), 682
copy() (*angr.sim_variable.SimVariable method*), 681
copy() (*angr.SimFile method*), 180
copy() (*angr.SimFileBase method*), 182
copy() (*angr.SimFileDescriptor method*), 184
copy() (*angr.SimFileDescriptorDuplex method*), 186
copy() (*angr.SimFileStream method*), 188
copy() (*angr.SimHeapBrk method*), 189
copy() (*angr.SimHeapPTMalloc method*), 191
copy() (*angr.SimHostFilesystem method*), 194
copy() (*angr.SimPackets method*), 198
copy() (*angr.SimPacketsStream method*), 200
copy() (*angr.SimState method*), 210
copy() (*angr.SimStatePlugin method*), 211
copy() (*angr.SimulationManager method*), 215
copy() (*angr.state_plugins.CallStack method*), 248
copy() (*angr.state_plugins.callstack.CallStack method*), 372
copy() (*angr.state_plugins.cgc.SimStateCGC method*), 380
copy() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 359
copy() (*angr.state_plugins.filesystem.SimFilesystem method*), 356
copy() (*angr.state_plugins.filesystem.SimHostFilesystem method*), 360
copy() (*angr.state_plugins.GDB method*), 247
copy() (*angr.state_plugins.gdb.GDB method*), 379
copy() (*angr.state_plugins.globals.SimStateGlobals method*), 387
copy() (*angr.state_plugins.heap.heap_base.SimHeapBase method*), 412
copy() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 412
copy() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 418
copy() (*angr.state_plugins.heap.SimHeapBase method*), 406
copy() (*angr.state_plugins.heap.SimHeapBrk method*), 407
copy() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 409
copy() (*angr.state_plugins.history.SimStateHistory method*), 376
copy() (*angr.state_plugins.inspect.SimInspector method*), 341
copy() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 401
copy() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 403
copy() (*angr.state_plugins.libc.SimStateLibc method*), 345
copy() (*angr.state_plugins.light_registers.SimLightRegisters method*), 375
copy() (*angr.state_plugins.log.SimStateLog method*), 370
copy() (*angr.state_plugins.loop_data.SimStateLoopData method*), 400
copy() (*angr.state_plugins.plugin.SimStatePlugin method*), 329
copy() (*angr.state_plugins.posix.PosixDevFS method*), 348

copy() (*angr.state_plugins.posix.PosixProcFS method*), 350
copy() (*angr.state_plugins.posix.SimSystemPosix method*), 352
copy() (*angr.state_plugins.PosixDevFS method*), 260
copy() (*angr.state_plugins.PosixProcFS method*), 262
copy() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 391
copy() (*angr.state_plugins.scratch.SimStateScratch method*), 389
copy() (*angr.state_plugins.sim_action.SimAction method*), 638
copy() (*angr.state_plugins.sim_action_object.SimActionObject method*), 641
copy() (*angr.state_plugins.SimAction method*), 263
copy() (*angr.state_plugins.SimActionObject method*), 265
copy() (*angr.state_plugins.SimFilesystem method*), 268
copy() (*angr.state_plugins.SimHeapBase method*), 270
copy() (*angr.state_plugins.SimHeapBrk method*), 271
copy() (*angr.state_plugins.SimHeapPTMalloc method*), 273
copy() (*angr.state_plugins.SimHostFilesystem method*), 276
copy() (*angr.state_plugins.SimInspector method*), 277
copy() (*angr.state_plugins.SimJavaVmClassloader method*), 280
copy() (*angr.state_plugins.SimLightRegisters method*), 281
copy() (*angr.state_plugins.SimMemView method*), 284
copy() (*angr.state_plugins.SimRegNameView method*), 286
copy() (*angr.state_plugins.SimSolver method*), 289
copy() (*angr.state_plugins.SimStateCGC method*), 297
copy() (*angr.state_plugins.SimStateEdgeHitmap method*), 299
copy() (*angr.state_plugins.SimStateGlobals method*), 300
copy() (*angr.state_plugins.SimStateHistory method*), 302
copy() (*angr.state_plugins.SimStateIcicle method*), 304
copy() (*angr.state_plugins.SimStateJNIReferences method*), 306
copy() (*angr.state_plugins.SimStateLibc method*), 309
copy() (*angr.state_plugins.SimStateLog method*), 311
copy() (*angr.state_plugins.SimStateLoopData method*), 313
copy() (*angr.state_plugins.SimStatePlugin method*), 314
copy() (*angr.state_plugins.SimStatePreconstrainer method*), 317
copy() (*angr.state_plugins.SimStateScratch method*), 319
copy() (*angr.state_plugins.SimSymbolizer method*), 320
copy() (*angr.state_plugins.SimSystemPosix method*), 322
copy() (*angr.state_plugins.SimUCManager method*), 325
copy() (*angr.state_plugins.solver.SimSolver method*), 363
copy() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 421
copy() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 383
copy() (*angr.state_plugins.trace_additions.FormatInfo method*), 382
copy() (*angr.state_plugins.trace_additions.FormatInfoDontConstrain method*), 382
copy() (*angr.state_plugins.trace_additions.FormatInfoIntToStr method*), 382
copy() (*angr.state_plugins.trace_additions.FormatInfoStrToInt method*), 382
copy() (*angr.state_plugins.trace_additions.ZenPlugin method*), 384
copy() (*angr.state_plugins.uc_manager.SimUCManager method*), 387
copy() (*angr.state_plugins.Unicorn method*), 327
copy() (*angr.state_plugins.unicorn_engine.Unicorn method*), 397
copy() (*angr.state_plugins.view.SimMemView method*), 430
copy() (*angr.state_plugins.view.SimRegNameView method*), 426
copy() (*angr.storage.file.SimFile method*), 435
copy() (*angr.storage.file.SimFileBase method*), 433
copy() (*angr.storage.file.SimFileDescriptor method*), 445
copy() (*angr.storage.file.SimFileDescriptorDuplex method*), 447
copy() (*angr.storage.file.SimFileStream method*), 437
copy() (*angr.storage.file.SimPackets method*), 439
copy() (*angr.storage.file.SimPacketsSlots method*), 449
copy() (*angr.storage.file.SimPacketsStream method*), 441
copy() (*angr.storage.memory_mixins.address_concretization_mixin.Addre method*), 495
copy() (*angr.storage.memory_mixins.AddressConcretizationMixin method*), 456
copy() (*angr.storage.memory_mixins.ClemoryBackerMixin method*), 458
copy() (*angr.storage.memory_mixins.convenient_mappings_mixin.Conven method*), 499
copy() (*angr.storage.memory_mixins.ConvenientMappingsMixin method*), 459
copy() (*angr.storage.memory_mixins.default_filler_mixin.ExplicitFillerMi method*), 491
copy() (*angr.storage.memory_mixins.default_filler_mixin.SpecialFillerMix method*), 491
copy() (*angr.storage.memory_mixins.DictBackerMixin

method), 460
copy() (*angr.storage.memory_mixins.ExplicitFillerMixin method*), 461
copy() (*angr.storage.memory_mixins.HistoryTrackingMixin method*), 462
copy() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 465
copy() (*angr.storage.memory_mixins.KeyValueMemoryMixin method*), 466
copy() (*angr.storage.memory_mixins.label_merger_mixin.LabelMergerMixin method*), 498
copy() (*angr.storage.memory_mixins.LabelMergerMixin method*), 467
copy() (*angr.storage.memory_mixins.ListPage method*), 467
copy() (*angr.storage.memory_mixins.MemoryRegionMetaMixin method*), 471
copy() (*angr.storage.memory_mixins.multi_value_merger_mixin.MultiValueMergerMixin method*), 504
copy() (*angr.storage.memory_mixins.MultiValueMergerMixin method*), 473
copy() (*angr.storage.memory_mixins.MVListPage method*), 469
copy() (*angr.storage.memory_mixins.MVListPagesMixin method*), 471
copy() (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.DictPageBackerMixin method*), 508
copy() (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.DirtyBackerMixin method*), 509
copy() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.MVListPagesMixin method*), 507
copy() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 504
copy() (*angr.storage.memory_mixins.paged_memory.pages.cache_mixin.CachingMixin method*), 519
copy() (*angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.HistoryTrackingMixin method*), 510
copy() (*angr.storage.memory_mixins.paged_memory.pages.list_page.ListPage method*), 521
copy() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 511
copy() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 499
copy() (*angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.PermissionsMixin method*), 519
copy() (*angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.PermissionsMixin method*), 515
copy() (*angr.storage.memory_mixins.paged_memory.pages.refcount_mixin.RefcountMixin method*), 518
copy() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 516
copy() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 523
copy() (*angr.storage.memory_mixins.paged_memory.pages.UltraPage method*), 516
copy() (*angr.storage.memory_mixins.paged_memory.stack_allocation_mix method*), 509
copy() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 474
copy() (*angr.storage.memory_mixins.PermissionsMixin method*), 477
copy() (*angr.storage.memory_mixins.RefcountMixin method*), 478
copy() (*angr.storage.memory_mixins.regioned_memory.MemoryRegionMe method*), 524
copy() (*angr.storage.memory_mixins.regioned_memory.region_data.Regio method*), 534
copy() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixi method*), 536
copy() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixi method*), 537
copy() (*angr.storage.memory_mixins.regioned_memory.regioned_address_ method*), 538
copy() (*angr.storage.memory_mixins.regioned_memory.regioned_memory method*), 531
copy() (*angr.storage.memory_mixins.regioned_memory.RegionedAddressC method*), 526
copy() (*angr.storage.memory_mixins.regioned_memory.RegionedMemory method*), 528
copy() (*angr.storage.memory_mixins.RegionedAddressConcretizationMixi method*), 479
copy() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 480
copy() (*angr.storage.memory_mixins.size_resolution_mixin.SizeConcretiza method*), 495
copy() (*angr.storage.memory_mixins.SizeConcretizationMixin method*), 483
copy() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixi method*), 540
copy() (*angr.storage.memory_mixins.SlottedMemoryMixin method*), 484
copy() (*angr.storage.memory_mixins.SpecialFillerMixin method*), 486
copy() (*angr.storage.memory_mixins.StackAllocationMixin method*), 486
copy() (*angr.storage.memory_mixins.top_merger_mixin.TopMergerMixin method*), 503
copy() (*angr.storage.memory_mixins.TopMergerMixin method*), 487
copy() (*angr.storage.memory_mixins.UltraPage method*), 487
copy() (*angr.storage.memory_mixins.underconstrained_mixin.Underconstr method*), 492
copy() (*angr.storage.memory_mixins.UnderconstrainedMixin method*), 489
copy() (*angr.storage.SimFile method*), 424

copy() (*angr.utils.cowdict.ChainMapCOW method*), 1345
copy_contents() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 506
copy_contents() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 476
copy_contents() (*angr.storage.memory_mixins.unwrapper.UnwrapperMixin method*), 498
copy_contents() (*angr.storage.memory_mixins.UnwrapperMixin method*), 490
copy_graph() (*angr.analyses.decompiler.Clinic method*), 1045
copy_graph() (*angr.analyses.decompiler.clinic.Clinic method*), 1061
copy_graph() (*in module angr.analyses.decompiler.utils*), 1154
copy_without_tail() (*angr.state_plugins.CallStack method*), 248
copy_without_tail() (*angr.state_plugins.callstack.CallStack method*), 372
CouldNotResolveException, 1245
count (*angr.analyses.typehoon.typeconsts.Array attribute*), 1278
count (*angr.state_plugins.unicorn_engine.TRANSMIT_RECORD attribute*), 392
count() (*angr.exploration_techniques.spiller.PickledStates method*), 573
count() (*angr.state_plugins.history.TreeIter method*), 378
count() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 502
cprotos2py() (*in module angr.utils.library*), 1352
crash_windup() (*angr.exploration_techniques.Tracer class method*), 567
crash_windup() (*angr.exploration_techniques.tracer.Tracer class method*), 577
crc (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
create_extract_vtable() (*angr.analyses.vtable.VtableFinder method*), 1292
create_extract_vtable() (*angr.analyses.VtableFinder method*), 927
create_jump_target_var() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 1063
create_merged_subgraph() (*angr.analyses.decompiler.optimization_passes.DuplicateReversing method*), 1077
create_new_reference() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 402
create_new_reference() (*angr.state_plugins.SimStateJNIReferences method*), 305
creation_failed (*angr.knowledge_plugins.cfg.cfg_node.CFGENode property*), 769
creation_failed (*angr.knowledge_plugins.cfg.CFGENode property*), 741
creation_failure_info (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 769
creation_failure_info (*angr.knowledge_plugins.cfg.CFGENode attribute*), 741
CRegister (*class in angr.analyses.decompiler.structured_codegen.c*), 1142
CReturn (*class in angr.analyses.decompiler.structured_codegen.c*), 1137
CrossJumpReverter (*class in angr.analyses.decompiler.optimization_passes*), 1075
cs_operand (*angr.analyses.disassembly.Operand property*), 1295
CStatement (*class in angr.analyses.decompiler.structured_codegen.c*), 1132
CStatements (*class in angr.analyses.decompiler.structured_codegen.c*), 1133
CStructField (*class in angr.analyses.decompiler.structured_codegen.c*), 1138
CStructFieldNameDef (*class in angr.analyses.decompiler.structured_codegen.c*), 1144
CStructuredCodeGenerator (*class in angr.analyses.decompiler*), 1040
CStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen*), 1120
CStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.c*), 1144
CStructuredCodeWalker (*class in angr.analyses.decompiler.structured_codegen*), 1123
CStructuredCodeWalker (*class in angr.analyses.decompiler.structured_codegen.c*), 1147
cstyle_ifs (*angr.analyses.decompiler.structured_codegen.c.CIfBreak attribute*), 1134
cstyle_ifs (*angr.analyses.decompiler.structured_codegen.c.CIfElse attribute*), 1134
CSwitchCase (*class in angr.analyses.decompiler.structured_codegen.c*), 1135
ctype (*angr.analyses.flirt.FlirtSignatureParsed at-*

tribute), 1164
CTypeCast (class in angr.analyses.decompiler.structured_codegen.c), 1141
ctypes_crc (angr.analyses.flirt.FlirtSignatureParsed attribute), 1164
CUnaryOp (class in angr.analyses.decompiler.structured_codegen.c), 1140
CUnsupportedStatement (class in angr.analyses.decompiler.structured_codegen.c), 1137
current_function_address (angr.state_plugins.CallStack property), 249
current_function_address (angr.state_plugins.callstack.CallStack property), 373
current_function_address() (angr.analyses.reaching_definitions.call_trace.CallTrace method), 1213
current_return_target (angr.state_plugins.CallStack property), 249
current_return_target (angr.state_plugins.callstack.CallStack property), 373
current_stack_pointer (angr.analyses.cfg.cfg_job_base.CFGJobBase property), 994
current_stack_pointer (angr.state_plugins.CallStack property), 249
current_stack_pointer (angr.state_plugins.callstack.CallStack property), 373
custom_string (angr.ailment.tagged_object.TagDict attribute), 1407
CustomStrings (class in angr.knowledge_plugins), 714
CVariable (class in angr.analyses.decompiler.structured_codegen.c), 1139
CVariableField (class in angr.analyses.decompiler.structured_codegen.c), 1140
CVEXCCallExpression (class in angr.analyses.decompiler.structured_codegen.c), 1143
CWhileLoop (class in angr.analyses.decompiler.structured_codegen.c), 1133
cyclic (angr.analyses.decompiler.graph_region.GraphRegion attribute), 1071
cyclic_ancestor (angr.analyses.decompiler.graph_region.GraphRegion attribute), 1071
cyclomatic_complexity (angr.knowledge_plugins.Function property), 718
cyclomatic_complexity (angr.knowledge_plugins.functions.Function property), 778
cyclomatic_complexity (angr.knowledge_plugins.functions.function.Function property), 802

D

data (angr.ailment.expression.StringLiteral attribute), 1394
data (angr.ailment.statement.Store attribute), 1399
data (angr.analyses.variable_recovery.engine_base.RichR attribute), 1257
data (angr.engines.pcode.lifter.Lifter attribute), 610
data (angr.engines.pcode.lifter.PcodeLifter attribute), 613
data (angr.state_plugins.unicorn_engine.TRANSMIT_RECORD attribute), 392
Data (class in angr.analyses.reassembler), 1301
Data (class in angr.knowledge_plugins), 714
Data (class in angr.knowledge_plugins.data), 775
data_addr (angr.analyses.decompiler.clinic.DataRefDesc attribute), 1057
data_graph (angr.analyses.DDG property), 876
data_graph (angr.analyses.ddg.DDG property), 1159
data_hi (angr.ailment.statement.CAS attribute), 1404
data_lo (angr.ailment.statement.CAS attribute), 1404
data_ptr() (angr.PTChunk method), 167
data_ptr() (angr.state_plugins.heap.heap_freelist.Chunk method), 414
data_ptr() (angr.state_plugins.heap.heap_ptmalloc.PTChunk method), 417
data_ptr() (angr.state_plugins.heap.PTChunk method), 405
data_ptr() (angr.state_plugins.PTChunk method), 258
data_refs (angr.analyses.decompiler.clinic.Clinic attribute), 1060
data_refs (angr.engines.pcode.lifter.IRSB attribute), 608
data_size (angr.analyses.decompiler.clinic.DataRefDesc attribute), 1057
data_sub_graph() (angr.analyses.DDG method), 877
data_sub_graph() (angr.analyses.ddg.DDG method), 1160
data_type_str (angr.analyses.decompiler.clinic.DataRefDesc attribute), 1057
DataDependencyGraphAnalysis (class in angr.analyses), 899
DataDependencyGraphAnalysis (class in angr.analyses.data_dep), 1317
DataDependencyGraphAnalysis (class in angr.analyses.data_dep.data_dependency_analysis), 1312
DataGraphError, 1017
DataGraphMeta (class in angr.analyses.datagraph_meta), 1017

DataLabel (*class in angr.analyses.reassembler*), 1297
DataNormalizationMixin (*class in angr.storage.memory_mixins*), 460
DataNormalizationMixin (*class in angr.storage.memory_mixins.bvv_conversion_mixin*), 491
DATAREF_HINTS (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 987
DataRefDesc (*class in angr.analyses.decompiler.clinic*), 1056
db_compatible() (*angr.angrdb.AngrDB method*), 1019
db_compatible() (*angr.angrdb.db.AngrDB method*), 1021
DbCFGModel (*class in angr.angrdb.models*), 1022
DbComment (*class in angr.angrdb.models*), 1024
DbFunction (*class in angr.angrdb.models*), 1023
dbg_comments() (*angr.analyses.reassembler.Instruction method*), 1299
dbg_draw() (*angr.knowledge_plugins.Function method*), 723
dbg_draw() (*angr.knowledge_plugins.FunctionManager method*), 728
dbg_draw() (*angr.knowledge_plugins.functions.Function method*), 783
dbg_draw() (*angr.knowledge_plugins.functions.function.Function method*), 807
dbg_draw() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 797
dbg_draw() (*angr.knowledge_plugins.functions.FunctionManager method*), 788
dbg_get_repr() (*angr.analyses.decompiler.graph_region.GraphRegion static method*), 1071
dbg_print() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 1071
dbg_print() (*angr.knowledge_plugins.Function method*), 723
dbg_print() (*angr.knowledge_plugins.functions.Function method*), 783
dbg_print() (*angr.knowledge_plugins.functions.function.Function method*), 807
dbg_print() (*angr.storage.memory_mixins.MemoryRegionMetaMixin method*), 473
dbg_print() (*angr.storage.memory_mixins.regioned_memory.MemoryRegion method*), 526
dbg_print() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.MemoryRegionMetaMixin method*), 538
dbg_print_irsb() (*angr.annocfg.AnnotatedCFG method*), 1322
dbg_print_stack() (*angr.sim_state.SimState method*), 242
dbg_print_stack() (*angr.SimState method*), 211
dbg_repr() (*angr.ailment.Block method*), 1368
dbg_repr() (*angr.ailment.block.Block method*), 1407
dbg_repr() (*angr.analyses.backward_slice.BackwardSlice method*), 946
dbg_repr() (*angr.analyses.BackwardSlice method*), 882
dbg_repr() (*angr.analyses.cfg.cfb.CFBlanket method*), 973
dbg_repr() (*angr.analyses.cfg.CFBlanket method*), 959
dbg_repr() (*angr.analyses.DDG method*), 876
dbg_repr() (*angr.analyses.ddg.DDG method*), 1160
dbg_repr() (*angr.analyses.decompiler.Clinic method*), 1045
dbg_repr() (*angr.analyses.decompiler.clinic.Clinic method*), 1061
dbg_repr() (*angr.annocfg.AnnotatedCFG method*), 1322
dbg_repr() (*angr.Blade method*), 160
dbg_repr() (*angr.blade.Blade method*), 1320
dbg_repr() (*angr.keyed_region.KeyedRegion method*), 869
dbg_repr() (*angr.state_plugins.CallStack method*), 250
dbg_repr() (*angr.state_plugins.callstack.CallStack method*), 374
dbg_repr_run() (*angr.analyses.backward_slice.BackwardSlice method*), 947
dbg_repr_run() (*angr.analyses.BackwardSlice method*), 882
DbInformation (*class in angr.angrdb.models*), 1021
DbKnowledgeBase (*class in angr.angrdb.models*), 1022
DbLabel (*class in angr.angrdb.models*), 1025
DbObject (*class in angr.angrdb.models*), 1021
DbStructuredCode (*class in angr.angrdb.models*), 1023
DbVariableCollection (*class in angr.angrdb.models*), 1023
DbXRefs (*class in angr.angrdb.models*), 1024
DDG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
DDG (*class in angr.analyses*), 875
DDG (*class in angr.analyses.ddg*), 1159
DDGJob (*class in angr.analyses.ddg*), 1157
DDGView (*class in angr.analyses.ddg*), 1159
DDGViewInstruction (*class in angr.analyses.ddg*), 1158
DDGViewItem (*class in angr.analyses.ddg*), 1158
deactivate() (*angr.misc.plugins.PluginPreset method*), 1347
dead_ref() (*angr.state_hierarchy.StateHierarchy method*), 552
dead_ref() (*angr.StateHierarchy method*), 221
dead_vvar_ids (*angr.analyses.SPropagatorAnalysis property*), 920
DeadAssignment (*class in angr.analyses.binary_optimizer*), 1306
DeadblockRemover (*class in angr.analyses.decompiler.optimization_passes*), 1076

deadended (*angr.sim_manager.SimulationManager attribute*), 545
deadended (*angr.SimulationManager attribute*), 215
deadends (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 980
deadends (*angr.analyses.cfg.CFGEmulated property*), 966
deadends (*angr.analyses.CFGEmulated property*), 888
debug() (*angr.sim_manager.ErrorRecord method*), 551
DebugVariable (*class in angr.knowledge_plugins.debug_variables*), 820
DebugVariableContainer (*class in angr.knowledge_plugins.debug_variables*), 820
DebugVariableManager (*class in angr.knowledge_plugins*), 715
DebugVariableManager (*class in angr.knowledge_plugins.debug_variables*), 821
dec_active_workers() (*angr.distributed.Server method*), 1362
dec_active_workers() (*angr.distributed.server.Server method*), 1362
dec_active_workers() (*angr.Server method*), 173
decode_instruction() (*in module angr.analyses.disassembly_utils*), 1297
DecodingAssumption (*class in angr.analyses.cfg.cfg_fast*), 984
DecompilationCache (*class in angr.analyses.decompiler.decompilation_cache*), 1064
DecompilationOption (*class in angr.analyses.decompiler.decompilation_options*), 1063
decompilations (*angr.knowledge_base.KnowledgeBase attribute*), 711
decompilations (*angr.KnowledgeBase attribute*), 164
DECOMPILE (*angr.analyses.decompiler.clinic.ClinicMode attribute*), 1056
decompile_functions() (*in module angr.analyses.decompiler.utils*), 1154
Decompiler (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
Decompiler (*class in angr.analyses*), 900
Decompiler (*class in angr.analyses.decompiler*), 1046
Decompiler (*class in angr.analyses.decompiler.decompiler*), 1065
decorate() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler static method*), 1226
deep_copy() (*angr.ailment.Assignment method*), 1366
deep_copy() (*angr.ailment.BinaryOp method*), 1367
deep_copy() (*angr.ailment.Block method*), 1368
deep_copy() (*angr.ailment.block.Block method*), 1406
deep_copy() (*angr.ailment.Const method*), 1369
deep_copy() (*angr.ailment.expression.Array method*), 1395
deep_copy() (*angr.ailment.expression.BasePointerOffset method*), 1390
deep_copy() (*angr.ailment.expression.BinaryOp method*), 1385
deep_copy() (*angr.ailment.expression.Call method*), 1394
deep_copy() (*angr.ailment.expression.Const method*), 1377
deep_copy() (*angr.ailment.expression.Convert method*), 1383
deep_copy() (*angr.ailment.expression.DirtyExpression method*), 1388
deep_copy() (*angr.ailment.expression.Extract method*), 1392
deep_copy() (*angr.ailment.expression.FunctionLikeMacro method*), 1397
deep_copy() (*angr.ailment.expression.Insert method*), 1393
deep_copy() (*angr.ailment.expression.ITE method*), 1387
deep_copy() (*angr.ailment.expression.Let method*), 1396
deep_copy() (*angr.ailment.expression.Load method*), 1386
deep_copy() (*angr.ailment.expression.MultiStatementExpression method*), 1389
deep_copy() (*angr.ailment.expression.Phi method*), 1380
deep_copy() (*angr.ailment.expression.Register method*), 1378
deep_copy() (*angr.ailment.expression.Reinterpret method*), 1384
deep_copy() (*angr.ailment.expression.RustEnum method*), 1395
deep_copy() (*angr.ailment.expression.StackBaseOffset method*), 1391
deep_copy() (*angr.ailment.expression.StringLiteral method*), 1394
deep_copy() (*angr.ailment.expression.Struct method*), 1395
deep_copy() (*angr.ailment.expression.Tmp method*), 1377
deep_copy() (*angr.ailment.expression.UnaryOp method*), 1382
deep_copy() (*angr.ailment.expression.VEXCCallExpression method*), 1389
deep_copy() (*angr.ailment.expression.VirtualVariable method*), 1380
deep_copy() (*angr.ailment.Register method*), 1371
deep_copy() (*angr.ailment.statement.Assignment method*), 1398

deep_copy() (*angr.ailment.statement.CAS method*), 1404
deep_copy() (*angr.ailment.statement.ConditionalJump method*), 1401
deep_copy() (*angr.ailment.statement.DirtyStatement method*), 1405
deep_copy() (*angr.ailment.statement.Jump method*), 1400
deep_copy() (*angr.ailment.statement.Label method*), 1406
deep_copy() (*angr.ailment.statement.Return method*), 1403
deep_copy() (*angr.ailment.statement.SideEffectStatement method*), 1402
deep_copy() (*angr.ailment.statement.Store method*), 1400
deep_copy() (*angr.ailment.statement.WeakAssignment method*), 1399
deep_copy() (*angr.ailment.tagged_object.TaggedObject method*), 1408
deep_copy() (*angr.ailment.Tmp method*), 1372
deep_copy() (*angr.ailment.UnaryOp method*), 1373
DEF_EQ_SAME_BLOCK (*angr.analyses.decompiler.ail_simplifier.DefEqRelation attribute*), 1053
DEF_IN_EQ_PRED_BLOCK (*angr.analyses.decompiler.ail_simplifier.DefEqRelation attribute*), 1053
DEF_IS_FUNCARG (*angr.analyses.decompiler.ail_simplifier.DefEqRelation attribute*), 1053
default (*angr.analyses.decompiler.structured_codegen.c.CSwitchCase attribute*), 1135
default (*angr.sim_state_options.StateOption attribute*), 243
DEFAULT (*angr.state_plugins.sim_action.SimActionExit attribute*), 639
DEFAULT (*angr.state_plugins.SimActionExit attribute*), 264
default() (*angr.angrdb.serializers.loader.LoadArgsJSONEncoder method*), 1028
default_cc() (*in module angr*), 222
default_cc() (*in module angr.calling_conventions*), 679
default_engine (*angr.factory.AngrObjectFactory property*), 228
default_engine_factory (*angr.factory.AngrObjectFactory attribute*), 228
default_exit_target (*angr.engines.pcode.lifter.IRSB attribute*), 608
default_indirect_jump_resolvers() (*in module angr.analyses.cfg.indirect_jump_resolvers.default_resolvers*), 998
default_simtype_from_bits() (*angr.analyses.decompiler.CStructuredCodeGenerator method*), 1042
default_simtype_from_bits() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCode method*), 1147
default_simtype_from_bits() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeG method*), 1122
default_simtype_from_size() (*angr.analyses.decompiler.structured_codegen.RustStructuredCod method*), 1127
DEFAULT_STRUCTURER (*in module angr.analyses.decompiler.structuring*), 1032
DefaultChainMapCOW (*class in angr.utils.cowdict*), 1345
DefaultFillerMixin (*class in angr.storage.memory_mixins*), 460
DefaultFillerMixin (*class in angr.storage.memory_mixins.default_filler_mixin*), 490
DefaultListPagesMemory (*class in angr.storage.memory_mixins*), 460
DefaultMemory (*class in angr.storage*), 423
DefaultMemory (*class in angr.storage.memory_mixins*), 460
DefEqRelation (*class in angr.analyses.decompiler.ail_simplifier*), 1053
DefEqRelation (*class in*

cleanup() (*angr.state_hierarchy.StateHierarchy method*), 552
cleanup() (*angr.StateHierarchy method*), 221
definition (*angr.knowledge_plugins.key_definitions.live_definitions.Defin attribute*), 846
Definition (*class in angr.analyses.reaching_definitions*), 1181
Definition (*class in angr.knowledge_plugins.key_definitions*), 822
Definition (*class in angr.knowledge_plugins.key_definitions.definition*), 843
DefinitionAnnotation (*class in angr.knowledge_plugins.key_definitions.live_definitions*), 846
DefinitionMatchPredicate (*class in angr.knowledge_plugins.key_definitions.definition*), 841
definitions (*angr.analyses.ddg.DDGViewInstruction property*), 1159
defs (*angr.ailment.expression.Let attribute*), 1396
defs (*angr.knowledge_base.KnowledgeBase attribute*), 711
defs (*angr.KnowledgeBase attribute*), 164
DefUseChain (*class in angr.analyses.vsa_ddg*), 1290
delete() (*angr.SimMount method*), 195
delete() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 359

delete() (*angr.state_plugins.filesystem.SimFilesystem method*), 357
delete() (*angr.state_plugins.filesystem.SimMount method*), 358
delete() (*angr.state_plugins.posix.PosixDevFS method*), 347
delete() (*angr.state_plugins.posix.PosixProcFS method*), 349
delete() (*angr.state_plugins.PosixDevFS method*), 259
delete() (*angr.state_plugins.PosixProcFS method*), 261
delete() (*angr.state_plugins.SimFilesystem method*), 270
delete() (*angr.state_plugins.SimMount method*), 286
delete_reference() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 403
delete_reference() (*angr.state_plugins.SimStateJNIReferences method*), 306
delete_uc() (*angr.state_plugins.Unicorn static method*), 328
delete_uc() (*angr.state_plugins.unicorn_engine.Unicorn static method*), 398
delimiter (*angr.ailment.expression.Macro attribute*), 1396
demangled_name (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 1132
demangled_name (*angr.knowledge_plugins.Function property*), 724
demangled_name (*angr.knowledge_plugins.functions.Function property*), 784
demangled_name (*angr.knowledge_plugins.functions.function.Function property*), 808
demote() (*angr.state_plugins.history.SimStateHistory method*), 377
demote() (*angr.state_plugins.SimStateHistory method*), 302
dep_graph (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 1233
dep_graph (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 1216
dep_graph (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 1200
dep_graph (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 1207
dep_graph (*angr.analyses.ReachingDefinitionsAnalysis property*), 914
dependents (*angr.analyses.ddg.DDGViewItem property*), 1158
depends() (*angr.analyses.reaching_definitions.function_handler.FunctionCallDefs method*), 1224
depends() (*angr.analyses.reaching_definitions.FunctionCallDefs method*), 1184
depends_on (*angr.analyses.ddg.DDGViewItem property*), 1158
DepGraph (*class in angr.analyses.reaching_definitions.dep_graph*), 1217
DepNodeTypes (*class in angr.analyses.data_dep*), 1317
DepNodeTypes (*class in angr.analyses.data_dep.dep_nodes*), 1313
depth (*angr.ailment.Assignment property*), 1366
depth (*angr.ailment.Expression attribute*), 1369
depth (*angr.ailment.expression.Array attribute*), 1396
depth (*angr.ailment.expression.Expression attribute*), 1376
depth (*angr.ailment.expression.RustEnum attribute*), 1395
depth (*angr.ailment.Statement property*), 1372
depth (*angr.ailment.statement.Assignment property*), 1398
depth (*angr.ailment.statement.CAS property*), 1404
depth (*angr.ailment.statement.ConditionalJump property*), 1401
depth (*angr.ailment.statement.DirtyStatement property*), 1405
depth (*angr.ailment.statement.Jump property*), 1400
depth (*angr.ailment.statement.Label property*), 1406
depth (*angr.ailment.statement.Return property*), 1403
depth (*angr.ailment.statement.SideEffectStatement property*), 1402
depth (*angr.ailment.statement.Statement property*), 1397
depth (*angr.ailment.statement.Store property*), 1400
depth (*angr.ailment.statement.WeakAssignment property*), 1399
depth (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 769
depth (*angr.knowledge_plugins.cfg.CFGENode attribute*), 741
depth (*angr.state_plugins.sim_action_object.SimActionObject property*), 641
depth (*angr.state_plugins.SimActionObject property*), 265
deref (*angr.state_plugins.debug_variables.SimDebugVariable property*), 422
deref (*angr.state_plugins.SimDebugVariable property*), 267
deref (*angr.state_plugins.SimMemView property*), 285
deref (*angr.state_plugins.view.SimMemView property*), 431
deref() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1196
deref() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsS method*), 1237
deref() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1211
deref() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi method*), 854
deref() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 831
deref_src_addr (*angr.ailment.tagged_object.TagDict*

attribute), 1407
DerefSize (*class in angr.knowledge_plugins.key_definitions*), 823
DerefSize (*class in angr.knowledge_plugins.key_definitions.live_definitions*), 846
DerivedTypeVariable (*class in angr.analyses.typehoon.typevars*), 1270
desc (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.OrderWalker attribute*), 1102
describe_variables() (*angr.state_plugins.SimSolver method*), 288
describe_variables() (*angr.state_plugins.solver.SimSolver method*), 362
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.a_simplifier.AddressSimplifier attribute*), 1098
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.arg_spiller.ArgSpiller attribute*), 1072
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.arg_recovery.ArgRecovery attribute*), 1072
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.cascading_output_canonicalizer.CascadingOutputCanonicalizer attribute*), 1073
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.code_motion.CodeMotion attribute*), 1074
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.constant_derefs_simplifier.ConstantDerefsSimplifier attribute*), 1092
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.conversion_simplifier.ConversionSimplifier attribute*), 1075
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.copy_propagator.CopyPropagator attribute*), 1074
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.cross_jump_reverter.CrossJumpReverter attribute*), 1075
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.dead_assignments_remover.DeadAssignmentsRemover attribute*), 1076
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifier attribute*), 1098
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.duplicate_condition_remover.DuplicateConditionRemover attribute*), 1076
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.duplicate_return_remover.DuplicateReturnRemover attribute*), 1077
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifyerEngineBase attribute*), 1078
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.ExprOpSwapper attribute*), 1103
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.flag_page_accesses_simplifier.FlagPageAccessesSimplifier attribute*), 1079
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.flag_setting_cond_simplifier.FlagSettingCondSimplifier attribute*), 1079
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.fp_ldexp_simplifier.FpLdexpSimplifier attribute*), 1080
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.fp_mul_simplifier.FpMulSimplifier attribute*), 1080
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.fp_sqrt_simplifier.FpSqrtSimplifier attribute*), 1081
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.InlinedStringReplacer attribute*), 1081
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.InlinedStrlen attribute*), 1082
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.InlinedWcscpy attribute*), 1082
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 1099
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ITEExprConverter attribute*), 1079
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 1080
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier attribute*), 1100
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.LoweredSwitchSimplifier attribute*), 1083
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.MipsGpSetting attribute*), 1083
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 1101
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 1084
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute*), 1093
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.register_saver.RegisterSaveRemoverSimplifier attribute*), 1103
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.RegisterSaveRemoverSimplifier attribute*), 1085
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.RegisterSaveRemover attribute*), 1085
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ret_addr_saver.RetAddrSaver attribute*), 1104
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.RetAddrSaver attribute*), 1085
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.return_dedup.ReturnDedup attribute*), 1086
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator attribute*), 1086
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator attribute*), 1088
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 1097
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.StackCanarySimplifier attribute*), 1088
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.StaticVVarRemover attribute*), 1089
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.SwitchDefaultCase attribute*), 1089
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.SwitchReuseCase attribute*), 1090
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.TagSlicer attribute*), 1090
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.WinStackCanaryPrologue attribute*), 1090

DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_cast_simplifier.X86GccCastSimplifier attribute*), 1104
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_ffs_simplifier.X86GccFfsSimplifier attribute*), 1091
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.ConstSub attribute*), 1106
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.MultSub attribute*), 1105
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.PeepholeOptimizationStmtBase attribute*), 1104
description (*angr.sim_state_options.StateOption attribute*), 243
descriptions (*angr.state_plugins.history.SimStateHistory property*), 378
descriptions (*angr.state_plugins.SimStateHistory property*), 303
dest (*angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute*), 1221
destroy() (*angr.state_plugins.Unicorn method*), 329
destroy() (*angr.state_plugins.unicorn_engine.Unicorn method*), 399
desymbolize() (*angr.analyses.reassembler.Data method*), 1301
determine() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 1264
DFS (*class in angr.exploration_techniques*), 552
DFS (*class in angr.exploration_techniques.dfs*), 569
dfs_back_edges() (*in module angr.utils.graph*), 1347
dfs_postorder_nodes_deterministic() (*angr.utils.graph.GraphUtils static method*), 1350
diagnose_unsat() (*angr.analyses.Pathfinder method*), 911
dict_content (*angr.utils.dynamic_dictlist.DynamicDictList attribute*), 1346
dict_strkey_to_intkey() (*angr.angrdb.serializers.structured_code.StructuredCodeManagerSerializer static method*), 1032
DictBackerMixin (*class in angr.storage.memory_mixins*), 460
DictBackerMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 509
Difference (*class in angr.analyses.bindiff*), 947
difference() (*angr.sim_state_options.SimStateOptions method*), 245
differing_blocks (*angr.analyses.BinDiff property*), 883
differing_blocks (*angr.analyses.bindiff.BinDiff property*), 950
differing_blocks (*angr.analyses.bindiff.FunctionDiff property*), 948
differing_constants() (*in module angr.analyses.bindiff*), 947
differing_functions (*angr.analyses.BinDiff property*), 883
differing_functions (*angr.analyses.bindiff.BinDiff property*), 950
differing_functions() (*angr.analyses.BinDiff method*), 883
differing_functions() (*angr.analyses.bindiff.BinDiff method*), 950
direct_next_block (*angr.analyses.cfg.cfg_base.IRSB property*), 609
Director (*class in angr.exploration_techniques*), 554
Director (*class in angr.exploration_techniques.director*), 581
dirty (*angr.ailment.statement.DirtyStatement attribute*), 1405
dirty (*angr.analyses.decompiler.structured_codegen.c.CDirtyExpression attribute*), 1144
dirty (*angr.analyses.decompiler.structured_codegen.c.CDirtyStatement attribute*), 1137
dirty (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
dirty (*angr.knowledge_plugins.cfg.CFGNode property*), 750
dirty (*angr.knowledge_plugins.Function property*), 720
dirty (*angr.knowledge_plugins.functions.Function property*), 781
dirty (*angr.knowledge_plugins.functions.function.Function property*), 805
DIRTY (*angr.state_plugins.inspect.EventType attribute*), 332
Dirty() (*angr.ailment.converter_vex.VEXStmtConverter static method*), 1375
dirty_args (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
dirty_args (*angr.state_plugins.InspectAttrs attribute*), 255
dirty_func (*angr.knowledge_plugins.functions.function.FunctionManagerSerializer (in module angr.knowledge_plugins.functions.function)*), 799
dirty_handler (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
dirty_handler (*angr.state_plugins.InspectAttrs attribute*), 255
dirty_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1173
dirty_handler() (*angr.engines.light.SimEngineLightVEX static method*), 1170
dirty_name (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
dirty_name (*angr.state_plugins.InspectAttrs attribute*), 255
dirty_result (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
dirty_result (*angr.state_plugins.InspectAttrs at-*

tribute), 255
DirtyAddrsMixin (*class in angr.storage.memory_mixins*), 461
DirtyAddrsMixin (*class in angr.storage.memory_mixins.dirty_addrs_mixin*), 495
DirtyExpression (*class in angr.ailment.expression*), 1387
DirtyStatement (*class in angr.ailment.statement*), 1405
disable_profiling() (*in module angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 996
disable_timing() (*in module angr.state_plugins.solver*), 361
DisassemblerBlock (*class in angr.block*), 232
DisassemblerInsn (*class in angr.block*), 232
Disassembly (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
disassembly (*angr.Block property*), 161
disassembly (*angr.block.Block property*), 233
disassembly (*angr.engines.pcode.lifter.IRSB property*), 610
Disassembly (*class in angr.analyses*), 903
Disassembly (*class in angr.analyses.disassembly*), 1296
DisassemblyPiece (*class in angr.analyses.disassembly*), 1293
discard() (*angr.knowledge_plugins.structured_code.StructuredCodeManager method*), 822
discard() (*angr.knowledge_plugins.StructuredCodeManager method*), 734
discard() (*angr.sim_state_options.SimStateOptions method*), 245
discard_by_prefix() (*angr.knowledge_plugins.PropagationManager method*), 733
discard_by_prefix() (*angr.knowledge_plugins.propagations.PropagationManager method*), 774
discard_input() (*angr.state_plugins.cgc.SimStateCGC method*), 380
discard_input() (*angr.state_plugins.SimStateCGC method*), 297
discard_output() (*angr.state_plugins.cgc.SimStateCGC method*), 380
discard_output() (*angr.state_plugins.SimStateCGC method*), 297
discard_plugin_preset() (*angr.misc.plugins.PluginHub method*), 236
display_name (*angr.analyses.data_dep.dep_nodes.VarDepNode property*), 1315
display_name (*angr.analyses.data_dep.VarDepNode property*), 1319
dissect_instruction() (*angr.analyses.disassembly.Instruction method*), 1294
dissect_instruction_by_default() (*angr.analyses.disassembly.Instruction method*), 1294
dissect_instruction_for_arm() (*angr.analyses.disassembly.Instruction method*), 1294
DivSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1076
DivSimplifier (*class in angr.analyses.decompiler.optimization_passes.div_simplifier*), 1098
DivSimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.div_simplifier*), 1098
do_full_xrefs() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 992
do_full_xrefs() (*angr.analyses.cfg.CFGFast method*), 970
do_full_xrefs() (*angr.analyses.CFGFast method*), 892
do_trace() (*angr.analyses.Identifier method*), 907
do_trace() (*angr.analyses.identifier.identify.Identifier method*), 1283
dom (*angr.utils.graph.Dominators attribute*), 1348
dominance_frontiers (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase property*), 1250
DominanceFrontier (*class in angr.analyses*), 904
DominanceFrontier (*class in angr.analyses.dominance_frontier*), 1307
dominates() (*in module angr.utils.graph*), 1347
Dominators (*class in angr.utils.graph*), 1348
done (*angr.analyses.vfg.AnalysisTask property*), 1286
done (*angr.analyses.vfg.CallAnalysis property*), 1287
done (*angr.analyses.vfg.FunctionAnalysis property*), 1287
dowhile_known_tail_nodes (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer attribute*), 1038
downsize() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 979
downsize() (*angr.analyses.cfg.CFGEmulated method*), 964
downsize() (*angr.analyses.CFGEmulated method*), 887
downsize() (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method*), 936
downsize() (*angr.analyses.forward_analysis.ForwardAnalysis method*), 932
downsize() (*angr.analyses.ForwardAnalysis method*), 907
downsize() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1237

downsize() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1210
downsize() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 1251
downsize() (*angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFastState method*), 1252
downsize() (*angr.knowledge_plugins.cfg.cfg_node.CFGENode method*), 769
downsize() (*angr.knowledge_plugins.cfg.CFGENode method*), 741
downsize() (*angr.knowledge_plugins.propagations.PropagationModel method*), 775
downsize() (*angr.sim_state.SimState method*), 241
downsize() (*angr.SimState method*), 209
downsize() (*angr.state_plugins.inspect.SimInspector method*), 342
downsize() (*angr.state_plugins.sim_action.SimAction method*), 638
downsize() (*angr.state_plugins.sim_action.SimActionData method*), 640
downsize() (*angr.state_plugins.SimAction method*), 263
downsize() (*angr.state_plugins.SimActionData method*), 264
downsize() (*angr.state_plugins.SimInspector method*), 278
downsize() (*angr.state_plugins.SimSolver method*), 291
downsize() (*angr.state_plugins.solver.SimSolver method*), 364
downsize_region() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 1251
DreamStructurer (*class in angr.analyses.decompiler.structuring*), 1033
DreamStructurer (*class in angr.analyses.decompiler.structuring.dream*), 1035
DrillerCore (*class in angr.exploration_techniques*), 554
DrillerCore (*class in angr.exploration_techniques.driller_core*), 577
DROP (*angr.sim_manager.SimulationManager attribute*), 545
DROP (*angr.SimulationManager attribute*), 215
drop() (*angr.sim_manager.SimulationManager method*), 549
drop() (*angr.SimulationManager method*), 219
drop_bad_functions() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 992
drop_bad_functions() (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot method*), 1017
drop_bad_functions() (*angr.analyses.cfg.CFGFast method*), 970
drop_bad_functions() (*angr.analyses.cfg.CFGFastSoot method*), 977
drop_bad_functions() (*angr.analyses.CFGFast method*), 892
drop_bad_functions() (*angr.analyses.CFGFastSoot method*), 894
drop_db() (*angr.knowledge_plugins.RuntimeDb method*), 734
dst (*angr.ailment.Assignment attribute*), 1365
dst (*angr.ailment.statement.Assignment attribute*), 1397
dst (*angr.ailment.statement.WeakAssignment attribute*), 1398
dst (*angr.engines.pcode.lifter.ExitStatement attribute*), 605
dst (*angr.knowledge_plugins.xrefs.XRef attribute*), 861
dst (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 864
dst_addr (*angr.analyses.cfg.cfg_fast.FunctionCallEdge attribute*), 986
dst_addr (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge attribute*), 986
dst_addr (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute*), 986
dst_func_addr (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge attribute*), 987
dst_func_addr (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute*), 986
dst_type (*angr.analyses.decompiler.structured_codegen.c.CTypeCast attribute*), 1141
dummy (*angr.analyses.reaching_definitions.Definition attribute*), 1182
dummy (*angr.knowledge_plugins.key_definitions.Definition attribute*), 823
dummy (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 843
DummyStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen*), 1124
DummyStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.dummy*), 1149
dump() (*angr.angrdb.AngrDB method*), 1019
dump() (*angr.angrdb.db.AngrDB method*), 1021
dump() (*angr.angrdb.serializers.cfg_model.CFGModelSerializer static method*), 1026
dump() (*angr.angrdb.serializers.comments.CommentsSerializer static method*), 1027
dump() (*angr.angrdb.serializers.funcs.FunctionManagerSerializer static method*), 1027
dump() (*angr.angrdb.serializers.kb.KnowledgeBaseSerializer static method*), 1028
dump() (*angr.angrdb.serializers.KnowledgeBaseSerializer static method*), 1025

dump() (*angr.angrdb.serializers.labels.LabelsSerializer static method*), 1028
dump() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 1030
dump() (*angr.angrdb.serializers.LoaderSerializer static method*), 1026
dump() (*angr.angrdb.serializers.structured_code.StructuredCodeManagerSerializer static method*), 1031
dump() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 1031
dump() (*angr.angrdb.serializers.xrefs.XRefsSerializer static method*), 1030
dump() (*angr.calling_conventions.AllocHelper method*), 658
dump_constraint_graph() (*angr.analyses.typehoon.simple_solver.SimpleSolver static method*), 1265
dump_file_by_path() (*angr.state_plugins.posix.SimSystemPosix method*), 354
dump_file_by_path() (*angr.state_plugins.SimSystemPosix method*), 324
dump_graph() (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer static method*), 1039
dump_graph() (*angr.analyses.decompiler.structuring.PhoenixStructurer static method*), 1034
dump_internal() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 1031
dumps() (*angr.state_plugins.posix.SimSystemPosix method*), 354
dumps() (*angr.state_plugins.SimSystemPosix method*), 324
dumps() (*angr.vaults.Vault method*), 872
DUPLICATION_CHECK (*angr.analyses.decompiler.structured_codegen.base.PositionMapping attribute*), 1128
DUPLICATION_CHECK (*angr.analyses.decompiler.structured_codegen.PositionMapping attribute*), 1124
DuplicationReverter (*class in angr.analyses.decompiler.optimization_passes*), 1076
DURING_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 1092
DURING_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 1084
dwarf_cfa (*angr.state_plugins.debug_variables.SimDebugVariablePlugin property*), 422
dwarf_cfa (*angr.state_plugins.SimDebugVariablePlugin property*), 267
dwarf_cfa_approx (*angr.state_plugins.debug_variables.SimDebugVariablePlugin property*), 423
dwarf_cfa_approx (*angr.state_plugins.SimDebugVariablePlugin property*), 268
DYNAMIC_RET (*angr.sim_procedure.SimProcedure attribute*), 645
DYNAMIC_RET (*angr.SimProcedure attribute*), 204
dynamic_returns() (*angr.sim_procedure.SimProcedure method*), 645
dynamic_returns() (*angr.SimProcedure method*), 205
DynamicDictList (*class in angr.utils.dynamic_dictlist*), 1346

E

E2BIG (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
E2BIG (*angr.state_plugins.SimSystemPosix attribute*), 321
EACCES (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EACCES (*angr.state_plugins.SimSystemPosix attribute*), 321
EAGAIN (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EAGAIN (*angr.state_plugins.SimSystemPosix attribute*), 321
EagerStdStringConcatenationPass (*class in angr.analyses.decompiler.optimization_passes*), 1078
EBADF (*angr.state_plugins.cgc.SimStateCGC attribute*), 380
EBADF (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EBADF (*angr.state_plugins.SimStateCGC attribute*), 296
EBADF (*angr.state_plugins.SimSystemPosix attribute*), 321
EBUSY (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EBUSY (*angr.state_plugins.SimSystemPosix attribute*), 321
ECHILD (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ECHILD (*angr.state_plugins.SimSystemPosix attribute*), 321
edge_hitmap (*angr.state_plugins.history.SimStateHistory attribute*), 375
edges_to_remove (*angr.analyses.decompiler.clinic.Clinic attribute*), 1061
edges_to_repair (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 757
edges_to_repair (*angr.knowledge_plugins.cfg.CFGModel attribute*), 743
EDOM (*angr.state_plugins.posix.SimSystemPosix attribute*), 352
EDOM (*angr.state_plugins.SimSystemPosix attribute*), 322
EEXIST (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EEXIST (*angr.state_plugins.SimSystemPosix attribute*), 321

EFAULT (*angr.state_plugins.cgc.SimStateCGC attribute*), 380
EFAULT (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EFAULT (*angr.state_plugins.SimStateCGC attribute*), 296
EFAULT (*angr.state_plugins.SimSystemPosix attribute*), 321
EFBIG (*angr.state_plugins.posix.SimSystemPosix attribute*), 352
EFBIG (*angr.state_plugins.SimSystemPosix attribute*), 321
EffectiveSizeExtractor (*class in angr.analyses.decompiler.expression_narrower*), 1069
effects (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 1224
effects (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
EH4ScopeTable (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
EH4ScopeTable (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
EH_FRAME_HINTS (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 987
EHFuncInfo (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
EHFuncInfo (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
EHHandlerType (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
EHHandlerType (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
EHTryBlockMap (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
EHTryBlockMap (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
EHUnwindMapEntry (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
EHUnwindMapEntry (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
EINTR (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EINTR (*angr.state_plugins.SimSystemPosix attribute*), 321
EINVAL (*angr.state_plugins.cgc.SimStateCGC attribute*), 380
EINVAL (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EINVAL (*angr.state_plugins.SimStateCGC attribute*), 296
EINVAL (*angr.state_plugins.SimSystemPosix attribute*), 321
EIO (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EIO (*angr.state_plugins.SimSystemPosix attribute*), 321
EISDIR (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EISDIR (*angr.state_plugins.SimSystemPosix attribute*), 321
elem_count (*angr.analyses.typehoon.typevars.HasField attribute*), 1272
elem_type (*angr.sim_type.SimTypeArray attribute*), 696
element (*angr.analyses.typehoon.typeconsts.Array attribute*), 1278
ELFHeader (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
ELFHeader (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 755
eliminatable (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegC property*), 1000
eliminatable (*angr.analyses.data_dep.data_dependency_analysis.NodalA property*), 1312
eliminatable (*angr.analyses.decompiler.condition_processor.AILExprIdA property*), 1062
eliminatable (*angr.analyses.variable_recovery.annotations.StackLocatio property*), 1246
eliminatable (*angr.analyses.variable_recovery.annotations.VariableSour property*), 1246
eliminatable (*angr.analyses.variable_recovery.variable_recovery_base.V property*), 1247
eliminatable (*angr.knowledge_plugins.key_definitions.live_definitions.De property*), 846
eliminatable (*angr.storage.memory_mixins.address_concretization_mixi property*), 495
elseNode (*angr.analyses.decompiler.structured_codegen.c.CIfElse attribute*), 1134
EMFILE (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EMFILE (*angr.state_plugins.SimSystemPosix attribute*), 321
EMLINK (*angr.state_plugins.posix.SimSystemPosix attribute*), 352
EMLINK (*angr.state_plugins.SimSystemPosix attribute*), 322
EmptyPort (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 1309
empty_block() (*angr.engines.pcode.lifter.IRSB static method*), 608
EmptyNodeRemover (*class in angr.analyses.decompiler.empty_node_remover*), 1068
Emulator (*class in angr*), 161
EmulatorStopReason (*class in angr*), 162
enable_profiling() (*in module angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 996
enable_timing() (*in module angr.state_plugins.solver*), 361
end (*angr.keyed_region.RegionObject property*), 868
end_info_hook() (*in module angr.state_plugins.trace_additions*), 382

endness (*angr.ailment.expression.Extract attribute*), 1391
endness (*angr.ailment.expression.Insert attribute*), 1392
endness (*angr.ailment.expression.Load attribute*), 1386
endness (*angr.ailment.statement.CAS attribute*), 1404
endness (*angr.ailment.statement.Store attribute*), 1399
endness (*angr.analyses.reaching_definitions.MemoryLocation attribute*), 1197
endness (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation attribute*), 841
endness (*angr.storage.memory_object.SimMemoryObject attribute*), 451
endness (*angr.storage.SimMemoryObject attribute*), 425
endpoints (*angr.knowledge_plugins.Function property*), 720
endpoints (*angr.knowledge_plugins.functions.Function property*), 780
endpoints (*angr.knowledge_plugins.functions.function.Function property*), 804
endpoints_with_type (*angr.knowledge_plugins.Function property*), 720
endpoints_with_type (*angr.knowledge_plugins.functions.Function property*), 780
endpoints_with_type (*angr.knowledge_plugins.functions.function.Function property*), 804
ENFILE (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ENFILE (*angr.state_plugins.SimSystemPosix attribute*), 321
ENGINE_PROCESS (*angr.state_plugins.inspect.EventType attribute*), 332
ENODEV (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ENODEV (*angr.state_plugins.SimSystemPosix attribute*), 321
ENOENT (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ENOENT (*angr.state_plugins.SimSystemPosix attribute*), 321
ENOEXEC (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ENOEXEC (*angr.state_plugins.SimSystemPosix attribute*), 321
ENOMEM (*angr.state_plugins.cgc.SimStateCGC attribute*), 380
ENOMEM (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ENOMEM (*angr.state_plugins.SimStateCGC attribute*), 296
ENOMEM (*angr.state_plugins.SimSystemPosix attribute*), 321
ENOSPC (*angr.state_plugins.posix.SimSystemPosix attribute*), 352
ENOSPC (*angr.state_plugins.SimSystemPosix attribute*), 322
ENOSYS (*angr.state_plugins.cgc.SimStateCGC attribute*), 380
ENOSYS (*angr.state_plugins.SimStateCGC attribute*), 296
ENOTBLK (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ENOTBLK (*angr.state_plugins.SimSystemPosix attribute*), 321
ENOTDIR (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ENOTDIR (*angr.state_plugins.SimSystemPosix attribute*), 321
ENOTTY (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ENOTTY (*angr.state_plugins.SimSystemPosix attribute*), 321
entries (*angr.knowledge_plugins.cfg.indirect_jump.JumptableInfo attribute*), 770
entry_node_addr (*angr.analyses.decompiler.clinic.Clinic attribute*), 1060
entry_node_addr (*angr.analyses.decompiler.optimization_passes.optimiz attribute*), 1095
entry_size (*angr.knowledge_plugins.cfg.indirect_jump.JumptableInfo attribute*), 770
entry_state() (*angr.factory.AngrObjectFactory method*), 228
entrypoints (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSlice property*), 1241
entrypoints (*angr.analyses.cfg_slice_to_sink.CFGSliceToSink property*), 1239
Enum (*class in angr.analyses.typehoon.typeconsts*), 1279
EnumVariant (*class in angr.analyses.typehoon.typeconsts*), 1279
environment (*angr.analyses.reaching_definitions.rd_state.ReachingDefini property*), 1233
environment (*angr.analyses.reaching_definitions.ReachingDefinitionsStat property*), 1207
Environment (*class in angr.knowledge_plugins.key_definitions.environment*), 844
ENXIO (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ENXIO (*angr.state_plugins.SimSystemPosix attribute*), 321
eof() (*angr.SimFileDescriptor method*), 183
eof() (*angr.SimFileDescriptorDuplex method*), 185
eof() (*angr.storage.file.SimFileDescriptor method*), 444
eof() (*angr.storage.file.SimFileDescriptorBase method*), 443
eof() (*angr.storage.file.SimFileDescriptorDuplex method*), 447
EPERM (*angr.state_plugins.posix.SimSystemPosix attribute*), 351

EPERM (*angr.state_plugins.SimSystemPosix attribute*), 321
EPIPE (*angr.state_plugins.cgc.SimStateCGC attribute*), 380
EPIPE (*angr.state_plugins.posix.SimSystemPosix attribute*), 352
EPIPE (*angr.state_plugins.SimStateCGC attribute*), 296
EPIPE (*angr.state_plugins.SimSystemPosix attribute*), 322
EQ (*angr.analyses.decompiler.region_simplifiers.switch_cluster.RelativeCmpOp attribute*), 1115
Eq (*class in angr.analyses.stack_pointer_tracker*), 1243
eq() (*angr.ailment.Statement method*), 1371
eq() (*angr.ailment.statement.Statement method*), 1397
eqclass_constraints_count (*angr.analyses.typehoon.simple_solver.SimpleSolver attribute*), 1263
equivalence (*angr.knowledge_plugins.propagations.PropagationModel attribute*), 775
Equivalence (*class in angr.analyses.typehoon.typevars*), 1266
ERANGE (*angr.state_plugins.posix.SimSystemPosix attribute*), 352
ERANGE (*angr.state_plugins.SimSystemPosix attribute*), 322
erase() (*angr.storage.memory_mixins.ListPage method*), 467
erase() (*angr.storage.memory_mixins.MVListPage method*), 469
erase() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixins.PagedMemoryMixin method*), 505
erase() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixins.ListPage method*), 521
erase() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixins.MVListPage method*), 511
erase() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixins.ListPage method*), 500
erase() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixins.MVListPage method*), 513
erase() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 475
EROFS (*angr.state_plugins.posix.SimSystemPosix attribute*), 352
EROFS (*angr.state_plugins.SimSystemPosix attribute*), 322
errno (*angr.state_plugins.libc.SimStateLibc property*), 347
errno (*angr.state_plugins.SimStateLibc property*), 311
error (*angr.sim_manager.ErrorRecord attribute*), 551
error_converter() (*in module angr.state_plugins.solver*), 361
ERROR_REG (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 676
ERROR_REG (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 675
ERROR_REG (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 678
ERROR_REG (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 677
ERROR_REG (*angr.calling_conventions.SimCCSyscall attribute*), 670
errored (*angr.sim_manager.SimulationManager property*), 546
errored (*angr.SimulationManager property*), 215
ErrorRecord (*class in angr.sim_manager*), 551
errors (*angr.analyses.Analysis attribute*), 881
errors (*angr.analyses.analysis.Analysis attribute*), 930
errors (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 1065
errors (*angr.Analysis attribute*), 156
errors (*angr.angrdb.models.DbStructuredCode attribute*), 1024
ESPIPE (*angr.state_plugins.posix.SimSystemPosix attribute*), 352
ESPIPE (*angr.state_plugins.SimSystemPosix attribute*), 322
ESRCH (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ESRCH (*angr.state_plugins.SimSystemPosix attribute*), 321
ETXTBSY (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
ETXTBSY (*angr.state_plugins.SimSystemPosix attribute*), 321
eval() (*angr.state_plugins.SimSolver method*), 293
eval() (*angr.state_plugins.solver.SimSolver method*), 367
eval_atleast() (*angr.state_plugins.SimSolver method*), 295
eval_atleast() (*angr.state_plugins.solver.SimSolver method*), 368
eval_atmost() (*angr.state_plugins.SimSolver method*), 294
eval_atmost() (*angr.state_plugins.solver.SimSolver method*), 368
eval_exact() (*angr.state_plugins.SimSolver method*), 295
eval_exact() (*angr.state_plugins.solver.SimSolver method*), 369
eval_one() (*angr.state_plugins.SimSolver method*), 293
eval_one() (*angr.state_plugins.solver.SimSolver method*), 367
eval_to_ast() (*angr.state_plugins.SimSolver method*), 291
eval_to_ast() (*angr.state_plugins.solver.SimSolver method*), 364
eval_upto() (*angr.state_plugins.SimSolver method*), 292
eval_upto() (*angr.state_plugins.solver.SimSolver method*), 366
evaluate_binary() (*angr.engines.pcode.behavior.OpBehavior method*), 616

evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorBoolAnd method*), 629
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorBoolOr method*), 630
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorBoolXor method*), 629
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorEqual method*), 617
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorFloatAdd method*), 621
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorFloatMult method*), 624
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorFloatCurry method*), 622
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorFloatDiv method*), 627
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorLoffset method*), 625
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntLess method*), 619
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual method*), 620
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntMult method*), 626
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntOr method*), 625
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntRem method*), 628
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntRight method*), 626
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSborrow method*), 623
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntScarry method*), 622
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSdiv method*), 627
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSless method*), 618
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual method*), 619
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSrem method*), 628
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSright method*), 626
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSub method*), 621
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntXor method*), 624
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorNotEqual method*), 618
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorSubpiece method*), 634
evaluate_unary() (*angr.engines.pcode.behavior.OpBehavior method*), 616
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorBoolNegate method*), 629
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorCopy method*), 617
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorInt2Comp method*), 623
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorIntNegate method*), 623
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorIntSext method*), 620
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorIntZext method*), 620
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorLzcount method*), 635
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorPopcount method*), 635
events_are_zero() (*angr.analyses.decompiler.optimization_passes.InlinedWcscpySim static method*), 1082
events (*angr.analyses.vfg.VFGNode attribute*), 1288
events (*angr.state_plugins.history.SimStateHistory property*), 377
events (*angr.state_plugins.SimStateHistory property*), 303
events_of_type() (*angr.state_plugins.log.SimStateLog method*), 370
events_of_type() (*angr.state_plugins.SimStateLog method*), 311
EventType (*class in angr.state_plugins.inspect*), 331
evict_all_cached() (*angr.knowledge_plugins.functions.function_manag method*), 794
evicted (*angr.knowledge_plugins.Function attribute*), 717
evicted (*angr.knowledge_plugins.functions.Function attribute*), 777
evicted (*angr.knowledge_plugins.functions.function.Function attribute*), 802
evicted (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 810
evidence (*angr.analyses.LanguageDetector property*), 909
EXC_COUNTER (*angr.analyses.decompiler.condition_processor.ConditionPr attribute*), 1063
EXDEV (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
EXDEV (*angr.state_plugins.SimSystemPosix attribute*), 321
execute() (*angr.Project method*), 172
execute() (*angr.project.Project method*), 227
execute() (*angr.sim_procedure.SimProcedure method*), 644
execute() (*angr.SimProcedure method*), 204
ExecuteAddressGoal (*class in angr.exploration_techniques*), 555
ExecuteAddressGoal (*class in*

angr.exploration_techniques.director), 579
executed_instruction_count (angr.errors.SimError attribute), 1358
executed_instruction_count (angr.SimError attribute), 178
Existence (class in angr.analyses.typehoon.typevars), 1267
exists() (angr.knowledge_plugins.PropagationManager method), 733
exists() (angr.knowledge_plugins.propagations.PropagationManager method), 773
exists_in_replacements() (in module angr.analyses.cfg.indirect_jump_resolvers.const_resolver), 1003
EXIT (angr.EmulatorStopReason attribute), 162
EXIT (angr.state_plugins.inspect.EventType attribute), 331
Exit() (angr.ailment.converter_vex.VEXStmtConverter static method), 1375
exit() (angr.sim_procedure.SimProcedure method), 646
exit() (angr.SimProcedure method), 206
exit_guard (angr.state_plugins.inspect.InspectAttrs attribute), 335
exit_guard (angr.state_plugins.InspectAttrs attribute), 254
exit_hook() (in module angr.state_plugins.trace_additions), 382
exit_jumpkind (angr.state_plugins.inspect.InspectAttrs attribute), 335
exit_jumpkind (angr.state_plugins.InspectAttrs attribute), 254
exit_observe() (angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method), 1217
exit_observe() (angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method), 1201
exit_observe() (angr.analyses.ReachingDefinitionsAnalysis method), 915
exit_observed (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState attribute), 1231
exit_observed (angr.analyses.reaching_definitions.ReachingDefinitionsState attribute), 1205
exit_statements (angr.engines.pcode.lifter.IRSB property), 608
exit_target (angr.state_plugins.inspect.InspectAttrs attribute), 335
exit_target (angr.state_plugins.InspectAttrs attribute), 254
ExitStatement (class in angr.engines.pcode.lifter), 605
expd_hi (angr.ailment.statement.CAS attribute), 1404
expd_lo (angr.ailment.statement.CAS attribute), 1404
ExplicitFillerMixin (class in angr.storage.memory_mixins), 461
ExplicitFillerMixin (class in angr.storage.memory_mixins.default_filler_mixin), 491
ExplorationStatusNotifier (class in angr.distributed.worker), 1363
ExplorationTechnique (class in angr), 162
ExplorationTechnique (class in angr.exploration_techniques), 555
explore() (angr.sim_manager.SimulationManager method), 546
explore() (angr.SimulationManager method), 216
ExplorerMixin (class in angr.exploration_techniques), 557
Explorer (class in angr.exploration_techniques.explorer), 569
expr (angr.ailment.expression.MultiStatementExpression attribute), 1389
expr (angr.ailment.statement.SideEffectStatement attribute), 1402
expr (angr.analyses.decompiler.optimization_passes.lowered_switch_simpl attribute), 1100
expr (angr.analyses.decompiler.structured_codegen.c.CExpressionStatemen attribute), 1136
expr (angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpr attribute), 1143
expr (angr.analyses.decompiler.structured_codegen.c.CTypeCast attribute), 1141
EXPR (angr.state_plugins.inspect.EventType attribute), 331
expr (angr.state_plugins.inspect.InspectAttrs attribute), 335
expr (angr.state_plugins.InspectAttrs attribute), 254
expr_classes (angr.analyses.decompiler.peephole_optimizations.base.Pee attribute), 1106
expr_comments (angr.analyses.decompiler.structured_codegen.base.BaseS attribute), 1129
expr_comments (angr.angrdb.models.DbStructuredCode attribute), 1024
expr_handlers (angr.ailment.AILBlockWalker attribute), 1365
expr_idx (angr.analyses.decompiler.region_simplifiers.expr_folding.Expre attribute), 1110
expr_result (angr.state_plugins.inspect.InspectAttrs attribute), 335
expr_result (angr.state_plugins.InspectAttrs attribute), 254
expr_to_effective_bits (angr.analyses.decompiler.expression_narrower.EffectiveSizeExtr attribute), 1070
expr_used_as_call_arg_effective_bits (angr.analyses.decompiler.expression_narrower.EffectiveSizeExtr attribute), 1070
expr_used_as_insert_base (angr.analyses.decompiler.expression_narrower.EffectiveSizeExtr attribute), 1070
Expression (class in angr.ailment), 1369
Expression (class in angr.ailment.expression), 1375

ExpressionCounter (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1112
ExpressionFolder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1113
ExpressionLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1110
ExpressionNarrower (*class in angr.analyses.decompiler.expression_narrower*), 1070
ExpressionReplacer (*class in angr.analyses.decompiler.optimization_passes.expr_extractor*), 1102
ExpressionReplacer (*class in angr.analyses.decompiler.optimization_passes.ite_expr_converter*), 1098
ExpressionReplacer (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1113
expressions (*angr.engines.pcode.lifter.IRSB property*), 609
ExpressionSpotter (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1112
ExpressionUseFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1111
ExprNarrowingInfo (*class in angr.analyses.decompiler.expression_narrower*), 1069
ExprOpSwapper (*class in angr.analyses.decompiler.optimization_passes*), 1078
ExprOpSwapper (*class in angr.analyses.decompiler.optimization_passes.expr_extractor*), 1102
extend() (*angr.engines.pcode.lifter.IRSB method*), 608
extend_actions() (*angr.state_plugins.history.SimStateHistory method*), 377
extend_actions() (*angr.state_plugins.log.SimStateLog method*), 370
extend_actions() (*angr.state_plugins.SimStateHistory method*), 303
extend_actions() (*angr.state_plugins.SimStateLog method*), 311
extern (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 842
extern_idx (*angr.code_location.AILCodeLocation property*), 867
extern_idx_unwrap (*angr.code_location.AILCodeLocation property*), 867
ExternalCodeLocation (*class in angr.code_location*), 866
extern_vars (*angr.analyses.decompiler.clinic.Clinic attribute*), 1060
EXTRA_ARCHES (*angr.calling_conventions.SimCC attribute*), 664
EXTRA_ARCHES (*angr.calling_conventions.SimCCARMHF attribute*), 673
EXTRA_ARCHES (*angr.SimCC attribute*), 174
extra_def (*angr.ailment.tagged_object.TagDict attribute*), 1407
extra_defs (*angr.ailment.tagged_object.TagDict attribute*), 1407
Extract (*class in angr.ailment.expression*), 1391
extract() (*angr.sim_type.SimCppClass method*), 706
extract() (*angr.sim_type.SimStruct method*), 701
extract() (*angr.sim_type.SimType method*), 687
extract() (*angr.sim_type.SimTypeArray method*), 696
extract() (*angr.sim_type.SimTypeBitfield method*), 705
extract() (*angr.sim_type.SimTypeBool method*), 693
extract() (*angr.sim_type.SimTypeChar method*), 692
extract() (*angr.sim_type.SimTypeEnum method*), 704
extract() (*angr.sim_type.SimTypeFd method*), 694
extract() (*angr.sim_type.SimTypeFloat method*), 700
extract() (*angr.sim_type.SimTypeInt method*), 691
extract() (*angr.sim_type.SimTypeNum method*), 690
extract() (*angr.sim_type.SimTypeNumOffset method*), 707
extract() (*angr.sim_type.SimTypePointer method*), 695
extract() (*angr.sim_type.SimTypeReference method*), 695
extract() (*angr.sim_type.SimTypeString method*), 697
extract() (*angr.sim_type.SimTypeWideChar method*), 693
extract() (*angr.sim_type.SimTypeWString method*), 698
extract() (*angr.sim_type.SimUnion method*), 703
extract() (*angr.storage.memory_mixins.paged_memory.pages.multi_valu method*), 502
extract_claripy() (*angr.sim_type.SimType method*), 687
extract_components() (*angr.procedures.stubs.format_parser.FormatParser method*), 648
extract_defs() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 1191
extract_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDe method*), 1233
extract_defs() (*angr.analyses.reaching_definitions.ReachingDefinitions. method*), 1206
extract_defs() (*angr.knowledge_plugins.key_definitions.live_definitions. static method*), 849
extract_defs() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 826
extract_defs_from_annotations()

(*angr.analyses.reaching_definitions.LiveDefinitions static method*), 1191
extract_defs_from_annotations() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 849
extract_defs_from_annotations() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 826
extract_defs_from_mv() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 1191
extract_defs_from_mv() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 849
extract_defs_from_mv() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 826
extract_jump_targets() (*in module angr.analyses.decompiler.utils*), 1150
extract_offset_to_sp() (*angr.engines.light.engine.SimEngineLight static method*), 1172
extract_offset_to_sp() (*angr.engines.light.SimEngineLight static method*), 1168
extract_terms() (*in module angr.analyses.decompiler.structured_codegen.c*), 1130
extract_value_if_concrete() (*angr.analyses.xrefs.SimEngineXRefsVEX static method*), 1308
extract_variables() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 1249

F

f (*angr.utils.mp.Closure attribute*), 1354
FactCollector (*class in angr.analyses*), 904
FactCollector (*class in angr.analyses.calling_convention*), 952
FAILURE (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.Case2Result attribute*), 996
FAILURE (*angr.EmulatorStopReason attribute*), 162
false_target (*angr.ailment.statement.ConditionalJump attribute*), 1401
false_target_idx (*angr.ailment.statement.ConditionalJump attribute*), 1401
FAST (*angr.analyses.complete_calling_conventions.CallingConventionAnalysisMode attribute*), 953
fast_memory_load() (*angr.analyses.Reassembler method*), 919
fast_memory_load() (*angr.analyses.reassembler.Reassembler method*), 1305
FastConstantPropagation (*class in angr.analyses*), 905
FASTISH (*angr.analyses.complete_calling_conventions.CallingConventionAnalysisMode attribute*), 953
FastMemory (*class in angr.storage.memory_mixins*), 461
fd (*angr.state_plugins.unicorn_engine.TRANSMIT_RECORD attribute*), 392
Fd (*class in angr.analyses.typehoon.typeconsts*), 1275
FD_SETSIZE (*angr.state_plugins.cgc.SimStateCGC attribute*), 380
FD_SETSIZE (*angr.state_plugins.SimStateCGC attribute*), 296
features (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
FetchingZeroPageError, 396
field (*angr.analyses.decompiler.structured_codegen.c.CStructField attribute*), 1138
field_names (*angr.ailment.expression.Struct attribute*), 1394
field_offsets (*angr.ailment.expression.Struct attribute*), 1394
FieldReferenceCleanup (*class in angr.analyses.decompiler.structured_codegen.c*), 1148
fields (*angr.ailment.expression.Struct attribute*), 1394
fields (*angr.analyses.typehoon.typeconsts.EnumVariant attribute*), 1279
fields (*angr.sim_type.SimStruct attribute*), 701
fields (*angr.sim_type.SimUnion attribute*), 702
file_exists (*angr.SimFileDescriptor property*), 183
file_exists (*angr.storage.file.SimFileDescriptor property*), 445
file_exists (*angr.storage.file.SimFileDescriptorBase property*), 444
file_types (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
fill_content() (*angr.knowledge_plugins.cfg.memory_data.MemoryData method*), 765
fill_content() (*angr.knowledge_plugins.cfg.MemoryData method*), 755
fill_reg_map() (*in module angr.analyses.reassembler*), 1297
filter() (*angr.analyses.cfg.indirect_jump_resolvers.AArch64MachOGotR method*), 1005
filter() (*angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got.AMD method*), 994
filter() (*angr.analyses.cfg.indirect_jump_resolvers.AMD64ElfGotResolv method*), 1005
filter() (*angr.analyses.cfg.indirect_jump_resolvers.AMD64PeIatResolve method*), 1006
filter() (*angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast.ArmElf method*), 995
filter() (*angr.analyses.cfg.indirect_jump_resolvers.ArmElfFastResolver method*), 1007

filter() (*angr.analyses.cfg.indirect_jump_resolvers.const_resolver.ConstantResolver method*), 1003
filter() (*angr.analyses.cfg.indirect_jump_resolvers.ConstantResolver method*), 1008
filter() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver method*), 1002
filter() (*angr.analyses.cfg.indirect_jump_resolvers.JumpTableResolver method*), 1009
filter() (*angr.analyses.cfg.indirect_jump_resolvers.MemoryLoadResolver method*), 1010
filter() (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.MipsElfFastResolver method*), 996
filter() (*angr.analyses.cfg.indirect_jump_resolvers.MipsElfFastResolver method*), 1011
filter() (*angr.analyses.cfg.indirect_jump_resolvers.MipsElfGotResolver method*), 1011
filter() (*angr.analyses.cfg.indirect_jump_resolvers.resolver.IndirectJumpResolver method*), 1004
filter() (*angr.analyses.cfg.indirect_jump_resolvers.SyscallResolver method*), 1012
filter() (*angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt.X86ElfPicPltResolver method*), 997
filter() (*angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat.X86PeIatResolver method*), 995
filter() (*angr.analyses.cfg.indirect_jump_resolvers.X86ElfPicPltResolver method*), 1013
filter() (*angr.analyses.cfg.indirect_jump_resolvers.X86PeIatResolver method*), 1014
filter() (*angr.analyses.veritesting.CallTracingFilter method*), 1284
filter() (*angr.exploration_techniques.ExplorationTechnique method*), 556
filter() (*angr.exploration_techniques.Explorer method*), 558
filter() (*angr.exploration_techniques.explorer.Explorer method*), 570
filter() (*angr.exploration_techniques.local_loop_seer.LocalLoopSeer method*), 583
filter() (*angr.exploration_techniques.LocalLoopSeer method*), 559
filter() (*angr.exploration_techniques.loop_seer.LoopSeer method*), 582
filter() (*angr.exploration_techniques.LoopSeer method*), 560
filter() (*angr.exploration_techniques.Slicecutor method*), 562
filter() (*angr.exploration_techniques.slicecutor.Slicecutor method*), 578
filter() (*angr.exploration_techniques.Tracer method*), 567
filter() (*angr.exploration_techniques.tracer.Tracer method*), 576
filter() (*angr.ExplorationTechnique method*), 163
filter() (*angr.sim_manager.SimulationManager method*), 548
filter() (*angr.SimulationManager method*), 218
filter_actions() (*angr.state_plugins.history.SimStateHistory method*), 377
filter_actions() (*angr.state_plugins.SimStateHistory method*), 302
filter_cond_regions() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl*
filter_constraints() (*angr.state_plugins.trace_additions.ZenPlugin method*), 385
final_states (*angr.analyses.vfg.VFG attribute*), 1290
final_states (*angr.analyses.vfg.VFGNode attribute*), 1288
final_states (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 769
final_states (*angr.knowledge_plugins.cfg.CFGENode attribute*), 741
find() (*angr.storage.memory_mixins.regioned_memory.regioned_memory method*), 571
find() (*angr.storage.memory_mixins.regioned_memory.RegionedMemory*
find() (*angr.storage.memory_mixins.regioned_memory.static_find_mixin.*
find() (*angr.storage.memory_mixins.regioned_memory.StaticFindMixin method*), 530
find() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 482
find() (*angr.storage.memory_mixins.smart_find_mixin.SmartFindMixin method*), 490
find() (*angr.storage.memory_mixins.SmartFindMixin method*), 485
find() (*angr.storage.memory_mixins.StaticFindMixin method*), 487
find() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 498
find() (*angr.storage.memory_mixins.UnwrapperMixin method*), 490
find_all_predecessors() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 1219
find_all_successors() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 1220
find_best_hypothesis_path() (*angr.analyses.Pathfinder method*), 911
find_block_by_addr() (*in module angr.analyses.decompiler.utils*), 1155
find_block_by_addr_and_idx() (*in module angr.analyses.decompiler.utils*), 1156
find_cc() (*angr.calling_conventions.SimCC static method*), 666
find_cc() (*angr.SimCC static method*), 177

find_conflicting_call_args() (*angr.analyses.decompiler.optimization_passes.ConstPropOptimizationPass static method*), 1074
find_consumers() (*angr.analyses.DDG method*), 877
find_consumers() (*angr.analyses.ddg.DDG method*), 1161
find_data_references_and_update_memory_data() (*angr.analyses.Decompiler method*), 902
find_data_references_and_update_memory_data() (*angr.analyses.decompiler.Decompiler method*), 1047
find_data_references_and_update_memory_data() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1067
find_declaration() (*angr.knowledge_plugins.Function method*), 723
find_declaration() (*angr.knowledge_plugins.functions.Function method*), 783
find_declaration() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 808
find_definition() (*angr.analyses.decompiler.peephole_optimizations.PeepholeOptimizationExprBase static method*), 1107
find_definitions() (*angr.analyses.DDG method*), 877
find_definitions() (*angr.analyses.ddg.DDG method*), 1160
find_definitions() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 1219
find_defs_at() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 1202
find_defs_at() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 856
find_defs_at() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 833
find_function_for_reflow_into_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 763
find_function_for_reflow_into_addr() (*angr.knowledge_plugins.cfg.CFGModel method*), 749
find_killers() (*angr.analyses.DDG method*), 877
find_killers() (*angr.analyses.ddg.DDG method*), 1161
find_merge_points() (*angr.utils.graph.GraphUtils static method*), 1349
find_path() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 1220
find_paths() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 1220
find_sources() (*angr.analyses.DDG method*), 878
find_sources() (*angr.analyses.ddg.DDG method*), 1161
find_stack_vars_x86() (*angr.analyses.Identifier method*), 908
find_stack_vars_x86() (*angr.analyses.identifier.identify.Identifier method*), 1283
find_variable_by_atom() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 815
find_variable_by_stmt() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 815
find_variables_by_atom() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 815
find_variables_by_insn() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 814
find_variables_by_register() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 815
find_variables_by_stack_offset() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 815
find_variables_by_stmt() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 815
find_widening_points() (*angr.utils.graph.GraphUtils static method*), 1349
FindFirstNodeInSet (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl*), 1118
finish() (*angr.state_plugins.unicorn.Unicorn method*), 328
finish() (*angr.state_plugins.unicorn_engine.Unicorn method*), 399
fire() (*angr.BP method*), 155
fire() (*angr.state_plugins.inspect.BP method*), 340
first_label() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 1270
first_nonlabel_node() (*in module angr.analyses.decompiler.utils*), 1152
first_nonlabel_nonphi_node() (*in module angr.analyses.decompiler.utils*), 1153
first_nonlabel_nonphi_statement() (*in module angr.analyses.decompiler.utils*), 1152
first_nonlabel_statement() (*in module angr.analyses.decompiler.utils*), 1152
first_nonlabel_statement_id() (*in module angr.analyses.decompiler.utils*), 1152
fix_prototype_returnty() (*angr.sim_procedure.SimProcedure method*), 646
fix_prototype_returnty() (*angr.SimProcedure method*), 205
flags (*angr.procedures.stubs.format_parser.FormatParser attribute*), 648

flags (*angr.sim_type.SimTypeBitfield attribute*), 704
Flags (*class in angr.storage.file*), 431
flatten_typevar() (*angr.analyses.typehoon.simple_solver.Sketch static method*), 1261
flavor (*angr.angrdb.models.DbStructuredCode attribute*), 1024
FlipBooleanCmp (*class in angr.analyses.decompiler.optimization_passes*), 1079
flirt_arch_to_arch_name() (*in module angr.analyses.flirt*), 1166
flirt_os_type_to_os_name() (*in module angr.analyses.flirt*), 1166
FlirtAnalysis (*class in angr.analyses*), 905
FlirtAnalysis (*class in angr.analyses.flirt*), 1161
FlirtSignature (*class in angr.analyses.flirt*), 1162
FlirtSignature (*class in angr.flirt*), 1342
FlirtSignatureError, 1163
FlirtSignatureParsed (*class in angr.analyses.flirt*), 1163
FLOAT (*angr.analyses.decompiler.structured_codegen.base.CConstantType attribute*), 1128
Float (*class in angr.analyses.typehoon.typeconsts*), 1277
Float32 (*class in angr.analyses.typehoon.typeconsts*), 1277
Float64 (*class in angr.analyses.typehoon.typeconsts*), 1277
float_len_mod (*angr.procedures.stubs.format_parser.ScanfFormatPlace attribute*), 648
float_spec (*angr.procedures.stubs.format_parser.ScanfFormatPlace attribute*), 648
float_type() (*in module angr.analyses.typehoon.typeconsts*), 1281
floating_point (*angr.ailment.BinaryOp attribute*), 1367
floating_point (*angr.ailment.expression.BinaryOp attribute*), 1385
FloatingPoint (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
FloatingPoint (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
floor_addr() (*angr.analyses.cfg.cfb.CFBlanket method*), 973
floor_addr() (*angr.analyses.cfg.CFBlanket method*), 958
floor_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 759
floor_addr() (*angr.knowledge_plugins.cfg.CFGModel method*), 745
floor_addr() (*angr.knowledge_plugins.FunctionManager method*), 727
floor_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionDict method*), 792
floor_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionDictBase method*), 790
floor_addr() (*angr.knowledge_plugins.functions.function_manager.Func method*), 797
floor_addr() (*angr.knowledge_plugins.functions.FunctionManager method*), 787
floor_func() (*angr.knowledge_plugins.FunctionManager method*), 727
floor_func() (*angr.knowledge_plugins.functions.function_manager.Func method*), 797
floor_func() (*angr.knowledge_plugins.functions.FunctionManager method*), 787
floor_item() (*angr.analyses.cfg.cfb.CFBlanket method*), 973
floor_item() (*angr.analyses.cfg.CFBlanket method*), 958
floor_items() (*angr.analyses.cfg.cfb.CFBlanket method*), 973
floor_items() (*angr.analyses.cfg.CFBlanket method*), 958
flush_pages() (*angr.storage.memory_mixins.paged_memory.paged_mem method*), 506
flush_pages() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 476
fmt (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 1142
fmt_char (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 1142
fmt_double (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 1142
fmt_float (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 1142
fmt_hex (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 1142
fmt_int() (*angr.analyses.decompiler.structured_codegen.c.CConstant method*), 1142
fmt_neg (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 1142
force_unroll_loops() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 979
force_unroll_loops() (*angr.analyses.cfg.CFGEmulated method*), 964
force_unroll_loops() (*angr.analyses.CFGEmulated method*), 887
forget_last_label() (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode method*), 1262
forgotten (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode attribute*), 1262
FORGOTTEN (*class in angr.analyses.typehoon.simple_solver*), 1261
FORK (*angr.state_plugins.inspect.EventType attribute*), 732

format() (*angr.analyses.analysis.AnalysisLogEntry method*), 928
FormatInfo (*class in angr.state_plugins.trace_additions*), 381
FormatInfoDontConstrain (*class in angr.state_plugins.trace_additions*), 382
FormatInfoIntToStr (*class in angr.state_plugins.trace_additions*), 382
FormatInfoStrToInt (*class in angr.state_plugins.trace_additions*), 382
FormatParser (*class in angr.procedures.stubs.format_parser*), 647
FormatSpecifier (*class in angr.procedures.stubs.format_parser*), 647
FormatString (*class in angr.procedures.stubs.format_parser*), 646
ForwardAnalysis (*class in angr.analyses*), 906
ForwardAnalysis (*class in angr.analyses.forward_analysis*), 931
ForwardAnalysis (*class in angr.analyses.forward_analysis.forward_analysis*), 935
ForwardAnalysisForClinic (*class in angr.analyses.forward_analysis.forward_analysis*), 936
ForwardAnalysisForDummies (*class in angr.analyses.forward_analysis.forward_analysis*), 936
found (*angr.sim_manager.SimulationManager attribute*), 545
found (*angr.SimulationManager attribute*), 215
FP_ARG_REGS (*angr.calling_conventions.SimCC attribute*), 663
FP_ARG_REGS (*angr.calling_conventions.SimCCAArch64 attribute*), 673
FP_ARG_REGS (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 674
FP_ARG_REGS (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 672
FP_ARG_REGS (*angr.calling_conventions.SimCCARM attribute*), 672
FP_ARG_REGS (*angr.calling_conventions.SimCCARMHF attribute*), 673
FP_ARG_REGS (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 673
FP_ARG_REGS (*angr.calling_conventions.SimCCCdecl attribute*), 668
FP_ARG_REGS (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 669
FP_ARG_REGS (*angr.calling_conventions.SimCCN64 attribute*), 676
FP_ARG_REGS (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 676
FP_ARG_REGS (*angr.calling_conventions.SimCCO32 attribute*), 675
FP_ARG_REGS (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 675
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPC attribute*), 676
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPC64 attribute*), 677
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 677
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 677
FP_ARG_REGS (*angr.calling_conventions.SimCCRISCV64 attribute*), 674
FP_ARG_REGS (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 675
FP_ARG_REGS (*angr.calling_conventions.SimCCS390X attribute*), 679
FP_ARG_REGS (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 679
FP_ARG_REGS (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 671
FP_ARG_REGS (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 670
FP_ARG_REGS (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 671
FP_ARG_REGS (*angr.engines.pcode.cc.SimCCM68k attribute*), 636
FP_ARG_REGS (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 637
FP_ARG_REGS (*angr.engines.pcode.cc.SimCCXtensa attribute*), 638
FP_ARG_REGS (*angr.SimCC attribute*), 174
fp_args (*angr.calling_conventions.SimCC property*), 664
fp_args (*angr.SimCC property*), 174
fp_iter (*angr.calling_conventions.ArgSession attribute*), 663
fp_iter (*angr.calling_conventions.MicrosoftAMD64ArgSession attribute*), 669
fp_iter (*angr.calling_conventions.SimCC.ArgSession attribute*), 664
fp_iter (*angr.SimCC.ArgSession attribute*), 175
fp_ret_expr (*angr.ailment.statement.SideEffectStatement attribute*), 1402
FP_RETURN_VAL (*angr.calling_conventions.SimCC attribute*), 664
FP_RETURN_VAL (*angr.calling_conventions.SimCCARMHF attribute*), 673
FP_RETURN_VAL (*angr.calling_conventions.SimCCCdecl attribute*), 668
FP_RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 670
FP_RETURN_VAL (*angr.calling_conventions.SimCCRISCV64 attribute*), 674

FP_RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 671
FP_RETURN_VAL (*angr.SimCC attribute*), 174
free() (*angr.analyses.reaching_definitions.heap_allocator.HeapAllocator method*), 1221
free() (*angr.SimHeapPTMalloc method*), 192
free() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 415
free() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 419
free() (*angr.state_plugins.heap.SimHeapLibc method*), 408
free() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 410
free() (*angr.state_plugins.SimHeapLibc method*), 273
free() (*angr.state_plugins.SimHeapPTMalloc method*), 274
free_chunks() (*angr.SimHeapPTMalloc method*), 192
free_chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 415
free_chunks() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 418
free_chunks() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 409
free_chunks() (*angr.state_plugins.SimHeapPTMalloc method*), 274
freeze() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 1245
fresh_block() (*angr.factory.AngrObjectFactory method*), 231
fresh_constraints (*angr.state_plugins.log.SimStateLog property*), 370
fresh_constraints (*angr.state_plugins.SimStateLog property*), 311
from_ail_expr() (*angr.analyses.reaching_definitions.Atom static method*), 1179
from_ail_expr() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 836
from_argument() (*angr.analyses.reaching_definitions.Atom static method*), 1179
from_argument() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 837
from_bits (*angr.ailment.expression.Convert attribute*), 1382
from_bits (*angr.ailment.expression.Reinterpret attribute*), 1383
from_cle_variable() (*angr.state_plugins.debug_variables.SimDebugVariable static method*), 421
from_cle_variable() (*angr.state_plugins.SimDebugVariable static method*), 266
from_codeloc() (*angr.code_location.AILCodeLocation static method*), 867
from_digraph() (*angr.annocfg.AnnotatedCFG method*), 1321
from_env() (*angr.LLMClient class method*), 166
from_json() (*angr.angrdb.serializers.structured_code.ConstFormatsSerial static method*), 1031
from_json() (*angr.knowledge_plugins.functions.function.FunctionInfo class method*), 800
from_json() (*angr.knowledge_plugins.functions.function_parser.CallingC static method*), 809
from_json() (*angr.procedures.definitions.SimLibrary static method*), 650
from_json() (*angr.procedures.definitions.SimTypeCollection class method*), 649
from_json() (*angr.sim_type.SimType static method*), 687
from_json() (*angr.sim_type.SimTypeRef static method*), 708
from_name() (*angr.knowledge_plugins.debug_variables.DebugVariableMa method*), 821
from_name() (*angr.knowledge_plugins.DebugVariableManager method*), 715
from_name_and_pc() (*angr.knowledge_plugins.debug_variables.DebugVa method*), 821
from_name_and_pc() (*angr.knowledge_plugins.DebugVariableManager method*), 715
from_pc() (*angr.knowledge_plugins.debug_variables.DebugVariable method*), 820
from_pc() (*angr.knowledge_plugins.debug_variables.DebugVariableCont method*), 820
from_signature (*angr.knowledge_plugins.Function attribute*), 717
from_signature (*angr.knowledge_plugins.functions.Function attribute*), 777
from_signature (*angr.knowledge_plugins.functions.function.Function attribute*), 802
from_signature (*angr.knowledge_plugins.functions.soot_function.SootFu attribute*), 810
from_state() (*angr.analyses.variable_recovery.annotations.VariableSour static method*), 1247
from_type (*angr.ailment.expression.Convert attribute*), 1383
from_type (*angr.ailment.expression.Reinterpret attribute*), 1383
FrozenStackPointerTrackerState (*class in angr.analyses.stack_pointer_tracker*), 1244
FS (*in module angr.flirt*), 1342
fstat() (*angr.state_plugins.posix.SimSystemPosix method*), 353
fstat() (*angr.state_plugins.SimSystemPosix method*), 323
fstat_with_result() (*angr.state_plugins.posix.SimSystemPosix method*), 353
fstat_with_result()

(*angr.state_plugins.SimSystemPosix method*), 323
full_c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 1132
full_graph (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 1071
full_init_state() (*angr.factory.AngrObjectFactory method*), 229
full_simplify() (*angr.state_hierarchy.StateHierarchy method*), 552
full_simplify() (*angr.StateHierarchy method*), 221
func_addr (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
func_addr (*angr.analyses.cfg.cfg_job_base.CFGJobBase property*), 994
func_addr (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase attribute*), 1107
func_addr (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationMultiStmtBase attribute*), 1106
func_addr (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase attribute*), 1105
func_addr (*angr.analyses.variable_recovery.engine_base.SimEngineVRBase property*), 1257
func_addr (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase property*), 1250
func_addr (*angr.angrdb.models.DbStructuredCode attribute*), 1024
func_addr (*angr.angrdb.models.DbVariableCollection attribute*), 1023
func_addr (*angr.knowledge_plugins.cfg.BlockID property*), 740
func_addr (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 771
func_addr (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 753
func_edge_type_from_pb() (*in module angr.utils.enums_conv*), 1346
func_edge_type_to_pb() (*in module angr.utils.enums_conv*), 1346
func_graph (*angr.analyses.reaching_definitions.subject.SubjectType property*), 1238
func_lookup() (*angr.analyses.Disassembly method*), 904
func_lookup() (*angr.analyses.disassembly.Disassembly method*), 1296
func_name (*angr.codenode.FuncNode attribute*), 1324
func_typevar (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 1065
FuncComment (*class in angr.analyses.disassembly*), 1296
FuncIn (*class in angr.analyses.typehoon.typevars*), 1271
FuncInfo (*class in angr.analyses.identifier.identify*), 1283
FuncNode (*class in angr.codenode*), 1324
FuncOut (*class in angr.analyses.typehoon.typevars*), 1271
funcs (*angr.angrdb.models.DbKnowledgeBase attribute*), 1022
Function (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 1309
function (*angr.analyses.reaching_definitions.function_handler.FunctionC attribute*), 1224
function (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
Function (*angr.analyses.reaching_definitions.subject.SubjectType attribute*), 1238
Function (*class in angr.analyses.typehoon.typeconsts*), 1280
Function (*class in angr.knowledge_plugins*), 716
Function (*class in angr.knowledge_plugins.functions*), 776
Function (*class in angr.knowledge_plugins.functions.function*), 800
function() (*angr.knowledge_plugins.FunctionManager method*), 727
function() (*angr.knowledge_plugins.functions.function_manager.FunctionManager*), 797
function() (*angr.knowledge_plugins.functions.FunctionManager method*), 788
function_address (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
function_address (*angr.knowledge_plugins.cfg.CFGNode property*), 750
function_address (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
function_address (*angr.state_plugins.InspectAttrs attribute*), 254
function_address (*angr.storage.memory_mixins.regioned_memory.regio attribute*), 533
function_block_count (*angr.knowledge_plugins.propagations.PropagationModel attribute*), 775
function_codeloc (*angr.analyses.reaching_definitions.function_handler. attribute*), 1223
function_codeloc (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
function_dependency_graph() (*angr.analyses.DDG method*), 877
function_dependency_graph() (*angr.analyses.ddg.DDG method*), 1160
function_final_states (*angr.analyses.VFG property*), 880
function_final_states (*angr.analyses.vfg.VFG property*), 1290
function_initial_states (*angr.analyses.VFG property*), 880
function_initial_states (*angr.analyses.vfg.VFG property*), 1290

function_managers (*angr.knowledge_plugins.variables.variable_manager.VariableManager attribute*), 819
function_name (*angr.analyses.reassembler.FunctionLabel property*), 1298
function_name_changed() (*angr.knowledge_plugins.FunctionManager method*), 725
function_name_changed() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 795
function_name_changed() (*angr.knowledge_plugins.functions.FunctionManager method*), 786
function_needs_variable_recovery() (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis static method*), 955
function_needs_variable_recovery() (*angr.analyses.CompleteCallingConventionsAnalysis static method*), 898
FUNCTION_PROLOGUE (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 987
function_prototype() (*angr.factory.AngrObjectFactory method*), 231
function_summary (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 1065
FunctionAnalysis (*class in angr.analyses.vfg*), 1286
FunctionCall (*angr.analyses.proximity_graph.ProxiNodeType attribute*), 1309
FunctionCallData (*class in angr.analyses.reaching_definitions*), 1182
FunctionCallData (*class in angr.analyses.reaching_definitions.function_handler*), 1222
FunctionCallDataUnwrapped (*class in angr.analyses.reaching_definitions.function_handler*), 1226
FunctionCallEdge (*class in angr.analyses.cfg.cfg_fast*), 986
FunctionCallRelationships (*class in angr.analyses.reaching_definitions*), 1185
FunctionCallRelationships (*class in angr.analyses.reaching_definitions.function_handler*), 1226
FunctionDict (*class in angr.knowledge_plugins.functions.function_manager*), 791
FunctionDictBase (*class in angr.knowledge_plugins.functions.function_manager*), 790
FunctionDiff (*class in angr.analyses.bindiff*), 948
FunctionEdge (*class in angr.analyses.cfg.cfg_fast*), 985
FunctionEffect (*class in angr.analyses.reaching_definitions.function_handler*), 1221
FunctionFakeRetEdge (*class in angr.analyses.cfg.cfg_fast*), 986
FunctionGraphVisitor (*class in angr.analyses.forward_analysis*), 932
FunctionGraphVisitor (*class in angr.analyses.forward_analysis.visitors*), 938
FunctionGraphVisitor (*class in angr.analyses.forward_analysis.visitors.function_graph*), 941
FunctionHandler (*class in angr.analyses.reaching_definitions*), 1186
FunctionHandler (*class in angr.analyses.reaching_definitions.function_handler*), 1227
FunctionInfo (*class in angr.knowledge_plugins.functions.function*), 800
FunctionKey (*class in angr.analyses.cfg.cfg_job_base*), 993
FunctionLabel (*class in angr.analyses.reassembler*), 1298
FunctionLikeMacro (*class in angr.ailment.expression*), 1296
FunctionManager (*class in angr.knowledge_plugins*), 724
FunctionManager (*class in angr.knowledge_plugins.functions*), 785
FunctionManager (*class in angr.knowledge_plugins.functions.function_manager*), 794
FunctionManagerSerializer (*class in angr.angrdb.serializers.funcs*), 1027
FunctionParser (*class in angr.knowledge_plugins.functions.function_parser*), 809
FunctionProxiNode (*class in angr.analyses.proximity_graph*), 1309
FunctionReturn (*class in angr.analyses.cfg.cfg_fast*), 984
FunctionReturnEdge (*class in angr.analyses.cfg.cfg_fast*), 986
functions (*angr.analyses.cfg.cfg_base.CFGBase property*), 982
functions (*angr.analyses.cfg.CFGBase property*), 961
functions (*angr.knowledge_base.KnowledgeBase attribute*), 711
functions (*angr.KnowledgeBase attribute*), 164
functions_probably_identical() (*angr.analyses.BinDiff method*), 883
functions_probably_identical() (*angr.analyses.bindiff.BinDiff method*), 949
functions_reachable()

(*angr.knowledge_plugins.Function method*), 724
functions_reachable() (*angr.knowledge_plugins.functions.Function method*), 784
functions_reachable() (*angr.knowledge_plugins.functions.function.Function method*), 809
FunctionStart (*class in angr.analyses.disassembly*), 1293
FunctionTag (*class in angr.knowledge_plugins.key_definitions.tag*), 857
FunctionTransitionEdge (*class in angr.analyses.cfg.cfg_fast*), 986
functy (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 1132
fwd_chunk() (*angr.PTChunk method*), 168
fwd_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 414
fwd_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 417
fwd_chunk() (*angr.state_plugins.heap.PTChunk method*), 405
fwd_chunk() (*angr.state_plugins.PTChunk method*), 258

G

g_label_ctr (*angr.analyses.reassembler.Label attribute*), 1297
GDB (*class in angr.state_plugins*), 247
GDB (*class in angr.state_plugins.gdb*), 378
generate_gdt() (*angr.SimOS method*), 197
generate_gdt() (*angr.simos.SimOS method*), 1330
generate_gdt() (*angr.simos.simos.SimOS method*), 1335
generate_index() (*angr.analyses.cfg.cfg_base.CFGBase method*), 982
generate_index() (*angr.analyses.cfg.CFGBase method*), 961
generate_symbolic_cmd_line_arg() (*angr.simos.javavm.SimJavaVM static method*), 1340
generate_symbolic_cmd_line_arg() (*angr.simos.SimJavaVM static method*), 1325
generic_compare() (*angr.engines.pcode.behavior.OpBehavior static method*), 617
generic_info_hook() (*in module angr.state_plugins.trace_additions*), 382
generic_name_from_vex_op() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
Get() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
get() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 1245
get() (*angr.knowledge_plugins.functions.function_manager.FunctionDict method*), 792
get() (*angr.knowledge_plugins.functions.function_manager.FunctionDictB method*), 790
get() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctio method*), 793
get() (*angr.knowledge_plugins.key_definitions.environment.Environment method*), 844
get() (*angr.knowledge_plugins.Labels method*), 731
get() (*angr.knowledge_plugins.labels.Labels method*), 776
get() (*angr.knowledge_plugins.PropagationManager method*), 733
get() (*angr.knowledge_plugins.propagations.PropagationManager method*), 773
get() (*angr.knowledge_plugins.structured_code.StructuredCodeManager method*), 822
get() (*angr.knowledge_plugins.StructuredCodeManager method*), 734
get() (*angr.procedures.definitions.SimCppLibrary method*), 653
get() (*angr.procedures.definitions.SimLibrary method*), 652
get() (*angr.procedures.definitions.SimSyscallLibrary method*), 656
get() (*angr.procedures.definitions.SimTypeCollection method*), 649
get() (*angr.SimMount method*), 194
get() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 358
get() (*angr.state_plugins.filesystem.SimFilesystem method*), 357
get() (*angr.state_plugins.filesystem.SimMount method*), 358
get() (*angr.state_plugins.globals.SimStateGlobals method*), 387
get() (*angr.state_plugins.posix.PosixDevFS method*), 347
get() (*angr.state_plugins.posix.PosixProcFS method*), 349
get() (*angr.state_plugins.PosixDevFS method*), 259
get() (*angr.state_plugins.PosixProcFS method*), 261
get() (*angr.state_plugins.SimFilesystem method*), 269
get() (*angr.state_plugins.SimMount method*), 285
get() (*angr.state_plugins.SimRegNameView method*), 287
get() (*angr.state_plugins.SimStateGlobals method*), 300
get() (*angr.state_plugins.view.SimRegNameView method*), 427
get() (*angr.utils.cowdict.ChainMapCOW method*), 1345

get() (*angr.utils.mp.Initializer class method*), 1354
get_abstract_locations() (*angr.storage.memory_mixins.MemoryRegionMetaMixin method*), 472
get_abstract_locations() (*angr.storage.memory_mixins.regioned_memory.MemoryRegionMetaMixin method*), 525
get_abstract_locations() (*angr.storage.memory_mixins.regioned_memory.region_meta.RegionMetaMixin method*), 537
get_addr() (*angr.annocfg.AnnotatedCFG method*), 1321
get_addr_of_native_method() (*angr.simos.javavm.SimJavaVM method*), 1341
get_addr_of_native_method() (*angr.simos.SimJavaVM method*), 1326
get_addrs_by_name() (*angr.knowledge_plugins.FunctionManager method*), 726
get_addrs_by_name() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 796
get_addrs_by_name() (*angr.knowledge_plugins.functions.FunctionManager method*), 787
get_all_definitions() (*in module angr.analyses.reaching_definitions*), 1212
get_all_nodes() (*angr.analyses.VFG method*), 880
get_all_nodes() (*angr.analyses.vfg.VFG method*), 1290
get_all_nodes() (*angr.analyses.VSA_DDG method*), 880
get_all_nodes() (*angr.analyses.vsa_ddg.VSA_DDG method*), 1291
get_all_nodes() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 759
get_all_nodes() (*angr.knowledge_plugins.cfg.CFGModel method*), 745
get_all_nodes_intersecting_region() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 759
get_all_nodes_intersecting_region() (*angr.knowledge_plugins.cfg.CFGModel method*), 745
get_all_patches() (*angr.knowledge_plugins.patches.PatchManager method*), 738
get_all_patches() (*angr.knowledge_plugins.PatchManager method*), 732
get_all_predecessors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 761
get_all_predecessors() (*angr.knowledge_plugins.cfg.CFGModel method*), 747
get_all_successors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 762
get_all_successors() (*angr.knowledge_plugins.cfg.CFGModel method*), 748
get_all_variables() (*angr.storage.memory_mixins.KeyedRegionMetaMixin method*), 870
get_alloc_depth() (*angr.state_plugins.SimUCManager method*), 325
get_alloc_depth() (*angr.state_plugins.uc_manager.SimUCManager method*), 387
get_any_node() (*angr.analyses.VFG method*), 880
get_any_node() (*angr.analyses.vfg.VFG method*), 1290
get_any_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 758
get_any_node() (*angr.knowledge_plugins.cfg.CFGModel method*), 744
get_arg_info() (*angr.calling_conventions.SimCC method*), 667
get_arg_info() (*angr.SimCC method*), 177
get_args() (*angr.calling_conventions.SimCC method*), 665
get_args() (*angr.SimCC method*), 176
get_ast_subexprs() (*in module angr.analyses.decompiler.utils*), 1151
get_base_addr() (*angr.keyed_region.KeyedRegion method*), 870
get_basic_info() (*in module angr.flirt.build_sig*), 1343
get_behavior_for_opcode() (*angr.engines.pcode.behavior.BehaviorFactory method*), 636
get_binary_name_cached() (*angr.knowledge_plugins.FunctionManager method*), 725
get_binary_name_cached() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 795
get_binary_name_cached() (*angr.knowledge_plugins.functions.FunctionManager method*), 786
get_bits() (*in module angr.ailment.utils*), 1408
get_block() (*angr.knowledge_plugins.Function method*), 718
get_block() (*angr.knowledge_plugins.functions.Function method*), 779
get_block() (*angr.knowledge_plugins.functions.function.Function method*), 803
get_block_size() (*angr.knowledge_plugins.Function method*), 719
get_block_size() (*angr.knowledge_plugins.functions.Function*

method), 779
get_block_size() (angr.knowledge_plugins.functions.function.Function method), 803
get_branching_nodes() (angr.knowledge_plugins.cfg.cfg_model.CFGModel method), 762
get_branching_nodes() (angr.knowledge_plugins.cfg.CFGModel method), 748
get_by_addr() (angr.knowledge_plugins.FunctionManager method), 726
get_by_addr() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 796
get_by_addr() (angr.knowledge_plugins.functions.FunctionManager method), 786
get_by_name() (angr.knowledge_plugins.FunctionManager method), 726
get_by_name() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 796
get_by_name() (angr.knowledge_plugins.functions.FunctionManager method), 786
get_byte() (angr.state_plugins.trace_additions.ChallRespInfo static method), 384
get_call_args() (angr.analyses.Identifier method), 907
get_call_args() (angr.analyses.identifier.identify.Identifier method), 1283
get_call_return() (angr.knowledge_plugins.Function method), 721
get_call_return() (angr.knowledge_plugins.functions.Function method), 782
get_call_return() (angr.knowledge_plugins.functions.function.Function method), 806
get_call_sites() (angr.knowledge_plugins.Function method), 721
get_call_sites() (angr.knowledge_plugins.functions.Function method), 781
get_call_sites() (angr.knowledge_plugins.functions.function.Function method), 806
get_call_stack_suffix() (angr.analyses.cfg.cfg_job_base.CFGJobBase method), 993
get_call_target() (angr.knowledge_plugins.Function method), 721
get_call_target() (angr.knowledge_plugins.functions.Function method), 782
get_call_target() (angr.knowledge_plugins.functions.function.Function method), 806
get_cc() (angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method), 739
get_cc() (angr.knowledge_plugins.CallsitePrototypes method), 713
get_cfg_edge_cache_limit() (angr.Project method), 173
get_cfg_edge_cache_limit() (angr.project.Project method), 227
get_cfg_node_cache_limit() (angr.Project method), 173
get_cfg_node_cache_limit() (angr.project.Project method), 227
get_class() (angr.state_plugins.javavm_classloader.SimJavaVmClassloader method), 401
get_class() (angr.state_plugins.SimJavaVmClassloader method), 279
get_class_hierarchy() (angr.state_plugins.javavm_classloader.SimJavaVmClassloader method), 401
get_class_hierarchy() (angr.state_plugins.SimJavaVmClassloader method), 279
get_cmd_line_args() (angr.simos.javavm.SimJavaVM static method), 1341
get_cmd_line_args() (angr.simos.SimJavaVM static method), 1326
get_concrete_fd() (angr.state_plugins.posix.SimSystemPosix method), 353
get_concrete_fd() (angr.state_plugins.SimSystemPosix method), 323
get_concrete_state() (angr.analyses.variable_recovery.variable_recovery.VariableRecovery method), 1255
get_concrete_value() (angr.analyses.reaching_definitions.LiveDefinitions method), 1194
get_concrete_value() (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method), 1237
get_concrete_value() (angr.analyses.reaching_definitions.ReachingDefinitionsState method), 1210
get_concrete_value() (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method), 852
get_concrete_value() (angr.knowledge_plugins.key_definitions.LiveDefinitions method), 829
get_cpp_function_name() (in module angr.utils.library), 1353
get_cpp_function_name_and_metadata() (in module angr.utils.library), 1353
get_data_dep() (angr.analyses.data_dep.data_dependency_analysis.DataDependencyGraphAnalysis method), 1312
get_data_dep() (angr.analyses.data_dep.DataDependencyGraphAnalysis method), 1317
get_data_dep() (angr.analyses.DataDependencyGraphAnalysis method), 900
get_data_references() (angr.knowledge_plugins.cfg.cfg_node.CFGNode

method), 767
get_data_references() (angr.knowledge_plugins.cfg.CFGNode method), 751
get_data_size() (angr.PTChunk method), 167
get_data_size() (angr.state_plugins.heap.heap_freelist.Chunk method), 414
get_data_size() (angr.state_plugins.heap.heap_ptmalloc.PTChunk method), 416
get_data_size() (angr.state_plugins.heap.PTChunk method), 404
get_data_size() (angr.state_plugins.PTChunk method), 257
get_dbinfo() (angr.angrdb.AngrDB method), 1019
get_dbinfo() (angr.angrdb.db.AngrDB method), 1020
get_default_cache_limit() (angr.knowledge_plugins.FunctionManager method), 725
get_default_cache_limit() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 795
get_default_cache_limit() (angr.knowledge_plugins.functions.FunctionManager method), 785
get_default_value_by_type() (angr.simos.javavm.SimJavaVM static method), 1340
get_default_value_by_type() (angr.simos.SimJavaVM static method), 1325
get_definitions() (angr.analyses.reaching_definitions.LiveDefinitions method), 1193
get_definitions() (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method), 1236
get_definitions() (angr.analyses.reaching_definitions.ReachingDefinitionsState method), 1210
get_definitions() (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method), 851
get_definitions() (angr.knowledge_plugins.key_definitions.LiveDefinitions method), 828
get_defs() (angr.analyses.reaching_definitions.ReachingDefinitionsModel method), 1202
get_defs() (angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method), 856
get_defs() (angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method), 833
get_dependants() (angr.analyses.CDG method), 873
get_dependants() (angr.analyses.cdg.CDG method), 1017
get_element() (angr.analyses.decompiler.structured_codegen.base.PositionMapping method), 1128
get_element() (angr.analyses.decompiler.structured_codegen.PositionMapping method), 1125
get_exit_livedefinitions() (in module angr.analyses.reaching_definitions.function_handler), 1221
get_exit_stmt_idx() (angr.knowledge_plugins.cfg.cfg_model.CFGModel method), 762
get_exit_stmt_idx() (angr.knowledge_plugins.cfg.CFGModel method), 748
get_expr_depth() (angr.state_plugins.trace_additions.ZenPlugin method), 384
get_fd() (angr.state_plugins.posix.SimSystemPosix method), 353
get_fd() (angr.state_plugins.SimSystemPosix method), 323
get_field() (angr.ailment.expression.Struct method), 1395
get_flag_bytes() (angr.state_plugins.trace_additions.ZenPlugin method), 385
get_flag_rand_args() (angr.state_plugins.trace_additions.ZenPlugin static method), 384
get_footprint() (angr.calling_conventions.SimArrayArg method), 662
get_footprint() (angr.calling_conventions.SimComboArg method), 661
get_footprint() (angr.calling_conventions.SimFunctionArgument method), 660
get_footprint() (angr.calling_conventions.SimReferenceArgument method), 662
get_footprint() (angr.calling_conventions.SimRegArg method), 660
get_footprint() (angr.calling_conventions.SimStackArg method), 661
get_footprint() (angr.calling_conventions.SimStructArg method), 662
get_func_block_count() (angr.knowledge_plugins.functions.FunctionManager method), 729
get_func_block_count() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 798
get_func_block_count() (angr.knowledge_plugins.functions.FunctionManager method), 789
get_func_info() (angr.analyses.Identifier method), 907
get_func_info() (angr.analyses.identifier.identify.Identifier method), 1283
get_function_cache_limit() (angr.Project method), 73
get_function_cache_limit() (angr.project.Project method), 227
get_function_diff() (angr.analyses.BinDiff method), 884

get_function_diff() (*angr.analyses.bindiff.BinDiff method*), 950
get_function_manager() (*angr.knowledge_plugins.VariableManager method*), 735
get_function_manager() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 819
get_function_manager() (*angr.knowledge_plugins.variables.VariableManager method*), 811
get_function_name() (*in module angr.utils.library*), 1351
get_function_subgraph() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 980
get_function_subgraph() (*angr.analyses.cfg.CFGEmulated method*), 966
get_function_subgraph() (*angr.analyses.CFGEmulated method*), 888
get_global_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 816
get_guardians() (*angr.analyses.CDG method*), 873
get_guardians() (*angr.analyses.cdg.CDG method*), 1017
get_heap_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1194
get_heap_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 852
get_heap_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 828
get_heap_offset() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 1196
get_heap_offset() (*angr.analyses.reaching_definitions.rda_model.RDAMemoryAddressJob static method*), 1232
get_heap_offset() (*angr.analyses.reaching_definitions.RegionalizedDataAccessCollector static method*), 1205
get_heap_offset() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 854
get_heap_offset() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 832
get_implementers() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 955
get_implementers() (*angr.analyses.SootClassHierarchy method*), 922
get_info() (*angr.angrdb.AngrDB static method*), 1019
get_info() (*angr.angrdb.db.AngrDB static method*), 1020
get_intersecting_functions() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 763
get_intersecting_functions() (*angr.knowledge_plugins.cfg.CFGModel method*), 749
get_irsb_at() (*angr.analyses.datagraph_meta.DataGraphMeta method*), 1018
get_kb() (*angr.Project method*), 170
get_kb() (*angr.project.Project method*), 224
get_key_func_addrs() (*angr.knowledge_plugins.FunctionManager method*), 729
get_key_func_addrs() (*angr.knowledge_plugins.functions.function_manager.FunctionM method*), 799
get_key_func_addrs() (*angr.knowledge_plugins.functions.FunctionManager method*), 789
get_knowledge() (*angr.knowledge_base.KnowledgeBase method*), 712
get_knowledge() (*angr.KnowledgeBase method*), 165
get_last_statement() (*angr.analyses.decompiler.condition_processor.ConditionProcess class method*), 1063
get_last_statement_index() (*angr.annocfg.AnnotatedCFG method*), 1322
get_last_statements() (*angr.analyses.decompiler.condition_processor.ConditionProcess class method*), 1063
get_lifter() (*angr.engines.pcode.lifter.PcodeLifter class method*), 613
get_loop_back_edges() (*angr.analyses.cfg.cfg_base.CFGBase method*), 982
get_loop_back_edges() (*angr.analyses.cfg.CFGBase method*), 961
get_loops() (*angr.annocfg.AnnotatedCFG method*), 1322
get_make_symbolic() (*angr.state_plugins.cgc.SimStateCGC method*), 381
get_max_symbolic() (*angr.state_plugins.SimStateCGC method*), 298
get_memory_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1194
get_memory_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi method*), 852
get_memory_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 829
get_model() (*angr.knowledge_plugins.key_definitions.key_definition_man method*), 846
get_model() (*angr.knowledge_plugins.key_definitions.KeyDefinitionMana

method), 823
get_model() (angr.knowledge_plugins.KeyDefinitionManager method), 730
get_most_accurate() (angr.knowledge_plugins.cfg.cfg_manager.CFGManager method), 766
get_most_accurate() (angr.knowledge_plugins.cfg.CFGManager method), 742
get_most_accurate() (angr.knowledge_plugins.CFGManager method), 712
get_mountpoint() (angr.state_plugins.filesystem.SimFilesystem method), 357
get_mountpoint() (angr.state_plugins.SimFilesystem method), 270
get_native_cc() (angr.simos.javavm.SimJavaVM method), 1341
get_native_cc() (angr.simos.SimJavaVM method), 1326
get_native_type() (angr.simos.javavm.SimJavaVM method), 1341
get_native_type() (angr.simos.SimJavaVM method), 1326
get_nearest_pos() (angr.analyses.decompiler.structured_codegen.base.StructuredCodeGenerator method), 1129
get_nearest_pos() (angr.analyses.decompiler.structured_codegen.base.StructuredCodeWriter method), 1124
get_new_uuid() (angr.storage.memory_mixins.JavaVmMemoryMixin static method), 464
get_node() (angr.analyses.decompiler.structured_codegen.base.PositionMappingStructureMixin method), 1128
get_node() (angr.analyses.decompiler.structured_codegen.base.StructuredCodeWriter method), 1125
get_node() (angr.knowledge_plugins.cfg.cfg_model.CFGModel method), 758
get_node() (angr.knowledge_plugins.cfg.CFGModel method), 744
get_node() (angr.knowledge_plugins.Function method), 719
get_node() (angr.knowledge_plugins.functions.Function method), 779
get_node() (angr.knowledge_plugins.functions.function.Function method), 803
get_normalized_block() (angr.analyses.bindiff.FunctionDiff static method), 949
get_objects_by_offset() (angr.keyed_region.KeyedRegion method), 870
get_observation_by_exit() (angr.analyses.reaching_definitions.ReachingDefinitionsModel method), 1203
get_observation_by_exit() (angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method), 857
get_observation_by_exit() (angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method), 834
get_observation_by_insn() (angr.analyses.reaching_definitions.ReachingDefinitionsModel method), 1202
get_observation_by_insn() (angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method), 856
get_observation_by_insn() (angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method), 833
get_observation_by_node() (angr.analyses.reaching_definitions.ReachingDefinitionsModel method), 1203
get_observation_by_node() (angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method), 856
get_observation_by_node() (angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method), 834
get_observation_by_stmt() (angr.analyses.reaching_definitions.ReachingDefinitionsModel method), 1203
get_observation_by_stmt() (angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method), 857
get_observation_by_stmt() (angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method), 834
get_live_values() (angr.analyses.reaching_definitions.LiveDefinitions method), 1194
get_one_value() (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method), 1237
get_one_value() (angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method), 1210
get_one_value() (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method), 852
get_one_value() (angr.knowledge_plugins.key_definitions.LiveDefinitions method), 829
get_optimization_passes() (in module angr.analyses.decompiler.optimization_passes), 1091
get_own() (angr.knowledge_plugins.types.TypesStore method), 772
get_own() (angr.knowledge_plugins.TypesStore method), 735
get_patch() (angr.knowledge_plugins.patches.PatchManager method), 737
get_patch() (angr.knowledge_plugins.PatchManager method), 732
get_phi_subvariables()

(*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 817
get_phi_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 817
get_plugin() (*angr.knowledge_base.KnowledgeBase method*), 712
get_plugin() (*angr.KnowledgeBase method*), 165
get_plugin() (*angr.misc.plugins.PluginHub method*), 236
get_plugin() (*angr.sim_state.SimState method*), 240
get_plugin() (*angr.SimState method*), 209
get_possible_len() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 384
get_post_dominators() (*angr.analyses.CDG method*), 873
get_post_dominators() (*angr.analyses.cdg.CDG method*), 1017
get_predecessors() (*angr.analyses.DDG method*), 876
get_predecessors() (*angr.analyses.ddg.DDG method*), 1160
get_predecessors() (*angr.analyses.VSA_DDG method*), 880
get_predecessors() (*angr.analyses.vsa_ddg.VSA_DDG method*), 1291
get_predecessors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 760
get_predecessors() (*angr.knowledge_plugins.cfg.CFGModel method*), 746
get_predecessors_and_jumpkind() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 761
get_predecessors_and_jumpkind() (*angr.knowledge_plugins.cfg.CFGModel method*), 747
get_predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 761
get_predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.CFGModel method*), 747
get_prototype() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 739
get_prototype() (*angr.knowledge_plugins.CallsitePrototypes method*), 713
get_prototype() (*angr.procedures.definitions.SimCppLibrary method*), 654
get_prototype() (*angr.procedures.definitions.SimLibrary method*), 652
get_prototype() (*angr.procedures.definitions.SimSyscallLibrary method*), 657
get_reaching_definitions_by_insn() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 1216
get_reaching_definitions_by_insn() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 1200
get_reaching_definitions_by_insn() (*angr.analyses.ReachingDefinitionsAnalysis method*), 915
get_reaching_definitions_by_node() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 1216
get_reaching_definitions_by_node() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 1200
get_reaching_definitions_by_node() (*angr.analyses.ReachingDefinitionsAnalysis method*), 915
get_real_len() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 384
get_recent_bbl_addrs() (*angr.state_plugins.Unicorn method*), 328
get_recent_bbl_addrs() (*angr.state_plugins.unicorn_engine.Unicorn method*), 399
get_recent_n() (*angr.exploration_techniques.spiller.PickledStatesDb method*), 573
get_ref() (*angr.state_hierarchy.StateHierarchy method*), 552
get_ref() (*angr.StateHierarchy method*), 221
get_reg_name() (*angr.analyses.Identifier static method*), 907
get_reg_name() (*angr.analyses.identifier.identify.Identifier static method*), 1283
get_register_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1193
get_register_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi method*), 851
get_register_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 828
get_regs() (*angr.state_plugins.Unicorn method*), 329
get_regs() (*angr.state_plugins.unicorn_engine.Unicorn method*), 399
get_run() (*angr.annocfg.AnnotatedCFG method*), 1322
get_rust_function_name() (*in module angr.utils.library*), 1353
get_same_length_constraints() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 384
get_segment_register_name() (*angr.simos.linux.SimLinux method*), 1336
get_segment_register_name() (*angr.simos.SimLinux method*), 1328

get_segment_register_name() (*angr.simos.SimWindows method*), 1333
get_segment_register_name() (*angr.simos.windows.SimWindows method*), 1340
get_single_footprint() (*angr.calling_conventions.SimStructArg method*), 662
get_size() (*angr.PTChunk method*), 167
get_size() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 414
get_size() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 416
get_size() (*angr.state_plugins.heap.PTChunk method*), 404
get_size() (*angr.state_plugins.PTChunk method*), 257
get_sp() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1191
get_sp() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1233
get_sp() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1207
get_sp() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 850
get_sp() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 826
get_sp_offset() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1192
get_sp_offset() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 850
get_sp_offset() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 826
get_stack_address() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1192
get_stack_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1233
get_stack_address() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1207
get_stack_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 850
get_stack_address() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 826
get_stack_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1193
get_stack_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 852
get_stack_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 828
get_stack_offset() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 1191
get_stack_offset() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1232
get_stack_offset() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1206
get_stack_offset() (*angr.analyses.variable_recovery.variable_recovery.VariableRecoveryState method*), 1250
get_stack_offset() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 849
get_stack_offset() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 826
get_stack_values() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1193
get_stack_values() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 851
get_stack_values() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 828
get_stackvar_max_sizes() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 819
get_stdin_indices() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 384
get_stdout_indices() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 389
get_stop_details() (*angr.state_plugins.Unicorn method*), 328
get_stop_details() (*angr.state_plugins.unicorn_engine.Unicorn method*), 399
get_stop_msg() (*angr.state_plugins.unicorn_engine.STOP static method*), 395
get_structurer_option() (*in module angr.analyses.decompiler.decompilation_options*), 1064
get_stub() (*angr.procedures.definitions.SimCppLibrary method*), 653
get_stub() (*angr.procedures.definitions.SimLibrary method*), 652
get_stub() (*angr.procedures.definitions.SimSyscallLibrary method*), 656
get_sub_classes() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 956
get_sub_classes() (*angr.analyses.SootClassHierarchy method*), 922
get_sub_classes_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 956
get_sub_classes_including() (*angr.analyses.SootClassHierarchy method*), 922

get_sub_interfaces() (angr.analyses.soot_class_hierarchy.SootClassHierarchy method), 955
get_sub_interfaces() (angr.analyses.SootClassHierarchy method), 922
get_sub_interfaces_including() (angr.analyses.soot_class_hierarchy.SootClassHierarchy method), 955
get_sub_interfaces_including() (angr.analyses.SootClassHierarchy method), 922
get_subgraph() (angr.analyses.cfg.cfg_emulated.CFGEmulated method), 980
get_subgraph() (angr.analyses.cfg.CFGEmulated method), 965
get_subgraph() (angr.analyses.CFGEmulated method), 888
get_successors() (angr.knowledge_plugins.cfg.cfg_model.CFGModel method), 760
get_successors() (angr.knowledge_plugins.cfg.CFGModel method), 746
get_successors_and_jumpkind() (angr.knowledge_plugins.cfg.cfg_model.CFGModel method), 761
get_successors_and_jumpkind() (angr.knowledge_plugins.cfg.CFGModel method), 747
get_successors_and_jumpkinds() (angr.knowledge_plugins.cfg.cfg_model.CFGModel method), 760
get_successors_and_jumpkinds() (angr.knowledge_plugins.cfg.CFGModel method), 746
get_super_classes() (angr.analyses.soot_class_hierarchy.SootClassHierarchy method), 955
get_super_classes() (angr.analyses.SootClassHierarchy method), 922
get_super_classes_including() (angr.analyses.soot_class_hierarchy.SootClassHierarchy method), 955
get_super_classes_including() (angr.analyses.SootClassHierarchy method), 922
get_superclass() (angr.state_plugins.javavm_classloader.SimJavaVmClassloader method), 401
get_superclass() (angr.state_plugins.SimJavaVmClassloader method), 279
get_symbolic_addrs() (angr.storage.memory_mixins.convenient_mappings_mixins.ConvenientMappingsMixin method), 499
get_symbolic_addrs() (angr.storage.memory_mixins.ConvenientMappingsMixin method), 459
get_targets() (angr.annocfg.AnnotatedCFG method), 1322
get_tmp_definitions() (angr.analyses.reaching_definitions.LiveDefinitions method), 1193
get_tmp_definitions() (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi method), 851
get_tmp_definitions() (angr.knowledge_plugins.key_definitions.LiveDefinitions method), 828
get_topological_order() (angr.analyses.cfg.cfg_emulated.CFGEmulated method), 980
get_topological_order() (angr.analyses.cfg.CFGEmulated method), 965
get_topological_order() (angr.analyses.CFGEmulated method), 888
get_tv_max_stack_size() (angr.analyses.typehoon.simple_solver.SimpleSolver method), 1263
get_type() (angr.state_plugins.trace_additions.FormatInfo method), 382
get_type() (angr.state_plugins.trace_additions.FormatInfoDontConstrain method), 382
get_type() (angr.state_plugins.trace_additions.FormatInfoIntToStr method), 382
get_type() (angr.state_plugins.trace_additions.FormatInfoStrToInt method), 382
get_type_variable() (angr.analyses.typehoon.typevars.TypeVariables method), 1271
get_unambiguous_name() (angr.knowledge_plugins.Function method), 724
get_unambiguous_name() (angr.knowledge_plugins.functions.Function method), 784
get_unambiguous_name() (angr.knowledge_plugins.functions.function.Function method), 808
get_unconstrained_simprocedure() (angr.engines.soot.engine.SootMixin method), 592
get_unconstrained_simprocedure() (angr.engines.soot.SootMixin method), 602
get_unconstrained_simprocedure() (angr.engines.SootMixin method), 592
get_unified_local_vars() (angr.analyses.decompiler.structured_codegen.c.CFunction method), 1132

get_unified_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 816
get_unique_label() (*angr.knowledge_plugins.Labels method*), 731
get_unique_label() (*angr.knowledge_plugins.labels.Labels method*), 776
get_unique_strings() (*in module angr.flirt.build_sig*), 1343
get_unique_symbol_name() (*angr.analyses.reassembler.SymbolManager method*), 1298
get_uses() (*angr.knowledge_plugins.key_definitions.Uses method*), 834
get_uses() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 859
get_uses_by_insaddr() (*angr.knowledge_plugins.key_definitions.Uses method*), 835
get_uses_by_insaddr() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 860
get_uses_by_location() (*angr.knowledge_plugins.key_definitions.Uses method*), 835
get_uses_by_location() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 860
get_uses_with_expr() (*angr.knowledge_plugins.key_definitions.Uses method*), 835
get_uses_with_expr() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 859
get_value() (*angr.calling_conventions.SimArrayArg method*), 662
get_value() (*angr.calling_conventions.SimComboArg method*), 661
get_value() (*angr.calling_conventions.SimFunctionArgument method*), 659
get_value() (*angr.calling_conventions.SimLyingRegArg method*), 667
get_value() (*angr.calling_conventions.SimReferenceArgument method*), 663
get_value() (*angr.calling_conventions.SimRegArg method*), 660
get_value() (*angr.calling_conventions.SimStackArg method*), 661
get_value() (*angr.calling_conventions.SimStructArg method*), 662
get_values() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1194
get_values() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1236
get_values() (*angr.analyses.reaching_definitions.ReachingDefinitionsStateMixin method*), 1210
get_values() (*angr.knowledge_plugins.internal method*), 852
get_values() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 829
get_values() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 1101
get_variable() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifyingEngineBase method*), 422
get_variable() (*angr.state_plugins.debug_variables.SimDebugVariablePlugin method*), 267
get_variable() (*angr.state_plugins.SimDebugVariablePlugin method*), 736
get_variable_accesses() (*angr.knowledge_plugins.VariableManager method*), 819
get_variable_accesses() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 815
get_variable_accesses() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 811
get_variable_accesses() (*angr.knowledge_plugins.variables.VariableManager method*), 1248
get_variable_definitions() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase method*), 1250
get_variable_definitions() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase method*), 818
get_variable_type() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 816
get_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 288
get_variables() (*angr.state_plugins.SimSolver method*), 361
get_variables() (*angr.state_plugins.solver.SimSolver method*), 870
get_variables_by_offset() (*angr.keyed_region.KeyedRegion method*), 817
get_variables_without_writes() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1279
get_variant() (*angr.analyses.typehoon.typeconsts.RustEnum method*), 1322
get_whitelisted_statements() (*angr.annocfg.AnnotatedCFG method*), 736
get_xrefs_by_dst() (*angr.knowledge_plugins.XRefManager method*), 865
get_xrefs_by_dst() (*angr.knowledge_plugins.xrefs.xref_manager.XRefMixin method*), 1101
get_xrefs_by_dst() (*angr.knowledge_plugins.xrefs.XRefManager*

method), 862
get_xrefs_by_dst_region() (angr.knowledge_plugins.XRefManager method), 736
get_xrefs_by_dst_region() (angr.knowledge_plugins.xrefs.xref_manager.XRefManager method), 865
get_xrefs_by_dst_region() (angr.knowledge_plugins.xrefs.XRefManager method), 862
get_xrefs_by_ins_addr() (angr.knowledge_plugins.XRefManager method), 736
get_xrefs_by_ins_addr() (angr.knowledge_plugins.xrefs.xref_manager.XRefManager method), 865
get_xrefs_by_ins_addr() (angr.knowledge_plugins.xrefs.XRefManager method), 862
get_xrefs_by_ins_addr_region() (angr.knowledge_plugins.XRefManager method), 736
get_xrefs_by_ins_addr_region() (angr.knowledge_plugins.xrefs.xref_manager.XRefManager method), 865
get_xrefs_by_ins_addr_region() (angr.knowledge_plugins.xrefs.XRefManager method), 862
get_xrefs_in_range() (angr.knowledge_plugins.xrefs.xref_manager.XRefManager method), 864
GetBitsTypeParams (in module angr.ailment.utils), 1408
getpiece() (angr.analyses.disassembly.DisassemblyPiece method), 1293
getstate() (angr.calling_conventions.ArgSession method), 663
getstate() (angr.calling_conventions.SerializableCounter method), 659
getstate() (angr.calling_conventions.SerializableIterator method), 659
getstate() (angr.calling_conventions.SerializableListIterator method), 659
getstate() (angr.calling_conventions.SimCC.ArgSession method), 664
getstate() (angr.calling_conventions.UsercallArgSession method), 663
getstate() (angr.SimCC.ArgSession method), 175
give_up_on_memory_tracking() (angr.analyses.stack_pointer_tracker.StackPointerTracker method), 1244
global_addr (angr.knowledge_plugins.key_definitions.definitions.Atom attribute), 842
global_region (angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase attribute), 1249
GlobalDescriptorTable (class in angr.simos.simos), 1335
GotoSimplifier (class in angr.analyses.decompiler.region_simplifiers.goto), 1114
GOTPLTEntry (angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute), 764
GOTPLTEntry (angr.knowledge_plugins.cfg.MemoryDataSort attribute), 755
gp (angr.analyses.cfg.cfg_fast.CFGJob attribute), 988
gp_register_read_hook() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.MIPSGPHeuristic method), 1002
gp_register_write_hook() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.MIPSGPHeuristic method), 1002
graph (angr.analyses.CDG property), 873
graph (angr.analyses.cdg.CDG property), 1017
graph (angr.analyses.cfg.cfg_base.CFGBase property), 982
graph (angr.analyses.cfg.cfg_emulated.CFGEmulated property), 980
graph (angr.analyses.cfg.cfg_fast.CFGFast property), 992
graph (angr.analyses.cfg.CFGBase property), 961
graph (angr.analyses.cfg.CFGEmulated property), 966
graph (angr.analyses.cfg.CFGFast property), 969
graph (angr.analyses.CFGEmulated property), 888
graph (angr.analyses.CFGFast property), 892
graph (angr.analyses.data_dep.data_dependency_analysis.DataDependencyGraphAnalysis property), 1312
graph (angr.analyses.data_dep.DataDependencyGraphAnalysis property), 1317
graph (angr.analyses.DataDependencyGraphAnalysis property), 900
graph (angr.analyses.DDG property), 876
graph (angr.analyses.ddg.DDG property), 1159
graph (angr.analyses.decompiler.graph_region.GraphRegion attribute), 1071
graph (angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis property), 936
graph (angr.analyses.forward_analysis.ForwardAnalysis property), 932
graph (angr.analyses.ForwardAnalysis property), 907
graph (angr.analyses.proximity_graph.ProximityGraphAnalysis attribute), 1311
graph (angr.analyses.reaching_definitions.dep_graph.DepGraph property), 1217
graph (angr.analyses.typehoon.simple_solver.Sketch attribute), 1260
graph (angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute), 756
graph (angr.knowledge_plugins.cfg.CFGModel attribute), 1249

tribute), 742
graph (*angr.knowledge_plugins.Function property*), 722
graph (*angr.knowledge_plugins.functions.Function property*), 782
graph (*angr.knowledge_plugins.functions.function.Function property*), 806
graph_ex() (*angr.knowledge_plugins.Function method*), 722
graph_ex() (*angr.knowledge_plugins.functions.Function method*), 782
graph_ex() (*angr.knowledge_plugins.functions.function.Function method*), 806
graph_visitor (*angr.knowledge_plugins.propagations.PropagationModel attribute*), 775
graph_with_successors (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 1071
GraphChangedNotification, 1037
GraphDephication (*class in angr.analyses.decompiler*), 1049
GraphEdgeFilter (*class in angr.analyses.decompiler.structuring.phoenix*), 1037
GraphRegion (*class in angr.analyses.decompiler.graph_region*), 1070
GraphUtils (*class in angr.utils.graph*), 1349
GraphVisitor (*class in angr.analyses.forward_analysis.visitors.graph*), 942
GT (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.CmpOp attribute*), 1115
guard (*angr.ailment.expression.DirtyExpression attribute*), 1388
guard (*angr.ailment.expression.Load attribute*), 1386
guard (*angr.ailment.statement.Store attribute*), 1399
GUARD (*angr.analyses.reaching_definitions.AtomKind attribute*), 1181
guard (*angr.errors.SimError attribute*), 1358
GUARD (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 836
guard (*angr.SimError attribute*), 178
GuardUse (*class in angr.analyses.reaching_definitions*), 1188
GuardUse (*class in angr.knowledge_plugins.key_definitions.atoms*), 838
guess_prototype() (*angr.calling_conventions.SimCC static method*), 665
guess_prototype() (*angr.calling_conventions.SimCCSoot static method*), 678
guess_prototype() (*angr.SimCC static method*), 176
guess_value_type() (*in module angr.analyses.decompiler.structured_codegen.c*), 1130
GUESSED (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 799
guessed_cc (*angr.analyses.reaching_definitions.function_handler.Function attribute*), 1224
guessed_cc (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1184
guessed_prototype (*angr.analyses.reaching_definitions.function_handler attribute*), 1224
guessed_prototype (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1184

H

handle() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCod method*), 1147
handle() (*angr.analyses.decompiler.structured_codegen.CStructuredCode method*), 1123
handle_CAssignment() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCod method*), 1147
handle_CAssignment() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecastsI method*), 1148
handle_CAssignment() (*angr.analyses.decompiler.structured_codegen.c.PointerArithmeti method*), 1149
handle_CAssignment() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeV method*), 1123
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.c.CS method*), 1148
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.c.Ma method*), 1148
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.c.Po method*), 1149
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.CStr method*), 1123
handle_CDoWhileLoop() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCod method*), 1147
handle_CDoWhileLoop() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeV method*), 1123
handle_CExpressionStatement() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCod method*), 1147
handle_CExpressionStatement() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeV method*), 1123
handle_CForLoop() (*angr.analyses.decompiler.structured_codegen.c.CSt method*), 1147
handle_CForLoop() (*angr.analyses.decompiler.structured_codegen.CStru method*), 1123
handle_CFunction() (*angr.analyses.decompiler.structured_codegen.c.CS method*), 1147

handle_CFunction() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1148
handle_CFunction() (*angr.analyses.decompiler.structured_codegen.c.MakeTypes.MakeTypesWalker method*), 1123
handle_CFunctionCall() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1147
handle_CFunctionCall() (*angr.analyses.decompiler.structured_codegen.c.MakeTypes.MakeTypesWalker method*), 1148
handle_CFunctionCall() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1123
handle_CGoto() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1148
handle_CGoto() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_CIfBreak() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1147
handle_CIfBreak() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_CIfElse() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1147
handle_CIfElse() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_CIndexedVariable() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1148
handle_CIndexedVariable() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_CITE() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1148
handle_CITE() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1124
handle_CReturn() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1148
handle_CReturn() (*angr.analyses.decompiler.structured_codegen.c.MakeTypes.MakeTypesWalker method*), 1148
handle_CReturn() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_CStatements() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1147
handle_CStatements() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_CSwitchCase() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1147
handle_CSwitchCase() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1148
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.MakeTypes.MakeTypesWalker method*), 1148
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1123
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.MakeTypes.MakeTypesWalker method*), 1148
handle_CUnaryOp() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1148
handle_CUnaryOp() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_CVariableField() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1148
handle_CVariableField() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_CWhileLoop() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1147
handle_CWhileLoop() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_default() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 1147
handle_default() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeWalker method*), 1123
handle_exception() (*angr.SimOS method*), 196
handle_exception() (*angr.simos.SimOS method*), 1319
handle_exception() (*angr.simos.simos.SimOS method*), 1324
handle_exception() (*angr.simos.SimWindows method*), 1332
handle_exception() (*angr.simos.windows.SimWindows method*), 1339
handle_external_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 1228
handle_external_function() (*angr.analyses.reaching_definitions.FunctionHandler method*), 1187
handle_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 1227
handle_function() (*angr.analyses.reaching_definitions.FunctionHandler method*), 1187
handle_generic_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 1228
handle_generic_function() (*angr.analyses.reaching_definitions.FunctionHandler method*), 1187
handle_indirect_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 1228
handle_indirect_function() (*angr.analyses.reaching_definitions.FunctionHandler method*), 1187

(*angr.analyses.reaching_definitions.FunctionHandler method*), 1187
handle_local_function() (*angr.analyses.find_objects_static.NewFunctionHandler method*), 1292
handle_local_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 1228
handle_local_function() (*angr.analyses.reaching_definitions.FunctionHandler method*), 1187
handle_pcode_block() (*angr.engines.pcode.emulate.PcodeEmulatorMixin method*), 615
handle_vex_block() (*angr.engines.SimInspectMixin method*), 591
handle_vex_block() (*angr.engines.SuperFastpathMixin method*), 594
handle_vex_block() (*angr.engines.TrackActionsMixin method*), 594
handle_vex_block() (*angr.engines.vex.SimInspectMixin method*), 600
handle_vex_block() (*angr.engines.vex.SuperFastpathMixin method*), 600
handle_vex_block() (*angr.engines.vex.TrackActionsMixin method*), 600
handle_vex_block() (*angr.engines.vex.VEXMixin method*), 601
handle_vex_block() (*angr.engines.vex.VEXSlicingMixin method*), 601
hardcopy (*angr.state_plugins.history.TreeIter property*), 378
has_addr_dups() (*in module angr.analyses.decompiler.utils*), 1155
has_atom() (*angr.ailment.BinaryOp method*), 1367
has_atom() (*angr.ailment.Expression method*), 1369
has_atom() (*angr.ailment.expression.BinaryOp method*), 1385
has_atom() (*angr.ailment.expression.Call method*), 1394
has_atom() (*angr.ailment.expression.Expression method*), 1376
has_atom() (*angr.ailment.expression.ITE method*), 1387
has_atom() (*angr.ailment.expression.Load method*), 1386
has_atom() (*angr.ailment.expression.UnaryOp method*), 1382
has_atom() (*angr.ailment.UnaryOp method*), 1373
HAS_BITSHIFTS (*angr.analyses.code_tagging.CodeTags attribute*), 1018
has_bitshifts() (*angr.analyses.code_tagging.CodeTagging method*), 1018
has_bitshifts() (*angr.analyses.CodeTagging method*), 897
has_calls (*angr.analyses.decompiler.region_simplifiers.expr_folding.Expr attribute*), 1112
has_clobbered() (*angr.analyses.reaching_definitions.function_handler.F method*), 1224
has_clobbered() (*angr.analyses.reaching_definitions.FunctionCallData method*), 1184
has_default_value (*angr.sim_state_options.StateOption property*), 243
has_function_manager() (*angr.knowledge_plugins.VariableManager method*), 735
has_function_manager() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 819
has_function_manager() (*angr.knowledge_plugins.variables.VariableManager method*), 811
has_implementation() (*angr.procedures.definitions.SimCppLibrary method*), 654
has_implementation() (*angr.procedures.definitions.SimLibrary method*), 652
has_implementation() (*angr.procedures.definitions.SimSyscallLibrary method*), 657
has_job() (*angr.analyses.forward_analysis.forward_analysis.ForwardAna method*), 936
has_job() (*angr.analyses.forward_analysis.ForwardAnalysis method*), 932
has_job() (*angr.analyses.ForwardAnalysis method*), 907
has_load (*angr.analyses.decompiler.region_simplifiers.expr_folding.Expre attribute*), 1112
has_loads (*angr.analyses.decompiler.region_simplifiers.expr_folding.Expr attribute*), 1112
has_memory_dep (*angr.state_plugins.unicorn_engine.VEXStmtDetails attribute*), 393
has_metadata() (*angr.procedures.definitions.SimCppLibrary method*), 654
has_metadata() (*angr.procedures.definitions.SimLibrary method*), 652
has_metadata() (*angr.procedures.definitions.SimSyscallLibrary method*), 657
has_model() (*angr.knowledge_plugins.key_definitions.key_definition_man method*), 845
has_model() (*angr.knowledge_plugins.key_definitions.KeyDefinitionMana method*), 823
has_model() (*angr.knowledge_plugins.KeyDefinitionManager method*), 730
has_node_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 757
has_node_addr() (*angr.knowledge_plugins.cfg.CFGModel method*), 743

has_node_id() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 758
has_node_id() (*angr.knowledge_plugins.cfg.CFGModel method*), 744
has_nonlabel_nonphi_statements() (*angr.analyses.decompiler.structuring.structurer_base.StructuredBlock static method*), 1037
has_nonlabel_nonphi_statements() (*in module angr.analyses.decompiler.utils*), 1152
has_nonlabel_statements() (*in module angr.analyses.decompiler.utils*), 1151
has_plugin() (*angr.knowledge_base.KnowledgeBase method*), 712
has_plugin() (*angr.KnowledgeBase method*), 164
has_plugin() (*angr.misc.plugins.PluginHub method*), 236
has_plugin() (*angr.sim_state.SimState method*), 241
has_plugin() (*angr.SimState method*), 209
has_plugin_preset (*angr.misc.plugins.PluginHub property*), 235
has_prototype() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 740
has_prototype() (*angr.knowledge_plugins.CallsitePrototypes method*), 714
has_prototype() (*angr.procedures.definitions.SimCppLibrary method*), 654
has_prototype() (*angr.procedures.definitions.SimLibrary method*), 653
has_prototype() (*angr.procedures.definitions.SimSyscallLibrary method*), 657
has_return (*angr.knowledge_plugins.Function property*), 723
has_return (*angr.knowledge_plugins.functions.Function property*), 783
has_return (*angr.knowledge_plugins.functions.function.Function property*), 808
has_some_prototype() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 740
has_some_prototype() (*angr.knowledge_plugins.CallsitePrototypes method*), 714
HAS_SQL (*angr.analyses.code_tagging.CodeTags attribute*), 1018
has_sql() (*angr.analyses.code_tagging.CodeTagging method*), 1018
has_sql() (*angr.analyses.CodeTagging method*), 897
has_statements (*angr.engines.pcode.lifter.IRSB property*), 608
has_store() (*angr.analyses.decompiler.region_simplifiers.cond_proximate.CondProximate method*), 1114
has_super_class() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 955
has_super_class() (*angr.analyses.SootClassHierarchy method*), 922
has_symbolic_exit (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 393
has_type_variable_for() (*angr.analyses.typehoon.typevars.TypeVariables method*), 1271
has_unknown_bits() (*angr.sim_type.SimTypeBitfield method*), 705
has_unresolved_calls (*angr.knowledge_plugins.Function property*), 719
has_unresolved_calls (*angr.knowledge_plugins.functions.Function property*), 779
has_unresolved_calls (*angr.knowledge_plugins.functions.function.Function property*), 804
has_unresolved_jumps (*angr.knowledge_plugins.Function property*), 719
has_unresolved_jumps (*angr.knowledge_plugins.functions.Function property*), 779
has_unresolved_jumps (*angr.knowledge_plugins.functions.function.Function property*), 804
HAS_XOR (*angr.analyses.code_tagging.CodeTags attribute*), 1018
has_xor() (*angr.analyses.code_tagging.CodeTagging method*), 1018
has_xor() (*angr.analyses.CodeTagging method*), 897
HasCallExprWalker (*class in angr.analyses.decompiler.block_simplifier*), 1054
HasCallNotification, 1052
HasField (*class in angr.analyses.typehoon.typevars*), 1272
HasRetVarNotification, 1052
HasVVarNotification, 1052
have_opposite_edge_conditions() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 1062
head (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 1071
head (*angr.analyses.decompiler.structured_codegen.c.CIncompleteSwitchCase attribute*), 1135
headerless_c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 1132
heap (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
heap (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 1233
heap (*angr.analyses.reaching_definitions.ReachingDefinitionsState*

property), 1206
heap (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 848
heap (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 825
heap_address() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1196
heap_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1231
heap_address() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1205
heap_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 855
heap_address() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 832
heap_allocator (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState attribute*), 1231
heap_allocator (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 1205
heap_offset (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 842
heap_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
heap_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 1233
heap_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 1206
heap_uses (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 848
heap_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 825
HeapAddress (*class in angr.knowledge_plugins.key_definitions.heap_address*), 845
HeapAllocator (*class in angr.analyses.reaching_definitions.heap_allocator*), 1220
HeavyPcodeMixin (*class in angr.engines.pcode*), 603
HeavyPcodeMixin (*class in angr.engines.pcode.engine*), 604
HeavyResilienceMixin (*class in angr.engines*), 587
HeavyResilienceMixin (*class in angr.engines.vex*), 599
HeavyState (*in module angr.engines.successors*), 596
HeavyVEXMixin (*class in angr.engines*), 587
HeavyVEXMixin (*class in angr.engines.vex*), 599
height() (*angr.analyses.disassembly.Comment method*), 1296
height() (*angr.analyses.disassembly.DisassemblyPiece method*), 1293
height() (*angr.analyses.disassembly.FunctionStart method*), 1293
hex_dump() (*angr.storage.memory_mixins.hex_dumper_mixin.HexDumperMixin method*), 492
hex_dump() (*angr.storage.memory_mixins.HexDumperMixin method*), 461
HexDumperMixin (*class in angr.storage.memory_mixins*), 461
HexDumperMixin (*class in angr.storage.memory_mixins.hex_dumper_mixin*), 491
hexrepr() (*angr.analyses.disassembly.DisassemblyPiece method*), 1293
history (*angr.state_plugins.SimState attribute*), 239
history (*angr.SimState attribute*), 208
history_contains() (*angr.state_hierarchy.StateHierarchy method*), 552
history_contains() (*angr.StateHierarchy method*), 221
history_predecessors() (*angr.state_hierarchy.StateHierarchy method*), 552
history_predecessors() (*angr.StateHierarchy method*), 221
history_predicate (*angr.state_plugins.SimStateEdgeHitmap attribute*), 298
history_successors() (*angr.state_hierarchy.StateHierarchy method*), 552
history_successors() (*angr.StateHierarchy method*), 221
HistoryStmt (*class in angr.state_plugins.history*), 378
HistoryTrackingMixin (*class in angr.storage.memory_mixins*), 462
HistoryTrackingMixin (*class in angr.storage.memory_mixins.paged_memory.pages*), 510
HistoryTrackingMixin (*class in angr.storage.memory_mixins.paged_memory.pages.history_tracki*
HITMAP_SIZE (*angr.state_plugins.SimStateEdgeHitmap attribute*), 298
holes() (*angr.knowledge_plugins.Function method*), 724
holes() (*angr.knowledge_plugins.functions.Function method*), 785
holes() (*angr.knowledge_plugins.functions.function.Function method*), 809
Hook (*class in angr.analyses.disassembly*), 1294
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.PutHook static method*), 1001
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegisterInitia method*), 1001
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.StoreHook static method*), 1001
hook() (*angr.analyses.find_objects_static.NewFunctionHandler method*), 1292
hook() (*angr.analyses.reaching_definitions.function_handler.FunctionHan method*), 1227
hook() (*angr.analyses.reaching_definitions.FunctionHandler*

method), 1186
hook() (angr.Project method), 170
hook() (angr.project.Project method), 225
hook() (angr.state_plugins.Unicorn method), 328
hook() (angr.state_plugins.unicorn_engine.Unicorn method), 398
hook_add() (angr.state_plugins.unicorn_engine.Uniwrapper method), 396
hook_after() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver method), 1001
hook_before() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver method), 1001
hook_del() (angr.state_plugins.unicorn_engine.Uniwrapper method), 396
hook_reset() (angr.state_plugins.unicorn_engine.Uniwrapper method), 396
hook_symbol() (angr.Project method), 171
hook_symbol() (angr.project.Project method), 226
hooked_by() (angr.Project method), 171
hooked_by() (angr.project.Project method), 225
HookNode (class in angr.codenode), 1324
HooksMixin (class in angr.engines), 588
HooksMixin (class in angr.engines.hook), 597

I

id (angr.angrdb.models.DbCFGModel attribute), 1022
id (angr.angrdb.models.DbComment attribute), 1024
id (angr.angrdb.models.DbFunction attribute), 1023
id (angr.angrdb.models.DbInformation attribute), 1021
id (angr.angrdb.models.DbKnowledgeBase attribute), 1022
id (angr.angrdb.models.DbLabel attribute), 1025
id (angr.angrdb.models.DbObject attribute), 1022
id (angr.angrdb.models.DbStructuredCode attribute), 1024
id (angr.angrdb.models.DbVariableCollection attribute), 1023
id (angr.angrdb.models.DbXRefs attribute), 1024
id (angr.exploration_techniques.spiller_db.PickledState attribute), 574
ident (angr.analyses.disassembly.DisassemblyPiece attribute), 1293
ident (angr.analyses.disassembly.OperandPiece attribute), 1295
ident (angr.angrdb.models.DbCFGModel attribute), 1023
ident (angr.angrdb.models.DbVariableCollection attribute), 1023
ident (angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute), 756
ident (angr.knowledge_plugins.cfg.CFGModel attribute), 742
ident (angr.sim_variable.SimVariable attribute), 680
identical_blocks (angr.analyses.BinDiff property), 883
identical_blocks (angr.analyses.bindiff.BinDiff property), 950
identical_blocks (angr.analyses.bindiff.FunctionDiff property), 948
identical_functions (angr.analyses.BinDiff property), 883
identical_functions (angr.analyses.bindiff.BinDiff property), 950
identifier (angr.analyses.analysis.KnownAnalysesPlugin attribute), 928
Identifier (class in angr.analyses), 907
Identifier (class in angr.analyses.identifier.identify), 1283
identify_func() (angr.analyses.Identifier method), 907
identify_func() (angr.analyses.identifier.identify.Identifier method), 1283
idx (angr.ailment.Block attribute), 1368
idx (angr.ailment.block.Block attribute), 1406
idx (angr.ailment.tagged_object.TaggedObject attribute), 1408
idx (angr.analyses.decompiler.structured_codegen.c.CConstruct attribute), 1131
idx (angr.analyses.decompiler.structured_codegen.c.CStatement attribute), 1132
idx (angr.analyses.typehoon.typevars.TypeVariable attribute), 1269
idx_counters (angr.analyses.decompiler.structured_codegen.base.BaseSt attribute), 1129
IfElseFlattener (class in angr.analyses.decompiler.region_simplifiers.ifelse), 1114
iffalse (angr.ailment.expression.ITE attribute), 1387
iffalse (angr.analyses.decompiler.structured_codegen.c.CITE attribute), 1143
IfSimplifier (class in angr.analyses.decompiler.region_simplifiers.if_), 1114
iftrue (angr.ailment.expression.ITE attribute), 1387
iftrue (angr.analyses.decompiler.structured_codegen.c.CITE attribute), 1143
IFUNC_HINTS (angr.analyses.cfg.cfg_fast.CFGJobType attribute), 987
immediate_dominators() (angr.analyses.cfg.cfg_emulated.CFGEmulated method), 979
immediate_dominators() (angr.analyses.cfg.CFGEmulated method), 965
immediate_dominators() (angr.analyses.CFGEmulated method), 887
immediate_postdominators()

(*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 979
immediate_postdominators() (*angr.analyses.cfg.CFGEmulated method*), 965
immediate_postdominators() (*angr.analyses.CFGEmulated method*), 887
ImportedLine (*class in angr.analyses.decompiler.structured_codegen.dwarf_import*), 1149
ImportSourceCode (*class in angr.analyses.decompiler*), 1049
ImportSourceCode (*class in angr.analyses.decompiler.structured_codegen*), 1124
ImportSourceCode (*class in angr.analyses.decompiler.structured_codegen.dwarf_import*), 1149
IN (*angr.sim_type.PointerDisposition attribute*), 694
IN_OUT (*angr.sim_type.PointerDisposition attribute*), 694
IN_OUTMAYBE (*angr.sim_type.PointerDisposition attribute*), 694
inc_active_workers() (*angr.distributed.Server method*), 1362
inc_active_workers() (*angr.distributed.server.Server method*), 1362
inc_active_workers() (*angr.Server method*), 173
includes() (*angr.keyed_region.RegionObject method*), 868
includes() (*angr.storage.memory_object.SimMemoryObject method*), 451
includes() (*angr.storage.SimMemoryObject method*), 426
includes_function() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 1213
inconsistent (*angr.analyses.stack_pointer_tracker.StackPointerTracker property*), 1246
inconsistent (*angr.analyses.StackPointerTracker property*), 923
inconsistent_for() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1246
inconsistent_for() (*angr.analyses.StackPointerTracker method*), 923
increase_lmdb_map_size() (*angr.knowledge_plugins.RuntimeDb method*), 734
indent_str() (*angr.analyses.decompiler.structured_codegen.c.CComment static method*), 1131
index (*angr.analyses.decompiler.structured_codegen.c.CIndexedVariable attribute*), 1139
index (*angr.utils.graph.ContainerNode attribute*), 1348
indirect_jumps (*angr.analyses.cfg.cfg_base.CFGBase attribute*), 982
indirect_jumps (*angr.analyses.cfg.CFGBase attribute*), 960
IndirectJump (*class in angr.knowledge_plugins.cfg*), 751
IndirectJump (*class in angr.knowledge_plugins.cfg.indirect_jump*), 770
IndirectJumpResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.resolver*), 1004
IndirectJumps (*class in angr.knowledge_plugins*), 730
IndirectJumps (*class in angr.knowledge_plugins.indirect_jumps*), 775
IndirectJumpType (*class in angr.knowledge_plugins.cfg*), 753
IndirectJumpType (*class in angr.knowledge_plugins.cfg.indirect_jump*), 769
infer_provider() (*angr.LLMClient method*), 166
infer_shapes() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 1263
INFERRED (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypeKn attribute*), 739
info (*angr.code_location.CodeLocation attribute*), 866
info (*angr.knowledge_plugins.Function property*), 718
info (*angr.knowledge_plugins.functions.Function property*), 778
info (*angr.knowledge_plugins.functions.function.Function property*), 802
init_class() (*angr.state_plugins.javavm_classloader.SimJavaVmClasslo method*), 401
init_class() (*angr.state_plugins.SimJavaVmClassloader method*), 279
init_hierarchy() (*angr.analyses.soot_class_hierarchy.SootClassHierarc method*), 955
init_hierarchy() (*angr.analyses.SootClassHierarchy method*), 922
init_state() (*angr.SimHeapPTMalloc method*), 193
init_state() (*angr.SimStatePlugin method*), 213
init_state() (*angr.state_plugins.heap.heap_base.SimHeapBase method*), 412
init_state() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 420
init_state() (*angr.state_plugins.heap.SimHeapBase method*), 406
init_state() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 411
init_state() (*angr.state_plugins.history.SimStateHistory method*), 375
init_state() (*angr.state_plugins.plugin.SimStatePlugin method*), 331
init_state() (*angr.state_plugins.posix.SimSystemPosix method*), 352

init_state() (*angr.state_plugins.SimHeapBase method*), 270
init_state() (*angr.state_plugins.SimHeapPTMalloc method*), 275
init_state() (*angr.state_plugins.SimStateHistory method*), 301
init_state() (*angr.state_plugins.SimStatePlugin method*), 316
init_state() (*angr.state_plugins.SimSymbolizer method*), 320
init_state() (*angr.state_plugins.SimSystemPosix method*), 322
init_state() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 420
init_static_field() (*angr.simos.javavm.SimJavaVM static method*), 1341
init_static_field() (*angr.simos.SimJavaVM static method*), 1326
init_str() (*angr.procedures.definitions.SimTypeCollection method*), 649
INITIAL_SP_16BIT (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1189
INITIAL_SP_16BIT (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 847
INITIAL_SP_16BIT (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 824
INITIAL_SP_32BIT (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1189
INITIAL_SP_32BIT (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 847
INITIAL_SP_32BIT (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 824
INITIAL_SP_64BIT (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1189
INITIAL_SP_64BIT (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 847
INITIAL_SP_64BIT (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 824
INITIALIZATION (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1057
InitializationFinder (*class in angr.analyses*), 908
InitializationFinder (*class in angr.analyses.init_finder*), 1307
initialize() (*angr.utils.mp.Initializer method*), 1355
initialize_dominance_frontiers() (*angr.analyses.variable_recovery.variable_recovery.VariableRecoveryBase method*), 1248
initialize_gdt_x86() (*angr.simos.linux.SimLinux method*), 1336
initialize_gdt_x86() (*angr.simos.SimLinux method*), 1327
initialize_gdt_x86() (*angr.simos.SimWindows method*), 1333
initialize_gdt_x86() (*angr.simos.windows.SimWindows method*), 1339
initialize_segment_register_x64() (*angr.simos.linux.SimLinux method*), 1336
initialize_segment_register_x64() (*angr.simos.SimLinux method*), 1327
initialize_segment_register_x64() (*angr.simos.SimWindows method*), 1333
initialize_segment_register_x64() (*angr.simos.windows.SimWindows method*), 1339
initialize_variable_names() (*angr.knowledge_plugins.VariableManager method*), 735
initialize_variable_names() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 819
initialize_variable_names() (*angr.knowledge_plugins.variables.VariableManager method*), 811
initialized_classes (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader property*), 401
initialized_classes (*angr.state_plugins.SimJavaVmClassloader property*), 280
initializer (*angr.analyses.decompiler.structured_codegen.c.CForLoop attribute*), 1134
initializer (*class in angr.utils.mp*), 1354
initializers (*angr.utils.mp.Initializer attribute*), 1355
InitialValueTag (*class in angr.knowledge_plugins.key_definitions.tag*), 858
inline_call() (*angr.sim_procedure.SimProcedure method*), 645
inline_call() (*angr.SimProcedure method*), 205
InlineDefinitionSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1080
InlinedMemcpySimplifierLate (*class in angr.analyses.decompiler.optimization_passes*), 1080
InlinedMemsetSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1080
InlinedMemsetSimplifierLate (*class in angr.analyses.decompiler.optimization_passes*), 1081
InlinedStrcpySimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1081
InlinedStrcpySimplifierLate (*class in angr.analyses.decompiler.optimization_passes*), 1081

InlinedStringTransformationSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1081
InlinedStrlenSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1082
InlinedWcscpySimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1082
InlinedWcscpySimplifierLate (*class in angr.analyses.decompiler.optimization_passes*), 1082
inner_step() (*angr.exploration_techniques.Threading method*), 565
inner_step() (*angr.exploration_techniques.threading.Threading method*), 574
input_state (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 769
input_state (*angr.knowledge_plugins.cfg.CFGENode attribute*), 741
input_states (*angr.knowledge_plugins.propagations.PropagationModel attribute*), 774
input_variables (*angr.analyses.vfg.VFGNode attribute*), 1288
input_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 817
ins_addr (*angr.ailment.Manager property*), 1370
ins_addr (*angr.ailment.manager.Manager property*), 1407
ins_addr (*angr.ailment.tagged_object.TagDict attribute*), 1407
ins_addr (*angr.analyses.cfg.cfg_fast.FunctionEdge attribute*), 985
ins_addr (*angr.analyses.decompiler.clinic.DataRefDesc attribute*), 1057
ins_addr (*angr.analyses.decompiler.structured_codegen.base.InstructionMappingElement attribute*), 1128
ins_addr (*angr.analyses.decompiler.structured_codegen.InstructionMappingElement attribute*), 1124
ins_addr (*angr.code_location.AILCodeLocation property*), 867
ins_addr (*angr.code_location.CodeLocation attribute*), 866
ins_addr (*angr.engines.light.engine.SimEngineLight attribute*), 1171
ins_addr (*angr.engines.light.SimEngineLight attribute*), 1168
ins_addr (*angr.errors.SimError attribute*), 1358
ins_addr (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 771
ins_addr (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 752
ins_addr (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 842
ins_addr (*angr.knowledge_plugins.xrefs.XRef attribute*), 861
ins_addr (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 863
ins_addr (*angr.SimError attribute*), 178
ins_addrs (*angr.state_plugins.history.SimStateHistory property*), 378
ins_addrs (*angr.state_plugins.SimStateHistory property*), 303
Insert (*class in angr.ailment.expression*), 1392
insert() (*angr.SimMount method*), 194
insert() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 359
insert() (*angr.state_plugins.filesystem.SimFilesystem method*), 357
insert() (*angr.state_plugins.filesystem.SimMount method*), 358
insert() (*angr.state_plugins.posix.PosixDevFS method*), 347
insert() (*angr.state_plugins.posix.PosixProcFS method*), 349
insert() (*angr.state_plugins.PosixDevFS method*), 259
insert() (*angr.state_plugins.PosixProcFS method*), 261
insert() (*angr.state_plugins.SimFilesystem method*), 270
insert() (*angr.state_plugins.SimMount method*), 286
insert_asm() (*angr.analyses.Reassembler method*), 917
insert_asm() (*angr.analyses.reassembler.Reassembler method*), 1303
insert_node() (*in module angr.analyses.decompiler.utils*), 1151
inserted_asm_after_label (*angr.analyses.Reassembler property*), 916
inserted_asm_after_label (*angr.analyses.reassembler.Reassembler property*), 1302
inserted_asm_before_label (*angr.analyses.Reassembler property*), 916
inserted_asm_before_label (*angr.analyses.reassembler.Reassembler property*), 1302
insn (*angr.block.CapstoneInsn attribute*), 232
insn (*angr.block.PCodeInsn attribute*), 233
insn_addr (*angr.code_location.AILCodeLocation attribute*), 867
insn_addr_to_memory_data (*angr.analyses.cfg.cfg_fast.CFGFast property*), 992
insn_addr_to_memory_data (*angr.analyses.cfg.CFGFast property*), 970
insn_addr_to_memory_data (*angr.analyses.CFGFast property*), 882
insn_addr_to_memory_data

(*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 757
insn_addr_to_memory_data (*angr.knowledge_plugins.cfg.CFGModel attribute*), 743
insn_addr_unwrap (*angr.code_location.AILCodeLocation property*), 867
insn_observe() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 1216
insn_observe() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 1200
insn_observe() (*angr.analyses.ReachingDefinitionsAnalysis method*), 915
insn_op_idx (*angr.knowledge_plugins.xrefs.XRef attribute*), 861
insn_op_idx (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 863
insn_op_type (*angr.knowledge_plugins.xrefs.XRef attribute*), 862
insn_op_type (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 864
insns (*angr.block.DisassemblerBlock attribute*), 232
insns (*angr.engines.pcode.lifter.PcodeDisassemblerBlock attribute*), 606
inspect (*angr.sim_state.SimState attribute*), 239
inspect (*angr.SimState attribute*), 208
InspectAttrs (*class in angr.state_plugins*), 251
InspectAttrs (*class in angr.state_plugins.inspect*), 332
InspectMixin (*class in angr.storage.memory_mixins*), 463
INSTRUCTION (*angr.state_plugins.inspect.EventType attribute*), 331
instruction (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
instruction (*angr.state_plugins.InspectAttrs attribute*), 254
Instruction (*class in angr.analyses.disassembly*), 1294
Instruction (*class in angr.analyses.reassembler*), 1299
instruction_addresses (*angr.engines.pcode.lifter.IRSB property*), 609
instruction_addresses() (*angr.analyses.reassembler.BasicBlock method*), 1300
instruction_addresses() (*angr.analyses.reassembler.Procedure method*), 1301
instruction_addrs (*angr.Block property*), 161
instruction_addrs (*angr.block.Block property*), 234
instruction_addrs (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 767
instruction_addrs (*angr.knowledge_plugins.cfg.CFGNode attribute*), 750
INSTRUCTION_LIMIT (*angr.EmulatorStopReason attribute*), 162
instruction_size() (*angr.knowledge_plugins.Function method*), 722
instruction_size() (*angr.knowledge_plugins.functions.Function method*), 783
instruction_size() (*angr.knowledge_plugins.functions.function.Function method*), 807
InstructionError (*class in angr.analyses.disassembly*), 1297
InstructionMapping (*class in angr.analyses.decompiler.structured_codegen*), 1124
InstructionMapping (*class in angr.analyses.decompiler.structured_codegen.base*), 1129
InstructionMappingElement (*class in angr.analyses.decompiler.structured_codegen*), 1124
InstructionMappingElement (*class in angr.analyses.decompiler.structured_codegen.base*), 1128
instructions (*angr.analyses.Reassembler property*), 916
instructions (*angr.analyses.reassembler.Reassembler property*), 1302
instructions (*angr.Block property*), 161
instructions (*angr.block.Block property*), 234
instructions (*angr.engines.pcode.lifter.IRSB property*), 609
INT (*angr.analyses.decompiler.structured_codegen.base.CConstantType attribute*), 1127
Int (*class in angr.analyses.typehoon.typeconsts*), 1275
Int1 (*class in angr.analyses.typehoon.typeconsts*), 1275
Int128 (*class in angr.analyses.typehoon.typeconsts*), 1276
Int16 (*class in angr.analyses.typehoon.typeconsts*), 1275
Int256 (*class in angr.analyses.typehoon.typeconsts*), 1276
int2base() (*in module angr.state_plugins.trace_additions*), 382
Int32 (*class in angr.analyses.typehoon.typeconsts*), 1275
Int512 (*class in angr.analyses.typehoon.typeconsts*), 1276
Int64 (*class in angr.analyses.typehoon.typeconsts*), 1275
Int8 (*class in angr.analyses.typehoon.typeconsts*), 1275
int_args (*angr.calling_conventions.SimCC property*), 664
int_args (*angr.SimCC property*), 174
int_iter (*angr.calling_conventions.ArgSession attribute*), 663
int_iter (*angr.calling_conventions.MicrosoftAMD64ArgSession attribute*), 669
int_iter (*angr.calling_conventions.SimCC.ArgSession attribute*), 664
int_iter (*angr.SimCC.ArgSession attribute*), 175

int_len_mod (*angr.procedures.stubs.format_parser.FormatParser attribute*), 648
int_sign (*angr.procedures.stubs.format_parser.FormatParser attribute*), 648
int_type() (*in module angr.analyses.typehoon.typeconsts*), 1281
Integer (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 1309
Integer (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
Integer (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 755
IntegerProxiNode (*class in angr.analyses.proximity_graph*), 1310
interfered_assignments (*angr.analyses.decompiler.region_simplifiers.expr_folding.InterferenceChecker attribute*), 1113
interference_graph() (*angr.analyses.SLivenessAnalysis method*), 919
InterferenceChecker (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1112
interfunction_level (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 1227
interfunction_level (*angr.analyses.reaching_definitions.FunctionHandler attribute*), 1186
internal_objects (*angr.keyed_region.RegionObject property*), 868
interpret() (*angr.procedures.stubs.format_parser.FormatParser method*), 647
intersection() (*angr.state_plugins.sim_action_object.SimActionObject method*), 641
intersection() (*angr.state_plugins.SimActionObject method*), 265
interval_to_hasstore (*angr.analyses.decompiler.region_simplifiers.expr_folding.SomeStatementFinder attribute*), 1113
IntVar (*class in angr.analyses.typehoon.typeconsts*), 1277
invalidate() (*angr.angrdb.AngrDB method*), 1019
invalidate() (*angr.angrdb.db.AngrDB method*), 1021
invalidate_direct_next() (*angr.engines.pcode.lifter.IRSB method*), 609
inverse() (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode method*), 1262
inverse_wo_tag() (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode method*), 1262
inverted_idoms() (*in module angr.utils.graph*), 1347
ip (*angr.sim_state.SimState property*), 240
ip (*angr.SimState property*), 209
irange() (*angr.knowledge_plugins.functions.function_manager.FunctionDict method*), 791
irange() (*angr.knowledge_plugins.variables.variable_manager.SortedDict method*), 813
IROp (*class in angr.analyses.disassembly*), 1293
irsb (*angr.analyses.disassembly.IROp attribute*), 1294
irsb (*angr.engines.pcode.lifter.Lifter attribute*), 611
irsb (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
irsb (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 767
irsb (*angr.knowledge_plugins.cfg.CFGNode attribute*), 750
IRSB (*angr.state_plugins.inspect.EventType attribute*), 331
IRSB (*class in angr.engines.pcode.lifter*), 606
irsb_from_node() (*angr.analyses.VFG method*), 880
irsb_from_node() (*angr.analyses.vfg.VFG method*), 1290
IRSBConverter (*class in angr.ailment*), 1370
IRTop (*class in angr.engines.light.engine*), 1171
is_jump_target() (*angr.analyses.decompiler.structuring.structurer_base.StructuringBase static method*), 1037
is_addr_known (*angr.codenode.FuncNode property*), 1334
is_ail (*angr.state_plugins.scratch.SimStateScratch property*), 388
is_ail (*angr.state_plugins.SimStateScratch property*), 318
is_alignment (*angr.knowledge_plugins.Function property*), 718
is_alignment (*angr.knowledge_plugins.functions.Function property*), 778
is_alignment (*angr.knowledge_plugins.functions.function.Function property*), 802
is_alignment_mask() (*in module angr.utils.constants*), 1344
is_arm (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 756
is_arm (*angr.knowledge_plugins.cfg.CFGModel attribute*), 742
is_base (*angr.engines.light.data.SpOffset attribute*), 1167
is_base (*angr.engines.light.SpOffset attribute*), 1170
is_bit_set_be() (*angr.analyses.flirt.FlirtSignatureParsed static method*), 1165
is_bool_expr() (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase static method*), 1107
is_bounded() (*angr.state_plugins.SimUCManager method*), 325
is_bounded() (*angr.state_plugins.uc_manager.SimUCManager method*), 388
is_bytes (*angr.storage.memory_object.SimMemoryObject attribute*), 451

is_bytes (*angr.storage.SimMemoryObject attribute*), 425
is_cached() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 794
is_class_initialized() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 401
is_class_initialized() (*angr.state_plugins.SimJavaVmClassloader method*), 279
is_concrete() (*angr.analyses.init_finder.SimEngineInitFinderVEX static method*), 1307
is_cross_referenced() (*angr.analyses.vtable.VtableFinder method*), 1291
is_cross_referenced() (*angr.analyses.VtableFinder method*), 927
is_default_name (*angr.knowledge_plugins.Function attribute*), 717
is_default_name (*angr.knowledge_plugins.functions.Function attribute*), 777
is_default_name (*angr.knowledge_plugins.functions.function.Function attribute*), 802
is_default_name (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 810
is_dump (*angr.simos.SimWindows property*), 1331
is_dump (*angr.simos.windows.SimWindows property*), 1338
is_empty (*angr.engines.SimSuccessors property*), 592
is_empty (*angr.engines.successors.SimSuccessors property*), 595
is_empty (*angr.keyed_region.RegionObject property*), 868
is_empty (*angr.storage.memory_mixins.regioned_memory.RegionedObjectMap property*), 534
is_empty() (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink method*), 1241
is_empty() (*angr.analyses.cfg_slice_to_sink.CFGSliceToSink method*), 1239
is_empty_node() (*in module angr.analyses.decompiler.utils*), 1151
is_empty_or_label_only_node() (*in module angr.analyses.decompiler.utils*), 1151
is_exception (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute*), 986
is_extern (*angr.code_location.AILCodeLocation property*), 867
is_false() (*angr.state_plugins.SimSolver method*), 292
is_false() (*angr.state_plugins.solver.SimSolver method*), 365
is_fp_arg() (*angr.calling_conventions.SimCC method*), 664
is_fp_arg() (*angr.SimCC method*), 175
is_fp_value() (*angr.calling_conventions.SimCC static method*), 665
is_fp_value() (*angr.SimCC static method*), 176
is_free() (*angr.PTChunk method*), 167
is_free() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 414
is_free() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 417
is_free() (*angr.state_plugins.heap.PTChunk method*), 405
is_free() (*angr.state_plugins.PTChunk method*), 258
is_func_nonreturning() (*angr.knowledge_plugins.FunctionManager method*), 728
is_func_nonreturning() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 798
is_func_nonreturning() (*angr.knowledge_plugins.functions.FunctionManager method*), 789
is_func_returning_unknown() (*angr.knowledge_plugins.FunctionManager method*), 728
is_func_returning_unknown() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 798
is_func_returning_unknown() (*angr.knowledge_plugins.functions.FunctionManager method*), 789
IS_FUNCTION (*angr.sim_procedure.SimProcedure attribute*), 645
IS_FUNCTION (*angr.SimProcedure attribute*), 204
is_function() (*angr.analyses.vtable.VtableFinder method*), 1291
is_function() (*angr.analyses.VtableFinder method*), 927
is_function_argument (*angr.sim_variable.SimVariable property*), 681
is_global_variable_address() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase method*), 1250
is_global_variable_updated() (*angr.analyses.SPropagatorAnalysis static method*), 920
is_heap_address() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 1196
is_heap_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionState static method*), 1232
is_heap_address() (*angr.analyses.reaching_definitions.ReachingDefinitions static method*), 1205
is_heap_address() (*angr.knowledge_plugins.key_definitions.live_definitions static method*), 854
is_heap_address() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 832

is_hex() (*in module angr.analyses.reassembler*), 1297
is_hook (*angr.codenode.BlockNode attribute*), 1323
is_hook (*angr.codenode.CodeNode attribute*), 1323
is_hook (*angr.codenode.HookNode attribute*), 1324
is_hook (*angr.codenode.SyscallNode attribute*), 1324
is_hooked() (*angr.Project method*), 170
is_hooked() (*angr.project.Project method*), 225
is_immediate (*angr.analyses.reassembler.Operand property*), 1299
is_in_readonly_section() (*in module angr.utils.loader*), 1351
is_in_readonly_segment() (*in module angr.utils.loader*), 1351
is_inlined_strcpy() (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier static method*), 1081
is_inlined_wcsncpy() (*angr.analyses.decompiler.optimization_passes.InlinedWcscpySimplifier static method*), 1082
is_int (*angr.ailment.Const property*), 1369
is_int (*angr.ailment.expression.Const property*), 1377
is_integer_likely_a_string() (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier static method*), 1081
is_integer_likely_a_wide_string() (*angr.analyses.decompiler.optimization_passes.InlinedWcscpySimplifier static method*), 1082
is_java (*angr.sim_procedure.SimProcedure property*), 646
is_java (*angr.SimProcedure property*), 206
is_jumptable (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTable attribute*), 1000
is_leaf() (*angr.state_plugins.sim_action_object.SimActionObject method*), 641
is_leaf() (*angr.state_plugins.SimActionObject method*), 265
is_live (*angr.state_plugins.SimStateIcicle property*), 304
is_lsb_extract() (*angr.ailment.expression.Extract method*), 1391
is_lsb_overwrite() (*angr.ailment.expression.Insert method*), 1392
is_machine_word_size_type() (*in module angr.analyses.decompiler.structured_codegen.c*), 1130
is_none_or_likeable() (*in module angr.ailment.utils*), 1409
is_none_or_matchable() (*in module angr.ailment.utils*), 1409
is_noop_block (*angr.engines.pcode.lifter.IRSB property*), 609
is_not_in_cfg() (*angr.analyses.Veritesting method*), 927
is_not_in_cfg() (*angr.analyses.veritesting.Veritesting method*), 1285
is_on_stack (*angr.analyses.reaching_definitions.MemoryLocation property*), 1197
is_on_stack (*angr.knowledge_plugins.key_definitions.atoms.MemoryLoca property*), 841
is_on_stack (*angr.storage.memory_mixins.regioned_memory.region_data attribute*), 533
is_overbound() (*angr.analyses.Veritesting method*), 927
is_overbound() (*angr.analyses.veritesting.Veritesting method*), 1285
is_pc() (*in module angr.utils.loader*), 1351
is_phi_variable() (*angr.knowledge_plugins.variables.variable_manage method*), 816
is_plt (*angr.analyses.reassembler.Procedure property*), 1300
is_plt (*angr.knowledge_plugins.Function property*), 718
is_plt (*angr.knowledge_plugins.functions.Function property*), 778
is_plt (*angr.knowledge_plugins.functions.function.Function property*), 802
is_plt (*angr.knowledge_plugins.FunctionManager method*), 725
is_plt_cached() (*angr.knowledge_plugins.functions.function_manager.F method*), 795
is_plt_cached() (*angr.knowledge_plugins.functions.FunctionManager method*), 786
is_prev_free() (*angr.PTChunk method*), 167
is_prev_free() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 448
is_prev_free() (*angr.state_plugins.heap.PTChunk method*), 405
is_prev_free() (*angr.state_plugins.PTChunk method*), 258
is_prototype_certain() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 740
is_prototype_certain() (*angr.knowledge_plugins.CallsitePrototypes method*), 713
is_prototype_guessed (*angr.ailment.tagged_object.TagDict attribute*), 1407
is_prototype_guessed (*angr.knowledge_plugins.Function property*), 717
is_prototype_guessed (*angr.knowledge_plugins.functions.Function property*), 778
is_prototype_guessed (*angr.knowledge_plugins.functions.function.Function property*), 802
is_prototype_manual()

(*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 739
is_prototype_manual() (*angr.knowledge_plugins.CallsitePrototypes method*), 713
is_pyinstaller() (*in module angr.utils*), 1344
is_pyinstaller() (*in module angr.utils.env*), 1346
is_returning() (*angr.procedures.definitions.SimLibrary method*), 653
is_rust_binary (*angr.Project property*), 172
is_rust_binary (*angr.project.Project property*), 227
is_rust_function() (*angr.knowledge_plugins.Function method*), 724
is_rust_function() (*angr.knowledge_plugins.functions.Function method*), 784
is_rust_function() (*angr.knowledge_plugins.functions.function.Function method*), 808
is_signed (*angr.ailment.expression.Convert attribute*), 1383
is_simple_jump_node() (*in module angr.analyses.decompiler.region_simplifiers.switch_simplifier*), 1117
is_simprocedure (*angr.knowledge_plugins.cfg.cfg_node.CFGNodeB property*), 768
is_simprocedure (*angr.knowledge_plugins.cfg.CFGNode property*), 751
is_simprocedure (*angr.knowledge_plugins.Function property*), 718
is_simprocedure (*angr.knowledge_plugins.functions.Function property*), 778
is_simprocedure (*angr.knowledge_plugins.functions.function.Function property*), 802
is_special (*angr.engines.pcode.behavior.OpBehavior attribute*), 616
is_special (*angr.engines.pcode.behavior.OpBehaviorBoolAnd attribute*), 630
is_special (*angr.engines.pcode.behavior.OpBehaviorBoolNegate attribute*), 629
is_special (*angr.engines.pcode.behavior.OpBehaviorBoolOr attribute*), 630
is_special (*angr.engines.pcode.behavior.OpBehaviorBoolXor attribute*), 629
is_special (*angr.engines.pcode.behavior.OpBehaviorCopy attribute*), 617
is_special (*angr.engines.pcode.behavior.OpBehaviorEqual attribute*), 618
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatAbs attribute*), 632
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatAdd attribute*), 631
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatCeil attribute*), 633
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatDiv attribute*), 632
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatEqual attribute*), 630
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatFloat2Float attribute*), 633
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatFloor attribute*), 634
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatInt2Float attribute*), 633
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatLess attribute*), 631
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatLessEqual attribute*), 631
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatMult attribute*), 632
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatNan attribute*), 631
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatNeg attribute*), 632
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatNotEqual attribute*), 631
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatRound attribute*), 634
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatSqrt attribute*), 633
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatSub attribute*), 632
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatTrunc attribute*), 633
is_special (*angr.engines.pcode.behavior.OpBehaviorInt2Comp attribute*), 623
is_special (*angr.engines.pcode.behavior.OpBehaviorIntAdd attribute*), 621
is_special (*angr.engines.pcode.behavior.OpBehaviorIntAnd attribute*), 625
is_special (*angr.engines.pcode.behavior.OpBehaviorIntCarry attribute*), 622
is_special (*angr.engines.pcode.behavior.OpBehaviorIntDiv attribute*), 627
is_special (*angr.engines.pcode.behavior.OpBehaviorIntLeft attribute*), 625
is_special (*angr.engines.pcode.behavior.OpBehaviorIntLess attribute*), 619
is_special (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual attribute*), 620
is_special (*angr.engines.pcode.behavior.OpBehaviorIntMult attribute*), 627
is_special (*angr.engines.pcode.behavior.OpBehaviorIntNegate attribute*), 624
is_special (*angr.engines.pcode.behavior.OpBehaviorIntOr attribute*), 625
is_special (*angr.engines.pcode.behavior.OpBehaviorIntRem attribute*), 628
is_special (*angr.engines.pcode.behavior.OpBehaviorIntRight attribute*), 626

is_special (*angr.engines.pcode.behavior.OpBehaviorIntSless attribute*), 623
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSless attribute*), 622
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSlr attribute*), 628
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSext attribute*), 621
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSless attribute*), 619
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSless attribute*), 619
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSrem attribute*), 628
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSright attribute*), 626
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSub attribute*), 622
is_special (*angr.engines.pcode.behavior.OpBehaviorIntXor attribute*), 624
is_special (*angr.engines.pcode.behavior.OpBehaviorIntZext attribute*), 620
is_special (*angr.engines.pcode.behavior.OpBehaviorLzcount attribute*), 635
is_special (*angr.engines.pcode.behavior.OpBehaviorNotEqual attribute*), 618
is_special (*angr.engines.pcode.behavior.OpBehaviorPiece attribute*), 634
is_special (*angr.engines.pcode.behavior.OpBehaviorPopcount attribute*), 635
is_special (*angr.engines.pcode.behavior.OpBehaviorSubpiece attribute*), 635
is_stack (*angr.storage.memory_mixins.MemoryRegionMetaMixin property*), 472
is_stack (*angr.storage.memory_mixins.regioned_memory.MemoryRegionMetaMixin property*), 525
is_stack (*angr.storage.memory_mixins.regioned_memory.region_meta.MemoryRegionMetaMixin property*), 537
is_stack_address() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 1191
is_stack_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1232
is_stack_address() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1205
is_stack_address() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 1250
is_stack_address() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 849
is_stack_address() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 825
is_statement_terminating() (*in module angr.analyses.decompiler.utils*), 1154
is_stored() (*angr.vaults.Vault method*), 872
is_stored() (*angr.vaults.VaultDict method*), 872
is_subclass() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 955
is_subclass() (*angr.analyses.SootClassHierarchy method*), 922
is_subclass_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 955
is_subclass_including() (*angr.analyses.SootClassHierarchy method*), 922
is_symbol_hooked() (*angr.Project method*), 172
is_symbol_hooked() (*angr.project.Project method*), 226
is_symbolic (*angr.state_plugins.sim_action.SimAction property*), 638
is_symbolic (*angr.state_plugins.sim_action.SimActionConstraint property*), 639
is_symbolic (*angr.state_plugins.sim_action.SimActionData property*), 640
is_symbolic (*angr.state_plugins.sim_action.SimActionExit property*), 639
is_symbolic (*angr.state_plugins.sim_action.SimActionOperation property*), 639
is_symbolic (*angr.state_plugins.SimAction property*), 263
is_symbolic (*angr.state_plugins.SimActionConstraint property*), 263
is_symbolic (*angr.state_plugins.SimActionData property*), 264
is_symbolic (*angr.state_plugins.SimActionExit property*), 264
is_symbolic (*angr.state_plugins.SimActionOperation property*), 266
is_syscall (*angr.analyses.cfg.cfg_emulated.CFGJob property*), 716
is_syscall (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
is_syscall (*angr.knowledge_plugins.cfg.CFGNode property*), 750
is_syscall (*angr.knowledge_plugins.Function property*), 718
is_syscall (*angr.knowledge_plugins.functions.Function property*), 778
is_syscall (*angr.knowledge_plugins.functions.function.Function property*), 802
is_syscall_addr() (*angr.SimOS method*), 197
is_syscall_addr() (*angr.simos.SimOS method*), 1329
is_syscall_addr() (*angr.simos.simos.SimOS method*), 1355
is_syscall_addr() (*angr.simos.SimUserland method*), 1331
is_syscall_addr() (*angr.simos.SimWindows method*), 1332
is_syscall_addr() (*angr.simos.userland.SimUserland*

method), 1337
is_syscall_addr() (angr.simos.windows.SimWindows method), 1339
is_taint_impacting_stack_pointers() (angr.analyses.backward_slice.BackwardSlice method), 947
is_taint_impacting_stack_pointers() (angr.analyses.BackwardSlice method), 882
is_taint_related_to_ip() (angr.analyses.backward_slice.BackwardSlice method), 947
is_taint_related_to_ip() (angr.analyses.BackwardSlice method), 882
is_thumb_addr() (angr.analyses.cfg.cfg_base.CFGBase method), 982
is_thumb_addr() (angr.analyses.cfg.CFGBase method), 961
is_top() (angr.analyses.reaching_definitions.LiveDefinitions static method), 1190
is_top() (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method), 1231
is_top() (angr.analyses.reaching_definitions.ReachingDefinitionsState method), 1205
is_top() (angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method), 1249
is_top() (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method), 848
is_top() (angr.knowledge_plugins.key_definitions.LiveDefinitions static method), 825
is_tracking_memory (angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerState attribute), 1244
is_tracking_memory (angr.analyses.stack_pointer_tracker.StackPointerTrackerState attribute), 1244
is_true() (angr.state_plugins.SimSolver method), 292
is_true() (angr.state_plugins.solver.SimSolver method), 365
is_unary (angr.engines.pcode.behavior.OpBehavior attribute), 616
is_unary (angr.engines.pcode.behavior.OpBehaviorBoolAnd attribute), 630
is_unary (angr.engines.pcode.behavior.OpBehaviorBoolNegate attribute), 629
is_unary (angr.engines.pcode.behavior.OpBehaviorBoolOr attribute), 630
is_unary (angr.engines.pcode.behavior.OpBehaviorBoolXor attribute), 629
is_unary (angr.engines.pcode.behavior.OpBehaviorCopy attribute), 617
is_unary (angr.engines.pcode.behavior.OpBehaviorEqual attribute), 618
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatAbs attribute), 632
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatAdd attribute), 631
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatCeil attribute), 634
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatDiv attribute), 632
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatEqual attribute), 630
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatFloat2Float attribute), 633
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatFloor attribute), 634
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatInt2Float attribute), 633
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatLess attribute), 631
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatLessEqual attribute), 631
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatMult attribute), 632
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatNan attribute), 631
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatNeg attribute), 632
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatNotEqual attribute), 631
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatRound attribute), 634
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatSqrt attribute), 633
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatSub attribute), 632
is_unary (angr.engines.pcode.behavior.OpBehaviorFloatTrunc attribute), 633
is_unary (angr.engines.pcode.behavior.OpBehaviorInt2Comp attribute), 623
is_unary (angr.engines.pcode.behavior.OpBehaviorIntAdd attribute), 621
is_unary (angr.engines.pcode.behavior.OpBehaviorIntAnd attribute), 625
is_unary (angr.engines.pcode.behavior.OpBehaviorIntCarry attribute), 622
is_unary (angr.engines.pcode.behavior.OpBehaviorIntDiv attribute), 627
is_unary (angr.engines.pcode.behavior.OpBehaviorIntLeft attribute), 625
is_unary (angr.engines.pcode.behavior.OpBehaviorIntLess attribute), 619
is_unary (angr.engines.pcode.behavior.OpBehaviorIntLessEqual attribute), 620
is_unary (angr.engines.pcode.behavior.OpBehaviorIntMult attribute), 627
is_unary (angr.engines.pcode.behavior.OpBehaviorIntNegate attribute), 624
is_unary (angr.engines.pcode.behavior.OpBehaviorIntOr attribute), 625

is_unary (*angr.engines.pcode.behavior.OpBehaviorIntRem attribute*), 628
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntRight attribute*), 626
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSborrow attribute*), 623
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntScarry attribute*), 622
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSdiv attribute*), 628
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSext attribute*), 621
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSless attribute*), 619
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual attribute*), 619
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSrem attribute*), 628
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSright attribute*), 626
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSub attribute*), 622
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntXor attribute*), 624
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntZext attribute*), 620
is_unary (*angr.engines.pcode.behavior.OpBehaviorLzcount attribute*), 635
is_unary (*angr.engines.pcode.behavior.OpBehaviorNotEqual attribute*), 618
is_unary (*angr.engines.pcode.behavior.OpBehaviorPiece attribute*), 634
is_unary (*angr.engines.pcode.behavior.OpBehaviorPopcount attribute*), 635
is_unary (*angr.engines.pcode.behavior.OpBehaviorSubpiece attribute*), 635
is_va_start_amd64() (*angr.analyses.calling_convention.CallingConventionAnalysis method*), 952
is_va_start_amd64() (*angr.analyses.CallingConventionAnalysis method*), 896
is_value_set (*angr.state_plugins.unicorn_engine.MemoryItem attribute*), 393
is_value_symbolic (*angr.state_plugins.unicorn_engine.MemoryValue attribute*), 393
is_variable_used_at() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 814
is_visible_class() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 955
is_visible_class() (*angr.analyses.SootClassHierarchy method*), 922
is_visible_method() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 955
is_visible_method() (*angr.analyses.SootClassHierarchy method*), 922
is_vvar_used_for_addr_loading_switch_case() (*angr.analyses.SPropagatorAnalysis static method*), 920
IsArray (*class in angr.analyses.typehoon.typevars*), 1272
ISPOMixin (*class in angr.storage.memory_mixins*), 463
ISPOMixin (*class in angr.storage.memory_mixins.paged_memory.pages*), 511
ISPOMixin (*class in angr.storage.memory_mixins.paged_memory.pages.isp* 520
ITE (*class in angr.ailment.expression*), 1386
ITE() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1375
ite_exprs (*angr.analyses.decompiler.decompilation_cache.Decompilation attribute*), 1065
ite_exprs (*angr.angrdb.models.DbStructuredCode attribute*), 1024
ITEExprConverter (*class in angr.analyses.decompiler.optimization_passes*), 1079
ITEExprConverter (*class in angr.analyses.decompiler.optimization_passes.ite_expr_converter* 1099
items() (*angr.analyses.ddg.LiveDefinitions method*), 1158
items() (*angr.analyses.decompiler.structured_codegen.base.InstructionMa method*), 1129
items() (*angr.analyses.decompiler.structured_codegen.base.PositionMapp method*), 1128
items() (*angr.analyses.decompiler.structured_codegen.InstructionMappin method*), 1124
items() (*angr.analyses.decompiler.structured_codegen.PositionMapping method*), 1125
items() (*angr.knowledge_plugins.FunctionManager method*), 726
items() (*angr.knowledge_plugins.functions.function_manager.FunctionDic method*), 790
items() (*angr.knowledge_plugins.functions.function_manager.FunctionMa method*), 795
items() (*angr.knowledge_plugins.functions.function_manager.SpillingFun method*), 793
items() (*angr.knowledge_plugins.functions.FunctionManager method*), 781
items() (*angr.knowledge_plugins.Labels method*), 731
items() (*angr.knowledge_plugins.labels.Labels method*), 776
items() (*angr.knowledge_plugins.patches.PatchManager method*), 738
items() (*angr.knowledge_plugins.PatchManager*

method), 732
items() (*angr.state_plugins.globals.SimStateGlobals method*), 387
items() (*angr.state_plugins.SimStateGlobals method*), 300
items() (*angr.storage.memory_mixins.paged_memory.paged_memory.PageMixin method*), 502
iter_own() (*angr.knowledge_plugins.types.TypesStore method*), 772
iter_own() (*angr.knowledge_plugins.TypesStore method*), 735
iter_own_keys() (*angr.knowledge_plugins.types.TypesStore method*), 773
iter_own_keys() (*angr.knowledge_plugins.TypesStore method*), 735
iterator (*angr.analyses.decompiler.structured_codegen.c.CForLoop attribute*), 1134
ITERegionConverter (*class in angr.analyses.decompiler.optimization_passes*), 1079
itervariables() (*angr.analyses.ddg.LiveDefinitions method*), 1158

J

javavm_memory (*angr.sim_state.SimState property*), 241
javavm_memory (*angr.SimState property*), 209
javavm_registers (*angr.sim_state.SimState property*), 241
javavm_registers (*angr.SimState property*), 209
JavaVmMemory (*class in angr.storage.memory_mixins*), 463
JavaVmMemoryMixin (*class in angr.storage.memory_mixins*), 463
jni_references (*angr.sim_state.SimState attribute*), 239
jni_references (*angr.SimState attribute*), 208
job (*angr.analyses.forward_analysis.job_info.JobInfo property*), 937
job_type (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
JobInfo (*class in angr.analyses.forward_analysis.job_info*), 937
jobs (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis property*), 936
jobs (*angr.analyses.forward_analysis.ForwardAnalysis property*), 932
jobs (*angr.analyses.ForwardAnalysis property*), 907
join() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 1263
join_simtypes() (*angr.analyses.typehoon.simple_solver.SimpleSolver class method*), 1264
json_serialize_load_args() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 1030
json_serialize_load_args() (*angr.angrdb.serializers.LoaderSerializer static method*), 1026
Jump (*class in angr.ailment.statement*), 1400
jump() (*angr.sim_procedure.SimProcedure method*), 646
jump() (*angr.SimProcedure method*), 206
jump_guard (*angr.state_plugins.history.SimStateHistory attribute*), 375
jump_guards (*angr.state_plugins.history.SimStateHistory property*), 377
jump_guards (*angr.state_plugins.SimStateHistory property*), 303
jump_sources (*angr.state_plugins.history.SimStateHistory property*), 378
jump_sources (*angr.state_plugins.SimStateHistory property*), 303
jump_tables (*angr.analyses.cfg.cfg_fast.CFGFast property*), 992
jump_tables (*angr.analyses.cfg.CFGFast property*), 970
jump_tables (*angr.analyses.CFGFast property*), 892
jump_tables (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 756
jump_tables (*angr.knowledge_plugins.cfg.CFGModel attribute*), 742
jump_targets (*angr.state_plugins.history.SimStateHistory property*), 377
jump_targets (*angr.state_plugins.SimStateHistory property*), 303
jumpkind (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
jumpkind (*angr.engines.pcode.lifter.ExitStatement attribute*), 605
jumpkind (*angr.engines.pcode.lifter.IRSB attribute*), 608
jumpkind (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 771
jumpkind (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 753
jumpkind (*angr.state_plugins.history.SimStateHistory attribute*), 375
jumpkinds (*angr.state_plugins.history.SimStateHistory property*), 377
jumpkinds (*angr.state_plugins.SimStateHistory property*), 303
jumpout_sites (*angr.knowledge_plugins.Function property*), 720
jumpout_sites (*angr.knowledge_plugins.functions.Function property*), 780
jumpout_sites (*angr.knowledge_plugins.functions.function.Function property*), 804
jumptable (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 771
jumptable (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 753

jumptable_addr (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump property*), 772
jumptable_addr (*angr.knowledge_plugins.cfg.IndirectJump property*), 753
Jumptable_AddressComputed (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 769
Jumptable_AddressComputed (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 754
Jumptable_AddressLoadedFromMemory (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 769
Jumptable_AddressLoadedFromMemory (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 754
jumptable_entries (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump property*), 772
jumptable_entries (*angr.knowledge_plugins.cfg.IndirectJump property*), 753
jumptable_entry_size (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump property*), 772
jumptable_entry_size (*angr.knowledge_plugins.cfg.IndirectJump property*), 753
jumptable_size (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump property*), 772
jumptable_size (*angr.knowledge_plugins.cfg.IndirectJump property*), 753
JumpTableEntryConditionRewriter (*class in angr.analyses.decompiler.jumptable_entry_condition_rewriter*), 1071
JumptableInfo (*class in angr.knowledge_plugins.cfg.indirect_jump*), 770
JumpTableProcessor (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 1000
JumpTableProcessorState (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 1000
JumpTableResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1009
JumpTableResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 1002
jumptables (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 771
jumptables (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 753
JumpTargetBaseAddr (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 999
JumpTargetCollector (*class in angr.analyses.decompiler.jump_target_collector*), 1071

K

kb (*angr.analyses.Analysis attribute*), 881
kb (*angr.analyses.analysis.Analysis attribute*), 930
kb (*angr.analyses.decompiler.optimization_passes.optimization_pass.BaseO property*), 1093
kb (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimi attribute*), 1107
kb (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimi attribute*), 1106
kb (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimi attribute*), 1105
kb (*angr.Analysis attribute*), 156
kb (*angr.angrdb.models.DbCFGModel attribute*), 1023
kb (*angr.angrdb.models.DbComment attribute*), 1025
kb (*angr.angrdb.models.DbFunction attribute*), 1023
kb (*angr.angrdb.models.DbLabel attribute*), 1025
kb (*angr.angrdb.models.DbStructuredCode attribute*), 1024
kb (*angr.angrdb.models.DbVariableCollection attribute*), 1023
kb (*angr.angrdb.models.DbXRefs attribute*), 1024
kb (*angr.Project property*), 170
kb (*angr.project.Project property*), 224
kb_id (*angr.angrdb.models.DbCFGModel attribute*), 1022
kb_id (*angr.angrdb.models.DbComment attribute*), 1024
kb_id (*angr.angrdb.models.DbFunction attribute*), 1023
kb_id (*angr.angrdb.models.DbLabel attribute*), 1025
kb_id (*angr.angrdb.models.DbStructuredCode attribute*), 1024
kb_id (*angr.angrdb.models.DbVariableCollection attribute*), 1023
kb_id (*angr.angrdb.models.DbXRefs attribute*), 1024
keep_in_slice (*angr.ailment.tagged_object.TagDict attribute*), 1407
keep_path() (*angr.annocfg.AnnotatedCFG method*), 1322
key (*angr.angrdb.models.DbInformation attribute*), 1021
key (*angr.knowledge_plugins.propagations.PropagationModel attribute*), 774
key (*angr.sim_variable.SimConstantVariable property*), 681
key (*angr.sim_variable.SimMemoryVariable property*), 685
key (*angr.sim_variable.SimRegisterVariable property*), 683
key (*angr.sim_variable.SimStackVariable property*), 686
key (*angr.sim_variable.SimTemporaryVariable property*), 682

key (*angr.sim_variable.SimVariable property*), 681
KeyDefinitionManager (*class in angr.knowledge_plugins*), 730
KeyDefinitionManager (*class in angr.knowledge_plugins.key_definitions*), 823
KeyDefinitionManager (*class in angr.knowledge_plugins.key_definitions.key_definition_manager*), 845
KeyedRegion (*class in angr.keyed_region*), 868
keys() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict method*), 793
keys() (*angr.knowledge_plugins.patches.PatchManager method*), 738
keys() (*angr.knowledge_plugins.PatchManager method*), 732
keys() (*angr.state_plugins.globals.SimStateGlobals method*), 387
keys() (*angr.state_plugins.SimStateGlobals method*), 300
keys() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 502
keys() (*angr.vaults.Vault method*), 871
keys() (*angr.vaults.VaultDict method*), 872
keys() (*angr.vaults.VaultDir method*), 872
keys() (*angr.vaults.VaultDirShelf method*), 873
KeyValueMemory (*class in angr.storage.memory_mixins*), 466
KeyValueMemoryMixin (*class in angr.storage.memory_mixins*), 466
kill_and_add_definition() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1192
kill_and_add_definition() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1234
kill_and_add_definition() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1207
kill_and_add_definition() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 850
kill_and_add_definition() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 827
kill_def() (*angr.analyses.ddg.LiveDefinitions method*), 1158
kill_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 1201
kill_def() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 855
kill_def() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 832
kill_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1192
kill_definitions() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1234
kill_definitions() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1207
kill_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 850
kill_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 827
kind (*angr.knowledge_plugins.key_definitions.definition.DefinitionMatchPredicate attribute*), 842
KnowledgeBase (*class in angr*), 164
KnowledgeBase (*class in angr.knowledge_base*), 711
KnowledgeBasePlugin (*class in angr.knowledge_plugins*), 730
KnowledgeBasePlugin (*class in angr.knowledge_plugins.plugin*), 738
KnowledgeBaseSerializer (*class in angr.angrdb.serializers*), 1025
KnowledgeBaseSerializer (*class in angr.angrdb.serializers.kb*), 1027
KnownAnalysesPlugin (*class in angr.analyses.analysis*), 928
kwargs (*angr.utils.mp.Closure attribute*), 1354

L

label (*angr.storage.memory_object.SimLabeledMemoryObject attribute*), 451
Label (*class in angr.ailment.statement*), 1405
Label (*class in angr.analyses.disassembly*), 1293
Label (*class in angr.analyses.reassembler*), 1297
label_got() (*angr.analyses.reassembler.SymbolManager method*), 1298
LabeledMemory (*class in angr.storage.memory_mixins*), 467
LabeledPagesMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 507
LabelMergerMixin (*class in angr.storage.memory_mixins*), 466
LabelMergerMixin (*class in angr.storage.memory_mixins.label_merger_mixin*), 497
labels (*angr.analyses.typehoon.typevars.DerivedTypeVariable attribute*), 1270
labels (*angr.angrdb.models.DbKnowledgeBase attribute*), 1022
Labels (*class in angr.knowledge_plugins*), 731
Labels (*class in angr.knowledge_plugins.labels*), 775
LabelsSerializer (*class in angr.angrdb.serializers.labels*), 1028
LambdaAttriter (*class in angr.state_plugins.history*), 378

LambdaIterIter (*class in angr.state_plugins.history*), 378
language (*angr.analyses.LanguageDetector property*), 908
LANGUAGE (*angr.engines.pcode.cc.SimCCM68k attribute*), 636
LANGUAGE (*angr.engines.pcode.cc.SimCCPARISC attribute*), 637
LANGUAGE (*angr.engines.pcode.cc.SimCCPCodeBase attribute*), 636
LANGUAGE (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 637
LANGUAGE (*angr.engines.pcode.cc.SimCCRISCV attribute*), 636
LANGUAGE (*angr.engines.pcode.cc.SimCCSH4 attribute*), 637
LANGUAGE (*angr.engines.pcode.cc.SimCCSPARC attribute*), 637
LANGUAGE (*angr.engines.pcode.cc.SimCCXtensa attribute*), 638
LanguageDetector (*class in angr.analyses*), 908
languages() (*angr.Project method*), 172
languages() (*angr.project.Project method*), 227
LARGE_SWITCH (*angr.analyses.code_tagging.CodeTags attribute*), 1018
last_addr (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
last_addr (*angr.storage.memory_object.SimMemoryObject property*), 451
last_addr (*angr.storage.SimMemoryObject property*), 426
last_edge_hitmap (*angr.state_plugins.history.SimStateHistory property*), 378
last_edge_hitmap (*angr.state_plugins.SimStateHistory property*), 303
last_node() (*in module angr.analyses.decompiler.utils*), 1152
last_nonlabel_statement() (*in module angr.analyses.decompiler.utils*), 1152
lazy_import() (*in module angr.utils.lazy_import*), 1351
LEFT (*angr.analyses.typehoon.simple_solver.ConstraintGraphKeys attribute*), 1261
length (*angr.ailment.expression.Array property*), 1395
length (*angr.analyses.decompiler.structured_codegen.base.BaseNodeMapping attribute*), 1128
length (*angr.analyses.decompiler.structured_codegen.PositionMappingElement attribute*), 1125
length (*angr.sim_type.SimTypeArray attribute*), 696
length (*angr.state_plugins.sim_action_object.SimActionObject property*), 641
length (*angr.state_plugins.SimActionObject property*), 265
length (*angr.state_plugins.unicorn_engine.MEM_PATCH attribute*), 392
length (*angr.storage.memory_object.SimMemoryObject attribute*), 451
length (*angr.storage.SimMemoryObject attribute*), 425
length_spec (*angr.procedures.stubs.format_parser.FormatSpecifier attribute*), 647
LengthLimiter (*class in angr.exploration_techniques*), 558
LengthLimiter (*class in angr.exploration_techniques.lengthlimiter*), 571
Let (*class in angr.ailment.expression*), 1396
lhs (*angr.analyses.decompiler.structured_codegen.c.CAssignment attribute*), 1135
lhs (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp attribute*), 1141
libname (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
lift() (*angr.analyses.typehoon.translator.TypeTranslator method*), 1266
lift() (*angr.engines.light.engine.SimEngineLight method*), 1171
lift() (*angr.engines.light.SimEngineLight method*), 1168
lift() (*angr.engines.pcode.lifter.Lifter method*), 611
lift() (*angr.engines.pcode.lifter.PcodeBasicBlockLifter method*), 612
lift() (*angr.engines.pcode.lifter.PcodeLifter method*), 613
lift() (*in module angr.engines.pcode.lifter*), 611
lift_pcode() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 614
lift_soot() (*angr.engines.soot.engine.SootMixin method*), 602
lift_soot() (*angr.engines.soot.SootMixin method*), 601
lift_soot() (*angr.engines.SootMixin method*), 592
lift_vex() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 614
lift_vex() (*angr.engines.vex.VEXLifter method*), 600
Lifter (*class in angr.engines.pcode.lifter*), 610
likes() (*angr.ailment.Assignment method*), 1366
likes() (*angr.ailment.BinaryOp method*), 1367
likes() (*angr.ailment.Block method*), 1368
likes() (*angr.ailment.manager.Block.Block method*), 1407
likes() (*angr.ailment.Const method*), 1369
likes() (*angr.ailment.Expression method*), 1369
likes() (*angr.ailment.expression.Array method*), 1395
likes() (*angr.ailment.expression.BasePointerOffset method*), 1390
likes() (*angr.ailment.expression.BinaryOp method*), 1385
likes() (*angr.ailment.expression.Call method*), 1393
likes() (*angr.ailment.expression.ComboRegister*

method), 1378
likes() (angr.ailment.expression.Const method), 1377
likes() (angr.ailment.expression.Convert method), 1383
likes() (angr.ailment.expression.DirtyExpression method), 1388
likes() (angr.ailment.expression.Expression method), 1376
likes() (angr.ailment.expression.Extract method), 1392
likes() (angr.ailment.expression.FunctionLikeMacro method), 1396
likes() (angr.ailment.expression.Insert method), 1393
likes() (angr.ailment.expression.ITE method), 1387
likes() (angr.ailment.expression.Let method), 1396
likes() (angr.ailment.expression.Load method), 1386
likes() (angr.ailment.expression.MultiStatementExpression method), 1389
likes() (angr.ailment.expression.Phi method), 1380
likes() (angr.ailment.expression.Register method), 1378
likes() (angr.ailment.expression.Reinterpret method), 1383
likes() (angr.ailment.expression.RustEnum method), 1395
likes() (angr.ailment.expression.StringLiteral method), 1394
likes() (angr.ailment.expression.Struct method), 1395
likes() (angr.ailment.expression.Tmp method), 1377
likes() (angr.ailment.expression.UnaryOp method), 1381
likes() (angr.ailment.expression.VEXCCallExpression method), 1389
likes() (angr.ailment.expression.VirtualVariable method), 1380
likes() (angr.ailment.Register method), 1371
likes() (angr.ailment.Statement method), 1372
likes() (angr.ailment.statement.Assignment method), 1398
likes() (angr.ailment.statement.CAS method), 1404
likes() (angr.ailment.statement.ConditionalJump method), 1401
likes() (angr.ailment.statement.DirtyStatement method), 1405
likes() (angr.ailment.statement.Jump method), 1400
likes() (angr.ailment.statement.Label method), 1406
likes() (angr.ailment.statement.Return method), 1403
likes() (angr.ailment.statement.SideEffectStatement method), 1402
likes() (angr.ailment.statement.Statement method), 1397
likes() (angr.ailment.statement.Store method), 1400
likes() (angr.ailment.statement.WeakAssignment method), 1398
likes() (angr.ailment.Tmp method), 1372
likes() (angr.ailment.UnaryOp method), 1373
lineage (angr.state_plugins.history.SimStateHistory property), 377
lineage (angr.state_plugins.SimStateHistory property), 303
lineage() (angr.state_hierarchy.StateHierarchy method), 552
lineage() (angr.StateHierarchy method), 221
linux_syscall_update_error_reg() (angr.calling_conventions.SimCCSyscall method), 670
list_content (angr.utils.dynamic_dictlist.DynamicDictList attribute), 1346
list_default_plugins() (angr.misc.plugins.PluginPreset method), 237
ListPage (class in angr.storage.memory_mixins), 467
ListPage (class in angr.storage.memory_mixins.paged_memory.pages), 511
ListPage (class in angr.storage.memory_mixins.paged_memory.pages.list_ 521
ListPagesMixin (class in angr.storage.memory_mixins), 468
ListPagesMixin (class in angr.storage.memory_mixins.paged_memory.paged_memory_mix 507
ListPagesWithLabelsMixin (class in angr.storage.memory_mixins), 469
ListPagesWithLabelsMixin (class in angr.storage.memory_mixins.paged_memory.paged_memory_mix 508
live_definitions (angr.analyses.reaching_definitions.rd_state.Reaching attribute), 1231
live_definitions (angr.analyses.reaching_definitions.ReachingDefinitio attribute), 1205
live_vars_by_stmt() (angr.analyses.SLivenessAnalysis method), 919
LiveDefinitions (class in angr.analyses.ddg), 1157
LiveDefinitions (class in angr.analyses.reaching_definitions), 1189
LiveDefinitions (class in angr.knowledge_plugins.key_definitions), 824
LiveDefinitions (class in angr.knowledge_plugins.key_definitions.live_definitions), 846
LiveVariables (class in angr.knowledge_plugins.variables.variable_manager), 813
llm_client (angr.Project property), 170
llm_client (angr.project.Project property), 224
llm_refine() (angr.analyses.Decompiler method), 902
llm_refine() (angr.analyses.decompiler.Decompiler

method), 1048
llm_refine() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1067
llm_suggest_function_name() (*angr.analyses.Decompiler method*), 902
llm_suggest_function_name() (*angr.analyses.decompiler.Decompiler method*), 1048
llm_suggest_function_name() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1067
llm_suggest_variable_names() (*angr.analyses.Decompiler method*), 902
llm_suggest_variable_names() (*angr.analyses.decompiler.Decompiler method*), 1048
llm_suggest_variable_names() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1067
llm_suggest_variable_types() (*angr.analyses.Decompiler method*), 903
llm_suggest_variable_types() (*angr.analyses.decompiler.Decompiler method*), 1048
llm_suggest_variable_types() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1068
llm_summarize_function() (*angr.analyses.Decompiler method*), 903
llm_summarize_function() (*angr.analyses.decompiler.Decompiler method*), 1049
llm_summarize_function() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1068
LLMClient (*class in angr*), 165
Load (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTarget attribute*), 998
Load (*class in angr.ailment.expression*), 1385
Load (*class in angr.analyses.typehoon.typevars*), 1271
Load() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
load() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1245
load() (*angr.angrdb.AngrDB method*), 1020
load() (*angr.angrdb.db.AngrDB method*), 1021
load() (*angr.angrdb.serializers.cfg_model.CFGModelSerializer static method*), 1027
load() (*angr.angrdb.serializers.comments.CommentsSerializer static method*), 1027
load() (*angr.angrdb.serializers.funcs.FunctionManagerSerializer static method*), 1027
load() (*angr.angrdb.serializers.kb.KnowledgeBaseSerializer static method*), 1028
load() (*angr.angrdb.serializers.KnowledgeBaseSerializer static method*), 1025
load() (*angr.angrdb.serializers.labels.LabelsSerializer static method*), 1028
load() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 1030
load() (*angr.angrdb.serializers.LoaderSerializer static method*), 1026
load() (*angr.angrdb.serializers.structured_code.StructuredCodeManagerS static method*), 1032
load() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 1031
load() (*angr.angrdb.serializers.xrefs.XRefsSerializer static method*), 1030
load() (*angr.state_plugins.light_registers.SimLightRegisters method*), 375
load() (*angr.state_plugins.SimLightRegisters method*), 281
load() (*angr.storage.memory_mixins.actions_mixin.ActionsMixinHigh method*), 493
load() (*angr.storage.memory_mixins.actions_mixin.ActionsMixinLow method*), 493
load() (*angr.storage.memory_mixins.ActionsMixinHigh method*), 456
load() (*angr.storage.memory_mixins.ActionsMixinLow method*), 456
load() (*angr.storage.memory_mixins.address_concretization_mixin.Addre method*), 497
load() (*angr.storage.memory_mixins.AddressConcretizationMixin method*), 458
load() (*angr.storage.memory_mixins.bvv_conversion_mixin.DataNormaliz method*), 491
load() (*angr.storage.memory_mixins.conditional_store_mixin.Conditional method*), 497
load() (*angr.storage.memory_mixins.ConditionalMixin method*), 459
load() (*angr.storage.memory_mixins.DataNormalizationMixin method*), 460
load() (*angr.storage.memory_mixins.InspectMixin method*), 463
load() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 464
load() (*angr.storage.memory_mixins.KeyValueMemoryMixin method*), 466
load() (*angr.storage.memory_mixins.ListPage method*), 467
load() (*angr.storage.memory_mixins.MemoryRegionMetaMixin method*), 472
load() (*angr.storage.memory_mixins.MVListPage method*), 469
load() (*angr.storage.memory_mixins.name_resolution_mixin.NameResolu method*), 490
load() (*angr.storage.memory_mixins.NameResolutionMixin method*), 474

load() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixins.PagedMemoryMixin method*), 504
load() (*angr.storage.memory_mixins.paged_memory.pages.ListPage method*), 521
load() (*angr.storage.memory_mixins.paged_memory.pages.MemoryObjectPage method*), 511
load() (*angr.storage.memory_mixins.paged_memory.pages.MultiMemoryObjectPage method*), 500
load() (*angr.storage.memory_mixins.paged_memory.pages.SimpleMemoryObjectPage method*), 513
load() (*angr.storage.memory_mixins.paged_memory.pages.SinglePage method*), 523
load() (*angr.storage.memory_mixins.paged_memory.pages.UltraPage method*), 517
load() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 474
load() (*angr.storage.memory_mixins.regioned_memory.MemoryRegionMetaMixin method*), 525
load() (*angr.storage.memory_mixins.regioned_memory.region_types.MemoryObjectRegion method*), 537
load() (*angr.storage.memory_mixins.regioned_memory.region_types.MemoryRegion method*), 531
load() (*angr.storage.memory_mixins.regioned_memory.RegionedMemoryMixin method*), 528
load() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 481
load() (*angr.storage.memory_mixins.simple_interface_mixin.SimpleInterfaceMixin method*), 493
load() (*angr.storage.memory_mixins.SimpleInterfaceMixin method*), 483
load() (*angr.storage.memory_mixins.size_resolution_mixin.SizeConcretizationMixin method*), 495
load() (*angr.storage.memory_mixins.size_resolution_mixin.SizeNormalizationMixin method*), 494
load() (*angr.storage.memory_mixins.SizeConcretizationMixin method*), 484
load() (*angr.storage.memory_mixins.SizeNormalizationMixin method*), 484
load() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method*), 540
load() (*angr.storage.memory_mixins.SlottedMemoryMixin method*), 485
load() (*angr.storage.memory_mixins.UltraPage method*), 487
load() (*angr.storage.memory_mixins.underconstrained_mixin.UnderconstrainedMixin method*), 493
load() (*angr.storage.memory_mixins.UnderconstrainedMixin method*), 489
load() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 498
load() (*angr.storage.memory_mixins.UnwrapperMixin method*), 490
load() (*angr.vaults.Vault method*), 872
load() (*angr.vaults.VaultDirShelf method*), 873
load_all_definitions() (*in module angr.procedures.definitions*), 658
load_all_listed() (*angr.knowledge_plugins.functions.function_manag method*), 794
load_annotations() (*angr.storage.memory_mixins.PagedMemoryMultiV method*), 477
LOAD_ARG_BLACKLIST (*angr.angrdb.serializers.loader.LoaderSerializer attribute*), 1029
LOAD_ARG_BLACKLIST (*angr.angrdb.serializers.LoaderSerializer attribute*), 1026
load_array_element() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 464
load_array_elements() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 464
load_definitions() (*in module angr.procedures.definitions*), 658
load_from_dwarf() (*angr.knowledge_plugins.debug_variables.DebugVar method*), 822
load_from_dwarf() (*angr.knowledge_plugins.DebugVariableManager method*), 716
load_from_dwarf() (*angr.knowledge_plugins.VariableManager method*), 736
load_from_dwarf() (*angr.knowledge_plugins.variables.variable_manage method*), 820
load_from_dwarf() (*angr.knowledge_plugins.variables.VariableManager method*), 811
load_internal() (*angr.angrdb.serializers.variables.VariableManagerSer static method*), 1031
load_shellcode() (*in module angr*), 222
load_shellcode() (*in module angr.project*), 223
load_signatures() (*in module angr.flirt*), 1342
load_signatures() (*in module angr.flirt*), 1343
load_type_collections() (*in module angr.procedures.definitions*), 658
load_win32_type_collections() (*in module angr.procedures.definitions*), 658
load_win32_api_definitions() (*in module angr.procedures.definitions*), 658
load_with_labels() (*angr.storage.memory_mixins.paged_memory.paged method*), 507
LoadArgsJSONDecoder (*class in angr.angrdb.serializers.loader*), 1029
LoadArgsJSONEncoderMixin (*class in angr.angrdb.serializers.loader*), 1028
LoaderSerializer (*class in angr.angrdb.serializers*), 1025
LoaderSerializer (*class in angr.angrdb.serializers.loader*), 1029
LoadG() (*angr.ailment.converter_vex.VEXStmtConverter static method*), 1375
LoadHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 1001

loads() (*angr.vaults.Vault method*), 872
loc (*angr.analyses.typehoon.typevars.FuncIn attribute*), 1271
loc (*angr.analyses.typehoon.typevars.FuncOut attribute*), 1271
loc_repr() (*angr.sim_variable.SimComboRegisterVariable method*), 684
loc_repr() (*angr.sim_variable.SimConstantVariable method*), 681
loc_repr() (*angr.sim_variable.SimMemoryVariable method*), 685
loc_repr() (*angr.sim_variable.SimRegisterVariable method*), 683
loc_repr() (*angr.sim_variable.SimStackVariable method*), 686
loc_repr() (*angr.sim_variable.SimTemporaryVariable method*), 682
loc_repr() (*angr.sim_variable.SimVariable method*), 681
local_runtime_values (*angr.knowledge_plugins.Function property*), 719
local_runtime_values (*angr.knowledge_plugins.functions.Function property*), 780
local_runtime_values (*angr.knowledge_plugins.functions.function.Function property*), 804
local_types (*angr.analyses.decompiler.decompilation_cache.DecompilationCache property*), 1065
local_vars (*angr.sim_procedure.SimProcedure attribute*), 645
local_vars (*angr.SimProcedure attribute*), 204
LOCALE_ARRAY (*angr.state_plugins.libc.SimStateLibc attribute*), 343
LOCALE_ARRAY (*angr.state_plugins.SimStateLibc attribute*), 307
LocalLoopSeer (*class in angr.exploration_techniques*), 558
LocalLoopSeer (*class in angr.exploration_techniques.local_loop_seer*), 583
LocalVariableTag (*class in angr.knowledge_plugins.key_definitions.tag*), 858
location (*angr.knowledge_plugins.variables.variable_access.VariableAccess attribute*), 812
LocationBase (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1109
log (*angr.analyses.Analysis attribute*), 881
log (*angr.analyses.analysis.Analysis attribute*), 930
log (*angr.Analysis attribute*), 156
long_reason (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 766
longest_prefix() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 1270
longest_prefix_lookup() (*in module angr.engines.light.engine*), 1172
looks_like_sql() (*in module angr.utils*), 1344
lookup() (*angr.analyses.typehoon.simple_solver.Sketch method*), 1260
lookup() (*angr.knowledge_plugins.Labels method*), 731
lookup() (*angr.knowledge_plugins.labels.Labels method*), 776
lookup() (*angr.SimMount method*), 195
lookup() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 359
lookup() (*angr.state_plugins.filesystem.SimMount method*), 358
lookup() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 402
lookup() (*angr.state_plugins.posix.PosixDevFS method*), 347
lookup() (*angr.state_plugins.posix.PosixProcFS method*), 349
lookup() (*angr.state_plugins.PosixDevFS method*), 259
lookup() (*angr.state_plugins.PosixProcFS method*), 261
lookup() (*angr.state_plugins.SimMount method*), 286
lookup() (*angr.state_plugins.SimStateJNIReferences method*), 305
lookup_defs() (*angr.analyses.ddg.LiveDefinitions method*), 1158
lookup_original() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 384
Loop (*class in angr.analyses.loopfinder*), 1283
loop_nesting_forest() (*angr.utils.graph.GraphUtils static method*), 1350
loop_nodes (*angr.analyses.decompiler.region_simplifiers.expr_folding.LoopNodeFinder attribute*), 1111
LoopAnalysis (*class in angr.analyses*), 909
LoopAnalysis (*class in angr.analyses.loop_analysis*), 1284
LoopFinder (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
LoopFinder (*class in angr.analyses*), 909
LoopFinder (*class in angr.analyses.loopfinder*), 1283
looping_times (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 769
looping_times (*angr.knowledge_plugins.cfg.CFGENode attribute*), 741
LoopNodeFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1111
LoopSeer (*class in angr.exploration_techniques*), 559
LoopSeer (*class in angr.exploration_techniques.loop_seer*), 582
LoopSimplifier (*class in*

angr.analyses.decompiler.region_simplifiers.loop), 1114
LoopUnroller (*class in angr.analyses*), 909
LoopVisitor (*class in angr.analyses.forward_analysis*), 933
LoopVisitor (*class in angr.analyses.forward_analysis.visitors*), 939
LoopVisitor (*class in angr.analyses.forward_analysis.visitors.loop*), 944
lower_bound (*angr.analyses.typehoon.simple_solver.SketchMake attribute*), 1260
LoweredSwitchSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1082
LoweredSwitchSimplifier (*class in angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier*), 1100
LShift (*angr.engines.light.ArithmeticExpression attribute*), 1167
LShift (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
LT (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier attribute*), 1115

M

Macro (*class in angr.ailment.expression*), 1396
maddr (*angr.ailment.expression.DirtyExpression attribute*), 1388
main() (*in module angr.flirt.build_sig*), 1343
main_executable_region_limbos_contain() (*angr.analyses.Reassembler method*), 917
main_executable_region_limbos_contain() (*angr.analyses.reassembler.Reassembler method*), 1303
main_executable_regions (*angr.analyses.Reassembler property*), 916
main_executable_regions (*angr.analyses.reassembler.Reassembler property*), 1302
main_executable_regions_contain() (*angr.analyses.Reassembler method*), 916
main_executable_regions_contain() (*angr.analyses.reassembler.Reassembler method*), 1302
main_nonexecutable_region_limbos_contain() (*angr.analyses.Reassembler method*), 917
main_nonexecutable_region_limbos_contain() (*angr.analyses.reassembler.Reassembler method*), 1303
main_nonexecutable_regions (*angr.analyses.Reassembler property*), 916
main_nonexecutable_regions (*angr.analyses.reassembler.Reassembler property*), 1302
main_nonexecutable_regions_contain() (*angr.analyses.Reassembler method*), 917
main_nonexecutable_regions_contain() (*angr.analyses.reassembler.Reassembler method*), 1303
main_object (*angr.angrdb.models.DbObject attribute*), 1022
make() (*angr.sim_type.SimTypePointer method*), 695
make() (*angr.sim_type.SimTypeReference method*), 695
MAKE_ARGUMENT_LIST (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1057
make_breakpoint() (*angr.state_plugins.inspect.SimInspector method*), 340
make_breakpoint() (*angr.state_plugins.SimInspector method*), 276
make_bvs_zero_fill() (*in module angr.engines.pcode.behavior*), 616
MAKE_CALLSITES (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1057
make_child() (*angr.state_plugins.history.SimStateHistory method*), 378
make_child() (*angr.state_plugins.SimStateHistory method*), 303
make_concrete_int() (*angr.sim_state.SimState method*), 242
make_concrete_int() (*angr.SimState method*), 211
make_continuation() (*angr.sim_procedure.SimProcedure method*), 645
make_continuation() (*angr.SimProcedure method*), 204
make_copy() (*angr.analyses.cfg.cfg_base.CFGBase method*), 982
make_copy() (*angr.analyses.cfg.CFGBase method*), 961
make_extern() (*angr.code_location.AILCodeLocation static method*), 867
make_function_codeloc() (*angr.analyses.reaching_definitions.function_handler.FunctionHa method*), 1227
make_function_codeloc() (*angr.analyses.reaching_definitions.FunctionHandler method*), 1186
make_functions() (*angr.analyses.cfg.cfg_base.CFGBase method*), 983
make_functions() (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot method*), 1017
make_functions() (*angr.analyses.cfg.CFGBase method*), 962
make_functions() (*angr.analyses.cfg.CFGFastSoot method*), 972
make_functions() (*angr.analyses.CFGFastSoot method*), 894

make_ident() (*angr.SimFileBase static method*), 181
make_ident() (*angr.storage.file.SimFileBase static method*), 433
make_initial_state() (*angr.analyses.Identifier static method*), 908
make_initial_state() (*angr.analyses.identifier.identify.Identifier static method*), 1283
make_liveness_snapshot() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 1202
make_liveness_snapshot() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 856
make_liveness_snapshot() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 833
make_phi_node() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 814
make_return_edges() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 992
make_return_edges() (*angr.analyses.cfg.CFGFast method*), 970
make_return_edges() (*angr.analyses.CFGFast method*), 892
MAKE_RETURN_SITES (*angr.analyses.decompiler.clinic.Clinic attribute*), 1057
make_symbolic_state() (*angr.analyses.Identifier static method*), 908
make_symbolic_state() (*angr.analyses.identifier.identify.Identifier static method*), 1283
MakeTypecastsImplicit (*class in angr.analyses.decompiler.structured_codegen.c*), 1148
malloc() (*angr.SimHeapPTMalloc method*), 192
malloc() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 415
malloc() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 418
malloc() (*angr.state_plugins.heap.SimHeapLibc method*), 408
malloc() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 410
malloc() (*angr.state_plugins.SimHeapLibc method*), 272
malloc() (*angr.state_plugins.SimHeapPTMalloc method*), 274
manager (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase attribute*), 1107
manager (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationMultiStmtBase attribute*), 1106
manager (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase attribute*), 1105
manager (*angr.knowledge_plugins.variables.variable_manager.VariableManager attribute*), 813
Manager (*class in angr.ailment*), 1370
Manager (*class in angr.ailment.manager*), 1407
MANUAL (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypeKind attribute*), 739
ManualMergepoint (*class in angr.exploration_techniques*), 560
ManualMergepoint (*class in angr.exploration_techniques.manual_mergepoint*), 571
map() (*angr.storage.memory_mixins.MemoryMixin method*), 534
map_addr_to_label (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWriter attribute*), 1146
map_ast_to_pos (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator attribute*), 1119
map_ast_to_pos (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWriter attribute*), 1146
map_callsites() (*angr.analyses.Identifier method*), 907
map_callsites() (*angr.analyses.identifier.identify.Identifier method*), 1283
map_region() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 497
map_region() (*angr.storage.memory_mixins.AddressConcretizationMixin method*), 458
map_region() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 506
map_region() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 476
mapped_address (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
mapped_address (*angr.state_plugins.InspectAttrs attribute*), 255
mapped_page (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
mapped_page (*angr.state_plugins.InspectAttrs attribute*), 255
mark_const() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1237
mark_const() (*angr.analyses.reaching_definitions.ReachingDefinitionsStateBase method*), 1210
mark_dirty() (*angr.knowledge_plugins.Function method*), 720
mark_dirty() (*angr.knowledge_plugins.functions.Function method*), 781
mark_dirty() (*angr.knowledge_plugins.functions.function.Function method*), 805
mark_function_alignments() (*angr.analyses.cfg.cfg_fast.CFGBase method*), 983
mark_function_alignments() (*angr.analyses.CFGBase*

(*angr.analyses.cfg.CFGBase method*), 962
mark_guard() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1237
mark_guard() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 1210
mark_node_addr_has_return() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 757
mark_node_addr_has_return() (*angr.knowledge_plugins.cfg.CFGModel method*), 743
mark_nofilter() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 571
mark_nofilter() (*angr.exploration_techniques.ManualMergepoint method*), 560
mark_nonreturning_calls_endpoints() (*angr.knowledge_plugins.Function method*), 721
mark_nonreturning_calls_endpoints() (*angr.knowledge_plugins.functions.Function method*), 781
mark_nonreturning_calls_endpoints() (*angr.knowledge_plugins.functions.function.Function method*), 806
mark_okfilter() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 571
mark_okfilter() (*angr.exploration_techniques.ManualMergepoint method*), 561
marker_to_state() (*angr.analyses.Pathfinder method*), 910
match_stmt_classes() (*in module angr.analyses.decompiler.utils*), 1154
matches() (*angr.ailment.Assignment method*), 1366
matches() (*angr.ailment.BinaryOp method*), 1367
matches() (*angr.ailment.Const method*), 1369
matches() (*angr.ailment.Expression method*), 1369
matches() (*angr.ailment.expression.Array method*), 1395
matches() (*angr.ailment.expression.BasePointerOffset method*), 1390
matches() (*angr.ailment.expression.BinaryOp method*), 1385
matches() (*angr.ailment.expression.Call method*), 1393
matches() (*angr.ailment.expression.ComboRegister method*), 1378
matches() (*angr.ailment.expression.Const method*), 1377
matches() (*angr.ailment.expression.Convert method*), 1383
matches() (*angr.ailment.expression.DirtyExpression method*), 1388
matches() (*angr.ailment.expression.Expression method*), 1376
matches() (*angr.ailment.expression.Extract method*), 1392
matches() (*angr.ailment.expression.FunctionLikeMacro method*), 1396
matches() (*angr.ailment.expression.Insert method*), 1393
matches() (*angr.ailment.expression.ITE method*), 1387
matches() (*angr.ailment.expression.Let method*), 1396
matches() (*angr.ailment.expression.Load method*), 1386
matches() (*angr.ailment.expression.MultiStatementExpression method*), 1389
matches() (*angr.ailment.expression.Phi method*), 1380
matches() (*angr.ailment.expression.Register method*), 1378
matches() (*angr.ailment.expression.Reinterpret method*), 1383
matches() (*angr.ailment.expression.RustEnum method*), 1395
matches() (*angr.ailment.expression.StringLiteral method*), 1394
matches() (*angr.ailment.expression.Struct method*), 1395
matches() (*angr.ailment.expression.Tmp method*), 1377
matches() (*angr.ailment.expression.UnaryOp method*), 1381
matches() (*angr.ailment.expression.VEXCCallExpression method*), 1389
matches() (*angr.ailment.expression.VirtualVariable method*), 1380
matches() (*angr.ailment.Register method*), 1371
matches() (*angr.ailment.Statement method*), 1372
matches() (*angr.ailment.statement.Assignment method*), 1398
matches() (*angr.ailment.statement.CAS method*), 1404
matches() (*angr.ailment.statement.ConditionalJump method*), 1401
matches() (*angr.ailment.statement.DirtyStatement method*), 1405
matches() (*angr.ailment.statement.Jump method*), 1400
matches() (*angr.ailment.statement.Label method*), 1406
matches() (*angr.ailment.statement.Return method*), 1403
matches() (*angr.ailment.statement.SideEffectStatement method*), 1402
matches() (*angr.ailment.statement.Statement method*), 1397
matches() (*angr.ailment.statement.Store method*), 1400
matches() (*angr.ailment.statement.WeakAssignment method*), 1398
matches() (*angr.ailment.Tmp method*), 1372
matches() (*angr.ailment.UnaryOp method*), 1373
matches() (*angr.analyses.reaching_definitions.Definition method*), 1182
matches() (*angr.knowledge_plugins.key_definitions.Definition method*), 823

matches() (*angr.knowledge_plugins.key_definitions.definition.Definition method*), 844
matches() (*angr.knowledge_plugins.key_definitions.definition.DefinitionMatchPredicate method*), 842
max() (*angr.state_plugins.SimSolver method*), 291
max() (*angr.state_plugins.solver.SimSolver method*), 365
max_allocation (*angr.state_plugins.cgc.SimStateCGC attribute*), 380
max_allocation (*angr.state_plugins.SimStateCGC attribute*), 297
max_bytes (*angr.engines.pcode.lifter.Lifter attribute*), 611
max_bytes (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
MAX_DATA_REFS (*angr.engines.pcode.lifter.IRSB attribute*), 607
MAX_EXITS (*angr.engines.pcode.lifter.IRSB attribute*), 607
max_inst (*angr.engines.pcode.lifter.Lifter attribute*), 610
max_inst (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
max_int() (*angr.state_plugins.SimSolver method*), 296
max_int() (*angr.state_plugins.solver.SimSolver method*), 369
MAX_ONE_CALL (*angr.analyses.decompiler.structuring.phoenix.MultiSwitchPatternMatcher attribute*), 1037
max_size (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 765
max_size (*angr.knowledge_plugins.cfg.MemoryData attribute*), 754
max_size (*angr.utils.dynamic_dictlist.DynamicDictList attribute*), 1346
max_tv_id (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 1065
max_tv_id (*angr.analyses.typehoon.typevars.TypeVariableManager property*), 1268
maximize_similarity_of_blocks() (*angr.analyses.decompiler.optimization_passes.DuplicateReturn method*), 1077
maximum_syscall_number() (*angr.procedures.definitions.SimSyscallLibrary method*), 655
meet() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 1264
meet_simtypes() (*angr.analyses.typehoon.simple_solver.SimpleSolver class method*), 1264
mem (*angr.sim_state.SimState attribute*), 239
mem (*angr.SimState attribute*), 208
mem (*angr.state_plugins.debug_variables.SimDebugVariable property*), 421
MEM (*angr.state_plugins.sim_action.SimAction attribute*), 638
MEM (*angr.state_plugins.SimAction attribute*), 263
mem (*angr.state_plugins.SimDebugVariable property*), 266
mem() (*angr.analyses.reaching_definitions.Atom static method*), 1280
mem() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 838
mem_concrete() (*angr.sim_state.SimState method*), 242
mem_concrete() (*angr.SimState method*), 210
mem_deltas (*angr.analyses.stack_pointer_tracker.StackPointerTracker attribute*), 1246
mem_map() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 396
mem_map_ptr() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 396
MEM_PATCH (*class in angr.state_plugins.unicorn_engine*), 392
MEM_READ (*angr.state_plugins.inspect.EventType attribute*), 331
mem_read_address (*angr.state_plugins.inspect.InspectAttrs attribute*), 334
mem_read_address (*angr.state_plugins.InspectAttrs attribute*), 253
mem_read_condition (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
mem_read_condition (*angr.state_plugins.InspectAttrs attribute*), 253
mem_read_endness (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
mem_read_endness (*angr.state_plugins.InspectAttrs attribute*), 253
mem_read_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 334
mem_read_expr (*angr.state_plugins.InspectAttrs attribute*), 253
mem_read_length (*angr.state_plugins.inspect.InspectAttrs attribute*), 334
mem_read_length (*angr.state_plugins.InspectAttrs attribute*), 253
mem_reset() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 396
mem_unmap() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 396
mem_untyped (*angr.state_plugins.debug_variables.SimDebugVariable property*), 421
mem_untyped (*angr.state_plugins.SimDebugVariable property*), 266
mem_values (*angr.analyses.stack_pointer_tracker.StackPointerTracker attribute*), 1246
MEM_WRITE (*angr.state_plugins.inspect.EventType attribute*), 331
mem_write_address (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
mem_write_address (*angr.state_plugins.InspectAttrs attribute*), 253
mem_write_condition

(*angr.state_plugins.inspect.InspectAttrs attribute*), 335
mem_write_condition (*angr.state_plugins.InspectAttrs attribute*), 253
mem_write_endness (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
mem_write_endness (*angr.state_plugins.InspectAttrs attribute*), 253
mem_write_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
mem_write_expr (*angr.state_plugins.InspectAttrs attribute*), 253
mem_write_length (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
mem_write_length (*angr.state_plugins.InspectAttrs attribute*), 253
member() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 422
member() (*angr.state_plugins.SimDebugVariable method*), 267
member() (*angr.state_plugins.SimMemView method*), 285
member() (*angr.state_plugins.view.SimMemView method*), 431
members (*angr.analyses.typehoon.typeconsts.Enum attribute*), 1280
members (*angr.sim_type.SimCppClass property*), 706
members (*angr.sim_type.SimTypeEnum attribute*), 703
MemDepNode (*class in angr.analyses.data_dep*), 1318
MemDepNode (*class in angr.analyses.data_dep.dep_nodes*), 1314
memo (*angr.analyses.typehoon.translator.TypeTranslator attribute*), 1266
memo() (*angr.SimStatePlugin static method*), 211
memo() (*angr.state_plugins.plugin.SimStatePlugin static method*), 329
memo() (*angr.state_plugins.SimStatePlugin static method*), 314
memoize() (*in module angr.analyses.typehoon.typeconsts*), 1274
MEMORY (*angr.ailment.expression.VirtualVariableCategory attribute*), 1379
Memory (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 1313
Memory (*angr.analyses.data_dep.DepNodeTypes attribute*), 1317
MEMORY (*angr.analyses.reaching_definitions.AtomKind attribute*), 1181
memory (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
memory (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 1233
memory (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 1206
memory (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerSta attribute*), 1244
memory (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState attribute*), 1244
MEMORY (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 836
memory (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefini attribute*), 848
memory (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 824
MEMORY (*angr.knowledge_plugins.variables.variable_manager.VariableType attribute*), 813
MEMORY (*angr.knowledge_plugins.variables.VariableType attribute*), 812
memory (*angr.sim_state.SimState attribute*), 239
memory (*angr.SimState attribute*), 208
memory() (*angr.analyses.reaching_definitions.Atom static method*), 1180
memory() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 838
memory_args (*angr.calling_conventions.SimCC property*), 664
memory_args (*angr.SimCC property*), 174
memory_data (*angr.analyses.cfg.cfg_fast.CFGFast property*), 992
memory_data (*angr.analyses.cfg.CFGFast property*), 970
memory_data (*angr.analyses.CFGFast property*), 892
memory_data (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 756
memory_data (*angr.knowledge_plugins.cfg.CFGModel attribute*), 743
memory_data (*angr.knowledge_plugins.xrefs.XRef attribute*), 861
memory_data (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 863
MEMORY_ERROR (*angr.EmulatorStopReason attribute*), 162
MEMORY_PAGE_MAP (*angr.state_plugins.inspect.EventType attribute*), 332
memory_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
memory_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefini property*), 1233
memory_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 1206
memory_uses (*angr.knowledge_plugins.key_definitions.live_definitions.Live attribute*), 848
memory_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 825
memory_values (*angr.state_plugins.unicorn_engine.VEXStmtDetails attribute*), 393
memory_values_count (*angr.state_plugins.unicorn_engine.VEXStmtDetails*

attribute), 393
MemoryData (*class in angr.knowledge_plugins.cfg*), 754
MemoryData (*class in angr.knowledge_plugins.cfg.memory_data*), 764
MemoryDataSort (*class in angr.knowledge_plugins.cfg*), 755
MemoryDataSort (*class in angr.knowledge_plugins.cfg.memory_data*), 764
MemoryLoadResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1010
MemoryLocation (*class in angr.analyses.reaching_definitions*), 1197
MemoryLocation (*class in angr.knowledge_plugins.key_definitions.atoms*), 840
MemoryMappingError, 396
MemoryObjectMixin (*class in angr.storage.memory_mixins*), 471
MemoryObjectMixin (*class in angr.storage.memory_mixins.paged_memory.pages*), 515
MemoryObjectMixin (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 520
MemoryObjectSetMixin (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 520
MemoryOperand (*class in angr.analyses.disassembly*), 1295
MemoryRegion (*class in angr.analyses.cfg.cfb*), 972
MemoryRegionMetaMixin (*class in angr.storage.memory_mixins*), 471
MemoryRegionMetaMixin (*class in angr.storage.memory_mixins.regioned_memory*), 524
MemoryRegionMetaMixin (*class in angr.storage.memory_mixins.regioned_memory.region_meta_mixin*), 536
MemoryValue (*class in angr.state_plugins.unicorn_engine*), 392
MemoryWatcher (*class in angr.exploration_techniques*), 561
MemoryWatcher (*class in angr.exploration_techniques.memory_watcher*), 585
merge() (*angr.analyses.decompiler.optimization_passes.engine_base.SimEngineRDUBase method*), 1101
merge() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1192
merge() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1234
merge() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 1202
merge() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1207
merge() (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerS method*), 1244
merge() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 1245
merge() (*angr.analyses.variable_recovery.variable_recovery.VariableReco method*), 1255
merge() (*angr.analyses.variable_recovery.variable_recovery_fast.Variable method*), 1252
merge() (*angr.concretization_strategies.norepeats.SimConcretizationStrate method*), 541
merge() (*angr.concretization_strategies.norepeats_range.SimConcretizatio method*), 543
merge() (*angr.concretization_strategies.SimConcretizationStrategy method*), 452
merge() (*angr.concretization_strategies.SimConcretizationStrategyNorepe method*), 454
merge() (*angr.concretization_strategies.SimConcretizationStrategyNorepe method*), 455
merge() (*angr.keyed_region.KeyedRegion method*), 868
merge() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 768
merge() (*angr.knowledge_plugins.cfg.CFGNode method*), 751
merge() (*angr.knowledge_plugins.key_definitions.environment.Environmen method*), 844
merge() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi method*), 850
merge() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 827
merge() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefin method*), 856
merge() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsMod method*), 833
merge() (*angr.knowledge_plugins.key_definitions.Uses method*), 836
merge() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 860
merge() (*angr.sim_manager.SimulationManager method*), 550
merge() (*angr.sim_state.SimState method*), 241
merge() (*angr.SimFile method*), 180
merge() (*angr.SimFileDescriptor method*), 184
merge() (*angr.SimFileDescriptorDuplex method*), 186
merge() (*angr.SimFileStream method*), 188
merge() (*angr.SimFileHeapBase method*), 190
merge() (*angr.SimHeapPTMalloc method*), 192
merge() (*angr.SimPackets method*), 199
merge() (*angr.SimPacketsStream method*), 201
merge() (*angr.SimSite method*), 210
merge() (*angr.SimStatePlugin method*), 212
merge() (*angr.SimulationManager method*), 220

merge() (*angr.state_plugins.CallStack method*), 248
merge() (*angr.state_plugins.callstack.CallStack method*), 372
merge() (*angr.state_plugins.cgc.SimStateCGC method*), 380
merge() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 359
merge() (*angr.state_plugins.filesystem.SimFilesystem method*), 356
merge() (*angr.state_plugins.globals.SimStateGlobals method*), 386
merge() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 413
merge() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 419
merge() (*angr.state_plugins.heap.SimHeapBrk method*), 407
merge() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 410
merge() (*angr.state_plugins.history.SimStateHistory method*), 375
merge() (*angr.state_plugins.inspect.SimInspector method*), 342
merge() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 401
merge() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 403
merge() (*angr.state_plugins.libc.SimStateLibc method*), 346
merge() (*angr.state_plugins.log.SimStateLog method*), 370
merge() (*angr.state_plugins.loop_data.SimStateLoopData method*), 399
merge() (*angr.state_plugins.plugin.SimStatePlugin method*), 330
merge() (*angr.state_plugins.posix.PosixDevFS method*), 347
merge() (*angr.state_plugins.posix.PosixProcFS method*), 349
merge() (*angr.state_plugins.posix.SimSystemPosix method*), 353
merge() (*angr.state_plugins.PosixDevFS method*), 259
merge() (*angr.state_plugins.PosixProcFS method*), 261
merge() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 390
merge() (*angr.state_plugins.scratch.SimStateScratch method*), 389
merge() (*angr.state_plugins.SimFilesystem method*), 268
merge() (*angr.state_plugins.SimHeapBrk method*), 271
merge() (*angr.state_plugins.SimHeapPTMalloc method*), 274
merge() (*angr.state_plugins.SimInspector method*), 278
merge() (*angr.state_plugins.SimJavaVmClassloader method*), 280
merge() (*angr.state_plugins.SimMemView method*), 284
merge() (*angr.state_plugins.SimRegNameView method*), 286
merge() (*angr.state_plugins.SimSolver method*), 290
merge() (*angr.state_plugins.SimStateCGC method*), 297
merge() (*angr.state_plugins.SimStateEdgeHitmap method*), 299
merge() (*angr.state_plugins.SimStateGlobals method*), 299
merge() (*angr.state_plugins.SimStateHistory method*), 301
merge() (*angr.state_plugins.SimStateIcicle method*), 304
merge() (*angr.state_plugins.SimStateJNIReferences method*), 306
merge() (*angr.state_plugins.SimStateLibc method*), 310
merge() (*angr.state_plugins.SimStateLog method*), 311
merge() (*angr.state_plugins.SimStateLoopData method*), 313
merge() (*angr.state_plugins.SimStatePlugin method*), 315
merge() (*angr.state_plugins.SimStatePreconstrainer method*), 316
merge() (*angr.state_plugins.SimStateScratch method*), 319
merge() (*angr.state_plugins.SimSystemPosix method*), 323
merge() (*angr.state_plugins.solver.SimSolver method*), 363
merge() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 383
merge() (*angr.state_plugins.trace_additions.ZenPlugin method*), 384
merge() (*angr.state_plugins.Unicorn method*), 327
merge() (*angr.state_plugins.unicorn_engine.Unicorn method*), 397
merge() (*angr.state_plugins.view.SimMemView method*), 430
merge() (*angr.state_plugins.view.SimRegNameView method*), 426
merge() (*angr.storage.file.SimFile method*), 435
merge() (*angr.storage.file.SimFileDescriptor method*), 445
merge() (*angr.storage.file.SimFileDescriptorDuplex method*), 448
merge() (*angr.storage.file.SimFileStream method*), 437
merge() (*angr.storage.file.SimPackets method*), 439
merge() (*angr.storage.file.SimPacketsSlots method*), 450
merge() (*angr.storage.file.SimPacketsStream method*), 441
merge() (*angr.storage.memory_mixins.address_concretization_mixin.Addr method*), 496
merge() (*angr.storage.memory_mixins.AddressConcretizationMixin method*), 457
merge() (*angr.storage.memory_mixins.JavaVmMemoryMixin

method), 465
merge() (*angr.storage.memory_mixins.ListPage method*), 468
merge() (*angr.storage.memory_mixins.MemoryRegionMetaMixin method*), 472
merge() (*angr.storage.memory_mixins.MVListPage method*), 470
merge() (*angr.storage.memory_mixins.paged_memory.pagedmemorymixin.PagedMemoryMixin method*), 505
merge() (*angr.storage.memory_mixins.paged_memory.pages.ListPage method*), 521
merge() (*angr.storage.memory_mixins.paged_memory.pages.multivalued_page.MultiValuedMemoryPage method*), 512
merge() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 502
merge() (*angr.storage.memory_mixins.paged_memory.pages.paged_memory_mixin.PagedMemoryMixin method*), 500
merge() (*angr.storage.memory_mixins.paged_memory.pages.multi_values_mixin.MultiValuedMemoryMixin method*), 513
merge() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 523
merge() (*angr.storage.memory_mixins.paged_memory.pages.vsa_list_page.VSAListPage method*), 517
merge() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 475
merge() (*angr.storage.memory_mixins.regioned_memory.MemoryRegionMetaMixin method*), 525
merge() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.RegionMetaMixin method*), 536
merge() (*angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin.RegionedAddressConcretizationMixin method*), 537
merge() (*angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin.RegionedAddressConcretizationMixin method*), 539
merge() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 531
merge() (*angr.storage.memory_mixins.regioned_memory.RegionedAddressConcretizationMixin method*), 527
merge() (*angr.storage.memory_mixins.regioned_memory.RegionedMemoryMixin method*), 528
merge() (*angr.storage.memory_mixins.RegionedAddressConcretizationMixin method*), 479
merge() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 481
merge() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method*), 540
merge() (*angr.storage.memory_mixins.SlottedMemoryMixin method*), 484
merge() (*angr.storage.memory_mixins.UltraPage method*), 488
merge() (*angr.storage.SimFile method*), 424
merge_jobs() (*angr.analyses.vfg.CallAnalysis method*), 1288
merge_points() (*angr.annocfg.AnnotatedCFG method*), 1322
merge_to_top() (*angr.keyed_region.KeyedRegion method*), 868
merge_transitions() (*in module angr.analyses.cfg_slice_to_sink.transitions*), 1243
merged_jobs (*angr.analyses.forward_analysis.job_info.JobInfo property*), 937
meta_only (*angr.knowledge_plugins.functions.function.Function attribute*), 717
meta_only (*angr.knowledge_plugins.functions.Function attribute*), 777
meta_only (*angr.knowledge_plugins.functions.function.Function attribute*), 802
meta_only (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 811
mfn (*angr.ailment.expression.DirtyExpression attribute*), 1388
MicrosoftAMD64ArgSession (*class in angr.calling_conventions*), 669
min() (*angr.state_plugins.SimSolver method*), 291
min() (*angr.state_plugins.solver.SimSolver method*), 365
MINIMAL_ASSIGNMENTS (*angr.analyses.decompiler.optimization_passes.InlinedStringTransformation attribute*), 1080
min_int() (*angr.state_plugins.SimSolver method*), 296
min_int() (*angr.state_plugins.solver.SimSolver method*), 369
minimum_syscall_number() (*angr.procedures.definitions.SimSyscallLibrary method*), 655
MipsElfFastResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1011
MipsElfFastResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 996
MipsElfGlobalResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1011
MIPSGPHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*) 1002
MipsGpSettingSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1083
MixedPermissionsError, 396
mnemonic (*angr.analyses.disassembly.Instruction property*), 1294
mnemonic (*angr.block.CapstoneInsn property*), 232
mnemonic (*angr.block.DisassemblerInsn property*), 232
mnemonic (*angr.block.PCodeInsn property*), 233
mnemonic (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 606
model (*angr.analyses.cfg.cfg_base.CFGBase property*), 982
model (*angr.analyses.cfg.CFGBase property*), 960

ModSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1083
ModSimplifier (*class in angr.analyses.decompiler.optimization_passes.mod_simplifier*), 1101
ModSimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.mod_simplifier*), 1101
module
angr, 155
angr.ailment, 1364
angr.ailment.block, 1406
angr.ailment.converter_common, 1373
angr.ailment.converter_pcode, 1373
angr.ailment.converter_vex, 1374
angr.ailment.expression, 1375
angr.ailment.manager, 1407
angr.ailment.statement, 1397
angr.ailment.tagged_object, 1407
angr.ailment.utils, 1408
angr.analyses, 873
angr.analyses.analysis, 928
angr.analyses.backward_slice, 946
angr.analyses.binary_optimizer, 1306
angr.analyses.bindiff, 947
angr.analyses.boyscout, 950
angr.analyses.callee_cleanup_finder, 1307
angr.analyses.calling_convention, 950
angr.analyses.cdg, 1017
angr.analyses.cfg, 956
angr.analyses.cfg.cfb, 972
angr.analyses.cfg.cfg, 973
angr.analyses.cfg.cfg_arch_options, 992
angr.analyses.cfg.cfg_base, 981
angr.analyses.cfg.cfg_emulated, 976
angr.analyses.cfg.cfg_fast, 983
angr.analyses.cfg.cfg_fast_soot, 1015
angr.analyses.cfg.cfg_job_base, 993
angr.analyses.cfg.indirect_jump_resolvers, 1005
angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_pic, 994
angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast, 994
angr.analyses.cfg.indirect_jump_resolvers.const_resolver, 1003
angr.analyses.cfg.indirect_jump_resolvers.default_resolvers, 998
angr.analyses.cfg.indirect_jump_resolvers.jumptable, 998
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast, 996
angr.analyses.cfg.indirect_jump_resolvers.resolver, 1004
angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic, 997
angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat, 995
angr.analyses.cfg_slice_to_sink, 1238
angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink, 1241
angr.analyses.cfg_slice_to_sink.graph, 1242
angr.analyses.cfg_slice_to_sink.transitions, 1243
angr.analyses.class_identifier, 1292
angr.analyses.code_tagging, 1018
angr.analyses.complete_calling_conventions, 953
angr.analyses.congruency_check, 1305
angr.analyses.data_dep, 1316
angr.analyses.data_dep.data_dependency_analysis, 1311
angr.analyses.data_dep.dep_nodes, 1313
angr.analyses.data_dep.sim_act_location, 1312
angr.analyses.datagraph_meta, 1017
angr.analyses.ddg, 1156
angr.analyses.decompiler, 1039
angr.analyses.decompiler.ail_simplifier, 1052
angr.analyses.decompiler.ailgraph_walker, 1054
angr.analyses.decompiler.block_simplifier, 1054
angr.analyses.decompiler.callsite_maker, 1055
angr.analyses.decompiler.ccall_rewriters, 1055
angr.analyses.decompiler.ccall_rewriters.amd64_ccalls, 1056
angr.analyses.decompiler.ccall_rewriters.rewriter_base, 1055
angr.analyses.decompiler.clinic, 1056
angr.analyses.decompiler.condition_processor, 1061
angr.analyses.decompiler.decompilation_cache, 1064
angr.analyses.decompiler.decompilation_options, 1063
angr.analyses.decompiler.decompiler, 1065
angr.analyses.decompiler.empty_node_remover, 1068
angr.analyses.decompiler.expression_narrower, 1069
angr.analyses.decompiler.graph_region,

1070
angr.analyses.decompiler.jump_target_collector, 1071
angr.analyses.decompiler.jumptable_entry_condition_recovery, 1071
angr.analyses.decompiler.optimization_passes, 1071
angr.analyses.decompiler.optimization_passes.base_optimization_pass, 1097
angr.analyses.decompiler.optimization_passes.call_stmt_rewriter, 1091
angr.analyses.decompiler.optimization_passes.div_simplifier, 1098
angr.analyses.decompiler.optimization_passes.engine_base, 1101
angr.analyses.decompiler.optimization_passes.expr_op_swapper, 1102
angr.analyses.decompiler.optimization_passes.expr_simplifier, 1098
angr.analyses.decompiler.optimization_passes.lower_switch_simplifier, 1099
angr.analyses.decompiler.optimization_passes.mod_simplifier, 1101
angr.analyses.decompiler.optimization_passes.optimization_pass, 1092
angr.analyses.decompiler.optimization_passes.register_assignment_simplifier, 1103
angr.analyses.decompiler.optimization_passes.rangeanalyses_simplifier, 1103
angr.analyses.decompiler.optimization_passes.stack_canary_simplifier, 1097
angr.analyses.decompiler.optimization_passes.x86_cgc_cgrandom_simplifier, 1104
angr.analyses.decompiler.peephole_optimizations, 1104
angr.analyses.decompiler.peephole_optimizations.base, 1104
angr.analyses.decompiler.redundant_label_remover, 1119
angr.analyses.decompiler.region_identifier, 1107
angr.analyses.decompiler.region_simplifiers, 1108
angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer, 1109
angr.analyses.decompiler.region_simplifiers.cascading_ifs, 1109
angr.analyses.decompiler.region_simplifiers.expr_folding, 1109
angr.analyses.decompiler.region_simplifiers.goto, 1114
angr.analyses.decompiler.region_simplifiers.if_, 1114
angr.analyses.decompiler.region_simplifiers.ifelse, 1114
angr.analyses.decompiler.region_simplifiers.loop, 1114
angr.analyses.decompiler.region_simplifiers.node_address_refiner, 1114
angr.analyses.decompiler.region_simplifiers.region_simplifier, 1114
angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier, 1115
angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier, 1119
angr.analyses.decompiler.region_walker, 1119
angr.analyses.decompiler.sequence_walker, 1119
angr.analyses.decompiler.structured_codegen, 1120
angr.analyses.decompiler.structured_codegen.base, 1127
angr.analyses.decompiler.structured_codegen.c, 1130
angr.analyses.decompiler.structured_codegen.dummy, 1149
angr.analyses.decompiler.structured_codegen.dwarf_import, 1149
angr.analyses.decompiler.structuring, 1032
angr.analyses.decompiler.structuring.dream, 1035
angr.analyses.decompiler.structuring.phoenix, 1037
angr.analyses.decompiler.structuring.recursive_structuring, 1032
angr.analyses.decompiler.structuring.structurer_base, 1036
angr.analyses.decompiler.utils, 1150
angr.analyses.disassembly, 1293
angr.analyses.disassembly_utils, 1297
angr.analyses.dominance_frontier, 1307
angr.analyses.find_objects_static, 1292
angr.analyses.flirt, 1161
angr.analyses.forward_analysis, 931
angr.analyses.forward_analysis.forward_analysis, 935
angr.analyses.forward_analysis.job_info, 937
angr.analyses.forward_analysis.visitors, 937
angr.analyses.forward_analysis.visitors.call_graph, 941
angr.analyses.forward_analysis.visitors.function_graph, 941
angr.analyses.forward_analysis.visitors.graph, 942

angr.analyses.forward_analysis.visitors.loop, 944
angr.analyses.forward_analysis.visitors.single_node_graph, 945
angr.analyses.identifier.identify, 1283
angr.analyses.init_finder, 1307
angr.analyses.loop_analysis, 1284
angr.analyses.loopfinder, 1283
angr.analyses.propagator, 1174
angr.analyses.propagator.engine_base, 1176
angr.analyses.propagator.engine_vex, 1177
angr.analyses.propagator.propagator, 1177
angr.analyses.propagator.top_checker_mixin, 1178
angr.analyses.propagator.values, 1175
angr.analyses.propagator.vex_vars, 1175
angr.analyses.proximity_graph, 1309
angr.analyses.reaching_definitions, 1178
angr.analyses.reaching_definitions.call_trace, 1212
angr.analyses.reaching_definitions.dep_graph, 1217
angr.analyses.reaching_definitions.engine_ail, 1238
angr.analyses.reaching_definitions.engine_vex, 1213
angr.analyses.reaching_definitions.function_handler, 1221
angr.analyses.reaching_definitions.heap_allocator, 1220
angr.analyses.reaching_definitions.rd_state, 1229
angr.analyses.reaching_definitions.reaching_definitions, 1213
angr.analyses.reaching_definitions.subject, 1237
angr.analyses.reassembler, 1297
angr.analyses.soot_class_hierarchy, 955
angr.analyses.stack_pointer_tracker, 1243
angr.analyses.static_hooker, 1305
angr.analyses.typehoon, 1281
angr.analyses.typehoon.simple_solver, 1259
angr.analyses.typehoon.translator, 1265
angr.analyses.typehoon.typeconsts, 1274
angr.analyses.typehoon.typehoon, 1272
angr.analyses.typehoon.typevars, 1266
angr.analyses.variable_recovery, 1257
angr.analyses.variable_recovery.annotations, 1246
angr.analyses.variable_recovery.engine_ail, 1255
angr.analyses.variable_recovery.engine_base, 1256
angr.analyses.variable_recovery.engine_vex, 1256
angr.analyses.variable_recovery.irsb_scanner, 1257
angr.analyses.variable_recovery.variable_recovery, 1254
angr.analyses.variable_recovery.variable_recovery_base, 1247
angr.analyses.variable_recovery.variable_recovery_fast, 1251
angr.analyses.veritesting, 1284
angr.analyses.vfg, 1285
angr.analyses.vsa_ddg, 1290
angr.analyses.vtable, 1291
angr.analyses.xrefs, 1308
angr.angrdb, 1018
angr.angrdb.db, 1020
angr.angrdb.models, 1021
angr.angrdb.serializers, 1025
angr.angrdb.serializers.cfg_model, 1026
angr.angrdb.serializers.comments, 1027
angr.angrdb.serializers.funcs, 1027
angr.angrdb.serializers.kb, 1027
angr.angrdb.serializers.labels, 1028
angr.angrdb.serializers.loader, 1028
angr.angrdb.serializers.structured_code, 1031
angr.angrdb.serializers.variables, 1030
angr.angrdb.serializers.xrefs, 1030
angr.annocfg, 1321
angr.blade, 1319
angr.block, 232
angr.callable, 710
angr.calling_conventions, 658
angr.code_location, 865
angr.codenode, 1322
angr.concretization_strategies, 452
angr.concretization_strategies.any, 543
angr.concretization_strategies.controlled_data, 543
angr.concretization_strategies.eval, 541
angr.concretization_strategies.max, 542
angr.concretization_strategies.nonzero, 543
angr.concretization_strategies.nonzero_range, 542
angr.concretization_strategies.norepeats, 541
angr.concretization_strategies.norepeats_range, 543
angr.concretization_strategies.range, 542
angr.concretization_strategies.single, 541

angr.concretization_strategies.solutions, 541
angr.concretization_strategies.unlimited_range, 544
angr.distributed, 1361
angr.distributed.server, 1362
angr.distributed.worker, 1363
angr.engines, 587
angr.engines.engine, 594
angr.engines.failure, 598
angr.engines.hook, 597
angr.engines.light, 1167
angr.engines.light.data, 1166
angr.engines.light.engine, 1170
angr.engines.pcode, 603
angr.engines.pcode.behavior, 616
angr.engines.pcode.cc, 636
angr.engines.pcode.emulate, 615
angr.engines.pcode.engine, 604
angr.engines.pcode.lifter, 605
angr.engines.procedure, 597
angr.engines.soot, 601
angr.engines.soot.engine, 602
angr.engines.successors, 594
angr.engines.syscall, 598
angr.engines.unicorn, 603
angr.engines.vex, 599
angr.errors, 1355
angr.exploration_techniques, 552
angr.exploration_techniques.bucketizer, 586
angr.exploration_techniques.common, 585
angr.exploration_techniques.dfs, 569
angr.exploration_techniques.director, 579
angr.exploration_techniques.driller_core, 577
angr.exploration_techniques.explorer, 569
angr.exploration_techniques.lengthlimiter, 571
angr.exploration_techniques.local_loop_seer, 583
angr.exploration_techniques.loop_seer, 582
angr.exploration_techniques.manual_mergepoint, 571
angr.exploration_techniques.memory_watcher, 585
angr.exploration_techniques.oppologist, 581
angr.exploration_techniques.slicecutor, 578
angr.exploration_techniques.spiller, 571
angr.exploration_techniques.spiller_db, 574
angr.exploration_techniques.stochastic, 584
angr.exploration_techniques.suggestions, 586
angr.exploration_techniques.tech_builder, 585
angr.exploration_techniques.threading, 574
angr.exploration_techniques.timeout, 569
angr.exploration_techniques.tracer, 575
angr.exploration_techniques.unique, 584
angr.exploration_techniques.veritesting, 574
angr.factory, 227
angr.flirt, 1342
angr.flirt.build_sig, 1343
angr.keyed_region, 867
angr.knowledge_base, 711
angr.knowledge_plugins, 712
angr.knowledge_plugins.callsite_prototypes, 739
angr.knowledge_plugins.cfg, 740
angr.knowledge_plugins.cfg.cfg_manager, 766
angr.knowledge_plugins.cfg.cfg_model, 756
angr.knowledge_plugins.cfg.cfg_node, 766
angr.knowledge_plugins.cfg.indirect_jump, 769
angr.knowledge_plugins.cfg.memory_data, 764
angr.knowledge_plugins.comments, 775
angr.knowledge_plugins.data, 775
angr.knowledge_plugins.debug_variables, 820
angr.knowledge_plugins.functions, 776
angr.knowledge_plugins.functions.function, 799
angr.knowledge_plugins.functions.function_manager, 790
angr.knowledge_plugins.functions.function_parser, 809
angr.knowledge_plugins.functions.soot_function, 810
angr.knowledge_plugins.indirect_jumps, 775
angr.knowledge_plugins.key_definitions, 822
angr.knowledge_plugins.key_definitions.atoms, 836
angr.knowledge_plugins.key_definitions.constants, 841
angr.knowledge_plugins.key_definitions.definition, 841
angr.knowledge_plugins.key_definitions.environment,

844
angr.knowledge_plugins.key_definitions.heap_address, 845
angr.knowledge_plugins.key_definitions.key_definition, 845
angr.knowledge_plugins.key_definitions.live_definitions, 846
angr.knowledge_plugins.key_definitions.rd_model, 855
angr.knowledge_plugins.key_definitions.tag, 857
angr.knowledge_plugins.key_definitions.undefined, 858
angr.knowledge_plugins.key_definitions.unknown_size, 859
angr.knowledge_plugins.key_definitions.uses, 859
angr.knowledge_plugins.labels, 775
angr.knowledge_plugins.patches, 737
angr.knowledge_plugins.plugin, 738
angr.knowledge_plugins.propagations, 773
angr.knowledge_plugins.structured_code, 822
angr.knowledge_plugins.types, 772
angr.knowledge_plugins.variables, 811
angr.knowledge_plugins.variables.variable_access, 812
angr.knowledge_plugins.variables.variable_manager, 812
angr.knowledge_plugins.xrefs, 861
angr.knowledge_plugins.xrefs.xref, 863
angr.knowledge_plugins.xrefs.xref_manager, 864
angr.knowledge_plugins.xrefs.xref_types, 864
angr.misc.plugins, 235
angr.procedures, 646
angr.procedures.definitions, 649
angr.procedures.stubs.format_parser, 646
angr.project, 223
angr.protos, 871
angr.serializable, 870
angr.sim_manager, 544
angr.sim_options, 242
angr.sim_procedure, 642
angr.sim_state, 238
angr.sim_state_options, 242
angr.sim_type, 686
angr.sim_variable, 680
angr.simos, 1325
angr.simos.cgc, 1337
angr.simos.javavm, 1340
angr.simos.linux, 1335
angr.simos.simos, 1333
angr.simos.userland, 1337
angr.simos.windows, 1338
angr.slicer, 1320
angr.state_hierarchy, 551
angr.state_plugins, 246
angr.state_plugins.callstack, 371
angr.state_plugins.cgc, 379
angr.state_plugins.debug_variables, 421
angr.state_plugins.filesystem, 354
angr.state_plugins.gdb, 378
angr.state_plugins.globals, 386
angr.state_plugins.heap, 404
angr.state_plugins.heap.heap_base, 411
angr.state_plugins.heap.heap_brk, 412
angr.state_plugins.heap.heap_freelist, 414
angr.state_plugins.heap.heap_libc, 415
angr.state_plugins.heap.heap_ptmalloc, 416
angr.state_plugins.heap.utils, 420
angr.state_plugins.history, 375
angr.state_plugins.inspect, 331
angr.state_plugins.javavm_classloader, 400
angr.state_plugins.jni_references, 402
angr.state_plugins.libc, 343
angr.state_plugins.light_registers, 374
angr.state_plugins.log, 370
angr.state_plugins.loop_data, 399
angr.state_plugins.plugin, 329
angr.state_plugins.posix, 347
angr.state_plugins.preconstrainer, 390
angr.state_plugins.scratch, 388
angr.state_plugins.sim_action, 638
angr.state_plugins.sim_action_object, 640
angr.state_plugins.sim_event, 642
angr.state_plugins.solver, 361
angr.state_plugins.symbolizer, 420
angr.state_plugins.trace_additions, 381
angr.state_plugins.uc_manager, 387
angr.state_plugins.unicorn_engine, 392
angr.state_plugins.view, 426
angr.storage, 423
angr.storage.file, 431
angr.storage.memory_mixins, 456
angr.storage.memory_mixins.actions_mixin, 493
angr.storage.memory_mixins.address_concretization_mixi
495
angr.storage.memory_mixins.bvv_conversion_mixin, 491
angr.storage.memory_mixins.conditional_store_mixin, 497

angr.storage.memory_mixins.convenient_mappings_mixin, 499
angr.storage.memory_mixins.default_filler_mixin, 490
angr.storage.memory_mixins.dirty_addrs_mixin, 495
angr.storage.memory_mixins.hex_dumper_mixin, 491
angr.storage.memory_mixins.label_merger_mixin, 497
angr.storage.memory_mixins.multi_value_merger_mixin, 503
angr.storage.memory_mixins.name_resolution_mixin, 490
angr.storage.memory_mixins.paged_memory, 504
angr.storage.memory_mixins.paged_memory.page_backers_mixin, 508
angr.storage.memory_mixins.paged_memory.paged_memory_mixin, 504
angr.storage.memory_mixins.paged_memory.pages, 510
angr.storage.memory_mixins.paged_memory.pages.cooperative, 520
angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin, 519
angr.storage.memory_mixins.paged_memory.pages.mapping_mixin, 520
angr.storage.memory_mixins.paged_memory.pages.ultra_page, 521
angr.storage.memory_mixins.paged_memory.pages.vectorized_page, 501
angr.storage.memory_mixins.paged_memory.pages.vmem_page, 499
angr.storage.memory_mixins.paged_memory.pages.vmem_utils_mixin, 519
angr.storage.memory_mixins.paged_memory.pages.vmem_page_mixin, 518
angr.storage.memory_mixins.paged_memory.pages.vmem_page, 522
angr.storage.memory_mixins.paged_memory.privileged_mixin, 510
angr.storage.memory_mixins.paged_memory.stack_allocation_mixin, 509
angr.storage.memory_mixins.regioned_memory, 524
angr.storage.memory_mixins.regioned_memory.abstract_region_mixin, 535
angr.storage.memory_mixins.regioned_memory.actions_mixin, 538
angr.storage.memory_mixins.regioned_memory.comparable_mixin, 535
angr.storage.memory_mixins.regioned_memory.region_meta, 533
angr.storage.memory_mixins.regioned_memory.region_meta, 536
angr.storage.memory_mixins.regioned_memory.regioned_ad, 538
angr.storage.memory_mixins.regioned_memory.regioned_me, 530
angr.storage.memory_mixins.regioned_memory.static_find, 535
angr.storage.memory_mixins.simple_interface_mixin, 493
angr.storage.memory_mixins.simplification_mixin, 498
angr.storage.memory_mixins.size_resolution_mixin, 494
angr.storage.memory_mixins.slotted_memory, 539
angr.storage.memory_mixins.smart_find_mixin, 490
angr.storage.memory_mixins.symbolic_merger_mixin, 493
angr.storage.memory_mixins.top_merger_mixin, 503
angr.storage.memory_mixins.underconstrained_mixin, 492
angr.storage.memory_mixins.unwrapper_mixin, 498
angr.storage.memory_object, 451
angr.utils, 1344
angr.utils.algo, 1344
angr.utils.constants, 1344
angr.utils.cowdict, 1344
angr.utils.dynamic_dictlist, 1346
angr.utils.enums_conv, 1346
angr.utils.env, 1346
angr.utils.formatting, 1353
angr.utils.graph, 1346
angr.utils.library_import, 1351
angr.utils.library, 1351
angr.utils.loader, 1351
angr.utils.mp, 1354
angr.utils.timing, 1353
angr.vaults, 871
most_mergeable() (*angr.state_hierarchy.StateHierarchy method*), 552
most_mergeable() (*angr.StateHierarchy method*), 221
mount() (*angr.state_plugins.filesystem.SimFilesystem method*), 357
mount() (*angr.state_plugins.SimFilesystem method*), 270
move() (*angr.managers.manager.SimulationManager method*), 548
move() (*angr.SimulationManager method*), 218
move_codelocs() (*angr.analyses.reaching_definitions.rd_state.ReachingD*
method), 1234
move_codelocs() (*angr.analyses.reaching_definitions.ReachingDefinition*

method), 1207
mp_context() (in module angr.utils.mp), 1355
msize (angr.ailment.expression.DirtyExpression attribute), 1388
Mul (angr.engines.light.ArithmeticExpression attribute), 1167
Mul (angr.engines.light.data.ArithmeticExpression attribute), 1166
mulpyplex() (angr.sim_manager.SimulationManager method), 546
mulpyplex() (angr.SimulationManager method), 215
MultipleBlocksException, 1092
MultiStatementExpression (class in angr.ailment.expression), 1389
MultiStatementExpressionAssignmentFinder (class in angr.analyses.decompiler.region_simplifiers.expr_folding), 1111
MultiStmtExprMode (class in angr.analyses.decompiler.structuring.phoenix), 1037
MultiValuedMemory (class in angr.storage.memory_mixins), 473
MultiValueMergerMixin (class in angr.storage.memory_mixins), 473
MultiValueMergerMixin (class in angr.storage.memory_mixins.multi_value_merger_mixin), 503
MultiValues (class in angr.storage.memory_mixins.paged_memory.pages.multi_values), 501
MultiwriteAnnotation (class in angr.storage.memory_mixins.address_concretization_mixin), 495
mv_is_bv() (in module angr.storage.memory_mixins.paged_memory.pages.multi_values), 503
mv_is_fp() (in module angr.storage.memory_mixins.paged_memory.pages.multi_values), 503
MVListPage (class in angr.storage.memory_mixins), 469
MVListPage (class in angr.storage.memory_mixins.paged_memory.pages), 513
MVListPage (class in angr.storage.memory_mixins.paged_memory.pages.mv_list.page), 499
MVListPagesMixin (class in angr.storage.memory_mixins), 471
MVListPagesMixin (class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin), 507
MVListPagesWithLabelsMixin (class in angr.storage.memory_mixins), 471
MVListPagesWithLabelsMixin (class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin), 508

N

n (angr.analyses.typehoon.typevars.AddN attribute), 1272
n (angr.analyses.typehoon.typevars.SubN attribute), 1272
name (angr.ailment.expression.Macro attribute), 1396
name (angr.ailment.expression.Struct attribute), 1394
name (angr.ailment.statement.Label attribute), 1405
NAME (angr.analyses.decompiler.optimization_passes.base_ptr_save_simplif attribute), 1098
NAME (angr.analyses.decompiler.optimization_passes.BasePointerSaveSimpl attribute), 1072
NAME (angr.analyses.decompiler.optimization_passes.CallStatementRewriter attribute), 1072
NAME (angr.analyses.decompiler.optimization_passes.CodeMotionOptimizat attribute), 1073
NAME (angr.analyses.decompiler.optimization_passes.ConditionConstantPro attribute), 1074
NAME (angr.analyses.decompiler.optimization_passes.const_derefs.Constant attribute), 1092
NAME (angr.analyses.decompiler.optimization_passes.ConstantDereferences attribute), 1075
NAME (angr.analyses.decompiler.optimization_passes.ConstPropOptReverter attribute), 1074
NAME (angr.analyses.decompiler.optimization_passes.CrossJumpReverter attribute), 1075
NAME (angr.analyses.decompiler.optimization_passes.DeadblockRemover attribute), 1076
NAME (angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimp attribute), 1098
NAME (angr.analyses.decompiler.optimization_passes.DivSimplifier attribute), 1076
NAME (angr.analyses.decompiler.optimization_passes.DuplicationReverter attribute), 1076
NAME (angr.analyses.decompiler.optimization_passes.EagerStdStringConcat attribute), 1078
NAME (angr.analyses.decompiler.optimization_passes.expr_op_swapper.Exp attribute), 1103
NAME (angr.analyses.decompiler.optimization_passes.ExprOpSwapper attribute), 1079
NAME (angr.analyses.decompiler.optimization_passes.FlipBooleanCmp attribute), 1079
NAME (angr.analyses.decompiler.optimization_passes.InlinedMemcpySimplif attribute), 1080
NAME (angr.analyses.decompiler.optimization_passes.InlinedMemcpySimplif attribute), 1080
NAME (angr.analyses.decompiler.optimization_passes.InlinedMemsetSimplifi attribute), 1080
NAME (angr.analyses.decompiler.optimization_passes.InlinedMemsetSimplifi attribute), 1081
NAME (angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifie attribute), 1081
NAME (angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifie attribute), 1081
NAME (angr.analyses.decompiler.optimization_passes.InlinedStringTransform attribute), 1081

NAME (*angr.analyses.decompiler.optimization_passes.InlineCopyPropagation attribute*), 1082
NAME (*angr.analyses.decompiler.optimization_passes.InlineCopyPropagationSimplifier attribute*), 1082
NAME (*angr.analyses.decompiler.optimization_passes.InlineCopyPropagationSimplifierLoad attribute*), 1082
NAME (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExpr2CondAssignConverter attribute*), 1099
NAME (*angr.analyses.decompiler.optimization_passes.ITEExpr2CondAssignConverter attribute*), 1079
NAME (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 1080
NAME (*angr.analyses.decompiler.optimization_passes.lower_switch_simplifier.LowerSwitchSimplifier attribute*), 1100
NAME (*angr.analyses.decompiler.optimization_passes.LowerSwitchSimplifier attribute*), 1083
NAME (*angr.analyses.decompiler.optimization_passes.MipsGpSetupSimplifier attribute*), 1083
NAME (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 1101
NAME (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 1084
NAME (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute*), 1093
NAME (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifier attribute*), 1103
NAME (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifier attribute*), 1085
NAME (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifierAdvanced attribute*), 1085
NAME (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 1103
NAME (*angr.analyses.decompiler.optimization_passes.RetAddrSaveSimplifier attribute*), 1085
NAME (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator attribute*), 1086
NAME (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorHigh attribute*), 1086
NAME (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorLow attribute*), 1088
NAME (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 1097
NAME (*angr.analyses.decompiler.optimization_passes.StackCanarySimplifier attribute*), 1088
NAME (*angr.analyses.decompiler.optimization_passes.StaticVariableRewriter attribute*), 1089
NAME (*angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseDuplicator attribute*), 1089
NAME (*angr.analyses.decompiler.optimization_passes.SwitchReusedEntryRewriter attribute*), 1090
NAME (*angr.analyses.decompiler.optimization_passes.TagSlicer attribute*), 1090
NAME (*angr.analyses.decompiler.optimization_passes.WinStackCanarySimplifier attribute*), 1090
NAME (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier attribute*), 1104
NAME (*angr.analyses.decompiler.optimization_passes.X86GccGetPcSimplifier attribute*), 1091
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExpr attribute*), 1106
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationMultiStmt attribute*), 1105
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmt attribute*), 1104
name (*angr.analyses.decompiler.structured_codegen.c.CFakeVariable attribute*), 1139
name (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 1132
name (*angr.analyses.decompiler.structured_codegen.c.CLabel attribute*), 1138
name (*angr.analyses.decompiler.structured_codegen.c.CStructFieldNameDefinition attribute*), 1144
name (*angr.analyses.decompiler.structured_codegen.c.CVariable property*), 1139
NAME (*angr.analyses.decompiler.structuring.dream.DreamStructurer attribute*), 1036
NAME (*angr.analyses.decompiler.structuring.DreamStructurer attribute*), 1033
NAME (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer attribute*), 1038
NAME (*angr.analyses.decompiler.structuring.PhoenixStructurer attribute*), 1033
NAME (*angr.analyses.decompiler.structuring.SAILRStructurer attribute*), 1035
NAME (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase attribute*), 1036
name (*angr.analyses.reaching_definitions.function_handler.FunctionCallDataHandler attribute*), 1224
name (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
name (*angr.analyses.reaching_definitions.Register property*), 1211
name (*angr.analyses.reassembler.Procedure property*), 1300
name (*angr.analyses.type_honer.SimpleTypeVariable attribute*), 1269
name (*angr.angrdb.models.DbKnowledgeBase attribute*), 1022
name (*angr.angrdb.models.DbLabel attribute*), 1025
name (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
name (*angr.knowledge_plugins.cfg.CFGNode property*), 750
name (*angr.knowledge_plugins.Function property*), 717
name (*angr.knowledge_plugins.functions.Function property*), 778
name (*angr.knowledge_plugins.functions.function.Function property*), 802

name (*angr.knowledge_plugins.key_definitions.atoms.Register property*), 839
name (*angr.procedures.definitions.SimLibrary property*), 650
name (*angr.sim_state_options.StateOption attribute*), 243
name (*angr.sim_type.NamedTypeMixin property*), 688
name (*angr.sim_type.SimTypeRef property*), 708
name (*angr.sim_type.TypeRef property*), 688
name (*angr.sim_variable.SimVariable attribute*), 680
name (*angr.SimOS attribute*), 195
name (*angr.simos.SimOS attribute*), 1328
name (*angr.simos.simos.SimOS attribute*), 1333
name_stop() (*angr.state_plugins.unicorn_engine.STOP static method*), 395
named_errors (*angr.analyses.Analysis attribute*), 881
named_errors (*angr.analyses.analysis.Analysis attribute*), 930
named_errors (*angr.Analysis attribute*), 156
named_struct_id_counter (*angr.analyses.typehoon.translator.TypeTranslator attribute*), 1266
NamedTypeMixin (*class in angr.sim_type*), 688
NameResolutionMixin (*class in angr.storage.memory_mixins*), 473
NameResolutionMixin (*class in angr.storage.memory_mixins.name_resolution_mixin*), 490
narrowable (*angr.analyses.decompiler.expression_narrower.ExprNarrowingInfo attribute*), 1069
narrowing_times (*angr.analyses.vfg.VFGNode attribute*), 1288
native_arch (*angr.simos.javavm.SimJavaVM property*), 1341
native_arch (*angr.simos.SimJavaVM property*), 1326
NE (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.CmpOp attribute*), 1115
negate() (*in module angr.ailment.expression*), 1391
NEVER (*angr.analyses.decompiler.structuring.phoenix.MultiNodeExamine attribute*), 1037
new() (*angr.analyses.cfg.cfg_job_base.FunctionKey static method*), 993
new() (*angr.analyses.typehoon.typeconsts.Pointer method*), 1277
new() (*angr.knowledge_plugins.cfg.BlockID static method*), 740
new_block_addr() (*angr.analyses.decompiler.Clinic method*), 1046
new_block_addr() (*angr.analyses.decompiler.clinic.Clinic method*), 1061
new_block_addr() (*angr.analyses.decompiler.optimization_passes.ar method*), 1095
new_child() (*angr.utils.cowdict.ChainMapCOW method*), 1345
new_child() (*angr.utils.cowdict.DefaultChainMapCOW method*), 1345
new_dtv() (*angr.analyses.typehoon.typevars.TypeVariableManager method*), 1269
new_dtv_with_merged_labels() (*angr.analyses.typehoon.typevars.TypeVariableManager method*), 1269
new_from_shared() (*angr.storage.memory_mixins.paged_memory.pages. class method*), 522
new_from_shared() (*angr.storage.memory_mixins.paged_memory.pages. class method*), 516
new_from_shared() (*angr.storage.memory_mixins.UltraPage class method*), 487
new_label() (*angr.analyses.reassembler.Label static method*), 1297
new_label() (*angr.analyses.reassembler.SymbolManager method*), 1298
new_model() (*angr.knowledge_plugins.cfg.cfg_manager.CFGManager method*), 766
new_model() (*angr.knowledge_plugins.cfg.CFGManager method*), 742
new_model() (*angr.knowledge_plugins.CFGManager method*), 712
new_tv() (*angr.analyses.typehoon.typevars.TypeVariableManager method*), 1269
new_vvar_sizes (*angr.analyses.decompiler.expression_narrower.Expressi attribute*), 1070
NewFunctionHandler (*class in angr.analyses.find_objects_static*), 1292
next (*angr.engines.pcode.lifter.IRSB attribute*), 608
next (*angr.state_plugins.unicorn_engine.MEM_PATCH attribute*), 392
next_addr (*angr.analyses.decompiler.optimization_passes.lowered_switch attribute*), 1100
next_arg() (*angr.calling_conventions.SimCC method*), 665
next_arg() (*angr.calling_conventions.SimCCARM method*), 672
next_arg() (*angr.calling_conventions.SimCCARMHF method*), 673
next_arg() (*angr.calling_conventions.SimCCCdecl method*), 668
next_arg() (*angr.calling_conventions.SimCCMicrosoftAMD64 method*), 670
next_arg() (*angr.calling_conventions.SimCCO32 method*), 675
next_arg() (*angr.calling_conventions.SimCCRISCV64 method*), 674
next_arg() (*angr.calling_conventions.SimCCSystemVAMD64 method*), 671
next_arg() (*angr.calling_conventions.SimCCUsercall method*), 667
next_arg() (*angr.SimCC method*), 176
next_atom() (*angr.ailment.Manager method*), 1370
next_atom() (*angr.ailment.manager.Manager method*),

1407
next_chunk() (*angr.PTChunk method*), 167
next_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 414
next_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 417
next_chunk() (*angr.state_plugins.heap.PTChunk method*), 405
next_chunk() (*angr.state_plugins.PTChunk method*), 258
next_id() (*angr.analyses.typehoon.typevars.TypeVariableManager method*), 1268
next_idx() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 1130
next_idx() (*angr.analyses.decompiler.structured_codegen.StructuredCodeGenerator method*), 1120
next_node() (*angr.analyses.forward_analysis.SingleNodeGraphVisitor method*), 934
next_node() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 943
next_node() (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor method*), 945
next_node() (*angr.analyses.forward_analysis.visitors.SingleNodeGraphVisitor method*), 940
next_variable_ident() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 814
nfuncs (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
NO_MAINBIN_BACKENDS (*angr.angrdb.serializers.loader.LoaderSerializer attribute*), 1029
NO_MAINBIN_BACKENDS (*angr.angrdb.serializers.LoaderSerializer attribute*), 1026
no_ret (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
no_ret (*angr.knowledge_plugins.cfg.CFGNode property*), 750
NO_RET (*angr.sim_procedure.SimProcedure attribute*), 645
NO_RET (*angr.SimProcedure attribute*), 204
NO_SUCCESSORS (*angr.EmulatorStopReason attribute*), 162
NoConcreteDispatch, 955
NodalAnnotation (*class in angr.analyses.data_dep.data_dependency_analysis*), 1311
node (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchCluster attribute*), 1116
node (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchCaseNode attribute*), 1116
node (*angr.analyses.forward_analysis.SingleNodeGraphVisitor attribute*), 934
node (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor attribute*), 945
node (*angr.analyses.forward_analysis.visitors.SingleNodeGraphVisitor attribute*), 940
node_addr (*angr.analyses.decompiler.region_simplifiers.expr_folding.ConditionalExpr attribute*), 1111
node_addr (*angr.analyses.decompiler.region_simplifiers.expr_folding.ConditionalExpr attribute*), 1111
node_addr_has_return() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 757
node_addr_has_return() (*angr.knowledge_plugins.cfg.CFGModel method*), 743
node_addrs (*angr.knowledge_plugins.cfg.cfg_model.CFGModel property*), 757
node_addrs (*angr.knowledge_plugins.cfg.CFGModel property*), 743
node_iterations (*angr.knowledge_plugins.propagations.PropagationModel attribute*), 774
node_mapping (*angr.analyses.decompiler.simple_solver.Sketch attribute*), 1260
node_observe() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 1216
node_observe() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysisInternal method*), 1200
node_observe() (*angr.analyses.ReachingDefinitionsAnalysis method*), 915
node_returned (*angr.analyses.forward_analysis.SingleNodeGraphVisitor attribute*), 934
node_returned (*angr.analyses.forward_analysis.visitors.single_node_graph attribute*), 945
node_returned (*angr.analyses.forward_analysis.visitors.SingleNodeGraphVisitor attribute*), 940
node_type (*angr.analyses.decompiler.optimization_passes.lowered_switch attribute*), 1099
NodeAddressFinder (*class in angr.analyses.decompiler.region_simplifiers.node_address_finder*), 1114
NodeFoundNotification, 1098
nodes (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 1241
nodes (*angr.analyses.cfg_slice_to_sink.CFGSliceToSink property*), 1239
nodes (*angr.knowledge_plugins.Function property*), 719
nodes (*angr.knowledge_plugins.functions.Function property*), 779
nodes (*angr.knowledge_plugins.functions.function.Function property*), 803
nodes() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 943
nodes() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 1218
nodes() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel*

method), 760
nodes() (angr.knowledge_plugins.cfg.CFGModel method), 746
nodes_by_addr() (angr.knowledge_plugins.cfg.cfg_model.CFGModel method), 757
nodes_by_addr() (angr.knowledge_plugins.cfg.CFGModel method), 743
NONE (angr.knowledge_plugins.functions.function.PrototypeSource attribute), 799
NONE (angr.sim_type.PointerDisposition attribute), 694
NONE (angr.simos.windows.SecurityCookieInit attribute), 1338
nonreturning_func_addrs() (angr.knowledge_plugins.FunctionManager method), 728
nonreturning_func_addrs() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 798
nonreturning_func_addrs() (angr.knowledge_plugins.functions.FunctionManager method), 788
NORMAL (angr.analyses.cfg.cfg_fast.CFGJobType attribute), 987
normalize() (angr.analyses.cfg.cfg_base.CFGBase method), 983
normalize() (angr.analyses.cfg.cfg_fast_soot.CFGFastSoot method), 1016
normalize() (angr.analyses.cfg.CFGBase method), 961
normalize() (angr.analyses.cfg.CFGFastSoot method), 972
normalize() (angr.analyses.CFGFastSoot method), 894
normalize() (angr.knowledge_plugins.Function method), 723
normalize() (angr.knowledge_plugins.functions.Function method), 783
normalize() (angr.knowledge_plugins.functions.function.Function method), 808
normalize() (angr.knowledge_plugins.functions.soot_function.SootFunction method), 810
normalize() (angr.knowledge_plugins.key_definitions.definition.DefinitionMatchPredicate method), 842
normalize_cpp_function_name() (in module angr.sim_type), 710
normalized (angr.analyses.cfg.cfg_base.CFGBase property), 982
normalized (angr.analyses.cfg.CFGBase property), 960
normalized (angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute), 757
normalized (angr.knowledge_plugins.cfg.CFGModel attribute), 743
normalized (angr.knowledge_plugins.Function attribute), 717
normalized (angr.knowledge_plugins.functions.Function attribute), 777
normalized (angr.knowledge_plugins.functions.function.Function attribute), 801
normalized (angr.knowledge_plugins.functions.soot_function.SootFunction attribute), 810
NormalizedBlock (class in angr.analyses.bindiff), 948
NormalizedFunction (class in angr.analyses.bindiff), 948
NotAJumpTableNotification, 998
notes (angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute), 1065
NotMemoryview (class in angr.storage.memory_mixins.paged_memory.page_backer_mixins), 508
NotypeLabel (class in angr.analyses.reassembler), 1298
NULL_TERMINATE (angr.knowledge_plugins.key_definitions.DerefSize attribute), 823
NULL_TERMINATE (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitionsManager attribute), 846
num_arguments (angr.knowledge_plugins.Function property), 720
num_arguments (angr.knowledge_plugins.functions.Function property), 780
num_arguments (angr.knowledge_plugins.functions.function.Function property), 804

O

O (in module angr.analyses.decompiler.decompilation_options), 1064
O_ACCMODE (angr.storage.file.Flags attribute), 432
O_APPEND (angr.storage.file.Flags attribute), 432
O_ASYNC (angr.storage.file.Flags attribute), 432
O_CLOEXEC (angr.storage.file.Flags attribute), 432
O_CREAT (angr.storage.file.Flags attribute), 432
O_DIRECT (angr.storage.file.Flags attribute), 432
O_DIRECTORY (angr.storage.file.Flags attribute), 432
O_DSYNC (angr.storage.file.Flags attribute), 432
O_EXCL (angr.storage.file.Flags attribute), 432
O_LARGEFILE (angr.storage.file.Flags attribute), 432
O_NDELAY (angr.storage.file.Flags attribute), 432
O_NOATIME (angr.storage.file.Flags attribute), 432
O_NOCTTY (angr.storage.file.Flags attribute), 432
O_NOFOLLOW (angr.storage.file.Flags attribute), 432
O_NONBLOCK (angr.storage.file.Flags attribute), 432
O_PATH (angr.storage.file.Flags attribute), 432
O_RDONLY (angr.storage.file.Flags attribute), 431
O_RDWR (angr.storage.file.Flags attribute), 432
O_SYNC (angr.storage.file.Flags attribute), 432
O_TMPFILE (angr.storage.file.Flags attribute), 432
O_TRUNC (angr.storage.file.Flags attribute), 432
O_WRONLY (angr.storage.file.Flags attribute), 431
obfuscations (angr.knowledge_base.KnowledgeBase attribute), 711
obfuscations (angr.KnowledgeBase attribute), 164
Obfuscations (class in angr.knowledge_plugins), 731

obj (*angr.analyses.decompiler.structured_codegen.base.PositionMappingElement attribute*), 1128
obj (*angr.analyses.decompiler.structured_codegen.PositionMappingElement attribute*), 1125
obj (*angr.analyses.disassembly.IROp attribute*), 1294
obj (*angr.keyed_region.StoredObject attribute*), 868
obj (*angr.utils.graph.ContainerNode property*), 1348
obj_bit_size() (*in module angr.storage.memory_object*), 451
obj_id (*angr.keyed_region.StoredObject property*), 868
object (*angr.storage.memory_object.SimMemoryObject attribute*), 451
object (*angr.storage.SimMemoryObject attribute*), 425
ObjectLabel (*class in angr.analyses.reassembler*), 1298
ObservationPointType (*class in angr.analyses.reaching_definitions*), 1197
ObservationPointType (*class in angr.knowledge_plugins.key_definitions.constants*), 841
observed_results (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 1216
observed_results (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 1200
observed_results (*angr.analyses.ReachingDefinitionsAnalysis property*), 914
odd_offsets_are_zero() (*angr.analyses.decompiler.optimization_passes.InlinedWcscpy.Simplifier static method*), 1082
offset (*angr.ailment.expression.BasePointerOffset attribute*), 1390
offset (*angr.ailment.expression.Extract attribute*), 1391
offset (*angr.ailment.expression.Insert attribute*), 1392
offset (*angr.ailment.statement.Store attribute*), 1400
offset (*angr.analyses.decompiler.structured_codegen.c.CStructField attribute*), 1138
offset (*angr.analyses.propagator.vex_vars.VEXReg attribute*), 1176
offset (*angr.analyses.reaching_definitions.Definition property*), 1182
offset (*angr.analyses.reassembler.Label property*), 1297
offset (*angr.analyses.stack_pointer_tracker.OffsetVal property*), 1243
offset (*angr.analyses.stack_pointer_tracker.Register attribute*), 1243
offset (*angr.analyses.typehoon.typevars.HasField attribute*), 1272
offset (*angr.engines.light.data.RegisterOffset attribute*), 1167
offset (*angr.engines.light.RegisterOffset attribute*), 1167
offset (*angr.knowledge_plugins.Function property*), 720
offset (*angr.knowledge_plugins.functions.Function property*), 780
offset (*angr.knowledge_plugins.functions.function.Function property*), 805
offset (*angr.knowledge_plugins.key_definitions.Definition property*), 823
offset (*angr.knowledge_plugins.key_definitions.definition.Definition property*), 844
offset (*angr.knowledge_plugins.variables.variable_access.VariableAccess attribute*), 812
Offset (*angr.knowledge_plugins.xrefs.xref_types.XRefType attribute*), 864
Offset (*angr.knowledge_plugins.xrefs.XRefType attribute*), 863
offset (*angr.sim_variable.SimStackVariable attribute*), 686
offset (*angr.state_plugins.unicorn_engine.RegisterValue attribute*), 393
offset_after() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1246
offset_after() (*angr.analyses.StackPointerTracker method*), 923
offset_after_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1246
offset_after_block() (*angr.analyses.StackPointerTracker method*), 923
offset_before() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1246
offset_before() (*angr.analyses.StackPointerTracker method*), 923
offset_before_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1246
offset_before_block() (*angr.analyses.StackPointerTracker method*), 923
offsets (*angr.sim_type.SimStruct property*), 701
offsets_for() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 1246
offsets_for() (*angr.analyses.StackPointerTracker method*), 923
OffsetVal (*class in angr.analyses.stack_pointer_tracker*), 1243
offsIP (*angr.engines.pcode.lifter.IRSB property*), 609
oident (*angr.ailment.expression.VirtualVariable attribute*), 1379
oident (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable attribute*), 840
old_hi (*angr.ailment.statement.CAS attribute*), 1404
old_lo (*angr.ailment.statement.CAS attribute*), 1404
omit_header (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 1132
on_worker_exit() (*angr.distributed.Server method*),

1362
on_worker_exit() (*angr.distributed.server.Server method*), 1363
on_worker_exit() (*angr.Server method*), 173
one_active (*angr.sim_manager.SimulationManager attribute*), 545
one_active (*angr.SimulationManager attribute*), 215
one_deadended (*angr.sim_manager.SimulationManager attribute*), 545
one_deadended (*angr.SimulationManager attribute*), 215
one_found (*angr.sim_manager.SimulationManager attribute*), 545
one_found (*angr.SimulationManager attribute*), 215
one_label() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 1270
one_pruned (*angr.sim_manager.SimulationManager attribute*), 545
one_pruned (*angr.SimulationManager attribute*), 215
one_result (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 1216
one_result (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 1200
one_result (*angr.analyses.ReachingDefinitionsAnalysis property*), 914
one_stashed (*angr.sim_manager.SimulationManager attribute*), 545
one_stashed (*angr.SimulationManager attribute*), 215
one_type() (*angr.sim_state_options.StateOption method*), 243
one_unconstrained (*angr.sim_manager.SimulationManager attribute*), 545
one_unconstrained (*angr.SimulationManager attribute*), 215
one_unsat (*angr.sim_manager.SimulationManager attribute*), 545
one_unsat (*angr.SimulationManager attribute*), 215
one_value() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValuedMemory method*), 502
op (*angr.ailment.expression.Call property*), 1394
op (*angr.ailment.expression.DirtyExpression property*), 1388
op (*angr.ailment.expression.FunctionLikeMacro property*), 1397
op (*angr.ailment.expression.Op attribute*), 1381
op (*angr.ailment.expression.Phi property*), 1380
op (*angr.ailment.expression.VEXCCallExpression property*), 1389
op (*angr.analyses.decompiler.region_simplifiers.switch_cluster.SwitchClusterFloodRegion attribute*), 1116
op (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp attribute*), 1140
op (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp attribute*), 1140
op (*angr.engines.light.ArithmeticExpression attribute*), 1167
op (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
op (*angr.state_plugins.sim_action_object.SimActionObject property*), 641
op (*angr.state_plugins.SimActionObject property*), 265
Op (*class in angr.ailment.expression*), 1381
OP_AFTER (*angr.analyses.reaching_definitions.ObservationPointType attribute*), 1197
OP_AFTER (*angr.knowledge_plugins.key_definitions.constants.ObservationPoint attribute*), 841
OP_BEFORE (*angr.analyses.reaching_definitions.ObservationPointType attribute*), 1197
OP_BEFORE (*angr.knowledge_plugins.key_definitions.constants.ObservationPoint attribute*), 841
op_precedence (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp property*), 1141
op_str (*angr.block.CapstoneInsn property*), 232
op_str (*angr.block.DisassemblerInsn property*), 232
op_str (*angr.block.PCodeInsn property*), 233
op_str (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 606
OpBehavior (*class in angr.engines.pcode.behavior*), 616
OpBehaviorBoolAnd (*class in angr.engines.pcode.behavior*), 629
OpBehaviorBoolNegate (*class in angr.engines.pcode.behavior*), 628
OpBehaviorBoolOr (*class in angr.engines.pcode.behavior*), 630
OpBehaviorBoolXor (*class in angr.engines.pcode.behavior*), 629
OpBehaviorCopy (*class in angr.engines.pcode.behavior*), 617
OpBehaviorEqual (*class in angr.engines.pcode.behavior*), 617
OpBehaviorFloatAbs (*class in angr.engines.pcode.behavior*), 632
OpBehaviorFloatAdd (*class in angr.engines.pcode.behavior*), 631
OpBehaviorFloatCeil (*class in angr.engines.pcode.behavior*), 633
OpBehaviorFloatDiv (*class in angr.engines.pcode.behavior*), 632
OpBehaviorFloatEqual (*class in angr.engines.pcode.behavior*), 630
OpBehaviorFloatFloat2Float (*class in angr.engines.pcode.behavior*), 633
OpBehaviorFloatFloor (*class in angr.engines.pcode.behavior*), 634
OpBehaviorFloatInt2Float (*class in angr.engines.pcode.behavior*), 633
OpBehaviorFloatLess (*class in angr.engines.pcode.behavior*), 631

OpBehaviorFloatLessEqual (*class in angr.engines.pcode.behavior*), 631
OpBehaviorFloatMult (*class in angr.engines.pcode.behavior*), 632
OpBehaviorFloatNan (*class in angr.engines.pcode.behavior*), 631
OpBehaviorFloatNeg (*class in angr.engines.pcode.behavior*), 632
OpBehaviorFloatNotEqual (*class in angr.engines.pcode.behavior*), 630
OpBehaviorFloatRound (*class in angr.engines.pcode.behavior*), 634
OpBehaviorFloatSqrt (*class in angr.engines.pcode.behavior*), 633
OpBehaviorFloatSub (*class in angr.engines.pcode.behavior*), 632
OpBehaviorFloatTrunc (*class in angr.engines.pcode.behavior*), 633
OpBehaviorInt2Comp (*class in angr.engines.pcode.behavior*), 623
OpBehaviorIntAdd (*class in angr.engines.pcode.behavior*), 621
OpBehaviorIntAnd (*class in angr.engines.pcode.behavior*), 624
OpBehaviorIntCarry (*class in angr.engines.pcode.behavior*), 622
OpBehaviorIntDiv (*class in angr.engines.pcode.behavior*), 627
OpBehaviorIntLeft (*class in angr.engines.pcode.behavior*), 625
OpBehaviorIntLess (*class in angr.engines.pcode.behavior*), 619
OpBehaviorIntLessEqual (*class in angr.engines.pcode.behavior*), 619
OpBehaviorIntMult (*class in angr.engines.pcode.behavior*), 626
OpBehaviorIntNegate (*class in angr.engines.pcode.behavior*), 623
OpBehaviorIntOr (*class in angr.engines.pcode.behavior*), 625
OpBehaviorIntRem (*class in angr.engines.pcode.behavior*), 628
OpBehaviorIntRight (*class in angr.engines.pcode.behavior*), 625
OpBehaviorIntSborrow (*class in angr.engines.pcode.behavior*), 622
OpBehaviorIntScarry (*class in angr.engines.pcode.behavior*), 622
OpBehaviorIntSdiv (*class in angr.engines.pcode.behavior*), 627
OpBehaviorIntSext (*class in angr.engines.pcode.behavior*), 620
OpBehaviorIntSless (*class in angr.engines.pcode.behavior*), 618
OpBehaviorIntSlessEqual (*class in angr.engines.pcode.behavior*), 619
OpBehaviorIntSrem (*class in angr.engines.pcode.behavior*), 628
OpBehaviorIntSright (*class in angr.engines.pcode.behavior*), 626
OpBehaviorIntSub (*class in angr.engines.pcode.behavior*), 621
OpBehaviorIntXor (*class in angr.engines.pcode.behavior*), 624
OpBehaviorIntZext (*class in angr.engines.pcode.behavior*), 620
OpBehaviorLzcount (*class in angr.engines.pcode.behavior*), 635
OpBehaviorNotEqual (*class in angr.engines.pcode.behavior*), 618
OpBehaviorPiece (*class in angr.engines.pcode.behavior*), 634
OpBehaviorPopcount (*class in angr.engines.pcode.behavior*), 635
OpBehaviorSubpiece (*class in angr.engines.pcode.behavior*), 634
opcode (*angr.analyses.disassembly.Instruction attribute*), 1294
opcode (*angr.engines.pcode.behavior.OpBehavior attribute*), 616
opcode (*angr.engines.pcode.behavior.OpBehaviorBoolAnd attribute*), 630
opcode (*angr.engines.pcode.behavior.OpBehaviorBoolNegate attribute*), 629
opcode (*angr.engines.pcode.behavior.OpBehaviorBoolOr attribute*), 630
opcode (*angr.engines.pcode.behavior.OpBehaviorBoolXor attribute*), 629
opcode (*angr.engines.pcode.behavior.OpBehaviorCopy attribute*), 617
opcode (*angr.engines.pcode.behavior.OpBehaviorEqual attribute*), 618
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatAbs attribute*), 633
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatAdd attribute*), 631
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatCeil attribute*), 634
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatDiv attribute*), 632
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatEqual attribute*), 630
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatFloat2Float attribute*), 633
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatFloor attribute*), 634
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatInt2Float attribute*), 633

opcode (*angr.engines.pcode.behavior.OpBehaviorFloatLess attribute*), 631
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatLessEqual attribute*), 631
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatMult attribute*), 632
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatNan attribute*), 631
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatNeg attribute*), 632
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatNotEqual attribute*), 631
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatRound attribute*), 634
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatSqrt attribute*), 633
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatSub attribute*), 632
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatTrunc attribute*), 633
opcode (*angr.engines.pcode.behavior.OpBehaviorInt2Comp attribute*), 623
opcode (*angr.engines.pcode.behavior.OpBehaviorIntAdd attribute*), 621
opcode (*angr.engines.pcode.behavior.OpBehaviorIntAnd attribute*), 625
opcode (*angr.engines.pcode.behavior.OpBehaviorIntCarry attribute*), 622
opcode (*angr.engines.pcode.behavior.OpBehaviorIntDiv attribute*), 627
opcode (*angr.engines.pcode.behavior.OpBehaviorIntLeft attribute*), 625
opcode (*angr.engines.pcode.behavior.OpBehaviorIntLess attribute*), 619
opcode (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual attribute*), 620
opcode (*angr.engines.pcode.behavior.OpBehaviorIntMult attribute*), 627
opcode (*angr.engines.pcode.behavior.OpBehaviorIntNegate attribute*), 624
opcode (*angr.engines.pcode.behavior.OpBehaviorIntOr attribute*), 625
opcode (*angr.engines.pcode.behavior.OpBehaviorIntRem attribute*), 628
opcode (*angr.engines.pcode.behavior.OpBehaviorIntRight attribute*), 626
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSborrow attribute*), 623
opcode (*angr.engines.pcode.behavior.OpBehaviorIntScarry attribute*), 622
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSdiv attribute*), 628
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSext attribute*), 621
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSless attribute*), 619
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual attribute*), 619
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSrem attribute*), 628
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSright attribute*), 626
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSub attribute*), 622
opcode (*angr.engines.pcode.behavior.OpBehaviorIntXor attribute*), 624
opcode (*angr.engines.pcode.behavior.OpBehaviorIntZext attribute*), 620
opcode (*angr.engines.pcode.behavior.OpBehaviorLzcount attribute*), 635
opcode (*angr.engines.pcode.behavior.OpBehaviorNotEqual attribute*), 618
opcode (*angr.engines.pcode.behavior.OpBehaviorPiece attribute*), 634
opcode (*angr.engines.pcode.behavior.OpBehaviorPopcount attribute*), 635
opcode (*angr.engines.pcode.behavior.OpBehaviorSubpiece attribute*), 635
Opcode (*class in angr.analyses.disassembly*), 1295
OpDescriptor (*class in angr.analyses.decompiler.optimization_passes.expr_op_swapper*), 1102
open() (*angr.state_plugins.posix.SimSystemPosix method*), 352
open() (*angr.state_plugins.SimSystemPosix method*), 322
open_db() (*angr.angrdb.AngrDB method*), 1018
open_db() (*angr.angrdb.db.AngrDB method*), 1020
open_db() (*angr.knowledge_plugins.RuntimeDb method*), 734
open_socket() (*angr.state_plugins.posix.SimSystemPosix method*), 352
open_socket() (*angr.state_plugins.SimSystemPosix method*), 322
opening_symbol (*angr.analyses.decompiler.structured_codegen.c.CClosin attribute*), 1144
operand (*angr.ailment.expression.UnaryOp attribute*), 1381
operand (*angr.ailment.UnaryOp attribute*), 1373
operand (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp attribute*), 1140
Operand (*class in angr.analyses.disassembly*), 1295
Operand (*class in angr.analyses.reassembler*), 1298
operand_str (*angr.analyses.reassembler.DataLabel property*), 1297
operand_str (*angr.analyses.reassembler.FunctionLabel property*), 1298
operand_str (*angr.analyses.reassembler.Label prop-*

erty), 1297
operand_str (*angr.analyses.reassembler.NotypeLabel property*), 1298
operand_str (*angr.analyses.reassembler.ObjectLabel property*), 1298
OperandPiece (*class in angr.analyses.disassembly*), 1295
operands (*angr.ailment.BinaryOp attribute*), 1367
operands (*angr.ailment.expression.BinaryOp attribute*), 1385
operands (*angr.ailment.expression.DirtyExpression attribute*), 1388
operands (*angr.ailment.expression.UnaryOp property*), 1381
operands (*angr.ailment.expression.VEXCCallExpression attribute*), 1389
operands (*angr.ailment.UnaryOp property*), 1373
operands (*angr.analyses.decompiler.structured_codegen.c.CVEXCCallExpression attribute*), 1143
operands (*angr.engines.light.ArithmeticExpression attribute*), 1167
operands (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
OPERATE (*angr.state_plugins.sim_action.SimActionData attribute*), 639
OPERATE (*angr.state_plugins.SimActionData attribute*), 263
operations (*angr.engines.pcode.lifter.IRSB property*), 609
operations (*angr.knowledge_plugins.Function property*), 719
operations (*angr.knowledge_plugins.functions.Function property*), 779
operations (*angr.knowledge_plugins.functions.function.Function property*), 804
Oppologist (*class in angr.exploration_techniques*), 561
Oppologist (*class in angr.exploration_techniques.oppologist*), 581
OPSTR_MAP (*angr.ailment.BinaryOp attribute*), 1366
OPSTR_MAP (*angr.ailment.expression.BinaryOp attribute*), 1384
opt_level (*angr.engines.pcode.lifter.Lifter attribute*), 610
opt_level (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
OptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 1093
OptimizationPassStage (*class in angr.analyses.decompiler.optimization_passes*), 1084
OptimizationPassStage (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 1092
optimize() (*angr.analyses.binary_optimizer.BinaryOptimizer method*), 1307
optimize() (*angr.analyses.BinaryOptimizer method*), 884
optimize() (*angr.analyses.decompiler.peephole_optimizations.base.Peeph method*), 1107
optimize() (*angr.analyses.decompiler.peephole_optimizations.base.Peeph method*), 1106
optimize() (*angr.analyses.decompiler.peephole_optimizations.base.Peeph method*), 1105
OPTIONS (*angr.analyses.cfg.cfg_arch_options.CFGArchOptions attribute*), 992
OPTIONS (*angr.analyses.cfg.CFGArchOptions attribute*), 959
OPTIONS (*angr.analyses.CFGArchOptions attribute*), 884
OPTIONS (*angr.sim_state_options.SimStateOptions attribute*), 243
options (*angr.sim_state.SimLightState attribute*), 1321
options_to_params() (*angr.analyses.Decompiler static method*), 903
options_to_params() (*angr.analyses.decompiler.Decompiler static method*), 1049
options_to_params() (*angr.analyses.decompiler.decompiler.Decompiler static method*), 1068
Or (*angr.engines.light.ArithmeticExpression attribute*), 1167
Or (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
Or1 (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransfor attribute*), 998
orig_ins_addr (*angr.ailment.tagged_object.TagDict attribute*), 1407
original_node (*angr.analyses.decompiler.optimization_passes.lowered_s attribute*), 1099
original_size (*angr.ailment.Block attribute*), 1368
original_size (*angr.ailment.block.Block attribute*), 1406
os_types (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
other_input_defns (*angr.analyses.reaching_definitions.function_handle attribute*), 1226
other_input_defns (*angr.analyses.reaching_definitions.FunctionCallRel attribute*), 1186
other_output_defns (*angr.analyses.reaching_definitions.function_handl attribute*), 1227
other_output_defns (*angr.analyses.reaching_definitions.FunctionCallRe attribute*), 1186
other_types (*angr.procedures.stubs.format_parser.FormatParser attribute*), 648
other_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
other_uses (*angr.knowledge_plugins.key_definitions.live_definitions.Live*

attribute), 848
other_uses (angr.knowledge_plugins.key_definitions.LiveDefinitions attribute), 825
others (angr.analyses.reaching_definitions.LiveDefinitions attribute), 1190
others (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property), 1233
others (angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property), 1206
others (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute), 848
others (angr.knowledge_plugins.key_definitions.LiveDefinitions attribute), 825
OUT (angr.sim_type.PointerDisposition attribute), 694
out_graph (angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute), 1095
OutdatedError, 973
outerscope_uses (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter attribute), 1112
OuterWalker (class in angr.analyses.decompiler.optimization_passes.expr_op_swapper), 1102
OUTMAYBE (angr.sim_type.PointerDisposition attribute), 694
output() (angr.analyses.cfg.cfg_base.CFGBase method), 982
output() (angr.analyses.cfg.cfg_fast.CFGFast method), 992
output() (angr.analyses.cfg.CFGBase method), 961
output() (angr.analyses.cfg.CFGFast method), 970
output() (angr.analyses.CFGFast method), 892
OVERFLOW_FP_RETURN_VAL (angr.calling_conventions.SimCCSystemVAMD64 attribute), 671
OVERFLOW_RETURN_VAL (angr.calling_conventions.SimCC attribute), 664
OVERFLOW_RETURN_VAL (angr.calling_conventions.SimCCARM attribute), 672
OVERFLOW_RETURN_VAL (angr.calling_conventions.SimCCARMHF attribute), 673
OVERFLOW_RETURN_VAL (angr.calling_conventions.SimCCCdecl attribute), 668
OVERFLOW_RETURN_VAL (angr.calling_conventions.SimCCMicrosoftAMD64 attribute), 669
OVERFLOW_RETURN_VAL (angr.calling_conventions.SimCCMicrosoftFastcall attribute), 669
OVERFLOW_RETURN_VAL (angr.calling_conventions.SimCCO32 attribute), 675
OVERFLOW_RETURN_VAL (angr.calling_conventions.SimCCPowerPC attribute), 677
OVERFLOW_RETURN_VAL (angr.calling_conventions.SimCCSystemVAMD64 attribute), 671
OVERFLOW_RETURN_VAL (angr.SimCC attribute), 174
overlap() (angr.knowledge_plugins.patches.PatchManager method), 738
overlap() (angr.knowledge_plugins.PatchManager static method), 732

P

pack (angr.sim_type.SimStruct property), 701
packed (angr.sim_type.SimStruct property), 701
PACKED_ENTROPY_MIN_THRESHOLD (angr.analyses.PackingDetector attribute), 910
PACKED_MIN_BYTES (angr.analyses.PackingDetector attribute), 910
PackingDetector (class in angr.analyses), 909
pad_chr (angr.procedures.stubs.format_parser.FormatSpecifier attribute), 647
PAGE_TYPE (angr.storage.memory_mixins.ListPagesMixin attribute), 469
PAGE_TYPE (angr.storage.memory_mixins.MVListPagesMixin attribute), 471
PAGE_TYPE (angr.storage.memory_mixins.paged_memory.paged_memory_m attribute), 507
PAGE_TYPE (angr.storage.memory_mixins.paged_memory.paged_memory_m attribute), 507
PAGE_TYPE (angr.storage.memory_mixins.paged_memory.paged_memory_m attribute), 504
PAGE_TYPE (angr.storage.memory_mixins.paged_memory.paged_memory_m attribute), 508
PAGE_TYPE (angr.storage.memory_mixins.PagedMemoryMixin attribute), 474
PAGE_TYPE (angr.storage.memory_mixins.UltraPagesMixin attribute), 489
PageBase (class in angr.storage.memory_mixins), 474
PageBase (class in angr.storage.memory_mixins.paged_memory.pages), 515
PagedMemoryMixin (class in angr.storage.memory_mixins), 474
PagedMemoryMixin (class in angr.storage.memory_mixins.paged_memory.paged_memory_mix 504
PagedMemoryMultiValueMixin (class in angr.storage.memory_mixins), 477
PARAMETER (angr.ailment.expression.VirtualVariableCategory attribute), 1379
parameter_category (angr.ailment.expression.VirtualVariable property), 1380

parameter_reg_offset (*angr.ailment.expression.VirtualVariable property*), 1380
parameter_stack_offset (*angr.ailment.expression.VirtualVariable property*), 1380
parameters (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 1064
ParameterTag (*class in angr.knowledge_plugins.key_definitions.tag*), 858
parent (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.ConditionalRegion attribute*), 1116
parent (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchCaseRegion attribute*), 1116
parentop (*angr.analyses.disassembly.OperandPiece attribute*), 1295
parents (*angr.state_plugins.history.SimStateHistory property*), 377
parents (*angr.state_plugins.SimStateHistory property*), 303
parents() (*angr.storage.memory_mixins.HistoryTrackingMixin method*), 462
parents() (*angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.HistoryTrackingMixin method*), 520
parents() (*angr.storage.memory_mixins.paged_memory.pages.HistoryTrackingMixin method*), 511
parse() (*angr.analyses.flirt.FlirtSignatureParsed class method*), 1165
parse() (*angr.serializable.Serializable class method*), 871
parse_block() (*angr.analyses.Disassembly method*), 904
parse_block() (*angr.analyses.disassembly.Disassembly method*), 1296
parse_cpp_file() (*in module angr.sim_type*), 710
parse_defns() (*in module angr.sim_type*), 709
parse_file() (*in module angr.sim_type*), 709
parse_from_cmessage() (*angr.Block class method*), 161
parse_from_cmessage() (*angr.block.Block class method*), 234
parse_from_cmessage() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel class method*), 757
parse_from_cmessage() (*angr.knowledge_plugins.cfg.cfg_node.CFGENode class method*), 769
parse_from_cmessage() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode class method*), 768
parse_from_cmessage() (*angr.knowledge_plugins.cfg.CFGENode class method*), 741
parse_from_cmessage() (*angr.knowledge_plugins.cfg.CFGModel class method*), 743
parse_from_cmessage() (*angr.knowledge_plugins.cfg.CFGNode class method*), 751
parse_from_cmessage() (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump class method*), 772
parse_from_cmessage() (*angr.knowledge_plugins.cfg.indirect_jump.JumptableInfo class method*), 770
parse_from_cmessage() (*angr.knowledge_plugins.cfg.IndirectJump class method*), 753
parse_from_cmessage() (*angr.knowledge_plugins.cfg.memory_data.MemoryData class method*), 766
parse_from_cmessage() (*angr.knowledge_plugins.cfg.MemoryData class method*), 755
parse_from_cmessage() (*angr.knowledge_plugins.Function class method*), 719
parse_from_cmessage() (*angr.knowledge_plugins.functions.Function class method*), 779
parse_from_cmessage() (*angr.knowledge_plugins.functions.function.Function class method*), 804
parse_from_cmessage() (*angr.knowledge_plugins.variables.variable_access.VariableAcce class method*), 812
parse_from_cmessage() (*angr.knowledge_plugins.variables.variable_manager.VariableMa class method*), 813
parse_from_cmessage() (*angr.knowledge_plugins.XRefManager class method*), 737
parse_from_cmessage() (*angr.knowledge_plugins.xrefs.XRef class method*), 862
parse_from_cmessage() (*angr.knowledge_plugins.xrefs.xref.XRef class method*), 864
parse_from_cmessage() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager class method*), 865
parse_from_cmessage() (*angr.knowledge_plugins.xrefs.XRefManager class method*), 862
parse_from_cmessage() (*angr.serializable.Serializable class method*), 871

parse_from_cmessage() (*angr.sim_variable.SimComboRegisterVariable class method*), 684
parse_from_cmessage() (*angr.sim_variable.SimConstantVariable class method*), 682
parse_from_cmessage() (*angr.sim_variable.SimMemoryVariable class method*), 685
parse_from_cmessage() (*angr.sim_variable.SimRegisterVariable class method*), 683
parse_from_cmessage() (*angr.sim_variable.SimStackVariable class method*), 686
parse_from_cmessage() (*angr.sim_variable.SimTemporaryVariable class method*), 682
parse_from_cmsg() (*angr.knowledge_plugins.functions.function_parser.FunctionParser static method*), 810
parse_header() (*angr.analyses.flirt.FlirtSignatureParsed class method*), 1165
parse_module() (*angr.analyses.flirt.FlirtSignatureParsed method*), 1165
parse_modules() (*angr.analyses.flirt.FlirtSignatureParsed method*), 1164
parse_node() (*angr.analyses.flirt.FlirtSignatureParsed static method*), 1165
parse_public_function() (*angr.analyses.flirt.FlirtSignatureParsed method*), 1164
parse_referenced_functions() (*angr.analyses.flirt.FlirtSignatureParsed method*), 1164
parse_signature() (*in module angr.sim_type*), 709
parse_stack_pointer() (*in module angr.analyses.variable_recovery.variable_recovery_base*), 1247
parse_tail_bytes() (*angr.analyses.flirt.FlirtSignatureParsed method*), 1164
parse_tree() (*angr.analyses.flirt.FlirtSignatureParsed method*), 1164
parse_type() (*in module angr.sim_type*), 709
parse_type_with_name() (*in module angr.sim_type*), 709
parse_types() (*in module angr.sim_type*), 709
parse_variable_addr() (*angr.analyses.decompiler.Clinic method*), 1045
parse_variable_addr() (*angr.analyses.decompiler.clinic.Clinic method*), 1061
parse_variant_mask() (*angr.analyses.flirt.FlirtSignatureParsed static method*), 1165
parsedcprotos2py() (*in module angr.utils.library*), 1352
ParsedInstruction (*class in angr.analyses.data_dep.sim_act_location*), 1313
PartialConstantExprRewriter (*class in angr.analyses.decompiler.ail_simplifier*), 1053
Patch (*class in angr.knowledge_plugins.patches*), 737
patch_addrs() (*angr.knowledge_plugins.patches.PatchManager method*), 737
patch_addrs() (*angr.knowledge_plugins.PatchManager method*), 732
patched_entry_state (*angr.knowledge_plugins.patches.PatchManager property*), 738
patched_entry_state (*angr.knowledge_plugins.PatchManager property*), 732
PatchFinderAnalysis (*class in angr.analyses*), 910
PatchManager (*class in angr.knowledge_plugins*), 731
PatchManager (*class in angr.knowledge_plugins.patches*), 737
path (*angr.angrdb.models.DbObject attribute*), 1022
path() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 1270
path_between() (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGS method*), 1242
path_between() (*angr.analyses.cfg_slice_to_sink.CFGSliceToSink method*), 1239
Pathfinder (*class in angr.analyses*), 910
PathUnreachableError, 168, 1355
pattern_size (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
pcode (*angr.Block property*), 161
pcode (*angr.block.Block property*), 234
PcodeBasicBlockLifter (*class in angr.engines.pcode.lifter*), 612
PCodeBlock (*class in angr.block*), 232
PcodeDisassemblerBlock (*class in angr.engines.pcode.lifter*), 605
PcodeDisassemblerInsn (*class in angr.engines.pcode.lifter*), 606
PcodeEmulatorMixin (*class in angr.engines.pcode.emulate*), 615
PCodeInsn (*class in angr.block*), 232
PCodeIRSBConverter (*class in angr.ailment*), 1370
PCodeIRSBConverter (*class in angr.ailment.converter_pcode*), 1373
PcodeLifter (*class in angr.engines.pcode.lifter*), 612
PcodeLifterEngineMixin (*class in angr.engines.pcode.lifter*), 613
PEDelayImportDirectory

(*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
PEDelayImportDirectory (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
peek_input() (*angr.state_plugins.cgc.SimStateCGC method*), 380
peek_input() (*angr.state_plugins.SimStateCGC method*), 297
peek_output() (*angr.state_plugins.cgc.SimStateCGC method*), 380
peek_output() (*angr.state_plugins.SimStateCGC method*), 297
peephole_optimize_expr() (*in module angr.analyses.decompiler.utils*), 1154
peephole_optimize_exprs() (*in module angr.analyses.decompiler.utils*), 1154
peephole_optimize_multistmts() (*in module angr.analyses.decompiler.utils*), 1154
peephole_optimize_stmts() (*in module angr.analyses.decompiler.utils*), 1154
PeepholeOptimizationExprBase (*class in angr.analyses.decompiler.peephole_optimizations.base*), 1106
PeepholeOptimizationMultiStmtBase (*class in angr.analyses.decompiler.peephole_optimizations.base*), 1105
PeepholeOptimizationStmtBase (*class in angr.analyses.decompiler.peephole_optimizations.base*), 1104
PEExportDirectory (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
PEExportDirectory (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
PEImportDirectory (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
PEImportDirectory (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 756
PendingJob (*class in angr.analyses.cfg.cfg_emulated*), 976
PendingJob (*class in angr.analyses.vfg*), 1285
PendingJobs (*class in angr.analyses.cfg.cfg_fast*), 984
perform_call() (*angr.callable.Callable method*), 711
perm_exec (*angr.storage.memory_mixins.paged_memory.pages.PermissionsMixin property*), 519
perm_exec (*angr.storage.memory_mixins.paged_memory.pages.PermissionsMixin property*), 516
perm_exec (*angr.storage.memory_mixins.PermissionsMixin property*), 477
perm_read (*angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.PermissionsMixin property*), 519
perm_read (*angr.storage.memory_mixins.paged_memory.pages.PermissionsMixin property*), 515
perm_read (*angr.storage.memory_mixins.PermissionsMixin property*), 477
perm_write (*angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.PermissionsMixin property*), 519
perm_write (*angr.storage.memory_mixins.paged_memory.pages.PermissionsMixin property*), 515
perm_write (*angr.storage.memory_mixins.PermissionsMixin property*), 477
permissions() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 497
permissions() (*angr.storage.memory_mixins.AddressConcretizationMixin method*), 458
permissions() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 506
permissions() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 476
PermissionsMixin (*class in angr.storage.memory_mixins*), 477
PermissionsMixin (*class in angr.storage.memory_mixins.paged_memory.pages*), 515
PermissionsMixin (*class in angr.storage.memory_mixins.paged_memory.pages.permissions_mixin*), 519
Permissive (*angr.exploration_techniques.tracer.TracingMode attribute*), 575
persistent_id() (*angr.vaults.VaultPickler method*), 871
persistent_load() (*angr.vaults.VaultUnpickler method*), 871
Phi (*class in angr.ailment.expression*), 1380
phi_stmt_ref (*angr.analyses.decompiler.region_simplifiers.expr_folding.ExprFolding attribute*), 1110
phi_stmt (*angr.analyses.decompiler.region_simplifiers.expr_folding.StatementFolding attribute*), 1110
phi_variables (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBaseAnalysis attribute*), 1249
phi_vars (*angr.analyses.decompiler.expression_narrower.ExprNarrowingBase attribute*), 1069
PhoenixStructurer (*class in angr.analyses.decompiler.structuring*), 1033
PhoenixStructurer (*class in angr.analyses.decompiler.structuring.phoenix*), 1038
PickledStates (*angr.exploration_techniques.spiller.PickledStatesMixin in angr.exploration_techniques.spiller_db*), 574
PickledStatesBase (*class in angr.exploration_techniques.spiller*), 571
PickledStatesDb (*class in angr.exploration_techniques.spiller*), 572
PickledStatesList (*class in angr.exploration_techniques.spiller*), 572
PLATFORMS (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePtrSaveSimplifier attribute*), 1097

PLATFORMS (*angr.analyses.decompiler.optimization_passes.LowerSwitch attribute*), 1072
PLATFORMS (*angr.analyses.decompiler.optimization_passes.MainSimplifier attribute*), 1072
PLATFORMS (*angr.analyses.decompiler.optimization_passes.MultiSimplifier attribute*), 1073
PLATFORMS (*angr.analyses.decompiler.optimization_passes.NullptrPropagation attribute*), 1074
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ConstantDereferenceSimplifier attribute*), 1091
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ReferenceSimplifier attribute*), 1075
PLATFORMS (*angr.analyses.decompiler.optimization_passes.OutputRewriter attribute*), 1074
PLATFORMS (*angr.analyses.decompiler.optimization_passes.PtrSimplifier attribute*), 1076
PLATFORMS (*angr.analyses.decompiler.optimization_passes.PtrDivSimplifier attribute*), 1098
PLATFORMS (*angr.analyses.decompiler.optimization_passes.PtrArithSimplifier attribute*), 1076
PLATFORMS (*angr.analyses.decompiler.optimization_passes.LogicalConsecutiveIfs attribute*), 1078
PLATFORMS (*angr.analyses.decompiler.optimization_passes.LogicalOrSimplifier attribute*), 1103
PLATFORMS (*angr.analyses.decompiler.optimization_passes.LogicalAndSimplifier attribute*), 1079
PLATFORMS (*angr.analyses.decompiler.optimization_passes.LibraryCallSimplifier attribute*), 1079
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ConstantSimplifier attribute*), 1080
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ArithmeticSimplifier attribute*), 1080
PLATFORMS (*angr.analyses.decompiler.optimization_passes.SystemCallSimplifier attribute*), 1081
PLATFORMS (*angr.analyses.decompiler.optimization_passes.TransformationSimplifier attribute*), 1081
PLATFORMS (*angr.analyses.decompiler.optimization_passes.PhiSimplifier attribute*), 1082
PLATFORMS (*angr.analyses.decompiler.optimization_passes.CopyPropagation attribute*), 1082
PLATFORMS (*angr.analyses.decompiler.optimization_passes.plugin_preset.TEEmpiricPlugins attribute*), 1099
PLATFORMS (*angr.analyses.decompiler.optimization_passes.PlugInHub attribute*), 1079
PLATFORMS (*angr.analyses.decompiler.optimization_passes.PlugInCondition attribute*), 1080
PLATFORMS (*angr.analyses.decompiler.optimization_passes.PluginVendor attribute*), 1100
PLATFORMS (*angr.analyses.decompiler.optimization_passes.LoweredSwitchSimplifier attribute*), 1083
PLATFORMS (*angr.analyses.decompiler.optimization_passes.MipsGpSettingSimplifier attribute*), 1083
PLATFORMS (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 1101
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 1084
PLATFORMS (*angr.analyses.decompiler.optimization_passes.optimization_pa attribute*), 1093
PLATFORMS (*angr.analyses.decompiler.optimization_passes.optimization_pa attribute*), 1096
PLATFORMS (*angr.analyses.decompiler.optimization_passes.register_save_a attribute*), 1103
PLATFORMS (*angr.analyses.decompiler.optimization_passes.RegisterSaveAre attribute*), 1085
PLATFORMS (*angr.analyses.decompiler.optimization_passes.RegisterSaveAre attribute*), 1085
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ret_addr_save_ attribute*), 1103
PLATFORMS (*angr.analyses.decompiler.optimization_passes.RetAddrSaveSim attribute*), 1085
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ReturnDeduplic attribute*), 1086
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ReturnDuplicat attribute*), 1086
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ReturnDuplicat attribute*), 1088
PLATFORMS (*angr.analyses.decompiler.optimization_passes.stack_canary_s attribute*), 1097
PLATFORMS (*angr.analyses.decompiler.optimization_passes.StackCanarySim attribute*), 1088
PLATFORMS (*angr.analyses.decompiler.optimization_passes.StaticVVarRewr attribute*), 1089
PLATFORMS (*angr.analyses.decompiler.optimization_passes.SwitchDefaultC attribute*), 1089
PLATFORMS (*angr.analyses.decompiler.optimization_passes.SwitchReusedE attribute*), 1090
PLATFORMS (*angr.analyses.decompiler.optimization_passes.TagSlicer attribute*), 1090
PLATFORMS (*angr.analyses.decompiler.optimization_passes.WinStackCanar attribute*), 1090
PLATFORMS (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_ attribute*), 1104
PLATFORMS (*angr.analyses.decompiler.optimization_passes.X86GccGetPcS attribute*), 1091
plugin_preset (*angr.misc.plugins.PluginHub property*), 235
PlugInHub (*class in angr.misc.plugins*), 235
PluginPreset (*class in angr.misc.plugins*), 236
plugins (*angr.SimState property*), 240
plugins (*angr.sim_state.SimState property*), 208
PlugInVendor (*class in angr.misc.plugin*), 237
Pointer (*class in angr.analyses.typehoon.typeconsts*), 1277
Pointer32 (*class in angr.analyses.typehoon.typeconsts*), 1278
Pointer64 (*class in angr.analyses.typehoon.typeconsts*), 1278
pointer_addr (*angr.knowledge_plugins.cfg.memory_data.MemoryData*

attribute), 765
pointer_addr (angr.knowledge_plugins.cfg.MemoryData attribute), 754
PointerArithmeticFixer (class in angr.analyses.decompiler.structured_codegen.c), 1148
PointerArray (angr.knowledge_plugins.cfg.memory_data.MemoryData attribute), 764
PointerArray (angr.knowledge_plugins.cfg.MemoryDataSpec attribute), 755
PointerDisposition (class in angr.sim_type), 694
PointerWrapper (class in angr), 168
PointerWrapper (class in angr.calling_conventions), 658
pop() (angr.state_plugins.CallStack method), 250
pop() (angr.state_plugins.callstack.CallStack method), 374
pop() (angr.state_plugins.globals.SimStateGlobals method), 387
pop() (angr.state_plugins.SimStateGlobals method), 300
pop() (angr.utils.cowdict.ChainMapCOW method), 1345
pop_dirty() (angr.analyses.stack_pointer_tracker.StackPointerTracker method), 1245
pop_from_backup() (angr.state_plugins.trace_additions.ConstRecorder method), 384
pop_job() (angr.analyses.cfg.cfg_fast.PendingJobs method), 985
pop_n() (angr.exploration_techniques.spiller.PickledStatesBase method), 572
pop_n() (angr.exploration_techniques.spiller.PickledStatesPool method), 573
pop_n() (angr.exploration_techniques.spiller.PickledStatesSpiller method), 572
pop_priv() (angr.state_plugins.scratch.SimStateScratch method), 388
pop_priv() (angr.state_plugins.SimStateScratch method), 318
pop_stack_frame() (angr.storage.memory_mixins.JavaVmMemoryMixin method), 464
populate() (angr.sim_manager.SimulationManager method), 548
populate() (angr.SimulationManager method), 218
pos (angr.SimFileBase attribute), 181
pos (angr.storage.file.SimFileBase attribute), 433
PositionMapping (class in angr.analyses.decompiler.structured_codegen), 1124
PositionMapping (class in angr.analyses.decompiler.structured_codegen.base), 1128
PositionMappingElement (class in angr.analyses.decompiler.structured_codegen), 1125
PositionMappingElement (class in angr.analyses.decompiler.structured_codegen.base), 1128
posix (angr.sim_state.SimState attribute), 239
posix (angr.SimState attribute), 208
PosixDevFS (class in angr.state_plugins), 259
PosixDevFS (class in angr.state_plugins.posix), 347
PosixProcFS (class in angr.state_plugins), 261
PosixProcFS (class in angr.state_plugins.posix), 349
posmap_pos (angr.analyses.decompiler.structured_codegen.base.InstructionMap attribute), 1128
posmap_pos (angr.analyses.decompiler.structured_codegen.InstructionMap attribute), 1124
PossibleObject (class in angr.analyses.find_objects_static), 1292
possibly_patched_out (angr.analyses.PatchFinderAnalysis attribute), 910
POST_CALLSITES (angr.analyses.decompiler.clinic.ClinicStage attribute), 1057
post_dom (angr.utils.graph.PostDominators property), 1349
post_filter() (angr.exploration_techniques.StubStasher static method), 564
POST_FORGOTTEN (angr.analyses.typehoon.simple_solver.FORGOTTEN attribute), 1261
POST_SSA_LEVEL1_SIMPLIFICATIONS (angr.analyses.decompiler.clinic.ClinicStage attribute), 1058
PostDominators (class in angr.utils.graph), 1348
pp() (angr.ailment.Block method), 1368
pp() (angr.ailment.block.Block method), 1407
pp() (angr.analyses.datagraph_meta.DataGraphMeta method), 1018
pp() (angr.analyses.DDG method), 876
pp() (angr.analyses.ddg.DDG method), 1160
pp() (angr.Block method), 160
pp() (angr.block.Block method), 233
pp() (angr.block.DisassemblerBlock method), 232
pp() (angr.engines.pcode.lifter.IRSB method), 609
pp() (angr.knowledge_plugins.Function method), 724
pp() (angr.knowledge_plugins.functions.Function method), 785
pp() (angr.knowledge_plugins.functions.function.Function method), 809
pp_constraints() (angr.analyses.Typehoon method), 925
pp_constraints() (angr.analyses.typehoon.Typehoon method), 1282
pp_constraints() (angr.analyses.typehoon.typehoon.Typehoon method), 1274
pp_solution() (angr.analyses.Typehoon method), 925
pp_solution() (angr.analyses.typehoon.Typehoon method), 1282
pp_solution() (angr.analyses.typehoon.typehoon.Typehoon

method), 1274
pp_str() (*angr.analyses.typehoon.typeconsts.TypeConstant* *method*), 1274
pp_str() (*angr.analyses.typehoon.typevars.Add* *method*), 1268
pp_str() (*angr.analyses.typehoon.typevars.DerivedTypeVariable* *method*), 1270
pp_str() (*angr.analyses.typehoon.typevars.Equivalence* *method*), 1266
pp_str() (*angr.analyses.typehoon.typevars.Existence* *method*), 1267
pp_str() (*angr.analyses.typehoon.typevars.Sub* *method*), 1268
pp_str() (*angr.analyses.typehoon.typevars.Subtype* *method*), 1267
pp_str() (*angr.analyses.typehoon.typevars.TypeConstraint* *method*), 1266
pp_str() (*angr.analyses.typehoon.typevars.TypeVariable* *method*), 1269
PRE_FORGOTTEN (*angr.analyses.typehoon.simple_solver.FORGOTTEN* *attribute*), 1261
PRE_SSA_LEVEL0_FIXUPS (*angr.analyses.decompiler.clinic.ClinicStage* *attribute*), 1057
PRE_SSA_LEVEL1_SIMPLIFICATIONS (*angr.analyses.decompiler.clinic.ClinicStage* *attribute*), 1057
preconstrain() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer* *method*), 391
preconstrain() (*angr.state_plugins.SimStatePreconstrainer* *method*), 317
preconstrain_file() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer* *method*), 391
preconstrain_file() (*angr.state_plugins.SimStatePreconstrainer* *method*), 317
preconstrain_flag_page() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer* *method*), 391
preconstrain_flag_page() (*angr.state_plugins.SimStatePreconstrainer* *method*), 317
predecessors (*angr.exploration_techniques.tracer.Tracer* *attribute*), 576
predecessors (*angr.knowledge_plugins.cfg.cfg_node.CFGNode* *property*), 767
predecessors (*angr.knowledge_plugins.cfg.CFGNode* *property*), 750
predecessors() (*angr.analyses.forward_analysis.CallGraphVisitor* *method*), 931
predecessors() (*angr.analyses.forward_analysis.FunctionGraphVisitor* *method*), 933
predecessors() (*angr.analyses.forward_analysis.LoopVisitor* *method*), 934
predecessors() (*angr.analyses.forward_analysis.SingleNodeGraphVisitor* *method*), 934
predecessors() (*angr.analyses.forward_analysis.visitors.call_graph.CallGraphVisitor* *method*), 941
predecessors() (*angr.analyses.forward_analysis.visitors.CallGraphVisitor* *method*), 938
predecessors() (*angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor* *method*), 942
predecessors() (*angr.analyses.forward_analysis.visitors.FunctionGraphVisitor* *method*), 939
predecessors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor* *method*), 943
predecessors() (*angr.analyses.forward_analysis.visitors.loop.LoopVisitor* *method*), 945
predecessors() (*angr.analyses.forward_analysis.visitors.LoopVisitor* *method*), 939
predecessors() (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor* *method*), 945
predecessors() (*angr.analyses.forward_analysis.visitors.SingleNodeGraphVisitor* *method*), 940
predecessors() (*angr.analyses.reaching_definitions.dep_graph.DepGraph* *method*), 1218
predecessors() (*angr.codenode.CodeNode* *method*), 1323
predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode* *method*), 767
predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.CFGNode* *method*), 751
prep() (*angr.analyses.analysis.AnalysisFactory* *method*), 930
prep_tracer() (*angr.state_plugins.trace_additions.ChallRespInfo* *static method*), 384
prep_tracer() (*angr.state_plugins.trace_additions.ZenPlugin* *static method*), 386
prepare_call_state() (*angr.SimOS* *method*), 196
prepare_call_state() (*angr.simos.SimOS* *method*), 1329
prepare_call_state() (*angr.simos.simos.SimOS* *method*), 1334
prepare_function_symbol() (*angr.SimOS* *method*), 196
prepare_function_symbol() (*angr.simos.linux.SimLinux* *method*), 1336
prepare_function_symbol() (*angr.simos.SimLinux* *method*), 1327
prepare_function_symbol() (*angr.simos.SimOS* *method*), 1329
prepare_function_symbol() (*angr.simos.simos.SimOS* *method*), 1334
prepare_native_return_state() (*angr.engines.soot.engine.SootMixin* *static*

method), 602
prepare_native_return_state() (angr.engines.soot.SootMixin static method), 602
prepare_native_return_state() (angr.engines.SootMixin static method), 593
prepare_native_return_state() (in module angr.simos.javavm), 1341
prepare_return_state() (angr.engines.soot.engine.SootMixin static method), 602
prepare_return_state() (angr.engines.soot.SootMixin static method), 602
prepare_return_state() (angr.engines.SootMixin static method), 593
preprocess() (angr.analyses.typehoon.simple_solver.SimpleSolver method), 1263
preserve_vvar_ids (angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase attribute), 1107
preserve_vvar_ids (angr.analyses.decompiler.peephole_optimizations.base_peephole_optimization.PeepholeOptimizationMultiStmtBase attribute), 1106
preserve_vvar_ids (angr.analyses.decompiler.peephole_optimizations.base_optimization.OptimizationBase attribute), 1105
prettify_thiscall (angr.analyses.decompiler.structured_codegen.CFunctionCall property), 1137
prev_chunk() (angr.PTChunk method), 167
prev_chunk() (angr.state_plugins.heap.heap_freelist.ChunkFreelist method), 414
prev_chunk() (angr.state_plugins.heap.heap_ptmalloc.PTChunkPtmalloc method), 417
prev_chunk() (angr.state_plugins.heap.PTChunk method), 405
prev_chunk() (angr.state_plugins.PTChunk method), 258
prev_size() (angr.PTChunk method), 167
prev_size() (angr.state_plugins.heap.heap_ptmalloc.PTChunkPtmalloc method), 417
prev_size() (angr.state_plugins.heap.PTChunk method), 405
prev_size() (angr.state_plugins.PTChunk method), 258
previous_names (angr.knowledge_plugins.Function attribute), 717
previous_names (angr.knowledge_plugins.functions.Function attribute), 777
previous_names (angr.knowledge_plugins.functions.function_manager.FunctionManager attribute), 802
previous_names (angr.knowledge_plugins.functions.soot_function.SootFunction attribute), 810
print_all_chunks() (angr.state_plugins.heap.heap_freelist.SimHeapFreelist method), 415
print_heap_state() (angr.state_plugins.heap.heap_freelist.SimHeapFreelist method), 415
print_timing_total() (in module angr.utils.timing), 1353
PRINTABLES (angr.analyses.cfg.cfg_fast.CFGFast attribute), 990
PRINTABLES (angr.analyses.cfg.CFGFast attribute), 967
PRINTABLES (angr.analyses.CFGFast attribute), 890
prioritize_functions() (angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis method), 954
prioritize_functions() (angr.analyses.CompleteCallingConventionsAnalysis method), 898
priority (angr.exploration_techniques.spiller_db.PickledState attribute), 574
priv (angr.state_plugins.scratch.SimStateScratch property), 388
priv (angr.state_plugins.SimStateScratch property), 318
PrivilegedPagingMixin (class in angr.storage.paging.paging_mixin), 477
PrivilegedPagingMixin (class in angr.storage.paging.multi_stmt_base_privileged_mixin), 510
probably_identical_to (angr.analyses.binsiff.FunctionDiff property), 948
procedure (angr.analyses.reassembler), 1300
procedure_engine (angr.factory.AngrObjectFactory attribute), 228
ProcedureChunk (class in angr.analyses.reassembler), 1301
ProcedureEngine (class in angr.engines), 588
ProcedureEngine (class in angr.engines.procedure), 597
ProcedureMixin (class in angr.engines), 589
ProcedureMixin (class in angr.engines.procedure), 597
process() (angr.analyses.propagator.engine_base.SimEnginePropagatorBase method), 1177
process() (angr.analyses.reaching_definitions.engine_ail.SimEngineRDAIL method), 1238
process() (angr.analyses.reaching_definitions.engine_vex.SimEngineRDVEX method), 1213
process() (angr.analyses.variable_recovery.irsb_scanner.VEXIRSBScanner method), 1257
process() (angr.engines.light.engine.SimEngineLight method), 1171
process() (angr.engines.light.engine.SimEngineLightAIL method), 1174
process() (angr.engines.light.engine.SimEngineLightVEX method), 1173
process() (angr.engines.light.SimEngineLight method), 1168
process() (angr.engines.light.SimEngineLightAIL method), 1169
process() (angr.engines.light.SimEngineLightVEX

method), 1170
`process()` (*angr.engines.successors.SuccessorsEngine method*), 596
`process()` (*angr.engines.SuccessorsEngine method*), 593
`process()` (*angr.engines.vex.VEXSlicingMixin method*), 601
`process_exc_file()` (*in module angr.flirt.build_sig*), 1343
`process_procedure()` (*angr.engines.procedure.ProcedureMixin method*), 597
`process_procedure()` (*angr.engines.ProcedureMixin method*), 589
`process_successors()` (*angr.engines.failure.SimEngineFailure method*), 598
`process_successors()` (*angr.engines.HeavyVEXMixin method*), 587
`process_successors()` (*angr.engines.hook.HooksMixin method*), 597
`process_successors()` (*angr.engines.HooksMixin method*), 588
`process_successors()` (*angr.engines.pcode.engine.HeavyPcodeMixin method*), 605
`process_successors()` (*angr.engines.pcode.HeavyPcodeMixin method*), 604
`process_successors()` (*angr.engines.procedure.ProcedureEngine method*), 597
`process_successors()` (*angr.engines.ProcedureEngine method*), 588
`process_successors()` (*angr.engines.SimEngineFailure method*), 589
`process_successors()` (*angr.engines.SimEngineSyscall method*), 590
`process_successors()` (*angr.engines.SimEngineUnicorn method*), 590
`process_successors()` (*angr.engines.soot.engine.SootMixin method*), 602
`process_successors()` (*angr.engines.soot.SootMixin method*), 601
`process_successors()` (*angr.engines.SootMixin method*), 592
`process_successors()` (*angr.engines.successors.SuccessorsEngine method*), 596
`process_successors()` (*angr.engines.SuccessorsEngine method*), 593
`process_successors()` (*angr.engines.syscall.SimEngineSyscall method*), 598
`process_successors()` (*angr.engines.unicorn.SimEngineUnicorn method*), 603
`process_successors()` (*angr.engines.vex.HeavyVEXMixin method*), 599
`processed_constraints_count` (*angr.analyses.typehoon.simple_solver.SimpleSolver attribute*), 1263
`ProgramVariable` (*class in angr.analyses.ddg*), 1156
`project` (*angr.analyses.Analysis attribute*), 881
`project` (*angr.analyses.analysis.Analysis attribute*), 930
`project` (*angr.analyses.decompiler.ccall_rewriters.rewriter_base.CCallRe attribute*), 1056
`project` (*angr.analyses.decompiler.optimization_passes.optimization_pass property*), 1093
`project` (*angr.analyses.decompiler.peephole_optimizations.base.Peephole attribute*), 1107
`project` (*angr.analyses.decompiler.peephole_optimizations.base.Peephole attribute*), 1106
`project` (*angr.analyses.decompiler.peephole_optimizations.base.Peephole attribute*), 1105
`project` (*angr.analyses.disassembly.Value property*), 1296
`project` (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
`project` (*angr.Analysis attribute*), 156
`project` (*angr.factory.AngrObjectFactory attribute*), 228
`project` (*angr.knowledge_plugins.cfg.cfg_model.CFGModel property*), 757
`project` (*angr.knowledge_plugins.cfg.CFGModel property*), 743
`project` (*angr.knowledge_plugins.Function property*), 717
`project` (*angr.knowledge_plugins.functions.Function property*), 778
`project` (*angr.knowledge_plugins.functions.function.Function property*), 802
`project` (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi attribute*), 848
`project` (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 824
`Project` (*class in angr*), 168
`Project` (*class in angr.project*), 223
`prop` (*angr.analyses.decompiler.clinic.BlockCache attribute*), 1056

prop (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute*), 1092
prop_key (*angr.analyses.propagator.propagator.PropagatorAnalysis property*), 1178
prop_key (*angr.analyses.propagator.PropagatorAnalysis property*), 1175
prop_key (*angr.analyses.PropagatorAnalysis property*), 912
PROPAGATED (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes attribute*), 739
PropagationManager (*class in angr.knowledge_plugins*), 733
PropagationManager (*class in angr.knowledge_plugins.propagations*), 773
PropagationModel (*class in angr.knowledge_plugins.propagations*), 774
propagations (*angr.knowledge_base.KnowledgeBase attribute*), 711
propagations (*angr.KnowledgeBase attribute*), 164
Propagator (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
PropagatorAnalysis (*class in angr.analyses*), 911
PropagatorAnalysis (*class in angr.analyses.propagator*), 1174
PropagatorAnalysis (*class in angr.analyses.propagator.propagator*), 1177
prototype (*angr.ailment.expression.Call attribute*), 1393
prototype (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall property*), 1137
prototype (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 1224
prototype (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
prototype (*angr.knowledge_plugins.Function property*), 717
prototype (*angr.knowledge_plugins.functions.Function property*), 778
prototype (*angr.knowledge_plugins.functions.function.Function property*), 802
prototype_libname (*angr.knowledge_plugins.Function property*), 717
prototype_libname (*angr.knowledge_plugins.functions.Function property*), 778
prototype_libname (*angr.knowledge_plugins.functions.function.Function property*), 802
prototype_libnames (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis attribute*), 954
prototype_returnty (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall property*), 1137
prototype_source (*angr.knowledge_plugins.Function property*), 717
prototype_source (*angr.knowledge_plugins.functions.Function property*), 778
prototype_source (*angr.knowledge_plugins.functions.function.Function property*), 802
PrototypeSource (*class in angr.knowledge_plugins.functions.function*), 799
ProximityGraphAnalysis (*class in angr.analyses*), 912
ProximityGraphAnalysis (*class in angr.analyses.proximity_graph*), 1311
ProximityNodeKind (*class in angr.analyses.proximity_graph*), 1309
prune() (*angr.sim_manager.SimulationManager method*), 548
prune() (*angr.SimulationManager method*), 218
pruned (*angr.sim_manager.SimulationManager attribute*), 545
pruned (*angr.SimulationManager attribute*), 215
pseudocode (*angr.knowledge_plugins.Function property*), 720
pseudocode (*angr.knowledge_plugins.functions.Function property*), 780
pseudocode (*angr.knowledge_plugins.functions.function.Function property*), 805
PTChunk (*class in angr*), 166
PTChunk (*class in angr.state_plugins*), 257
PTChunk (*class in angr.state_plugins.heap*), 404
PTChunk (*class in angr.state_plugins.heap.heap_ptmalloc*), 416
PTChunkIterator (*class in angr.state_plugins*), 259
PTChunkIterator (*class in angr.state_plugins.heap*), 406
PTChunkIteratorCallData (*class in angr.state_plugins.heap.heap_ptmalloc*), 417
push() (*angr.state_plugins.CallStack method*), 250
push() (*angr.state_plugins.callstack.CallStack method*), 373
push_priv() (*angr.state_plugins.scratch.SimStateScratch method*), 388
push_priv() (*angr.state_plugins.SimStateScratch method*), 318
push_stack_frame() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 464
put() (*angr.ailment.converter_vex.VEXStmtConverter static method*), 1375
put() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 1245
PutHook (*angr.analyses.indirect_jump_resolvers.jumptable*), 1001

Q

qop_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1173
qop_handler() (*angr.engines.light.SimEngineLightVEX static method*), 1169

qualifies_for_implicit_cast() (*in module angr.analyses.decompiler.structured_codegen.c*), 1130
qualifies_for_simple_cast() (*in module angr.analyses.decompiler.structured_codegen.c*), 1130
quasi_topological_sort_nodes() (*angr.utils.graph.GraphUtils static method*), 1350
query() (*angr.knowledge_plugins.FunctionManager method*), 727
query() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 797
query() (*angr.knowledge_plugins.functions.FunctionManager method*), 788

R

ram_usage (*angr.analyses.Analysis property*), 881
ram_usage (*angr.analyses.analysis.Analysis property*), 930
ram_usage (*angr.Analysis property*), 156
ran_cca (*angr.knowledge_plugins.Function attribute*), 717
ran_cca (*angr.knowledge_plugins.functions.Function attribute*), 777
ran_cca (*angr.knowledge_plugins.functions.function.Function attribute*), 802
ran_cca (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 811
RANDOM (*angr.simos.windows.SecurityCookieInit attribute*), 1338
randomize_procedures() (*angr.analyses.Reassembler method*), 918
randomize_procedures() (*angr.analyses.reassembler.Reassembler method*), 1304
raw_to_bv() (*angr.state_plugins.sim_action_object.SimActionObject method*), 641
raw_to_bv() (*angr.state_plugins.SimActionObject method*), 266
rd (*angr.analyses.decompiler.clinic.BlockCache attribute*), 1056
rd (*angr.analyses.decompiler.optimization_passes.optimization_pass.BlockCache attribute*), 1092
rda_observe_callback() (*angr.knowledge_plugins.key_definitions.key_definition_manager.RDAObserverControl method*), 845
RDAObserverControl (*class in angr.knowledge_plugins.key_definitions.key_definition_manager*), 845
RdTmp() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
reachable() (*angr.state_plugins.history.SimStateHistory method*), 377
reachable() (*angr.state_plugins.SimStateHistory method*), 302
reached_fixedpoint() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 944
reaching_definitions (*angr.analyses.decompiler.clinic.Clinic attribute*), 1060
ReachingDefinitions (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
ReachingDefinitionsAnalysis (*class in angr.analyses*), 912
ReachingDefinitionsAnalysis (*class in angr.analyses.reaching_definitions*), 1197
ReachingDefinitionsAnalysis (*class in angr.analyses.reaching_definitions.reaching_definitions*), 1213
ReachingDefinitionsModel (*class in angr.analyses.reaching_definitions*), 1201
ReachingDefinitionsModel (*class in angr.knowledge_plugins.key_definitions*), 832
ReachingDefinitionsModel (*class in angr.knowledge_plugins.key_definitions.rd_model*), 855
ReachingDefinitionsState (*class in angr.analyses.reaching_definitions*), 1203
ReachingDefinitionsState (*class in angr.analyses.reaching_definitions.rd_state*), 1229
READ (*angr.knowledge_plugins.variables.variable_access.VariableAccessSo attribute*), 812
Read (*angr.knowledge_plugins.xrefs.xref_types.XRefType attribute*), 864
Read (*angr.knowledge_plugins.xrefs.XRefType attribute*), 863
READ (*angr.state_plugins.sim_action.SimActionData attribute*), 639
READ (*angr.state_plugins.SimActionData attribute*), 263
read() (*angr.SimFile method*), 179
read() (*angr.SimFileBase method*), 181
read() (*angr.SimFileStream method*), 188
read() (*angr.SimPackets method*), 198
read() (*angr.SimPacketsStream method*), 200
read() (*angr.storage.file.SimFile method*), 434
read() (*angr.storage.file.SimFileBase method*), 433
read() (*angr.storage.file.SimFileDescriptorBase method*), 442
read() (*angr.storage.file.SimFileStream method*), 436
read() (*angr.storage.file.SimPackets method*), 439
read() (*angr.storage.file.SimPacketsSlots method*), 449
read() (*angr.storage.file.SimPacketsStream method*), 441

read() (*angr.storage.SimFile method*), 424
read_data() (*angr.SimFileDescriptor method*), 182
read_data() (*angr.SimFileDescriptorDuplex method*), 185
read_data() (*angr.storage.file.SimFileDescriptor method*), 444
read_data() (*angr.storage.file.SimFileDescriptorBase method*), 442
read_data() (*angr.storage.file.SimFileDescriptorDuplex method*), 446
read_from() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 814
read_msr() (*angr.state_plugins.Unicorn method*), 329
read_msr() (*angr.state_plugins.unicorn_engine.Unicorn method*), 399
read_pos (*angr.SimFileDescriptor property*), 183
read_pos (*angr.SimFileDescriptorDuplex property*), 186
read_pos (*angr.storage.file.SimFileDescriptor property*), 445
read_pos (*angr.storage.file.SimFileDescriptorBase property*), 443
read_pos (*angr.storage.file.SimFileDescriptorDuplex property*), 447
read_storage (*angr.SimFileDescriptor property*), 183
read_storage (*angr.SimFileDescriptorDuplex property*), 186
read_storage (*angr.storage.file.SimFileDescriptor property*), 445
read_storage (*angr.storage.file.SimFileDescriptorBase property*), 443
read_storage (*angr.storage.file.SimFileDescriptorDuplex property*), 447
real_args (*angr.calling_conventions.UsercallArgSession attribute*), 663
real_length() (*angr.utils.dynamic_dictlist.DynamicDictList method*), 1346
realloc() (*angr.SimHeapPTMalloc method*), 192
realloc() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 416
realloc() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 419
realloc() (*angr.state_plugins.heap.SimHeapLibc method*), 409
realloc() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 410
realloc() (*angr.state_plugins.SimHeapLibc method*), 273
realloc() (*angr.state_plugins.SimHeapPTMalloc method*), 274
reapply_options() (*angr.analyses.decompiler.CStructuredCodeGenerator method*), 1042
reapply_options() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 1129
reapply_options() (*angr.analyses.decompiler.structured_codegen.CodeGenerator method*), 1120
reapply_options() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 1146
reapply_options() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeGenerator method*), 1122
reapply_options() (*angr.analyses.decompiler.structured_codegen.RustStructuredCodeGenerator method*), 1127
Reassembler (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
Reassembler (*class in angr.analyses*), 916
Reassembler (*class in angr.analyses.reassembler*), 1302
ReassemblerFailureNotice, 1297
rebuild_callgraph() (*angr.knowledge_plugins.FunctionManager method*), 728
rebuild_callgraph() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 798
rebuild_callgraph() (*angr.knowledge_plugins.functions.FunctionManager method*), 788
recall() (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode method*), 1262
recent_actions (*angr.state_plugins.history.SimStateHistory property*), 377
recent_actions (*angr.state_plugins.SimStateHistory property*), 303
recent_constraints (*angr.state_plugins.history.SimStateHistory property*), 377
recent_constraints (*angr.state_plugins.SimStateHistory property*), 303
reconstrain() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 392
reconstrain() (*angr.state_plugins.SimStatePreconstrainer method*), 318
record_memory_data_addr() (*angr.analyses.cfg.cfg_base.CFGBase method*), 982
record_memory_data_addr() (*angr.analyses.cfg.CFGBase method*), 961
record_state() (*angr.errors.SimError method*), 1358
record_state() (*angr.SimError method*), 178
record_variable() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 814
recover_edge_condition() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 1062
recover_edge_conditions() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 1062
recover_reaching_conditions() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 1062
RECOVER_VARIABLES (*angr.analyses.decompiler.clinic.ClinicStage*

attribute), 1058
recurse_analysis() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 1228
recurse_analysis() (*angr.analyses.reaching_definitions.FunctionHandler method*), 1188
recursive_copy() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 1071
RecursiveRefNode (*class in angr.analyses.typehoon.simple_solver*), 1260
RecursiveStructurer (*class in angr.analyses.decompiler.structuring*), 1034
RecursiveStructurer (*class in angr.analyses.decompiler.structuring.recursive_structurer*), 1032
redefine_locals (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 1224
redefine_locals (*angr.analyses.reaching_definitions.FunctionCallback attribute*), 1183
RedundantLabelRemover (*class in angr.analyses.decompiler.redundant_label_remover*), 1119
RedundantStackVariable (*class in angr.analyses.binary_optimizer*), 1306
RefcountMixin (*class in angr.storage.memory_mixins*), 478
RefcountMixin (*class in angr.storage.memory_mixins.paged_memory.page_mixins*), 516
RefcountMixin (*class in angr.storage.memory_mixins.paged_memory.pages.cow_mixin*), 518
REFERENCE (*angr.knowledge_plugins.variables.variable_access.VariableAccessSort attribute*), 812
reference_at() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 814
reference_size (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 765
reference_size (*angr.knowledge_plugins.cfg.MemoryData attribute*), 754
reference_values (*angr.ailment.tagged_object.TagDict attribute*), 1407
reference_values (*angr.analyses.decompiler.structured_codegen.c.CConstant attribute*), 1142
reference_variable (*angr.ailment.tagged_object.TagDict attribute*), 1408
reference_variable_offset (*angr.ailment.tagged_object.TagDict attribute*), 1407
refine() (*angr.calling_conventions.SimFunctionArgument method*), 659
refine() (*angr.calling_conventions.SimLyingRegArg method*), 667
refine() (*angr.calling_conventions.SimRegArg method*), 660
refine() (*angr.calling_conventions.SimStackArg method*), 661
refine_locs_with_struct_type() (*in module angr.calling_conventions*), 658
reflow_variable_types() (*angr.analyses.Decompiler method*), 901
reflow_variable_types() (*angr.analyses.decompiler.Decompiler method*), 1047
reflow_variable_types() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1066
refresh() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 1132
refs (*angr.ailment.expression.TypecastTypeReference attribute*), 695
reg (*angr.analyses.decompiler.structured_codegen.c.CRegister attribute*), 1142
reg (*angr.analyses.stack_pointer_tracker.OffsetVal property*), 1243
reg (*angr.engines.light.data.RegisterOffset attribute*), 1167
reg (*angr.engines.light.RegisterOffset attribute*), 1167
reg (*angr.sim_variable.SimRegisterVariable attribute*), 683
REG (*angr.state_plugins.sim_action.SimAction attribute*), 638
REG (*angr.state_plugins.SimAction attribute*), 263
reg() (*angr.analyses.reaching_definitions.Atom static method*), 1179
reg() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 837
reg_concrete() (*angr.sim_state.SimState method*), 242
reg_concrete() (*angr.SimState method*), 210
reg_delta (*angr.analyses.stack_pointer_tracker.StackPointerTracker attribute*), 1246
reg_deps (*angr.state_plugins.sim_action.SimActionData property*), 638
reg_deps (*angr.state_plugins.sim_action.SimActionData property*), 640
reg_deps (*angr.state_plugins.sim_action_object.SimActionObject attribute*), 640
reg_deps (*angr.state_plugins.SimAction property*), 263
reg_deps (*angr.state_plugins.SimActionData property*), 264
reg_deps (*angr.state_plugins.SimActionObject attribute*), 265
reg_name (*angr.ailment.tagged_object.TagDict attribute*), 1408
reg_name (*angr.knowledge_plugins.key_definitions.definition.DefinitionMa attribute*), 842
reg_offset (*angr.ailment.expression.Register attribute*), 1378
reg_offset (*angr.ailment.expression.VirtualVariable property*), 1379

reg_offset (*angr.ailment.Register attribute*), 1371
reg_offset (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffsetAnnotation attribute*), 1000
reg_offset (*angr.analyses.reaching_definitions.Register attribute*), 1211
reg_offset (*angr.knowledge_plugins.key_definitions.atoms.Register attribute*), 839
reg_offset (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 840
reg_offsets (*angr.ailment.expression.VirtualVariable property*), 1379
reg_offsets (*angr.sim_variable.SimComboRegisterVariable attribute*), 684
REG_READ (*angr.state_plugins.inspect.EventType attribute*), 331
reg_read_condition (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_read_condition (*angr.state_plugins.InspectAttrs attribute*), 254
reg_read_endness (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_read_endness (*angr.state_plugins.InspectAttrs attribute*), 254
reg_read_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_read_expr (*angr.state_plugins.InspectAttrs attribute*), 254
reg_read_length (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_read_length (*angr.state_plugins.InspectAttrs attribute*), 254
reg_read_offset (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_read_offset (*angr.state_plugins.InspectAttrs attribute*), 253
reg_read_stmts_to_ignore (*angr.analyses.variable_recovery.engine_vex.SimEngineVRVEX attribute*), 1256
reg_size (*angr.analyses.data_dep.dep_nodes.RegDepNode property*), 1316
reg_size (*angr.analyses.data_dep.RegDepNode property*), 1318
reg_values (*angr.analyses.stack_pointer_tracker.StackPointerTracker attribute*), 1245
reg_vvars (*angr.ailment.expression.VirtualVariable attribute*), 1379
REG_WRITE (*angr.state_plugins.inspect.EventType attribute*), 331
reg_write_condition (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_write_condition (*angr.state_plugins.InspectAttrs attribute*), 254
reg_write_endness (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_write_endness (*angr.state_plugins.InspectAttrs attribute*), 254
reg_write_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_write_expr (*angr.state_plugins.InspectAttrs attribute*), 254
reg_write_length (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_write_length (*angr.state_plugins.InspectAttrs attribute*), 254
reg_write_offset (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
reg_write_offset (*angr.state_plugins.InspectAttrs attribute*), 254
RegDepNode (*class in angr.analyses.data_dep*), 1318
RegDepNode (*class in angr.analyses.data_dep.dep_nodes*), 1315
regenerate_text() (*angr.analyses.decompiler.CStructuredCodeGenerat method*), 1042
regenerate_text() (*angr.analyses.decompiler.ImportSourceCode method*), 1050
regenerate_text() (*angr.analyses.decompiler.structured_codegen.base. method*), 1130
regenerate_text() (*angr.analyses.decompiler.structured_codegen.BaseS method*), 1120
regenerate_text() (*angr.analyses.decompiler.structured_codegen.c.CSt method*), 1146
regenerate_text() (*angr.analyses.decompiler.structured_codegen.CStru method*), 1122
regenerate_text() (*angr.analyses.decompiler.structured_codegen.dwarf method*), 1149
regenerate_text() (*angr.analyses.decompiler.structured_codegen.Impor method*), 1124
regenerate_text() (*angr.analyses.decompiler.structured_codegen.RustS method*), 1127
region (*angr.sim_variable.SimVariable attribute*), 680
region (*angr.storage.memory_mixins.regioned_memory.region_data.Addre attribute*), 533
region (*angr.storage.memory_mixins.regioned_memory.region_meta_mixi property*), 536
region_base_addr (*angr.storage.memory_mixins.regioned_memory.regio attribute*), 533
region_id (*angr.storage.memory_mixins.regioned_memory.region_data.R attribute*), 534
region_ids (*angr.storage.memory_mixins.regioned_memory.region_data. property*), 534
RegionCategoryMixin (*class in angr.storage.memory_mixins*), 478
RegionCategoryMixin (*class in angr.storage.memory_mixins.regioned_memory*), 526
RegionCategoryMixin (*class in angr.storage.memory_mixins.regioned_memory.region_category_

535
RegionDescriptor (*class in angr.storage.memory_mixins.regioned_memory.regioned_memory*), 534
RegionedAddressConcretizationMixin (*class in angr.storage.memory_mixins*), 479
RegionedAddressConcretizationMixin (*class in angr.storage.memory_mixins.regioned_memory*), 526
RegionedAddressConcretizationMixin (*class in angr.storage.memory_mixins.regioned_memory.regioned_memory_mixins*), 538
RegionedMemory (*class in angr.storage.memory_mixins*), 480
RegionedMemoryMixin (*class in angr.storage.memory_mixins*), 480
RegionedMemoryMixin (*class in angr.storage.memory_mixins.regioned_memory*), 527
RegionedMemoryMixin (*class in angr.storage.memory_mixins.regioned_memory.regioned_memory_mixins*), 530
RegionIdentifier (*class in angr.analyses.decompiler*), 1050
RegionIdentifier (*class in angr.analyses.decompiler.region_identifier*), 1107
RegionMap (*class in angr.storage.memory_mixins.regioned_memory.regioned_memory*), 534
RegionObject (*class in angr.keyed_region*), 868
regions (*angr.analyses.cfg.cfb.CFBlanket property*), 973
regions (*angr.analyses.cfg.CFBlanket property*), 958
regions (*angr.analyses.SelfModifyingCodeAnalysis attribute*), 922
RegionSimplifier (*class in angr.analyses.decompiler*), 1050
RegionSimplifier (*class in angr.analyses.decompiler.region_simplifiers*), 1108
RegionSimplifier (*class in angr.analyses.decompiler.region_simplifiers.region_simplifier*), 1114
RegionWalker (*class in angr.analyses.decompiler.region_walker*), 1119
REGISTER (*angr.ailment.expression.VirtualVariableCategory attribute*), 1379
Register (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 1313
Register (*angr.analyses.data_dep.DepNodeTypes attribute*), 1318
register (*angr.analyses.disassembly.RegisterOperand property*), 1295
REGISTER (*angr.analyses.reaching_definitions.AtomKind attribute*), 1180
REGISTER (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 836
REGISTER (*angr.knowledge_plugins.variables.variable_manager.VariableTy attribute*), 813
REGISTER (*angr.knowledge_plugins.variables.VariableType attribute*), 812
Register (*class in angr.ailment*), 1371
Register (*class in angr.ailment.expression*), 1378
Register (*class in angr.analyses.disassembly*), 1295
Register (*class in angr.analyses.reaching_definitions*), 1211
Register (*class in angr.analyses.stack_pointer_tracker*), 1243
Register (*class in angr.knowledge_plugins.key_definitions.atoms*), 839
register() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
register() (*angr.analyses.reaching_definitions.Atom static method*), 1180
register() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 837
register() (*angr.utils.mp.Initializer method*), 1355
register_analysis() (*in module angr*), 222
register_analysis() (*in module angr.analyses*), 928
register_analysis() (*in module angr.analyses.analysis*), 930
register_bool_option() (*angr.sim_state_options.SimStateOptions class method*), 246
register_callbacks() (*angr.analyses.variable_recovery.variable_recovery.VariableReco method*), 1255
register_data_reference() (*angr.analyses.Reassembler method*), 917
register_data_reference() (*angr.analyses.reassembler.Reassembler method*), 1303
register_default() (*angr.knowledge_plugins.KnowledgeBasePlugin static method*), 731
register_default() (*angr.knowledge_plugins.plugin.KnowledgeBasePlu static method*), 738
register_default() (*angr.misc.plugins.PluginHub class method*), 235
register_default() (*angr.SimStatePlugin class method*), 213
register_default() (*angr.state_plugins.plugin.SimStatePlugin class method*), 331
register_default() (*angr.state_plugins.SimStatePlugin class method*), 316
register_default_cc() (*in module angr.calling_conventions*), 679
register_function_analysis()

(*angr.analyses.vfg.CallAnalysis method*), 1287
register_instruction_reference() (*angr.analyses.Reassembler method*), 917
register_instruction_reference() (*angr.analyses.reassembler.Reassembler method*), 1303
register_kernel_types() (*in module angr.utils.library*), 1352
register_optimization_pass() (*in module angr.analyses.decompiler.optimization_passes*), 1091
register_option() (*angr.sim_state_options.SimStateOptions class method*), 246
register_pcode_arch_default_cc() (*in module angr.engines.pcode*), 604
register_pcode_arch_default_cc() (*in module angr.engines.pcode.cc*), 638
register_plugin() (*angr.knowledge_base.KnowledgeBase method*), 712
register_plugin() (*angr.KnowledgeBase method*), 165
register_plugin() (*angr.misc.plugins.PluginHub method*), 236
register_plugin() (*angr.misc.plugins.PluginVendor method*), 238
register_plugin() (*angr.sim_state.SimState method*), 241
register_plugin() (*angr.SimState method*), 209
register_preset() (*angr.misc.plugins.PluginHub class method*), 235
register_region (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStandBase attribute*), 1249
register_region (*angr.knowledge_plugins.variables.variable_manager.LiveVariables attribute*), 813
register_syscall_cc() (*in module angr.calling_conventions*), 680
register_types() (*in module angr.sim_type*), 708
register_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
register_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 1233
register_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 1206
register_uses (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 848
register_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 825
register_values (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 393
register_values_count (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 394
register_variable() (*angr.state_plugins.SimSolver method*), 288
register_variable() (*angr.state_plugins.solver.SimSolver method*), 362
RegisterInitializerHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 1001
RegisterOffset (*class in angr.engines.light*), 1167
RegisterOffset (*class in angr.engines.light.data*), 1166
RegisterOperand (*class in angr.analyses.disassembly*), 1295
RegisterReallocation (*class in angr.analyses.binary_optimizer*), 1306
registers (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
registers (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 1233
registers (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 1206
registers (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 848
registers (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 824
registers (*angr.sim_state.SimState attribute*), 239
registers (*angr.SimState attribute*), 208
RegisterSaveAreaSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1084
RegisterSaveAreaSimplifier (*class in angr.analyses.decompiler.optimization_passes.register_save_area_simplifier_advanced*), 1103
RegisterSaveAreaSimplifierAdvanced (*class in angr.analyses.decompiler.optimization_passes*), 1085
RegisterValue (*class in angr.state_plugins.unicorn_engine*), 393
RegOffsetAnnotation (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 1000
regs (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerState attribute*), 1254
regs (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState attribute*), 1244
regs (*angr.sim_state.SimState attribute*), 239
regs (*angr.SimState attribute*), 208
regs (*angr.slicer.SimLightState attribute*), 1321
regs_for_initialize (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffsetAnnotation attribute*), 1000
rehook_symbol() (*angr.Project method*), 172
rehook_symbol() (*angr.project.Project method*), 226
Reinterpret (*class in angr.ailment.expression*), 1383
ReinterpretAs (*class in angr.analyses.typehoon.typevars*), 1272
related_function_addr

(*angr.storage.memory_mixins.MemoryRegionMetaMixin* *property*), 472
related_function_addr (*angr.storage.memory_mixins.regioned_memory.MemoryRegionMetaMixin* *property*), 525
related_function_addr (*angr.storage.memory_mixins.regioned_memory.region_meta.MemoryRegionMetaMixin* *property*), 537
related_function_address (*angr.storage.memory_mixins.regioned_memory.region_meta.RegionDescriptor* *attribute*), 534
relativize() (*angr.storage.memory_mixins.regioned_memory.RegionMap* *method*), 535
release() (*angr.SimHeapBrk method*), 190
release() (*angr.state_plugins.heap.heap_brk.SimHeapBrk* *method*), 413
release() (*angr.state_plugins.heap.SimHeapBrk* *method*), 407
release() (*angr.state_plugins.SimHeapBrk method*), 271
release_plugin() (*angr.knowledge_base.KnowledgeBase* *method*), 712
release_plugin() (*angr.KnowledgeBase method*), 165
release_plugin() (*angr.misc.plugins.PluginHub* *method*), 236
release_plugin() (*angr.misc.plugins.PluginVendor* *method*), 238
release_shared() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixins.PagedMemoryMixin* *method*), 519
release_shared() (*angr.storage.memory_mixins.paged_memory.page_mixins.RefcountMixin* *method*), 516
release_shared() (*angr.storage.memory_mixins.RefcountMixin* *method*), 478
reload_format() (*angr.analyses.disassembly.Instruction* *method*), 1294
reload_solver() (*angr.state_plugins.SimSolver* *method*), 288
reload_solver() (*angr.state_plugins.solver.SimSolver* *method*), 361
reload_variable_types() (*angr.analyses.decompiler.CStructuredCodeGenerator* *method*), 1042
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator* *method*), 1130
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.BaseStructuredCodeGenerator* *method*), 1120
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator* *method*), 1147
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.CSyntaxLikeCodeGenerator* *method*), 1122
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.RustStructuredCodeGenerator* *method*), 1127
relocatable (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffsetAnnotation* *property*), 1000
relocatable (*angr.analyses.data_dep.data_dependency_analysis.NodalAnnotation* *property*), 1312
relocatable (*angr.analyses.variable_recovery.annotations.StackLocationAnnotation* *property*), 1246
relocatable (*angr.analyses.variable_recovery.annotations.VariableSourceAnnotation* *property*), 1246
relocatable (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBaseAnnotation* *property*), 1247
relocatable (*angr.knowledge_plugins.key_definitions.live_definitions.DefinitionAnnotation* *property*), 846
relocateable (*angr.analyses.decompiler.condition_processor.AILExprIdAnnotation* *property*), 1062
relocateable (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationAnnotation* *property*), 495
Relocation (*class in angr.analyses.reassembler*), 1302
relocations (*angr.analyses.Reassembler property*), 916
relocations (*angr.analyses.reassembler.Reassembler* *property*), 1302
remove() (*angr.sim_state_options.SimStateOptions* *method*), 245
remove_breakpoint() (*angr.Emulator method*), 162
remove_breakpoint() (*angr.state_plugins.SimInspector* *method*), 341
remove_breakpoint() (*angr.state_plugins.SimInspectorMixin* *method*), 277
remove_cgc_attachments() (*angr.analyses.Reassembler method*), 918
remove_cgc_attachments() (*angr.analyses.reassembler.Reassembler* *method*), 1304
remove_claripy_bool_asts() (*angr.analyses.decompiler.condition_processor.ConditionProcessor* *method*), 1062
remove_cycles() (*angr.analyses.cfg.cfg_emulated.CFGEmulated* *method*), 979
remove_cycles() (*angr.analyses.cfg.CFGEmulated* *method*), 964
remove_cycles() (*angr.analyses.CFGEmulated* *method*), 887
remove_edge() (*angr.analyses.cfg.cfg_base.CFGBase* *method*), 982
remove_edge() (*angr.analyses.cfg.CFGBase method*), 961
remove_edges_in_ailgraph() (*in module angr.analyses.decompiler.utils*), 1156
remove_empty_nodes() (*angr.analyses.decompiler.Clinic static*

method), 1046
remove_empty_nodes() (*angr.analyses.decompiler.clinic.Clinic static method*), 1061
remove_fakerets() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 979
remove_fakerets() (*angr.analyses.cfg.CFGEmulated method*), 965
remove_fakerets() (*angr.analyses.CFGEmulated method*), 888
remove_instruction() (*angr.analyses.Reassembler method*), 918
remove_instruction() (*angr.analyses.reassembler.Reassembler method*), 1304
remove_labels() (*in module angr.analyses.decompiler.utils*), 1153
remove_last_statement() (*in module angr.analyses.decompiler.utils*), 1150
remove_last_statements() (*in module angr.analyses.decompiler.utils*), 1150
remove_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 758
remove_node() (*angr.knowledge_plugins.cfg.CFGModel method*), 744
remove_node_and_graph_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 763
remove_node_and_graph_node() (*angr.knowledge_plugins.cfg.CFGModel method*), 749
remove_patch() (*angr.knowledge_plugins.patches.PatchManager method*), 737
remove_patch() (*angr.knowledge_plugins.PatchManager method*), 732
remove_preconstraints() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 391
remove_preconstraints() (*angr.state_plugins.SimStatePreconstrainer method*), 318
remove_redundant_jumps() (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase static method*), 1036
remove_technique() (*angr.sim_manager.SimulationManager method*), 546
remove_technique() (*angr.SimulationManager method*), 216
remove_types() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 818
remove_unnecessary_stuff() (*angr.analyses.Reassembler method*), 918
remove_unnecessary_stuff() (*angr.analyses.reassembler.Reassembler method*), 1304
remove_unnecessary_stuff_glibc() (*angr.analyses.Reassembler method*), 918
remove_unnecessary_stuff_glibc() (*angr.analyses.reassembler.Reassembler method*), 1304
remove_use() (*angr.knowledge_plugins.key_definitions.Uses method*), 835
remove_use() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 859
remove_uses() (*angr.knowledge_plugins.key_definitions.Uses method*), 835
remove_uses() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 860
remove_variable() (*angr.analyses.decompiler.optimization_passes.engin method*), 1101
remove_variable_by_atom() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 814
RemoveNodeNotice, 1054
rename() (*angr.knowledge_plugins.types.TypesStore method*), 773
rename() (*angr.knowledge_plugins.TypesStore method*), 735
renamed (*angr.sim_variable.SimVariable attribute*), 681
render() (*angr.analyses.Disassembly method*), 904
render() (*angr.analyses.disassembly.Disassembly method*), 1297
render() (*angr.analyses.disassembly.DisassemblyPiece method*), 1293
render() (*angr.sim_type.SimTypeBitfield method*), 705
render_notes() (*angr.analyses.decompiler.CStructuredCodeGenerator method*), 1042
render_notes() (*angr.analyses.decompiler.structured_codegen.c.CStruct method*), 1146
render_notes() (*angr.analyses.decompiler.structured_codegen.CStructur method*), 1122
render_notes() (*angr.analyses.decompiler.structured_codegen.RustStruc method*), 1127
render_text() (*angr.analyses.decompiler.CStructuredCodeGenerator method*), 1042
render_text() (*angr.analyses.decompiler.structured_codegen.c.CStructu method*), 1146
render_text() (*angr.analyses.decompiler.structured_codegen.CStructure method*), 1122
render_text() (*angr.analyses.decompiler.structured_codegen.RustStructu method*), 1127
RENDER_TYPE (*angr.analyses.decompiler.structured_codegen.RustStructure attribute*), 1127
RenderResult (*in module angr.analyses.decompiler.structured_codegen.c*), 1130
reopen_lmdb() (*angr.knowledge_plugins.RuntimeDb method*), 734

RepHook (*class in angr.exploration_techniques.tracer*), 575
replace() (*angr.ailment.Assignment method*), 1366
replace() (*angr.ailment.BinaryOp method*), 1367
replace() (*angr.ailment.Expression method*), 1369
replace() (*angr.ailment.expression.Array method*), 1395
replace() (*angr.ailment.expression.BasePointerOffset method*), 1390
replace() (*angr.ailment.expression.BinaryOp method*), 1385
replace() (*angr.ailment.expression.Call method*), 1394
replace() (*angr.ailment.expression.Convert method*), 1383
replace() (*angr.ailment.expression.DirtyExpression method*), 1388
replace() (*angr.ailment.expression.Expression method*), 1376
replace() (*angr.ailment.expression.Extract method*), 1392
replace() (*angr.ailment.expression.Insert method*), 1393
replace() (*angr.ailment.expression.ITE method*), 1387
replace() (*angr.ailment.expression.Load method*), 1386
replace() (*angr.ailment.expression.MultiStatementExpression method*), 1389
replace() (*angr.ailment.expression.Phi method*), 1381
replace() (*angr.ailment.expression.Reinterpret method*), 1384
replace() (*angr.ailment.expression.RustEnum method*), 1395
replace() (*angr.ailment.expression.StringLiteral method*), 1394
replace() (*angr.ailment.expression.Struct method*), 1395
replace() (*angr.ailment.expression.UnaryOp method*), 1381
replace() (*angr.ailment.expression.VEXCCallExpression method*), 1389
replace() (*angr.ailment.Statement method*), 1371
replace() (*angr.ailment.statement.Assignment method*), 1398
replace() (*angr.ailment.statement.CAS method*), 1404
replace() (*angr.ailment.statement.ConditionalJump method*), 1401
replace() (*angr.ailment.statement.DirtyStatement method*), 1405
replace() (*angr.ailment.statement.Jump method*), 1400
replace() (*angr.ailment.statement.Label method*), 1406
replace() (*angr.ailment.statement.Return method*), 1403
replace() (*angr.ailment.statement.SideEffectStatement method*), 1402
replace() (*angr.ailment.statement.Statement method*), 1397
replace() (*angr.ailment.statement.Store method*), 1400
replace() (*angr.ailment.statement.WeakAssignment method*), 1398
replace() (*angr.ailment.UnaryOp method*), 1373
replace() (*angr.analyses.SPropagatorAnalysis static method*), 921
replace() (*angr.analyses.typehoon.typeconsts.Array method*), 1278
replace() (*angr.analyses.typehoon.typeconsts.Enum method*), 1280
replace() (*angr.analyses.typehoon.typeconsts.Function method*), 1280
replace() (*angr.analyses.typehoon.typeconsts.Pointer method*), 1278
replace() (*angr.analyses.typehoon.typeconsts.RustEnum method*), 1279
replace() (*angr.analyses.typehoon.typeconsts.TypeConstant method*), 1274
replace() (*angr.analyses.typehoon.typevars.Add method*), 1268
replace() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 1270
replace() (*angr.analyses.typehoon.typevars.Existence method*), 1267
replace() (*angr.analyses.typehoon.typevars.Sub method*), 1268
replace() (*angr.analyses.typehoon.typevars.Subtype method*), 1267
replace() (*angr.keyed_region.KeyedRegion method*), 869
replace() (*angr.procedures.stubs.format_parser.FormatString method*), 646
replace_all() (*angr.storage.memory_mixins.convenient_mappings_mixi method*), 499
replace_all() (*angr.storage.memory_mixins.ConvenientMappingsMixin method*), 459
replace_all() (*angr.storage.memory_mixins.regioned_memory.regioned method*), 532
replace_all() (*angr.storage.memory_mixins.regioned_memory.Regioned method*), 529
replace_all() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 482
replace_all_with_offsets() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.U method*), 524
replace_all_with_offsets() (*angr.storage.memory_mixins.paged_memory.pages.UltraPage method*), 518
replace_all_with_offsets() (*angr.storage.memory_mixins.UltraPage method*), 489
replace_last_statement() (*in module angr.analyses.decompiler.utils*), 1150

replace_node_in_node() (*angr.analyses.decompiler.structuring.structurer_base.StructuringBase static method*), 1037
replace_nodes() (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer method*), 1038
replace_nodes() (*angr.analyses.decompiler.structuring.Phoenix.PhoenixStructurer method*), 1034
replace_nodes() (*angr.analyses.decompiler.structuring.structurer_base.StructuringBase method*), 1037
replace_region() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 1071
replace_region_with_region() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 1071
replacement_core_vvars (*angr.analyses.decompiler.expression_narrower.ExpressionNarrower attribute*), 1070
replacements (*angr.analyses.propagator.propagator.PropagatorAnalysis property*), 1178
replacements (*angr.analyses.propagator.PropagatorAnalysis property*), 1175
replacements (*angr.analyses.PropagatorAnalysis property*), 912
replacements (*angr.analyses.SPropagatorAnalysis property*), 920
replacements (*angr.knowledge_plugins.propagations.PropagationModel attribute*), 775
report() (*angr.exploration_techniques.Suggestions static method*), 565
report() (*angr.exploration_techniques.suggestions.Suggestions static method*), 587
repr_addr() (*in module angr.codenode*), 1322
request_knowledge() (*angr.knowledge_base.KnowledgeBase method*), 712
request_knowledge() (*angr.KnowledgeBase method*), 165
request_plugin() (*angr.misc.plugins.PluginPreset method*), 237
REQUIRE_CFG_STATES (*angr.exploration_techniques.CallFunctionGoal attribute*), 553
REQUIRE_CFG_STATES (*angr.exploration_techniques.director.BaseGoal attribute*), 579
REQUIRE_CFG_STATES (*angr.exploration_techniques.director.CallFunctionGoal attribute*), 580
REQUIRE_DATA_C (*angr.engines.pcode.lifter.Lifter attribute*), 610
REQUIRE_DATA_PY (*angr.engines.pcode.lifter.Lifter attribute*), 610
reraise() (*angr.sim_manager.ErrorRecord method*), 551
reset() (*angr.ailment.Manager method*), 1370
reset() (*angr.ailment.manager.Manager method*), 1407
reset() (*angr.analyses.forward_analysis.SingleNodeGraphVisitor method*), 934
reset() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 943
reset() (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor method*), 945
reset() (*angr.analyses.forward_analysis.visitors.SingleNodeGraphVisitor method*), 940
reset() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 396
reset_block_counters() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 1130
reset_idx_counters() (*angr.analyses.decompiler.structured_codegen.BaseStructuredCodeGenerator method*), 1120
reset_initial_regs() (*angr.Block static method*), 160
reset_initial_regs() (*angr.block.Block static method*), 233
reset_memory_data_addrs() (*angr.analyses.cfg.cfg_base.CFGBase method*), 983
reset_memory_data_addrs() (*angr.analyses.cfg.CFGBase method*), 961
reset_prototype() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 1225
reset_prototype() (*angr.analyses.reaching_definitions.FunctionCallDataHelper method*), 1184
reset_uses() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1190
reset_uses() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 848
reset_uses() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 825
resilient (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerState attribute*), 1244
resilient (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState attribute*), 1244
resolvable (*angr.state_plugins.debug_variables.SimDebugVariable property*), 422
resolvable (*angr.state_plugins.SimDebugVariable property*), 267
resolvable (*angr.state_plugins.SimMemView property*), 485
resolvable (*angr.state_plugins.view.SimMemView property*), 431
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.AArch64MachOGotPlt method*), 1005
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got.AMD64ElfGotPlt method*), 994
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.AMD64ElfGotResolvedPlt method*), 1006
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.AMD64PeIatResolvedPlt method*), 1007

resolve() (*angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast.ArmElfFastResolver method*), 995
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.ArmElfFastResolver method*), 1007
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.const_resolver.ConstantResolver method*), 1004
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.ConstantResolver method*), 1008
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver method*), 1003
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.JumpTableResolver method*), 1009
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.MemoryLoadResolver method*), 1010
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.MipsElfFastResolver method*), 997
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.MipsElfFastResolver method*), 1011
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.MipsElfGotResolver method*), 1012
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.resolver.IndirectJumpResolver method*), 1004
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.SyscallResolver method*), 1013
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt.X86ElfPicPltResolver method*), 997
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat.SimProcedureResolver method*), 996
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.X86ElfPicPltResolver method*), 1013
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.X86PeIatResolver method*), 1014
resolve() (*angr.sim_type.SimTypeBitfield method*), 704
resolve() (*angr.sim_type.SimTypeEnum method*), 703
resolve_abstract_dispatch() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 956
resolve_abstract_dispatch() (*angr.analyses.SootClassHierarchy method*), 922
resolve_concrete_dispatch() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 956
resolve_concrete_dispatch() (*angr.analyses.SootClassHierarchy method*), 922
resolve_invoke() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 956
resolve_invoke() (*angr.analyses.SootClassHierarchy method*), 922
resolve_register() (*angr.state_plugins.light_registers.SimLightRegisters method*), 375
resolve_register() (*angr.state_plugins.SimLightRegisters method*), 281
resolve_special_dispatch() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 956
resolve_special_dispatch() (*angr.analyses.SootClassHierarchy method*), 922
resolved (*angr.state_plugins.debug_variables.SimDebugVariable property*), 422
resolved (*angr.state_plugins.SimDebugVariable property*), 267
resolved (*angr.state_plugins.SimMemView property*), 285
resolved (*angr.state_plugins.view.SimMemView property*), 431
resolved_indirect_jumps (*angr.knowledge_base.KnowledgeBase property*), 712
resolved_indirect_jumps (*angr.KnowledgeBase property*), 164
resolved_targets (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpResolver attribute*), 771
resolved_targets (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 753
resource_event() (*in module angr.state_plugins*), 329
resource_event() (*angr.state_plugins.sim_event.X86ElfPicPltResolver in module angr.state_plugins.sim_event*), 642
restore_graph() (*angr.analyses.decompiler.optimization_passes.lowered_constant_propagation.LoweredConstantPropagation static method*), 1100
restore_graph() (*angr.analyses.decompiler.optimization_passes.LoweredConstantPropagation static method*), 1083
result (*angr.analyses.decompiler.ccall_rewriters.rewriter_base.CCallRewriterBase attribute*), 1056
result (*angr.analyses.SelfModifyingCodeAnalysis attribute*), 922
result_type() (*angr.engines.light.engine.IRTop method*), 1171
RESUME (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.Case2Resolver attribute*), 996
resume() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 978
resume() (*angr.analyses.cfg.CFGEmulated method*), 964
resume() (*angr.analyses.CFGEmulated method*), 886
resume_with_new_graph() (*angr.analyses.forward_analysis.FunctionGraphVisitor method*), 932
resume_with_new_graph() (*angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor method*), 941
resume_with_new_graph() (*angr.analyses.forward_analysis.visitors.FunctionGraphVisitor method*), 938
resymbolize() (*angr.state_plugins.SimSymbolizer method*), 320

resymbolize() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 421
ret() (*angr.sim_procedure.SimProcedure method*), 646
ret() (*angr.SimProcedure method*), 205
ret() (*angr.state_plugins.CallStack method*), 250
ret() (*angr.state_plugins.callstack.CallStack method*), 374
ret_addr (*angr.analyses.cfg.cfg_fast.FunctionCallEdge attribute*), 986
ret_atoms (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 1224
ret_atoms (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
ret_defns (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 1226
ret_defns (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1186
ret_errno() (*angr.state_plugins.libc.SimStateLibc method*), 347
ret_errno() (*angr.state_plugins.SimStateLibc method*), 311
ret_expr (*angr.ailment.statement.SideEffectStatement attribute*), 1402
ret_exprs (*angr.ailment.statement.Return attribute*), 1403
ret_from_addr (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge attribute*), 987
ret_sites (*angr.knowledge_plugins.Function property*), 720
ret_sites (*angr.knowledge_plugins.functions.Function property*), 780
ret_sites (*angr.knowledge_plugins.functions.function.Function property*), 804
ret_target (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
ret_to_addr (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge attribute*), 987
ret_values (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 1224
ret_values (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
ret_values_deps (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 1224
ret_values_deps (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1184
retaddr_on_stack (*angr.knowledge_plugins.Function attribute*), 717
retaddr_on_stack (*angr.knowledge_plugins.functions.Function attribute*), 777
retaddr_on_stack (*angr.knowledge_plugins.functions.function.Function attribute*), 802
retaddr_on_stack (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 810
retaddr_popped (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 1224
retaddr_popped (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1184
RetAddrSaveSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1085
RetAddrSaveSimplifier (*class in angr.analyses.decompiler.optimization_passes.ret_addr_save_sim*), 1103
return_sites (*angr.knowledge_plugins.Function property*), 720
return_sites (*angr.knowledge_plugins.functions.Function property*), 780
return_sites (*angr.knowledge_plugins.functions.function.Function property*), 804
RETURN (*angr.state_plugins.inspect.EventType attribute*), 332
Return (*class in angr.ailment.statement*), 1402
RETURN_ADDR (*angr.calling_conventions.SimCC attribute*), 664
return_addr (*angr.calling_conventions.SimCC property*), 665
RETURN_ADDR (*angr.calling_conventions.SimCCAArch64 attribute*), 674
RETURN_ADDR (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 674
RETURN_ADDR (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 672
RETURN_ADDR (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 672
RETURN_ADDR (*angr.calling_conventions.SimCCARM attribute*), 672
RETURN_ADDR (*angr.calling_conventions.SimCCARMHF attribute*), 673
RETURN_ADDR (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 673
RETURN_ADDR (*angr.calling_conventions.SimCCCdecl attribute*), 668
RETURN_ADDR (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 670
RETURN_ADDR (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 667
RETURN_ADDR (*angr.calling_conventions.SimCCN64 attribute*), 676
RETURN_ADDR (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 676
RETURN_ADDR (*angr.calling_conventions.SimCCO32 attribute*), 675
RETURN_ADDR (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 675
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPC attribute*), 676
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPC64 attribute*), 677

RETURN_ADDR (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 677
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 677
RETURN_ADDR (*angr.calling_conventions.SimCCRISCV64 attribute*), 674
RETURN_ADDR (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 675
RETURN_ADDR (*angr.calling_conventions.SimCCS390X attribute*), 679
RETURN_ADDR (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 679
RETURN_ADDR (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 671
RETURN_ADDR (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 670
RETURN_ADDR (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 671
RETURN_ADDR (*angr.engines.pcode.cc.SimCCM68k attribute*), 636
RETURN_ADDR (*angr.engines.pcode.cc.SimCCPARISC attribute*), 637
RETURN_ADDR (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 638
RETURN_ADDR (*angr.engines.pcode.cc.SimCCRISCV attribute*), 636
RETURN_ADDR (*angr.engines.pcode.cc.SimCCSH4 attribute*), 637
RETURN_ADDR (*angr.engines.pcode.cc.SimCCSPARC attribute*), 637
RETURN_ADDR (*angr.engines.pcode.cc.SimCCXtensa attribute*), 638
RETURN_ADDR (*angr.SimCC attribute*), 174
return_addr (*angr.SimCC property*), 176
return_in_implicit_outparam() (*angr.calling_conventions.SimCC method*), 665
return_in_implicit_outparam() (*angr.calling_conventions.SimCCCdecl method*), 668
return_in_implicit_outparam() (*angr.calling_conventions.SimCCMicrosoftAMD64 method*), 670
return_in_implicit_outparam() (*angr.calling_conventions.SimCCSystemVAMD64 method*), 671
return_in_implicit_outparam() (*angr.SimCC method*), 175
return_target (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 769
return_target (*angr.knowledge_plugins.cfg.CFGENode attribute*), 741
return_to (*angr.analyses.cfg.cfg_fast.FunctionReturn attribute*), 984
return_type (*angr.sim_procedure.SimProcedure property*), 646
return_type (*angr.SimProcedure property*), 206
RETURN_VAL (*angr.calling_conventions.SimCC attribute*), 664
RETURN_VAL (*angr.calling_conventions.SimCCAArch64 attribute*), 674
RETURN_VAL (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 674
RETURN_VAL (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 672
RETURN_VAL (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 672
RETURN_VAL (*angr.calling_conventions.SimCCARM attribute*), 672
RETURN_VAL (*angr.calling_conventions.SimCCARMHF attribute*), 673
RETURN_VAL (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 673
RETURN_VAL (*angr.calling_conventions.SimCCCdecl attribute*), 668
RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 669
RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 669
RETURN_VAL (*angr.calling_conventions.SimCCN64 attribute*), 676
RETURN_VAL (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 676
RETURN_VAL (*angr.calling_conventions.SimCCO32 attribute*), 675
RETURN_VAL (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 675
RETURN_VAL (*angr.calling_conventions.SimCCPowerPC attribute*), 676
RETURN_VAL (*angr.calling_conventions.SimCCPowerPC64 attribute*), 677
RETURN_VAL (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 677
RETURN_VAL (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 677
RETURN_VAL (*angr.calling_conventions.SimCCRISCV64 attribute*), 674
RETURN_VAL (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 675
RETURN_VAL (*angr.calling_conventions.SimCCS390X attribute*), 679
RETURN_VAL (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 679
RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 671
RETURN_VAL (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 670
RETURN_VAL (*angr.calling_conventions.SimCCX86WindowsSyscall*

attribute), 671
RETURN_VAL (*angr.engines.pcode.cc.SimCCM68k attribute*), 636
RETURN_VAL (*angr.engines.pcode.cc.SimCCPARISC attribute*), 637
RETURN_VAL (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 638
RETURN_VAL (*angr.engines.pcode.cc.SimCCRISCV attribute*), 636
RETURN_VAL (*angr.engines.pcode.cc.SimCCSH4 attribute*), 637
RETURN_VAL (*angr.engines.pcode.cc.SimCCSPARC attribute*), 637
RETURN_VAL (*angr.engines.pcode.cc.SimCCXtensa attribute*), 638
RETURN_VAL (*angr.SimCC attribute*), 174
return_val() (*angr.calling_conventions.SimCC method*), 665
return_val() (*angr.calling_conventions.SimCCCdecl method*), 668
return_val() (*angr.calling_conventions.SimCCMicrosoftAMD64 method*), 670
return_val() (*angr.calling_conventions.SimCCSystemVAMD64 method*), 671
return_val() (*angr.calling_conventions.SimCCUsercall method*), 667
return_val() (*angr.SimCC method*), 175
ReturnDeduplicator (*class in angr.analyses.decompiler.optimization_passes*), 1085
ReturnDuplicatorHigh (*class in angr.analyses.decompiler.optimization_passes*), 1086
ReturnDuplicatorLow (*class in angr.analyses.decompiler.optimization_passes*), 1087
returning (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 1136
returning (*angr.knowledge_plugins.Function property*), 717
returning (*angr.knowledge_plugins.functions.Function property*), 778
returning (*angr.knowledge_plugins.functions.function.Function property*), 802
returning_source (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
returnty (*angr.ailment.expression.Macro attribute*), 1396
returnty (*angr.sim_type.SimTypeFunction attribute*), 698
ReturnValueTag (*class in angr.knowledge_plugins.key_definitions.tag*), 858
retval (*angr.analyses.decompiler.structured_codegen.c.CReturn attribute*), 1137
reverse_post_order_sort_nodes() (*angr.utils.graph.GraphUtils static method*), 1350
revisit_node() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 944
revisit_successors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 944
rhs (*angr.analyses.decompiler.structured_codegen.c.CAssignment attribute*), 1135
rhs (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp attribute*), 1141
RichR (*class in angr.analyses.variable_recovery.engine_base*), 1256
RIGHT (*angr.analyses.typehoon.simple_solver.ConstraintGraphTag attribute*), 1261
root (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
root (*angr.analyses.typehoon.simple_solver.Sketch attribute*), 1260
rounding_mode (*angr.ailment.BinaryOp attribute*), 1367
rounding_mode (*angr.ailment.expression.BinaryOp attribute*), 1385
rounding_mode (*angr.ailment.expression.Convert attribute*), 1383
RShift (*angr.engines.light.ArithmeticExpression attribute*), 1167
RShift (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
rtdb (*angr.knowledge_base.KnowledgeBase attribute*), 711
rtdb (*angr.KnowledgeBase attribute*), 164
run() (*angr.analyses.congruency_check.CongruencyCheck method*), 1305
run() (*angr.analyses.CongruencyCheck method*), 899
run() (*angr.analyses.CFG.CFGJob method*), 907
run() (*angr.analyses.identifier.identify.Identifier method*), 1283
run() (*angr.distributed.Server method*), 1362
run() (*angr.distributed.server.Server method*), 1363
run() (*angr.distributed.worker.Worker method*), 1363
run() (*angr.Emulator method*), 162
run() (*angr.exploration_techniques.tracer.RepHook method*), 575
run() (*angr.Server method*), 173
run() (*angr.sim_manager.SimulationManager method*), 546
run() (*angr.sim_procedure.SimProcedure method*), 645
run() (*angr.SimProcedure method*), 204
run() (*angr.SimulationManager method*), 216
run_pelf() (*in module angr.flirt.build_sig*), 1343
run_sigmake() (*in module angr.flirt.build_sig*), 1343
RuntimeDb (*class in angr.knowledge_plugins*), 733

RustEnum (*class in angr.ailment.expression*), 1395
RustEnum (*class in angr.analyses.typehoon.typeconsts*), 1279
RustStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen*), 1125

S

SAILRStructurer (*class in angr.analyses.decompiler.structuring*), 1035
satisfiable() (*angr.sim_state.SimState method*), 241
satisfiable() (*angr.SimState method*), 209
satisfiable() (*angr.state_plugins.SimSolver method*), 292
satisfiable() (*angr.state_plugins.solver.SimSolver method*), 366
save_info() (*angr.angrdb.AngrDB static method*), 1018
save_info() (*angr.angrdb.db.AngrDB static method*), 1020
SCANF_DELIMITERS (*angr.procedures.stubs.format_parser.FormatString attribute*), 646
ScanfFormatParser (*class in angr.procedures.stubs.format_parser*), 648
scc_id (*angr.utils.graph.SCCPlaceholder attribute*), 1349
SCCPlaceholder (*class in angr.utils.graph*), 1349
scratch (*angr.sim_state.SimState attribute*), 240
scratch (*angr.SimState attribute*), 208
SDiv() (*angr.state_plugins.sim_action_object.SimActionObject method*), 641
SDiv() (*angr.state_plugins.SimActionObject method*), 265
secondary_stackvars (*angr.analyses.decompiler.clinic.Clinic attribute*), 1061
section_alignment() (*angr.analyses.Reassembler method*), 916
section_alignment() (*angr.analyses.reassembler.Reassembler method*), 1302
SecurityCookieInit (*class in angr.simos.windows*), 1338
seek() (*angr.SimFileDescriptor method*), 183
seek() (*angr.SimFileDescriptorDuplex method*), 185
seek() (*angr.storage.file.SimFileDescriptor method*), 444
seek() (*angr.storage.file.SimFileDescriptorBase method*), 443
seek() (*angr.storage.file.SimFileDescriptorDuplex method*), 447
seekable (*angr.SimFileBase attribute*), 181
seekable (*angr.storage.file.SimFileBase attribute*), 433
seekable (*angr.storage.file.SimPacketsSlots attribute*), 449
SegfaultError, 396
Segment (*class in angr.storage.memory_mixins.regioned_memory.region_m*
536
SegmentBoundary (*angr.knowledge_plugins.cfg.memory_data.MemoryDa*
attribute), 764
SegmentBoundary (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 755
segments (*angr.storage.memory_mixins.regioned_memory.region_meta_m*
property), 536
selector() (*angr.exploration_techniques.ExplorationTechnique method*), 556
selector() (*angr.ExplorationTechnique method*), 163
selector() (*angr.sim_manager.SimulationManager method*), 548
selector() (*angr.SimulationManager method*), 218
SelfModifyingCodeAnalysis (*class in angr.analyses*), 921
SEMANTIC_VARIABLE_NAMING (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1058
seq (*angr.analyses.disassembly.IROp attribute*), 1294
SeqNodeDephication (*class in angr.analyses.decompiler*), 1051
sequence_matcher_similarity() (*angr.exploration_techniques.unique.UniqueSearch static method*), 585
sequence_matcher_similarity() (*angr.exploration_techniques.UniqueSearch static method*), 568
sequence_to_blocks() (*in module angr.analyses.decompiler.utils*), 1156
sequence_to_statements() (*in module angr.analyses.decompiler.utils*), 1156
SequenceOptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*
1095
SequenceWalker (*class in angr.analyses.decompiler.sequence_walker*), 1119
Serializable (*class in angr.serializable*), 870
SerializableCounter (*class in angr.calling_conventions*), 659
SerializableIterator (*class in angr.calling_conventions*), 659
SerializableListIterator (*class in angr.calling_conventions*), 659
serialize() (*angr.knowledge_plugins.functions.function_parser.Function*
static method), 809
serialize() (*angr.serializable.Serializable method*), 871
serialize_to_cmessage() (*angr.Block method*), 161
serialize_to_cmessage() (*angr.block.Block method*), 234
serialize_to_cmessage()

(*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 757
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.cfg_node.CFGENode method*), 769
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 768
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.CFGENode method*), 741
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.CFGModel method*), 743
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.CFGNode method*), 751
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump method*), 772
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.indirect_jump.JumptableInfo method*), 770
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.IndirectJump method*), 753
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.memory_data.MemoryData method*), 765
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.MemoryData method*), 755
serialize_to_cmessage() (*angr.knowledge_plugins.Function method*), 719
serialize_to_cmessage() (*angr.knowledge_plugins.functions.Function method*), 779
serialize_to_cmessage() (*angr.knowledge_plugins.functions.function.Function method*), 804
serialize_to_cmessage() (*angr.knowledge_plugins.variables.variable_access.VariableAccess method*), 812
serialize_to_cmessage() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 813
serialize_to_cmessage() (*angr.knowledge_plugins.XRefManager method*), 736
serialize_to_cmessage() (*angr.knowledge_plugins.xrefs.XRef method*), 861
serialize_to_cmessage() (*angr.knowledge_plugins.xrefs.xref.XRef method*), 864
serialize_to_cmessage() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 865
serialize_to_cmessage() (*angr.knowledge_plugins.xrefs.XRefManager method*), 862
serialize_to_cmessage() (*angr.serializable.Serializable method*), 870
serialize_to_cmessage() (*angr.sim_variable.SimComboRegisterVariable method*), 684
serialize_to_cmessage() (*angr.sim_variable.SimConstantVariable method*), 681
serialize_to_cmessage() (*angr.sim_variable.SimMemoryVariable method*), 685
serialize_to_cmessage() (*angr.sim_variable.SimRegisterVariable method*), 683
serialize_to_cmessage() (*angr.sim_variable.SimStackVariable method*), 686
serialize_to_cmessage() (*angr.sim_variable.SimTemporaryVariable method*), 682
Server (*class in angr*), 173
Server (*class in angr.distributed*), 1361
Server (*class in angr.distributed.server*), 1362
session_scope() (*angr.angrdb.AngrDB static method*), 1018
session_scope() (*angr.angrdb.db.AngrDB static method*), 1020
set() (*angr.knowledge_plugins.key_definitions.environment.Environment method*), 844
set_abi_cc() (*angr.procedures.definitions.SimSyscallLibrary method*), 655
set_alloc_depth() (*angr.state_plugins.SimUCManager method*), 325
set_alloc_depth() (*angr.state_plugins.uc_manager.SimUCManager method*), 388
set_args() (*angr.sim_procedure.SimProcedure method*), 645
set_args() (*angr.SimProcedure method*), 205
set_base_state() (*angr.callable.Callable method*), 711
set_bck_chunk() (*angr.PTChunk method*), 168
set_bck_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 414
set_bck_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 417
set_bck_chunk() (*angr.state_plugins.heap.PTChunk

method), 405
set_bck_chunk() (angr.state_plugins.PTChunk method), 259
set_brk() (angr.state_plugins.posix.SimSystemPosix method), 352
set_brk() (angr.state_plugins.SimSystemPosix method), 322
set_c_prototype() (angr.procedures.definitions.SimLibrary method), 651
set_data() (angr.state_plugins.GDB method), 247
set_data() (angr.state_plugins.gdb.GDB method), 379
set_default_cc() (angr.procedures.definitions.SimLibrary method), 650
set_entry_register_values() (angr.simos.linux.SimLinux method), 1336
set_entry_register_values() (angr.simos.SimLinux method), 1327
set_fd_data() (angr.exploration_techniques.Tracer method), 566
set_fd_data() (angr.exploration_techniques.tracer.Tracer method), 576
set_func_block_count() (angr.knowledge_plugins.FunctionManager method), 729
set_func_block_count() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 798
set_func_block_count() (angr.knowledge_plugins.functions.FunctionManager method), 789
set_function_returning() (angr.knowledge_plugins.FunctionManager method), 728
set_function_returning() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 798
set_function_returning() (angr.knowledge_plugins.functions.FunctionManager method), 788
set_fwd_chunk() (angr.PTChunk method), 168
set_fwd_chunk() (angr.state_plugins.heap.heap_freelist.Chunk method), 414
set_fwd_chunk() (angr.state_plugins.heap.heap_ptmalloc.PTChunk method), 417
set_fwd_chunk() (angr.state_plugins.heap.PTChunk method), 405
set_fwd_chunk() (angr.state_plugins.PTChunk method), 258
set_graph() (angr.codenode.CodeNode method), 1323
set_heap() (angr.state_plugins.GDB method), 247
set_heap() (angr.state_plugins.gdb.GDB method), 379
set_initial_regs() (angr.Block method), 160
set_initial_regs() (angr.block.Block method), 233
set_kb() (angr.knowledge_plugins.FunctionManager method), 725
set_kb() (angr.knowledge_plugins.functions.function_manager.FunctionM method), 794
set_kb() (angr.knowledge_plugins.functions.FunctionManager method), 785
set_kb() (angr.knowledge_plugins.KnowledgeBasePlugin method), 731
set_kb() (angr.knowledge_plugins.plugin.KnowledgeBasePlugin method), 738
set_last_block_details() (angr.state_plugins.Unicorn method), 328
set_last_block_details() (angr.state_plugins.unicorn_engine.Unicorn method), 398
set_last_statement() (angr.annocfg.AnnotatedCFG method), 1321
set_library_names() (angr.procedures.definitions.SimLibrary method), 650
set_live_variables() (angr.knowledge_plugins.variables.variable_manager.VariableMa method), 814
set_manager() (angr.knowledge_plugins.variables.variable_manager.Var method), 813
set_mode() (angr.sim_state.SimState method), 242
set_mode() (angr.SimState method), 211
set_names() (angr.procedures.definitions.SimTypeCollection method), 649
set_non_returning() (angr.procedures.definitions.SimLibrary method), 650
set_object() (angr.keyed_region.KeyedRegion method), 869
set_object() (angr.keyed_region.RegionObject method), 868
set_prev_freeness() (angr.PTChunk method), 167
set_prev_freeness() (angr.state_plugins.heap.heap_ptmalloc.PTChunk method), 416
set_prev_freeness() (angr.state_plugins.heap.PTChunk method), 405
set_prev_freeness() (angr.state_plugins.PTChunk method), 258
set_prototype() (angr.knowledge_plugins.callsite_prototypes.CallsitePr method), 739
set_prototype() (angr.knowledge_plugins.CallsitePrototypes method), 713
set_prototype() (angr.procedures.definitions.SimLibrary method), 651
set_prototype() (angr.procedures.definitions.SimSyscallLibrary method), 655
set_prototypes() (angr.procedures.definitions.SimLibrary method), 651

set_prototypes() (*angr.procedures.definitions.SimSyscallLibrary method*), 656
set_regs() (*angr.state_plugins.GDB method*), 247
set_regs() (*angr.state_plugins.gdb.GDB method*), 379
set_regs() (*angr.state_plugins.Unicorn method*), 329
set_regs() (*angr.state_plugins.unicorn_engine.Unicorn method*), 399
set_return_val() (*angr.calling_conventions.SimCC method*), 666
set_return_val() (*angr.calling_conventions.SimCCSyscall method*), 670
set_return_val() (*angr.SimCC method*), 176
set_simgr() (*angr.analyses.congruency_check.CongruencyCheck method*), 1305
set_simgr() (*angr.analyses.CongruencyCheck method*), 899
set_size() (*angr.PTChunk method*), 167
set_size() (*angr.sim_type.SimTypeRef method*), 708
set_size() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 414
set_size() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 416
set_size() (*angr.state_plugins.heap.PTChunk method*), 404
set_size() (*angr.state_plugins.PTChunk method*), 257
set_stack() (*angr.state_plugins.GDB method*), 247
set_stack() (*angr.state_plugins.gdb.GDB method*), 379
set_stack_address_mapping() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 532
set_stack_address_mapping() (*angr.storage.memory_mixins.regioned_memory.RegionedMemoryMixin method*), 529
set_stack_address_mapping() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 482
set_stack_size() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 533
set_stack_size() (*angr.storage.memory_mixins.regioned_memory.RegionedMemoryMixin method*), 530
set_stack_size() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 482
set_state() (*angr.SimFile method*), 179
set_state() (*angr.SimFileDescriptor method*), 183
set_state() (*angr.SimFileDescriptorDuplex method*), 185
set_state() (*angr.SimFileStream method*), 188
set_state() (*angr.SimPackets method*), 198
set_state() (*angr.SimStatePlugin method*), 211
set_state() (*angr.state_plugins.CallStack method*), 248
set_state() (*angr.state_plugins.callstack.CallStack method*), 372
set_state() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 359
set_state() (*angr.state_plugins.filesystem.SimFilesystem method*), 356
set_state() (*angr.state_plugins.globals.SimStateGlobals method*), 386
set_state() (*angr.state_plugins.inspect.SimInspector method*), 343
set_state() (*angr.state_plugins.light_registers.SimLightRegisters method*), 375
set_state() (*angr.state_plugins.plugin.SimStatePlugin method*), 329
set_state() (*angr.state_plugins.posix.SimSystemPosix method*), 352
set_state() (*angr.state_plugins.SimFilesystem method*), 268
set_state() (*angr.state_plugins.SimInspector method*), 279
set_state() (*angr.state_plugins.SimLightRegisters method*), 281
set_state() (*angr.state_plugins.SimMemView method*), 282
set_state() (*angr.state_plugins.SimStateEdgeHitmap method*), 298
set_state() (*angr.state_plugins.SimStateGlobals method*), 299
set_state() (*angr.state_plugins.SimStateIcicle method*), 304
set_state() (*angr.state_plugins.SimStatePlugin method*), 314
set_state() (*angr.state_plugins.SimSystemPosix method*), 322
set_state() (*angr.state_plugins.SimUCManager method*), 325
set_state() (*angr.state_plugins.uc_manager.SimUCManager method*), 388
set_state() (*angr.state_plugins.Unicorn method*), 328
set_state() (*angr.state_plugins.unicorn_engine.Unicorn method*), 398
set_state() (*angr.state_plugins.view.SimMemView method*), 428
set_state() (*angr.storage.file.SimFile method*), 434
set_state() (*angr.storage.file.SimFileDescriptor method*), 445
set_state() (*angr.storage.file.SimFileDescriptorDuplex method*), 447
set_state() (*angr.storage.file.SimFileStream method*), 436
set_state() (*angr.storage.file.SimPackets method*), 438
set_state() (*angr.storage.memory_mixins.address_concretization_mixin.* *method*), 495
set_state() (*angr.storage.memory_mixins.AddressConcretizationMixin method*), 456
set_state() (*angr.storage.memory_mixins.ISPOMixin*

method), 463
set_state() (angr.storage.memory_mixins.JavaVmMemoryMixin method), 465
set_state() (angr.storage.memory_mixins.paged_memory.pages.ISPOMixin method), 520
set_state() (angr.storage.memory_mixins.paged_memory.pages.ISPOMixin method), 511
set_state() (angr.storage.memory_mixins.regioned_memory.region_addresses_logic.SimRegionAddressConcretizatio method), 538
set_state() (angr.storage.memory_mixins.regioned_memory.stack_range_logic.StackRangeMemoryMixin method), 532
set_state() (angr.storage.memory_mixins.regioned_memory.stack_addresses_logic.StackAddressConcretizationMixin method), 526
set_state() (angr.storage.memory_mixins.regioned_memory.RegionedMemoryMixin method), 529
set_state() (angr.storage.memory_mixins.RegionedAddressConcretizationMixin method), 479
set_state() (angr.storage.memory_mixins.RegionedMemoryMixin method), 482
set_state() (angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method), 540
set_state() (angr.storage.memory_mixins.SlottedMemoryMixin method), 484
set_state() (angr.storage.SimFile method), 423
set_state_options() (angr.analyses.congruency_check.CongruencyCheck method), 1305
set_state_options() (angr.analyses.CongruencyCheck method), 899
set_states() (angr.analyses.congruency_check.CongruencyCheck method), 1305
set_states() (angr.analyses.CongruencyCheck method), 899
set_stops() (angr.state_plugins.Unicorn method), 328
set_stops() (angr.state_plugins.unicorn_engine.Unicorn method), 398
set_strongref_state() (angr.SimStatePlugin method), 211
set_strongref_state() (angr.state_plugins.history.SimStateHistory method), 375
set_strongref_state() (angr.state_plugins.plugin.SimStatePlugin method), 329
set_strongref_state() (angr.state_plugins.SimStateHistory method), 301
set_strongref_state() (angr.state_plugins.SimStatePlugin method), 314
set_symbolization_for_all_pages() (angr.state_plugins.SimSymbolizer method), 320
set_symbolization_for_all_pages() (angr.state_plugins.symbolizer.SimSymbolizer method), 420
set_symbolized_ISPOMixin_range() (angr.state_plugins.SimSymbolizer method), 320
set_symbolized_target_range() (angr.state_plugins.symbolizer.SimSymbolizer method), 420
set_tracking() (angr.state_plugins.SimMemory method), 328
set_tracking() (angr.state_plugins.unicorn_engine.Unicorn method), 398
set_type() (angr.state_plugins.scratch.SimStateScratch method), 388
set_type() (angr.state_plugins.SimStateScratch method), 318
set_type() (angr.analyses.decompiler.structured_codegen.c.CExpression method), 1133
set_variable() (angr.knowledge_plugins.variables.variable_manager.VariableMa method), 818
set_value() (angr.calling_conventions.SimArrayArg method), 662
set_value() (angr.calling_conventions.SimComboArg method), 661
set_value() (angr.calling_conventions.SimFunctionArgument method), 659
set_value() (angr.calling_conventions.SimLyingRegArg method), 667
set_value() (angr.calling_conventions.SimReferenceArgument method), 663
set_value() (angr.calling_conventions.SimRegArg method), 660
set_value() (angr.calling_conventions.SimStackArg method), 661
set_value() (angr.calling_conventions.SimStructArg method), 662
set_variable() (angr.keyed_region.KeyedRegion method), 869
set_variable() (angr.knowledge_plugins.variables.variable_manager.Va method), 814
set_variable_type() (angr.knowledge_plugins.variables.variable_manager.VariableMa method), 818
setstate() (angr.calling_conventions.ArgSession method), 663
setstate() (angr.calling_conventions.SerializableCounter method), 659
setstate() (angr.calling_conventions.SerializableIterator method), 659
setstate() (angr.calling_conventions.SerializableListIterator method), 659
setstate() (angr.calling_conventions.SimCC.ArgSession

method), 665
setstate() (*angr.calling_conventions.UsercallArgSession method*), 663
setstate() (*angr.SimCC.ArgSession method*), 175
setup() (*angr.exploration_techniques.DFS method*), 552
setup() (*angr.exploration_techniques.dfs.DFS method*), 569
setup() (*angr.exploration_techniques.driller_core.DrillerCore method*), 577
setup() (*angr.exploration_techniques.DrillerCore method*), 554
setup() (*angr.exploration_techniques.ExplorationTechnique method*), 555
setup() (*angr.exploration_techniques.Explorer method*), 557
setup() (*angr.exploration_techniques.explorer.Explorer method*), 570
setup() (*angr.exploration_techniques.local_loop_seer.LocalLoopSeer method*), 583
setup() (*angr.exploration_techniques.LocalLoopSeer method*), 559
setup() (*angr.exploration_techniques.loop_seer.LoopSeer method*), 582
setup() (*angr.exploration_techniques.LoopSeer method*), 560
setup() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 571
setup() (*angr.exploration_techniques.ManualMergepoint method*), 560
setup() (*angr.exploration_techniques.memory_watcher.MemoryWatcher method*), 586
setup() (*angr.exploration_techniques.MemoryWatcher method*), 561
setup() (*angr.exploration_techniques.Slicecutor method*), 562
setup() (*angr.exploration_techniques.slicecutor.Slicecutor method*), 578
setup() (*angr.exploration_techniques.Timeout method*), 565
setup() (*angr.exploration_techniques.timeout.Timeout method*), 569
setup() (*angr.exploration_techniques.Tracer method*), 566
setup() (*angr.exploration_techniques.tracer.Tracer method*), 576
setup() (*angr.exploration_techniques.unique.UniqueSearch method*), 584
setup() (*angr.exploration_techniques.UniqueSearch method*), 568
setup() (*angr.ExplorationTechnique method*), 162
setup() (*angr.state_plugins.Unicorn method*), 328
setup() (*angr.state_plugins.unicorn_engine.Unicorn method*), 398
setup_arguments() (*angr.engines.soot.engine.SootMixin static method*), 602
setup_arguments() (*angr.engines.soot.SootMixin static method*), 602
setup_arguments() (*angr.engines.SootMixin static method*), 593
setup_callsite() (*angr.calling_conventions.SimCC method*), 666
setup_callsite() (*angr.calling_conventions.SimCCSoot method*), 678
setup_callsite() (*angr.engines.soot.engine.SootMixin class method*), 602
setup_callsite() (*angr.engines.soot.SootMixin class method*), 602
setup_callsite() (*angr.engines.SootMixin class method*), 593
setup_callsite() (*angr.SimCC method*), 176
setup_flags() (*angr.state_plugins.Unicorn method*), 329
setup_flags() (*angr.state_plugins.unicorn_engine.Unicorn method*), 399
setup_gdt() (*angr.SimOS method*), 197
setup_gdt() (*angr.simos.SimOS method*), 1330
setup_gdt() (*angr.simos.simos.SimOS method*), 1335
setup_gdt() (*angr.state_plugins.Unicorn method*), 329
setup_gdt() (*angr.state_plugins.unicorn_engine.Unicorn method*), 399
setup_terminal() (*in module angr.utils.formatting*), 1353
shallow_reverse() (*in module angr.utils.graph*), 1346
shared_common_conditional_dom() (*angr.analyses.decompiler.optimization_passes.DuplicationRever method*), 1078
ShiftLeft (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressT attribute*), 998
ShiftRight (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.Address attribute*), 998
short_name (*angr.knowledge_plugins.Function property*), 724
short_name (*angr.knowledge_plugins.functions.Function property*), 784
short_name (*angr.knowledge_plugins.functions.function.Function property*), 808
short_reason (*angr.knowledge_plugins.cfg.cfg_node.CFGNodeCreationF attribute*), 766
short_repr (*angr.analyses.ddg.ProgramVariable property*), 1157
short_repr (*angr.code_location.CodeLocation property*), 866
should_abort (*angr.analyses.forward_analysis.forward_analysis.Forward property*), 936
should_abort (*angr.analyses.forward_analysis.ForwardAnalysis property*), 932
should_abort (*angr.analyses.ForwardAnalysis prop-*

erty), 907
should_add_successors (*angr.sim_procedure.SimProcedure property*), 645
should_add_successors (*angr.SimProcedure property*), 205
should_execute_statement() (*angr.annocfg.AnnotatedCFG method*), 1322
should_skip_main_binary() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 1030
should_skip_main_binary() (*angr.angrdb.serializers.LoaderSerializer static method*), 1026
should_take_exit() (*angr.annocfg.AnnotatedCFG method*), 1322
show_demangled_name (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 1132
show_demangled_name (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 1136
show_disambiguated_name (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 1136
shrink() (*angr.analyses.reassembler.Data method*), 1301
SideEffectStatement (*class in angr.ailment.statement*), 1401
SideEffectTag (*class in angr.knowledge_plugins.key_definitions.tag*), 857
SIG_BLOCK (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
SIG_BLOCK (*angr.state_plugins.SimSystemPosix attribute*), 321
SIG_SETMASK (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
SIG_SETMASK (*angr.state_plugins.SimSystemPosix attribute*), 321
SIG_UNBLOCK (*angr.state_plugins.posix.SimSystemPosix attribute*), 351
SIG_UNBLOCK (*angr.state_plugins.SimSystemPosix attribute*), 321
sigmask() (*angr.state_plugins.posix.SimSystemPosix method*), 353
sigmask() (*angr.state_plugins.SimSystemPosix method*), 323
sign_bit (*angr.ailment.Const property*), 1369
sign_bit (*angr.ailment.expression.Const property*), 1377
SIGNATURES (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 800
signed (*angr.ailment.BinaryOp attribute*), 1367
signed (*angr.ailment.expression.BinaryOp attribute*), 1385
signed (*angr.procedures.stubs.format_parser.FormatSpecifier property*), 647
signed (*angr.sim_type.SimTypeFloat attribute*), 700
signed_int_type() (*in module angr.analyses.typehoon.typeconsts*), 1281
SignedExtension (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.A attribute*), 998
sigprocmask() (*angr.state_plugins.posix.SimSystemPosix method*), 353
sigprocmask() (*angr.state_plugins.SimSystemPosix method*), 323
sim_engine (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
sim_engine (*angr.state_plugins.InspectAttrs attribute*), 255
sim_procedure (*angr.code_location.CodeLocation attribute*), 866
sim_procedure (*angr.codenode.HookNode attribute*), 1324
sim_procedure (*angr.codenode.SyscallNode attribute*), 1324
sim_successors (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
sim_successors (*angr.state_plugins.InspectAttrs attribute*), 255
SimAbstractMemoryError, 173, 1358
SimAction (*class in angr.state_plugins*), 263
SimAction (*class in angr.state_plugins.sim_action*), 638
SimActionConstraint (*class in angr.state_plugins*), 263
SimActionConstraint (*class in angr.state_plugins.sim_action*), 639
SimActionData (*class in angr.state_plugins*), 263
SimActionData (*class in angr.state_plugins.sim_action*), 639
SimActionError, 174, 1360
SimActionExit (*class in angr.state_plugins*), 264
SimActionExit (*class in angr.state_plugins.sim_action*), 638
SimActionObject (*class in angr.state_plugins*), 264
SimActionObject (*class in angr.state_plugins.sim_action_object*), 640
SimActionOperation (*class in angr.state_plugins*), 266
SimActionOperation (*class in angr.state_plugins.sim_action*), 639
SimActLocation (*class in angr.analyses.data_dep.sim_act_location*), 1312
SimArrayArg (*class in angr.calling_conventions*), 662
SimCC (*class in angr*), 174
SimCC (*class in angr.calling_conventions*), 663
SimCC.ArgSession (*class in angr*), 175

SimCC.ArgSession (*class in angr.calling_conventions*), 664
SimCCAArch64 (*class in angr.calling_conventions*), 673
SimCCAArch64LinuxSyscall (*class in angr.calling_conventions*), 674
SimCCallError, 177, 1359
SimCCAMD64LinuxSyscall (*class in angr.calling_conventions*), 672
SimCCAMD64WindowsSyscall (*class in angr.calling_conventions*), 672
SimCCARM (*class in angr.calling_conventions*), 672
SimCCARMHF (*class in angr.calling_conventions*), 673
SimCCARMLinuxSyscall (*class in angr.calling_conventions*), 673
SimCCCdecl (*class in angr.calling_conventions*), 668
SimCCError, 177, 1360
SimCCM68k (*class in angr.engines.pcode.cc*), 636
SimCCMicrosoftAMD64 (*class in angr.calling_conventions*), 669
SimCCMicrosoftCdecl (*class in angr.calling_conventions*), 668
SimCCMicrosoftFastcall (*class in angr.calling_conventions*), 669
SimCCMicrosoftThiscall (*class in angr.calling_conventions*), 668
SimCCN64 (*class in angr.calling_conventions*), 675
SimCCN64LinuxSyscall (*class in angr.calling_conventions*), 676
SimCCO32 (*class in angr.calling_conventions*), 675
SimCCO32LinuxSyscall (*class in angr.calling_conventions*), 675
SimCCO64 (*in module angr.calling_conventions*), 676
SimCCPARISC (*class in angr.engines.pcode.cc*), 637
SimCCPCodeBase (*class in angr.engines.pcode.cc*), 636
SimCCPowerPC (*class in angr.calling_conventions*), 676
SimCCPowerPC (*class in angr.engines.pcode.cc*), 637
SimCCPowerPC64 (*class in angr.calling_conventions*), 677
SimCCPowerPC64LinuxSyscall (*class in angr.calling_conventions*), 677
SimCCPowerPCLinuxSyscall (*class in angr.calling_conventions*), 677
SimCCRISCV (*class in angr.engines.pcode.cc*), 636
SimCCRISCV64 (*class in angr.calling_conventions*), 674
SimCCRISCV64LinuxSyscall (*class in angr.calling_conventions*), 674
SimCCS390X (*class in angr.calling_conventions*), 679
SimCCS390XLinuxSyscall (*class in angr.calling_conventions*), 679
SimCCSH4 (*class in angr.engines.pcode.cc*), 637
SimCCSoot (*class in angr.calling_conventions*), 678
SimCCSPARC (*class in angr.engines.pcode.cc*), 636
SimCCStdcall (*class in angr.calling_conventions*), 669
SimCCSyscall (*class in angr.calling_conventions*), 670
SimCCSystemVAMD64 (*class in angr.calling_conventions*), 671
SimCCUnknown (*class in angr.calling_conventions*), 679
SimCCUsercall (*class in angr.calling_conventions*), 667
SimCCX86LinuxSyscall (*class in angr.calling_conventions*), 670
SimCCX86WindowsSyscall (*class in angr.calling_conventions*), 671
SimCCXtensa (*class in angr.engines.pcode.cc*), 638
SimCGC (*class in angr.simos*), 1325
SimCGC (*class in angr.simos.cgc*), 1337
SimComboArg (*class in angr.calling_conventions*), 661
SimComboRegisterVariable (*class in angr.sim_variable*), 683
SimConcreteBreakpointError, 178, 1361
SimConcreteFilesystem (*class in angr.state_plugins.filesystem*), 358
SimConcreteMemoryError, 178, 1361
SimConcreteRegisterError, 178, 1361
SimConcretizationStrategy (*class in angr.concretization_strategies*), 452
SimConcretizationStrategyAny (*class in angr.concretization_strategies*), 452
SimConcretizationStrategyAny (*class in angr.concretization_strategies.any*), 543
SimConcretizationStrategyControlledData (*class in angr.concretization_strategies*), 452
SimConcretizationStrategyControlledData (*class in angr.concretization_strategies.controlled_data*), 543
SimConcretizationStrategyEval (*class in angr.concretization_strategies*), 453
SimConcretizationStrategyEval (*class in angr.concretization_strategies.eval*), 541
SimConcretizationStrategyMax (*class in angr.concretization_strategies*), 453
SimConcretizationStrategyMax (*class in angr.concretization_strategies.max*), 542
SimConcretizationStrategyNonzero (*class in angr.concretization_strategies*), 453
SimConcretizationStrategyNonzero (*class in angr.concretization_strategies.nonzero*), 543
SimConcretizationStrategyNonzeroRange (*class in angr.concretization_strategies*), 453
SimConcretizationStrategyNonzeroRange (*class in angr.concretization_strategies.nonzero_range*), 542
SimConcretizationStrategyNorepeats (*class in angr.concretization_strategies*), 454
SimConcretizationStrategyNorepeats (*class in angr.concretization_strategies.norepeats*), 541
SimConcretizationStrategyNorepeatsRange (*class in angr.concretization_strategies*), 454
SimConcretizationStrategyNorepeatsRange (*class

in angr.concretization_strategies.norepeats_range), 543
SimConcretizationStrategyRange (*class in angr.concretization_strategies*), 455
SimConcretizationStrategyRange (*class in angr.concretization_strategies.range*), 542
SimConcretizationStrategySingle (*class in angr.concretization_strategies*), 455
SimConcretizationStrategySingle (*class in angr.concretization_strategies.single*), 541
SimConcretizationStrategySolutions (*class in angr.concretization_strategies*), 455
SimConcretizationStrategySolutions (*class in angr.concretization_strategies.solutions*), 541
SimConcretizationStrategyUnlimitedRange (*class in angr.concretization_strategies*), 455
SimConcretizationStrategyUnlimitedRange (*class in angr.concretization_strategies.unlimited_range*), 544
SimConstantVariable (*class in angr.sim_variable*), 681
SimCppClass (*class in angr.sim_type*), 705
SimCppClassValue (*class in angr.sim_type*), 707
SimCppLibrary (*class in angr.procedures.definitions*), 653
SimDebugVariable (*class in angr.state_plugins*), 266
SimDebugVariable (*class in angr.state_plugins.debug_variables*), 421
SimDebugVariablePlugin (*class in angr.state_plugins*), 267
SimDebugVariablePlugin (*class in angr.state_plugins.debug_variables*), 422
SimEmptyCallStackError, 178, 1360
SimEngine (*class in angr.engines*), 589
SimEngine (*class in angr.engines.engine*), 594
SimEngineError, 178, 1359
SimEngineFailure (*class in angr.engines*), 589
SimEngineFailure (*class in angr.engines.failure*), 598
SimEngineInitFinderVEX (*class in angr.analyses.init_finder*), 1307
SimEngineLight (*class in angr.engines.light*), 1168
SimEngineLight (*class in angr.engines.light.engine*), 1171
SimEngineLightAIL (*class in angr.engines.light*), 1169
SimEngineLightAIL (*class in angr.engines.light.engine*), 1173
SimEngineLightVEX (*class in angr.engines.light*), 1169
SimEngineLightVEX (*class in angr.engines.light.engine*), 1172
SimEngineNoexprAIL (*class in angr.engines.light*), 1170
SimEngineNoexprAIL (*class in angr.engines.light.engine*), 1174
SimEngineNostmtAIL (*class in angr.engines.light*), 1170
SimEngineNostmtAIL (*class in angr.engines.light.engine*), 1174
SimEngineNostmtVEX (*class in angr.engines.light*), 1170
SimEngineNostmtVEX (*class in angr.engines.light.engine*), 1173
SimEnginePropagatorBaseMixin (*class in angr.analyses.propagator.engine_base*), 1176
SimEnginePropagatorVEX (*class in angr.analyses.propagator.engine_vex*), 1177
SimEngineRDAIL (*class in angr.analyses.reaching_definitions.engine_ail*), 1238
SimEngineRDVEX (*class in angr.analyses.reaching_definitions.engine_vex*), 1213
SimEngineSyscall (*class in angr.engines*), 589
SimEngineSyscall (*class in angr.engines.syscall*), 598
SimEngineUnicorn (*class in angr.engines*), 590
SimEngineUnicorn (*class in angr.engines.unicorn*), 603
SimEngineVRAIL (*class in angr.analyses.variable_recovery.engine_ail*), 1255
SimEngineVRBase (*class in angr.analyses.variable_recovery.engine_base*), 1257
SimEngineVRVEX (*class in angr.analyses.variable_recovery.engine_vex*), 1256
SimEngineXRefsVEX (*class in angr.analyses.xrefs*), 1308
SimError, 178, 1357
SimEvent (*class in angr.state_plugins*), 268
SimEvent (*class in angr.state_plugins.sim_event*), 642
SimEventError, 178, 1358
SimException, 178, 1361
SimExpressionError, 178, 1359
SimFastMemoryError, 178, 1358
SimFastPathError, 178, 1360
SimFile (*class in angr*), 178
SimFile (*class in angr.storage*), 423
SimFile (*class in angr.storage.file*), 434
SimFileBase (*class in angr*), 181
SimFileBase (*class in angr.storage.file*), 432
SimFileDescriptor (*class in angr*), 182
SimFileDescriptor (*class in angr.storage.file*), 444
SimFileDescriptorBase (*class in angr.storage.file*), 442
SimFileDescriptorDuplex (*class in angr*), 185
SimFileDescriptorDuplex (*class in angr.storage.file*), 446
SimFileError, 187, 1358
SimFileStream (*class in angr*), 187
SimFileStream (*class in angr.storage.file*), 436

SimFilesystem (*class in angr.state_plugins*), 268
SimFilesystem (*class in angr.state_plugins.filesystem*), 355
SimFilesystemError, 189, 1358
SimFunctionArgument (*class in angr.calling_conventions*), 659
simgr() (*angr.factory.AngrObjectFactory method*), 230
SimHeapBase (*class in angr.state_plugins*), 270
SimHeapBase (*class in angr.state_plugins.heap*), 406
SimHeapBase (*class in angr.state_plugins.heap.heap_base*), 411
SimHeapBrk (*class in angr*), 189
SimHeapBrk (*class in angr.state_plugins*), 270
SimHeapBrk (*class in angr.state_plugins.heap*), 406
SimHeapBrk (*class in angr.state_plugins.heap.heap_brk*), 412
SimHeapError, 191, 1358
SimHeapFreelist (*class in angr.state_plugins.heap.heap_freelist*), 415
SimHeapLibc (*class in angr.state_plugins*), 272
SimHeapLibc (*class in angr.state_plugins.heap*), 408
SimHeapLibc (*class in angr.state_plugins.heap.heap_libc*), 415
SimHeapPTMalloc (*class in angr*), 191
SimHeapPTMalloc (*class in angr.state_plugins*), 273
SimHeapPTMalloc (*class in angr.state_plugins.heap*), 409
SimHeapPTMalloc (*class in angr.state_plugins.heap.heap_ptmalloc*), 417
SimHostFilesystem (*class in angr*), 193
SimHostFilesystem (*class in angr.state_plugins*), 275
SimHostFilesystem (*class in angr.state_plugins.filesystem*), 360
similar_conditional_when_single_corrected() (*angr.analyses.decompiler.optimization_passes.DuplicateReturn method*), 1078
similarity() (*angr.exploration_techniques.unique.UniqueSearch static method*), 585
similarity() (*angr.exploration_techniques.UniqueSearch static method*), 568
SimInspectMixin (*class in angr.engines*), 590
SimInspectMixin (*class in angr.engines.vex*), 600
SimInspector (*class in angr.state_plugins*), 276
SimInspector (*class in angr.state_plugins.inspect*), 340
SimIRSBError, 194, 1359
SimIRSBNoDecodeError, 194, 1360
SimJavaVM (*class in angr.simos*), 1325
SimJavaVM (*class in angr.simos.javavm*), 1340
SimJavaVmClassloader (*class in angr.state_plugins*), 279
SimJavaVmClassloader (*class in angr.state_plugins.javavm_classloader*), 400
SimLabeledMemoryObject (*class in angr.storage.memory_object*), 451
SimLibrary (*class in angr.procedures.definitions*), 650
SimLightRegisters (*class in angr.state_plugins*), 281
SimLightRegisters (*class in angr.state_plugins.light_registers*), 374
SimLightState (*class in angr.slicer*), 1320
SimLinux (*class in angr.simos*), 1327
SimLinux (*class in angr.simos.linux*), 1335
SimLyingRegArg (*class in angr.calling_conventions*), 667
SimMemoryAddressError, 194, 1358
SimMemoryError, 194, 1358
SimMemoryLimitError, 194, 1358
SimMemoryMissingError, 194, 1358
SimMemoryObject (*class in angr.storage*), 425
SimMemoryObject (*class in angr.storage.memory_object*), 451
SimMemoryVariable (*class in angr.sim_variable*), 684
SimMemView (*class in angr.state_plugins*), 281
SimMemView (*class in angr.state_plugins.view*), 427
SimMergeError, 194, 1358
SimMissingTempError, 194, 1359
SimMount (*class in angr*), 194
SimMount (*class in angr.state_plugins*), 285
SimMount (*class in angr.state_plugins.filesystem*), 357
simop_from_vexop() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
SimOperationError, 197, 1359
SimOS (*class in angr*), 195
SimOS (*class in angr.simos*), 1328
SimOS (*class in angr.simos.simos*), 1333
SIMOS_CGC (*angr.state_plugins.unicorn_engine.SimOSEnum attribute*), 396
SIMOS_LINUX (*angr.state_plugins.unicorn_engine.SimOSEnum attribute*), 396
SIMOS_OTHER (*angr.state_plugins.unicorn_engine.SimOSEnum attribute*), 396
SimOSEnum (*class in angr.state_plugins.unicorn_engine*), 395
SimPackets (*class in angr*), 197
SimPackets (*class in angr.storage.file*), 438
SimPacketsSlots (*class in angr.storage.file*), 449
SimPacketsStream (*class in angr*), 200
SimPacketsStream (*class in angr.storage.file*), 440
SimpleInterfaceMixin (*class in angr.storage.memory_mixins*), 482
SimpleInterfaceMixin (*class in angr.storage.memory_mixins.simple_interface_mixin*), 493
SimpleSolver (*class in angr.analyses.typehoon.simple_solver*), 1262
SimplificationMixin (*class in angr.storage.memory_mixins*), 483

SimplificationMixin (*class in angr.storage.memory_mixins.simplification_mixin*), 498
simplified (*angr.analyses.decompiler.ail_simplifier.AILSimplifier attribute*), 1054
simplified_constraints_count (*angr.analyses.typehoon.simple_solver.SimpleSolver attribute*), 1263
simplified_data_graph (*angr.analyses.DDG property*), 876
simplified_data_graph (*angr.analyses.ddg.DDG property*), 1160
simplified_graph (*angr.analyses.data_dep.data_dependency_graph.DataDependencyGraphAnalysis property*), 1312
simplified_graph (*angr.analyses.data_dep.DataDependencyGraphAnalysis property*), 1317
simplified_graph (*angr.analyses.DataDependencyGraphAnalysis property*), 900
SimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.engine_base*), 1101
SimplifierAILState (*class in angr.analyses.decompiler.optimization_passes.engine_base*), 1101
simplify() (*angr.sim_state.SimState method*), 241
simplify() (*angr.SimState method*), 209
simplify() (*angr.state_hierarchy.StateHierarchy method*), 552
simplify() (*angr.state_plugins.SimSolver method*), 296
simplify() (*angr.state_plugins.solver.SimSolver method*), 370
simplify() (*angr.StateHierarchy method*), 221
simplify_condition() (*angr.analyses.decompiler.condition_processor.ConditionProcessor static method*), 1063
simplify_else_scope (*angr.analyses.decompiler.structured_codegen.c.CIfElseNode attribute*), 1134
simplify_lowered_switches() (*in module angr.analyses.decompiler.region_simplifiers.switch_clusters*), 1118
simplify_lowered_switches_core() (*in module angr.analyses.decompiler.region_simplifiers.switch_clusters*), 1118
simplify_switch_clusters() (*in module angr.analyses.decompiler.region_simplifiers.switch_clusters*), 1117
SimPosixError, 201, 1358
SIMPROC (*angr.knowledge_plugins.functions.function.PrototypeKnowledge attribute*), 800
SIMPROCEDURE (*angr.state_plugins.inspect.EventType attribute*), 332
simprocedure (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
simprocedure (*angr.state_plugins.InspectAttrs attribute*), 255
SimProcedure (*class in angr*), 201
SimProcedure (*class in angr.sim_procedure*), 642
simprocedure_addr (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
simprocedure_addr (*angr.state_plugins.InspectAttrs attribute*), 255
simprocedure_name (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 767
simprocedure_name (*angr.knowledge_plugins.cfg.CFGNode attribute*), 750
simprocedure_name (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
simprocedure_name (*angr.state_plugins.InspectAttrs attribute*), 255
simprocedure_result (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
simprocedure_result (*angr.state_plugins.InspectAttrs attribute*), 255
SimProcedureArgumentError, 206, 1360
SimProcedureError, 206, 1360
SimReferenceArgument (*class in angr.calling_conventions*), 662
SimRegArg (*class in angr.calling_conventions*), 660
SimRegionMapError, 206, 1358
SimRegisterVariable (*class in angr.sim_variable*), 683
SimRegNameView (*class in angr.state_plugins*), 286
SimRegNameView (*class in angr.state_plugins.view*), 426
SimReliftException, 206, 1360
SimSegfaultError (*in module angr*), 206
SimSegfaultError (*in module angr.errors*), 1361
SimSegfaultException, 206, 1361
SimShadowStackError, 206, 1360
SimSlicer (*class in angr.slicer*), 1321
SimSlicerError, 206, 1360
SimSnimmucNxp (*class in angr.simos*), 1330
SimSolver (*class in angr.state_plugins*), 287
SimSolver (*class in angr.state_plugins.solver*), 361
SimSolverError, 206, 1359
SimSolverModelError, 206, 1359
SimSolverOptionError, 206, 1359
SimStackArg (*class in angr.calling_conventions*), 660
SimStackVariable (*class in angr.sim_variable*), 685
SimState (*class in angr*), 206
SimState (*class in angr.sim_state*), 238
SimStateCGC (*class in angr.state_plugins*), 296
SimStateCGC (*class in angr.state_plugins.cgc*), 379
SimStateEdgeHitmap (*class in angr.state_plugins*), 298
SimStateError, 211, 1358
SimStateGlobals (*class in angr.state_plugins*), 299

SimStateGlobals (*class in angr.state_plugins.globals*), 386
SimStateHistory (*class in angr.state_plugins*), 301
SimStateHistory (*class in angr.state_plugins.history*), 375
SimStateIcicle (*class in angr.state_plugins*), 304
SimStateJNIReferences (*class in angr.state_plugins*), 305
SimStateJNIReferences (*class in angr.state_plugins.jni_references*), 402
SimStateLibc (*class in angr.state_plugins*), 307
SimStateLibc (*class in angr.state_plugins.libc*), 343
SimStateLog (*class in angr.state_plugins*), 311
SimStateLog (*class in angr.state_plugins.log*), 370
SimStateLoopData (*class in angr.state_plugins*), 312
SimStateLoopData (*class in angr.state_plugins.loop_data*), 399
SimStatementError, 213, 1359
SimStateOptions (*class in angr.sim_state_options*), 243
SimStateOptionsError, 211, 1360
SimStatePlugin (*class in angr*), 211
SimStatePlugin (*class in angr.state_plugins*), 314
SimStatePlugin (*class in angr.state_plugins.plugin*), 329
SimStatePreconstrainer (*class in angr.state_plugins*), 316
SimStatePreconstrainer (*class in angr.state_plugins.preconstrainer*), 390
SimStateScratch (*class in angr.state_plugins*), 318
SimStateScratch (*class in angr.state_plugins.scratch*), 388
SimStruct (*class in angr.sim_type*), 701
SimStructArg (*class in angr.calling_conventions*), 661
SimStructValue (*class in angr.sim_type*), 702
SimSuccessors (*class in angr.engines*), 591
SimSuccessors (*class in angr.engines.successors*), 594
SimSymbolicFilesystemError, 213, 1358
SimSymbolizer (*class in angr.state_plugins*), 320
SimSymbolizer (*class in angr.state_plugins.symbolizer*), 420
SimSyscallLibrary (*class in angr.procedures.definitions*), 654
SimSystemPosix (*class in angr.state_plugins*), 320
SimSystemPosix (*class in angr.state_plugins.posix*), 350
SimTemporaryVariable (*class in angr.sim_variable*), 682
SimTranslationError, 213, 1360
SimType (*class in angr.sim_type*), 686
simtype2tc() (*angr.analyses.typehoon.translator.TypeTranslator method*), 1266
SimTypeArray (*class in angr.sim_type*), 696
SimTypeBitfield (*class in angr.sim_type*), 704
SimTypeBool (*class in angr.sim_type*), 693
SimTypeBottom (*class in angr.sim_type*), 689
SimTypeChar (*class in angr.sim_type*), 692
SimTypeCollection (*class in angr.procedures.definitions*), 649
SimTypeCppFunction (*class in angr.sim_type*), 699
SimTypeDouble (*class in angr.sim_type*), 700
SimTypeEnum (*class in angr.sim_type*), 703
SimTypeFd (*class in angr.sim_type*), 693
SimTypeFixedSizeArray (*in module angr.sim_type*), 696
SimTypeFixedSizeInt (*class in angr.sim_type*), 691
SimTypeFloat (*class in angr.sim_type*), 700
SimTypeFunction (*class in angr.sim_type*), 698
SimTypeInt (*class in angr.sim_type*), 690
SimTypeInt128 (*class in angr.sim_type*), 692
SimTypeInt256 (*class in angr.sim_type*), 692
SimTypeInt512 (*class in angr.sim_type*), 692
SimTypeLength (*class in angr.sim_type*), 699
SimTypeLong (*class in angr.sim_type*), 691
SimTypeLongLong (*class in angr.sim_type*), 691
SimTypeNum (*class in angr.sim_type*), 690
SimTypeNumOffset (*class in angr.sim_type*), 707
SimTypePointer (*class in angr.sim_type*), 694
SimTypeRef (*class in angr.sim_type*), 707
SimTypeReference (*class in angr.sim_type*), 695
SimTypeReg (*class in angr.sim_type*), 689
SimTypeShort (*class in angr.sim_type*), 691
SimTypeString (*class in angr.sim_type*), 696
SimTypeTempRef (*class in angr.analyses.typehoon.translator*), 1265
SimTypeTop (*class in angr.sim_type*), 689
SimTypeWideChar (*class in angr.sim_type*), 692
SimTypeWString (*class in angr.sim_type*), 697
SimUCManager (*class in angr.state_plugins*), 324
SimUCManager (*class in angr.state_plugins.uc_manager*), 387
SimUCManagerAllocationError, 213, 1360
SimUCManagerError, 213, 1360
simulation_manager() (*angr.factory.AngrObjectFactory method*), 230
SimulationManager (*class in angr*), 214
SimulationManager (*class in angr.sim_manager*), 544
SimulationManagerError, 221, 1355
SimUnicornError, 213, 1360
SimUnicornSymbolic, 213, 1360
SimUnicornUnsupport, 213, 1360
SimUninitializedAccessError, 213, 1359
SimUnion (*class in angr.sim_type*), 702
SimUnionValue (*class in angr.sim_type*), 703
SimUnsatError, 213, 1359
SimUnsupportedError, 214, 1359
SimUserland (*class in angr.simos*), 1330
SimUserland (*class in angr.simos.userland*), 1337

SimValueError, 214, 1359
SimVariable (*class in angr.sim_variable*), 680
SimWindows (*class in angr.simos*), 1331
SimWindows (*class in angr.simos.windows*), 1338
SimZeroDivisionException, 214, 1361
single_valued() (*angr.state_plugins.SimSolver method*), 296
single_valued() (*angr.state_plugins.solver.SimSolver method*), 370
SingleNodeGraphVisitor (*class in angr.analyses.forward_analysis*), 934
SingleNodeGraphVisitor (*class in angr.analyses.forward_analysis.visitors*), 940
SingleNodeGraphVisitor (*class in angr.analyses.forward_analysis.visitors.single_node_graph*), 945
SInt16 (*class in angr.analyses.typehoon.typeconsts*), 1276
SInt32 (*class in angr.analyses.typehoon.typeconsts*), 1276
SInt64 (*class in angr.analyses.typehoon.typeconsts*), 1276
SInt8 (*class in angr.analyses.typehoon.typeconsts*), 1275
size (*angr.ailment.Expression property*), 1369
size (*angr.ailment.expression.Array property*), 1395
size (*angr.ailment.expression.ComboRegister property*), 1378
size (*angr.ailment.expression.Expression property*), 1376
size (*angr.ailment.expression.FunctionLikeMacro property*), 1396
size (*angr.ailment.expression.RustEnum property*), 1395
size (*angr.ailment.expression.StringLiteral property*), 1394
size (*angr.ailment.expression.Struct property*), 1395
size (*angr.ailment.statement.CAS property*), 1405
size (*angr.ailment.statement.SideEffectStatement property*), 1402
size (*angr.ailment.statement.Store attribute*), 1399
size (*angr.analyses.propagator.values.Top attribute*), 1175
size (*angr.analyses.propagator.vex_vars.VEXMemVar attribute*), 1176
size (*angr.analyses.propagator.vex_vars.VEXReg attribute*), 1176
size (*angr.analyses.reaching_definitions.Atom attribute*), 1179
size (*angr.analyses.reaching_definitions.Definition property*), 1182
size (*angr.analyses.typehoon.simple_solver.SketchNode property*), 1260
size (*angr.analyses.typehoon.typeconsts.Array property*), 1278
SIZE (*angr.analyses.typehoon.typeconsts.BottomType attribute*), 1274
size (*angr.analyses.typehoon.typeconsts.Enum property*), 1280
SIZE (*angr.analyses.typehoon.typeconsts.Fd attribute*), 1275
SIZE (*angr.analyses.typehoon.typeconsts.Float32 attribute*), 1277
SIZE (*angr.analyses.typehoon.typeconsts.Float64 attribute*), 1277
SIZE (*angr.analyses.typehoon.typeconsts.Int1 attribute*), 1275
SIZE (*angr.analyses.typehoon.typeconsts.Int128 attribute*), 1276
SIZE (*angr.analyses.typehoon.typeconsts.Int16 attribute*), 1275
SIZE (*angr.analyses.typehoon.typeconsts.Int256 attribute*), 1276
SIZE (*angr.analyses.typehoon.typeconsts.Int32 attribute*), 1275
SIZE (*angr.analyses.typehoon.typeconsts.Int512 attribute*), 1277
SIZE (*angr.analyses.typehoon.typeconsts.Int64 attribute*), 1275
SIZE (*angr.analyses.typehoon.typeconsts.Int8 attribute*), 1275
size (*angr.analyses.typehoon.typeconsts.IntVar property*), 1277
size (*angr.analyses.typehoon.typeconsts.RustEnum property*), 1279
size (*angr.analyses.typehoon.typeconsts.Struct property*), 1279
SIZE (*angr.analyses.typehoon.typeconsts.TopType attribute*), 1274
SIZE (*angr.analyses.typehoon.typeconsts.TypeConstant attribute*), 1274
size (*angr.analyses.typehoon.typeconsts.TypeConstant property*), 1274
size (*angr.Block attribute*), 160
size (*angr.block.Block attribute*), 233
size (*angr.block.CapstoneInsn property*), 232
size (*angr.block.DisassemblerInsn property*), 232
size (*angr.block.PCodeInsn property*), 233
size (*angr.block.SootBlock property*), 234
size (*angr.codenode.CodeNode attribute*), 1323
size (*angr.engines.pcode.lifter.IRSB property*), 609
size (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 606
size (*angr.keyed_region.RegionObject attribute*), 868
size (*angr.keyed_region.StoredObject attribute*), 868
size (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
size (*angr.knowledge_plugins.cfg.CFGNode property*), 750

size (*angr.knowledge_plugins.cfg.indirect_jump.JumptableInfo attribute*), 770
size (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 765
size (*angr.knowledge_plugins.cfg.MemoryData attribute*), 754
size (*angr.knowledge_plugins.Function property*), 720
size (*angr.knowledge_plugins.functions.Function property*), 780
size (*angr.knowledge_plugins.functions.function.Function property*), 805
size (*angr.knowledge_plugins.key_definitions.atoms.Atom attribute*), 836
size (*angr.knowledge_plugins.key_definitions.Definition property*), 823
size (*angr.knowledge_plugins.key_definitions.definition.Definition property*), 844
size (*angr.procedures.stubs.format_parser.FormatSpecifier property*), 647
size (*angr.sim_type.SimCppClass property*), 706
size (*angr.sim_type.SimStruct property*), 702
size (*angr.sim_type.SimType property*), 686
size (*angr.sim_type.SimTypeArray property*), 696
size (*angr.sim_type.SimTypeBitfield property*), 704
size (*angr.sim_type.SimTypeDouble property*), 701
size (*angr.sim_type.SimTypeEnum property*), 703
size (*angr.sim_type.SimTypeFd property*), 694
size (*angr.sim_type.SimTypeFixedSizeInt property*), 691
size (*angr.sim_type.SimTypeFloat property*), 700
size (*angr.sim_type.SimTypeFunction property*), 699
size (*angr.sim_type.SimTypeInt property*), 691
size (*angr.sim_type.SimTypeLength property*), 700
size (*angr.sim_type.SimTypeNum property*), 690
size (*angr.sim_type.SimTypePointer property*), 695
size (*angr.sim_type.SimTypeReference property*), 695
size (*angr.sim_type.SimTypeString property*), 697
size (*angr.sim_type.SimTypeWString property*), 698
size (*angr.sim_type.SimUnion property*), 702
size (*angr.sim_type.TypeRef property*), 688
size (*angr.sim_variable.SimVariable attribute*), 681
size (*angr.SimFile property*), 179
size (*angr.SimFileBase property*), 182
size (*angr.SimPackets property*), 198
size (*angr.state_plugins.unicorn_engine.RegisterValue attribute*), 393
size (*angr.storage.file.SimFile property*), 434
size (*angr.storage.file.SimFileBase property*), 433
size (*angr.storage.file.SimPackets property*), 438
size (*angr.storage.file.SimPacketsSlots property*), 449
size (*angr.storage.SimFile property*), 423
size() (*angr.calling_conventions.AllocHelper method*), 658
size() (*angr.SimFileDescriptor method*), 183
size() (*angr.SimFileDescriptorDuplex method*), 186
size() (*angr.storage.file.SimFileDescriptor method*), 444
size() (*angr.storage.file.SimFileDescriptorBase method*), 443
size() (*angr.storage.file.SimFileDescriptorDuplex method*), 447
size() (*angr.storage.memory_object.SimMemoryObject method*), 451
size() (*angr.storage.SimMemoryObject method*), 425
SizeConcretizationMixin (*class in angr.storage.memory_mixins*), 483
SizeConcretizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*), 494
SizeNormalizationMixin (*class in angr.storage.memory_mixins*), 484
SizeNormalizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*), 494
Sketch (*class in angr.analyses.typehoon.simple_solver*), 1260
SketchNode (*class in angr.analyses.typehoon.simple_solver*), 1259
SketchNodeBase (*class in angr.analyses.typehoon.simple_solver*), 1259
skip_stmts (*angr.engines.pcode.lifter.Lifter attribute*), 611
skip_stmts (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
SkipConversionNotice, 1373
slice (*angr.Blade property*), 160
slice (*angr.blade.Blade property*), 1320
slice_callgraph() (*in module angr.analyses.cfg_slice_to_sink*), 1240
slice_callgraph() (*in module angr.analyses.cfg_slice_to_sink.graph*), 1242
slice_cfg_graph() (*in module angr.analyses.cfg_slice_to_sink*), 1240
slice_cfg_graph() (*in module angr.analyses.cfg_slice_to_sink.graph*), 1242
slice_function_graph() (*in module angr.analyses.cfg_slice_to_sink*), 1240
slice_function_graph() (*in module angr.analyses.cfg_slice_to_sink.graph*), 1242
slice_graph() (*angr.analyses.decompiler.region_identifier.RegionIdentifier static method*), 1108
slice_graph() (*angr.analyses.decompiler.RegionIdentifier static method*), 1050
Slicecutor (*class in angr.exploration_techniques*), 562
Slicecutor (*class in angr.exploration_techniques.slicecutor*), 578
SLivenessAnalysis (*class in angr.analyses*), 919
SlottedMemoryMixin (*class in angr.storage.memory_mixins*), 484

SlottedMemoryMixin (*class in angr.storage.memory_mixins.slotted_memory*), 539
SmartFindMixin (*class in angr.storage.memory_mixins*), 485
SmartFindMixin (*class in angr.storage.memory_mixins.smart_find_mixin*), 490
SMod() (*angr.state_plugins.sim_action_object.SimActionObject method*), 641
SMod() (*angr.state_plugins.SimActionObject method*), 265
snippet() (*angr.factory.AngrObjectFactory method*), 228
solution() (*angr.state_plugins.SimSolver method*), 291
solution() (*angr.state_plugins.solver.SimSolver method*), 365
solve() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 1263
solver (*angr.analyses.typehoon.simple_solver.Sketch attribute*), 1260
solver (*angr.sim_state.SimState attribute*), 239
solver (*angr.SimState attribute*), 208
soot (*angr.block.SootBlock property*), 234
soot_block (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 767
soot_block (*angr.knowledge_plugins.cfg.CFGNode attribute*), 750
SootBlock (*class in angr.block*), 234
SootBlockNode (*class in angr.codenode*), 1323
SootClassHierarchy (*class in angr.analyses*), 922
SootClassHierarchy (*class in angr.analyses.soot_class_hierarchy*), 955
SootClassHierarchyError, 955
SootExpression (*class in angr.analyses.disassembly*), 1294
SootExpressionInvoke (*class in angr.analyses.disassembly*), 1294
SootExpressionStaticFieldRef (*class in angr.analyses.disassembly*), 1294
SootExpressionTarget (*class in angr.analyses.disassembly*), 1294
SootFunction (*class in angr.knowledge_plugins.functions.soot_function*), 810
SootMixin (*class in angr.engines*), 592
SootMixin (*class in angr.engines.soot*), 601
SootMixin (*class in angr.engines.soot.engine*), 602
SootStatement (*class in angr.analyses.disassembly*), 1295
sort (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 765
sort (*angr.knowledge_plugins.cfg.MemoryData attribute*), 754
sort (*angr.sim_type.SimTypeDouble attribute*), 701
sort (*angr.sim_type.SimTypeFloat attribute*), 700
sort() (*angr.exploration_techniques.spiller.PickledStatesBase method*), 571
sort() (*angr.exploration_techniques.spiller.PickledStatesDb method*), 572
sort() (*angr.exploration_techniques.spiller.PickledStatesList method*), 572
sort_key (*angr.ailment.Block property*), 1368
sort_key (*angr.ailment.block.Block property*), 1407
sort_local_vars() (*angr.analyses.decompiler.structured_codegen.c.CFu static method*), 1132
sort_nodes() (*angr.analyses.forward_analysis.CallGraphVisitor method*), 931
sort_nodes() (*angr.analyses.forward_analysis.FunctionGraphVisitor method*), 933
sort_nodes() (*angr.analyses.forward_analysis.LoopVisitor method*), 934
sort_nodes() (*angr.analyses.forward_analysis.SingleNodeGraphVisitor method*), 935
sort_nodes() (*angr.analyses.forward_analysis.visitors.call_graph.CallGr method*), 941
sort_nodes() (*angr.analyses.forward_analysis.visitors.CallGraphVisitor method*), 938
sort_nodes() (*angr.analyses.forward_analysis.visitors.function_graph.Fu method*), 942
sort_nodes() (*angr.analyses.forward_analysis.visitors.FunctionGraphVis method*), 939
sort_nodes() (*angr.analyses.forward_analysis.visitors.graph.GraphVisito method*), 943
sort_nodes() (*angr.analyses.forward_analysis.visitors.loop.LoopVisitor method*), 945
sort_nodes() (*angr.analyses.forward_analysis.visitors.LoopVisitor method*), 939
sort_nodes() (*angr.analyses.forward_analysis.visitors.single_node_graph method*), 946
sort_nodes() (*angr.analyses.forward_analysis.visitors.SingleNodeGraph method*), 940
SortedDict (*class in angr.knowledge_plugins.variables.variable_manager* 812
sources (*angr.analyses.reaching_definitions.function_handler.FunctionEff attribute*), 1222
sources_defns (*angr.analyses.reaching_definitions.function_handler.Fun attribute*), 1222
sp_delta (*angr.knowledge_plugins.Function attribute*), 717
sp_delta (*angr.knowledge_plugins.functions.Function attribute*), 777
sp_delta (*angr.knowledge_plugins.functions.function.Function attribute*), 802
sp_delta (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 810
sp_offset() (*angr.engines.light.engine.SimEngineLight static method*), 1172

sp_offset() (*angr.engines.light.SimEngineLight static method*), 1168
spec_type (*angr.procedures.stubs.format_parser.FormatSpecifier property*), 647
Special (*angr.analyses.disassembly.SootExpressionInvoke attribute*), 1295
SPECIAL_THUNKS (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 990
SPECIAL_THUNKS (*angr.analyses.cfg.CFGFast attribute*), 967
SPECIAL_THUNKS (*angr.analyses.CFGFast attribute*), 890
SpecialFillerMixin (*class in angr.storage.memory_mixins*), 485
SpecialFillerMixin (*class in angr.storage.memory_mixins.default_filler_mixin*), 491
spilled_count (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict property*), 794
spilled_function_count (*angr.knowledge_plugins.FunctionManager property*), 729
spilled_function_count (*angr.knowledge_plugins.functions.function_manager.FunctionManager property*), 799
spilled_function_count (*angr.knowledge_plugins.functions.FunctionManager property*), 790
Spiller (*class in angr.exploration_techniques*), 563
Spiller (*class in angr.exploration_techniques.spiller*), 573
SpillingFunctionDict (*class in angr.knowledge_plugins.functions.function_manager*), 792
split() (*angr.keyed_region.RegionObject method*), 868
split() (*angr.sim_manager.SimulationManager method*), 550
split() (*angr.SimulationManager method*), 220
split_arm_op_string() (*angr.analyses.disassembly.Instruction static method*), 1294
split_op_string() (*angr.analyses.disassembly.Instruction static method*), 1294
split_operands() (*in module angr.analyses.reassembler*), 1297
SpOffset (*class in angr.engines.light*), 1170
SpOffset (*class in angr.engines.light.data*), 1167
SPropagatorAnalysis (*class in angr.analyses*), 919
src (*angr.ailment.Assignment attribute*), 1366
src (*angr.ailment.expression.Let attribute*), 1396
src (*angr.ailment.statement.Assignment attribute*), 1398
src (*angr.ailment.statement.WeakAssignment attribute*), 1398
src_and_vvars (*angr.ailment.expression.Phi attribute*), 1380
src_block_id (*angr.analyses.vfg.PendingJob attribute*), 1286
src_func_addr (*angr.analyses.cfg.cfg_fast.FunctionEdge attribute*), 985
src_ins_addr (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
src_ins_addr (*angr.analyses.vfg.PendingJob attribute*), 1286
src_node (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
src_node (*angr.analyses.cfg.cfg_fast.FunctionCallEdge attribute*), 986
src_node (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge attribute*), 986
src_node (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute*), 986
src_stmt_idx (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
src_stmt_idx (*angr.analyses.vfg.PendingJob attribute*), 1286
src_type (*angr.analyses.decompiler.structured_codegen.c.CTypeCast attribute*), 1141
SReachingDefinitionsAnalysis (*class in angr.analyses*), 921
SSA_LEVEL0_TRANSFORMATION (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1057
SSA_LEVEL1_TRANSFORMATION (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1057
Ssailification (*class in angr.analyses.decompiler*), 1051
sse_extend() (*angr.calling_conventions.SimRegArg method*), 660
st_atime (*angr.state_plugins.filesystem.Stat attribute*), 355
st_atime (*angr.state_plugins.Stat attribute*), 325
st_atimensec (*angr.state_plugins.filesystem.Stat attribute*), 355
st_atimensec (*angr.state_plugins.Stat attribute*), 326
st_blksize (*angr.state_plugins.filesystem.Stat attribute*), 355
st_blksize (*angr.state_plugins.Stat attribute*), 326
st_blocks (*angr.state_plugins.filesystem.Stat attribute*), 355
st_blocks (*angr.state_plugins.Stat attribute*), 326
st_ctime (*angr.state_plugins.filesystem.Stat attribute*), 355
st_ctime (*angr.state_plugins.Stat attribute*), 326
st_ctimensec (*angr.state_plugins.filesystem.Stat attribute*), 355
st_ctimensec (*angr.state_plugins.Stat attribute*), 326
st_dev (*angr.state_plugins.filesystem.Stat attribute*), 355

st_dev (*angr.state_plugins.Stat attribute*), 326
st_gid (*angr.state_plugins.filesystem.Stat attribute*), 355
st_gid (*angr.state_plugins.Stat attribute*), 326
st_ino (*angr.state_plugins.filesystem.Stat attribute*), 355
st_ino (*angr.state_plugins.Stat attribute*), 326
st_mode (*angr.state_plugins.filesystem.Stat attribute*), 355
st_mode (*angr.state_plugins.Stat attribute*), 326
st_mtime (*angr.state_plugins.filesystem.Stat attribute*), 355
st_mtime (*angr.state_plugins.Stat attribute*), 326
st_mtimensec (*angr.state_plugins.filesystem.Stat attribute*), 355
st_mtimensec (*angr.state_plugins.Stat attribute*), 326
st_nlink (*angr.state_plugins.filesystem.Stat attribute*), 355
st_nlink (*angr.state_plugins.Stat attribute*), 326
st_rdev (*angr.state_plugins.filesystem.Stat attribute*), 355
st_rdev (*angr.state_plugins.Stat attribute*), 326
st_size (*angr.state_plugins.filesystem.Stat attribute*), 355
st_size (*angr.state_plugins.Stat attribute*), 326
st_uid (*angr.state_plugins.filesystem.Stat attribute*), 355
st_uid (*angr.state_plugins.Stat attribute*), 326
stable_hash() (*in module angr.ailment.utils*), 1408
StableVarExprHasher (*class in angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier*), 1100
STACK (*angr.ailment.expression.VirtualVariableCategory attribute*), 1379
stack (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
stack (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 1233
stack (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 1206
stack (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 848
stack (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 824
stack (*angr.storage.memory_mixins.JavaVmMemoryMixin property*), 464
stack_actions (*angr.state_plugins.history.SimStateHistory property*), 378
stack_actions (*angr.state_plugins.SimStateHistory property*), 303
stack_addr_from_offset() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 1250
stack_address() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1190
stack_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 1232
stack_address() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 1205
stack_address() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 1249
stack_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 849
stack_address() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 825
STACK_ALIGNMENT (*angr.calling_conventions.SimCC attribute*), 664
STACK_ALIGNMENT (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 670
STACK_ALIGNMENT (*angr.calling_conventions.SimCCRISCV64 attribute*), 674
STACK_ALIGNMENT (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 671
STACK_ALIGNMENT (*angr.SimCC attribute*), 174
stack_base (*angr.storage.memory_mixins.regioned_memory.region_data.RegionedMemoryMixin property*), 534
stack_id() (*angr.storage.memory_mixins.regioned_memory.regioned_memory.RegionedMemoryMixin method*), 532
stack_id() (*angr.storage.memory_mixins.regioned_memory.RegionedMemoryMixin method*), 530
stack_id() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 482
stack_items (*angr.analyses.decompiler.clinic.Clinic attribute*), 1061
stack_items (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute*), 1095
stack_loc() (*angr.calling_conventions.AllocHelper class method*), 658
stack_offset (*angr.ailment.expression.VirtualVariable property*), 1380
stack_offset (*angr.calling_conventions.SimStackArg attribute*), 661
stack_offset (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 840
stack_offset (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 842
stack_offset_to_stack_addr() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 1192
stack_offset_to_stack_addr() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 850
stack_offset_to_stack_addr() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 827
stack_offset_typevars (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 1065
stack_offset_typevars (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase attribute*), 1249

stack_offsets (*angr.slicer.SimLightState attribute*), 1321
stack_pointer_as_atom() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler static method*), 1229
stack_pointer_as_atom() (*angr.analyses.reaching_definitions.FunctionHandler static method*), 1188
stack_pop() (*angr.sim_state.SimState method*), 242
stack_pop() (*angr.SimState method*), 210
stack_push() (*angr.sim_state.SimState method*), 242
stack_push() (*angr.SimState method*), 210
stack_read() (*angr.sim_state.SimState method*), 242
stack_read() (*angr.SimState method*), 210
stack_region (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase attribute*), 1249
stack_region (*angr.knowledge_plugins.variables.variable_manager.LiveVariables attribute*), 813
stack_space() (*angr.calling_conventions.SimCC method*), 665
stack_space() (*angr.SimCC method*), 175
stack_suffix() (*angr.state_plugins.CallStack method*), 250
stack_suffix() (*angr.state_plugins.callstack.CallStack method*), 374
stack_suffix_to_string() (*angr.state_plugins.CallStack static method*), 250
stack_suffix_to_string() (*angr.state_plugins.callstack.CallStack static method*), 373
stack_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
stack_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 1233
stack_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 1206
stack_uses (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 848
stack_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 825
StackAllocationMixin (*class in angr.storage.memory_mixins*), 486
StackAllocationMixin (*class in angr.storage.memory_mixins.paged_memory.stack_allocation_mixin*), 509
STACKARG_SP_BUFF (*angr.calling_conventions.SimCC attribute*), 664
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 669
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCN64 attribute*), 676
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCO32 attribute*), 675
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCPowerPC attribute*), 676
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCPowerPC64 attribute*), 677
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCS390X attribute*), 679
STACKARG_SP_BUFF (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 637
STACKARG_SP_BUFF (*angr.SimCC attribute*), 174
STACKARG_SP_DIFF (*angr.calling_conventions.SimCC attribute*), 664
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCCdecl attribute*), 668
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 669
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 669
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 671
STACKARG_SP_DIFF (*angr.engines.pcode.cc.SimCCM68k attribute*), 636
STACKARG_SP_DIFF (*angr.SimCC attribute*), 174
StackBaseOffset (*class in angr.ailment.expression*), 1390
StackCanarySimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1088
StackCanarySimplifier (*class in angr.analyses.decompiler.optimization_passes.stack_canary_simp*), 1097
StackLocationAnnotation (*class in angr.analyses.variable_recovery.annotations*), 1246
StackPointerTracker (*class in angr.analyses*), 922
StackPointerTracker (*class in angr.analyses.stack_pointer_tracker*), 1245
StackPointerTrackerState (*class in angr.analyses.stack_pointer_tracker*), 1244
stackvar_max_sizes (*angr.analyses.decompiler.decompilation_cache.De attribute*), 1065
stage (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 992
stage (*angr.analyses.cfg.CFGFast attribute*), 969
STAGE (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simpl attribute*), 1097
STAGE (*angr.analyses.decompiler.optimization_passes.BasePointerSaveSim attribute*), 1072
STAGE (*angr.analyses.decompiler.optimization_passes.CallStatementRewrit attribute*), 1072
STAGE (*angr.analyses.decompiler.optimization_passes.CodeMotionOptimiz attribute*), 1073
STAGE (*angr.analyses.decompiler.optimization_passes.ConditionConstantP attribute*), 1074
STAGE (*angr.analyses.decompiler.optimization_passes.const_derefs.Constar*

attribute), 1092
STAGE (*angr.analyses.decompiler.optimization_passes.ConstantDerefRemovalSimplifier attribute*), 1075
STAGE (*angr.analyses.decompiler.optimization_passes.ConstantMultiplication attribute*), 1074
STAGE (*angr.analyses.decompiler.optimization_passes.CrossJumpReverter attribute*), 1075
STAGE (*angr.analyses.decompiler.optimization_passes.DeadassignmentRemover attribute*), 1076
STAGE (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifier attribute*), 1098
STAGE (*angr.analyses.decompiler.optimization_passes.DivSimplifier attribute*), 1076
STAGE (*angr.analyses.decompiler.optimization_passes.EagerReturnsSimplifier attribute*), 1078
STAGE (*angr.analyses.decompiler.optimization_passes.expr_simplifiers.ExprOpSubcomp attribute*), 1103
STAGE (*angr.analyses.decompiler.optimization_passes.ExprSubcomp attribute*), 1079
STAGE (*angr.analyses.decompiler.optimization_passes.FlipBoolEncmp attribute*), 1079
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors attribute*), 1080
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors attribute*), 1080
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors attribute*), 1080
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors.InlineAccessors attribute*), 1081
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors.InlineAccessors attribute*), 1081
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors.InlineAccessors attribute*), 1081
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors.InlineAccessors attribute*), 1081
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors.InlineAccessors attribute*), 1082
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors.InlineAccessors attribute*), 1082
STAGE (*angr.analyses.decompiler.optimization_passes.InlineAccessors.InlineAccessors.InlineAccessors attribute*), 1082
STAGE (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 1099
STAGE (*angr.analyses.decompiler.optimization_passes.ITEExprConverter attribute*), 1079
STAGE (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 1080
STAGE (*angr.analyses.decompiler.optimization_passes.MipsGpSetupSimplifier attribute*), 1083
STAGE (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 1101
STAGE (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 1084
STAGE (*angr.analyses.decompiler.optimization_passes.optimization_pass.BaseOptimizationPass attribute*), 1093
STAGE (*angr.analyses.decompiler.optimization_passes.optimization_pass.St attribute*), 1097
STAGE (*angr.analyses.decompiler.optimization_passes.register_save_area_s attribute*), 1103
STAGE (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSim attribute*), 1085
STAGE (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSim attribute*), 1085
STAGE (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simpl attribute*), 1103
STAGE (*angr.analyses.decompiler.optimization_passes.RetAddrSaveSimplifi attribute*), 1085
STAGE (*angr.analyses.decompiler.optimization_passes.ReturnDeduplicator attribute*), 1086
STAGE (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorHig attribute*), 1086
STAGE (*angr.analyses.decompiler.optimization_passes.stack_canary_simpli attribute*), 1097
STAGE (*angr.analyses.decompiler.optimization_passes.StackCanarySimplifi attribute*), 1088
STAGE (*angr.analyses.decompiler.optimization_passes.StaticVVarRewriter attribute*), 1089
STAGE (*angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseD attribute*), 1089
STAGE (*angr.analyses.decompiler.optimization_passes.SwitchReusedEntryR attribute*), 1090
STAGE (*angr.analyses.decompiler.optimization_passes.TagSlicer attribute*), 1090
STAGE (*angr.analyses.decompiler.optimization_passes.WinStackCanarySim attribute*), 1090
STAGE (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simp attribute*), 1104
STAGE (*angr.analyses.decompiler.optimization_passes.X86GccGetPcSimpli attribute*), 1091
start (*angr.analyses.decompiler.structured_codegen.base.PositionMappin attribute*), 1128
start (*angr.analyses.decompiler.structured_codegen.PositionMappingElem attribute*), 1125
start (*angr.keyed_region.RegionObject attribute*), 868
start (*angr.keyed_region.StoredObject attribute*), 868
start() (*angr.rlt.RLTExpCollapser.Worker method*), 1363
start() (*angr.state_plugins.Unicorn method*), 328
start() (*angr.state_plugins.unicorn_engine.Unicorn method*), 398
start_addr (*angr.knowledge_plugins.Function attribute*), 717
start_addr (*angr.knowledge_plugins.functions.Function attribute*), 777
start_addr (*angr.knowledge_plugins.functions.function.Function attribute*), 802
startpoint (*angr.knowledge_plugins.functions.soot_function.SootFunctio attribute*), 810
stash (*angr.storage.spiller_db.PickledState*

attribute), 574
stash() (*angr.sim_manager.SimulationManager method*), 549
stash() (*angr.SimulationManager method*), 218
stashed (*angr.sim_manager.SimulationManager attribute*), 545
stashed (*angr.SimulationManager attribute*), 215
stashes (*angr.sim_manager.SimulationManager property*), 546
stashes (*angr.SimulationManager property*), 215
Stat (*class in angr.state_plugins*), 325
Stat (*class in angr.state_plugins.filesystem*), 354
state (*angr.analyses.vfg.PendingJob attribute*), 1286
state (*angr.analyses.vfg.VFGNode attribute*), 1288
state (*angr.Emulator property*), 161
state (*angr.engines.engine.SimEngine attribute*), 594
state (*angr.engines.light.engine.SimEngineLight attribute*), 1171
state (*angr.engines.light.SimEngineLight attribute*), 1168
state (*angr.engines.SimEngine attribute*), 589
state (*angr.procedures.stubs.format_parser.FormatString property*), 646
state (*angr.sim_manager.ErrorRecord attribute*), 551
state (*angr.sim_procedure.SimProcedure attribute*), 644
state (*angr.SimProcedure attribute*), 204
state (*angr.SimStatePlugin attribute*), 211
state (*angr.state_plugins.plugin.SimStatePlugin attribute*), 329
state (*angr.state_plugins.SimMemView attribute*), 284
state (*angr.state_plugins.SimStatePlugin attribute*), 314
state (*angr.state_plugins.view.SimMemView attribute*), 430
state_blank() (*angr.SimOS method*), 195
state_blank() (*angr.simos.cgc.SimCGC method*), 1337
state_blank() (*angr.simos.javavm.SimJavaVM method*), 1340
state_blank() (*angr.simos.linux.SimLinux method*), 1336
state_blank() (*angr.simos.SimCGC method*), 1325
state_blank() (*angr.simos.SimJavaVM method*), 1325
state_blank() (*angr.simos.SimLinux method*), 1327
state_blank() (*angr.simos.SimOS method*), 1328
state_blank() (*angr.simos.simos.SimOS method*), 1333
state_blank() (*angr.simos.SimWindows method*), 1332
state_blank() (*angr.simos.windows.SimWindows method*), 1338
state_call() (*angr.SimOS method*), 196
state_call() (*angr.simos.javavm.SimJavaVM method*), 1340
state_call() (*angr.simos.SimJavaVM method*), 1325
state_call() (*angr.simos.SimOS method*), 1328
state_call() (*angr.simos.simos.SimOS method*), 1334
state_entry() (*angr.SimOS method*), 196
state_entry() (*angr.simos.cgc.SimCGC method*), 1337
state_entry() (*angr.simos.javavm.SimJavaVM method*), 1340
state_entry() (*angr.simos.linux.SimLinux method*), 1336
state_entry() (*angr.simos.SimCGC method*), 1325
state_entry() (*angr.simos.SimJavaVM method*), 1325
state_entry() (*angr.simos.SimLinux method*), 1327
state_entry() (*angr.simos.SimOS method*), 1328
state_entry() (*angr.simos.simos.SimOS method*), 1334
state_entry() (*angr.simos.SimWindows method*), 1332
state_entry() (*angr.simos.windows.SimWindows method*), 1338
state_full_init() (*angr.SimOS method*), 196
state_full_init() (*angr.simos.linux.SimLinux method*), 1336
state_full_init() (*angr.simos.SimLinux method*), 1327
state_full_init() (*angr.simos.SimOS method*), 1328
state_full_init() (*angr.simos.simos.SimOS method*), 1334
state_priority() (*angr.exploration_techniques.Spiller static method*), 564
state_priority() (*angr.exploration_techniques.spiller.Spiller static method*), 573
StateHierarchy (*class in angr*), 221
StateHierarchy (*class in angr.state_hierarchy*), 551
STATEMENT (*angr.state_plugins.inspect.EventType attribute*), 331
statement (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
statement (*angr.state_plugins.InspectAttrs attribute*), 254
Statement (*class in angr.ailment*), 1371
Statement (*class in angr.ailment.statement*), 1397
statement_id (*angr.storage.memory_mixins.regioned_memory.region_me property*), 536
statement_location() (*angr.analyses.decompiler.region_simplifiers.expr_folding.Expres method*), 1110
StatementLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1110
statements (*angr.ailment.Block attribute*), 1368
statements (*angr.ailment.block.Block attribute*), 1406
statements (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 1132
statements (*angr.analyses.decompiler.structured_codegen.c.CStatements attribute*), 1133
statements (*angr.engines.pcode.lifter.IRSB property*),

609
StateOption (*class in angr.sim_state_options*), 242
states (*angr.knowledge_plugins.propagations.PropagationModel attribute*), 775
Static (*angr.analyses.disassembly.SootExpressionInvoke attribute*), 1295
STATIC (*angr.simos.windows.SecurityCookieInit attribute*), 1338
static_exits() (*angr.sim_procedure.SimProcedure method*), 645
static_exits() (*angr.SimProcedure method*), 204
StaticFindMixin (*class in angr.storage.memory_mixins*), 486
StaticFindMixin (*class in angr.storage.memory_mixins.regioned_memory*), 530
StaticFindMixin (*class in angr.storage.memory_mixins.regioned_memory.static_find_mixin*), 535
StaticHooker (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
StaticHooker (*class in angr.analyses*), 923
StaticHooker (*class in angr.analyses.static_hooker*), 1305
StaticObjectFinder (*class in angr.analyses*), 923
StaticObjectFinder (*class in angr.analyses.find_objects_static*), 1292
StaticVVarRewriter (*class in angr.analyses.decompiler.optimization_passes*), 1088
step() (*angr.distributed.worker.BadStatesDropper method*), 1363
step() (*angr.distributed.worker.ExplorationStatusNotifier method*), 1363
step() (*angr.exploration_techniques.DFS method*), 552
step() (*angr.exploration_techniques.dfs.DFS method*), 569
step() (*angr.exploration_techniques.Director method*), 554
step() (*angr.exploration_techniques.director.Director method*), 581
step() (*angr.exploration_techniques.driller_core.DrillerCore method*), 577
step() (*angr.exploration_techniques.DrillerCore method*), 555
step() (*angr.exploration_techniques.ExplorationTechnique method*), 556
step() (*angr.exploration_techniques.Explorer method*), 557
step() (*angr.exploration_techniques.explorer.Explorer method*), 570
step() (*angr.exploration_techniques.LengthLimiter method*), 558
step() (*angr.exploration_techniques.lengthlimiter.LengthLimiter method*), 571
step() (*angr.exploration_techniques.manual_mergepoint.ManualMergepo method*), 571
step() (*angr.exploration_techniques.ManualMergepoint method*), 561
step() (*angr.exploration_techniques.memory_watcher.MemoryWatcher method*), 586
step() (*angr.exploration_techniques.MemoryWatcher method*), 561
step() (*angr.exploration_techniques.Spiller method*), 563
step() (*angr.exploration_techniques.spiller.Spiller method*), 573
step() (*angr.exploration_techniques.stochastic.StochasticSearch method*), 584
step() (*angr.exploration_techniques.StochasticSearch method*), 564
step() (*angr.exploration_techniques.StubStasher method*), 564
step() (*angr.exploration_techniques.Suggestions method*), 564
step() (*angr.exploration_techniques.suggestions.Suggestions method*), 586
step() (*angr.exploration_techniques.Threading method*), 565
step() (*angr.exploration_techniques.threading.Threading method*), 574
step() (*angr.exploration_techniques.Timeout method*), 565
step() (*angr.exploration_techniques.timeout.Timeout method*), 569
step() (*angr.exploration_techniques.Tracer method*), 567
step() (*angr.exploration_techniques.tracer.Tracer method*), 577
step() (*angr.exploration_techniques.unique.UniqueSearch method*), 584
step() (*angr.exploration_techniques.UniqueSearch method*), 568
step() (*angr.ExplorationTechnique method*), 163
step() (*angr.sim_manager.SimulationManager method*), 547
step() (*angr.sim_state.SimState method*), 241
step() (*angr.SimState method*), 209
step() (*angr.SimulationManager method*), 217
step_back() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 1213
step_state() (*angr.exploration_techniques.ExplorationTechnique method*), 556
step_state() (*angr.exploration_techniques.Slicecutor method*), 562
step_state() (*angr.exploration_techniques.slicecutor.Slicecutor method*), 578
step_state() (*angr.exploration_techniques.Tracer

method), 567
step_state() (*angr.exploration_techniques.tracer.Tracer method*), 577
step_state() (*angr.exploration_techniques.Veritesting method*), 568
step_state() (*angr.exploration_techniques.veritesting.Veritesting method*), 574
step_state() (*angr.ExplorationTechnique method*), 163
step_state() (*angr.sim_manager.SimulationManager method*), 548
step_state() (*angr.SimulationManager method*), 218
stmt (*angr.analyses.decompiler.structured_codegen.c.CUnsupportedStatement attribute*), 1137
stmt_can_move_to() (*angr.analyses.decompiler.optimization_passes.expr_duplicator.ExpressionDuplicator method*), 1077
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationMultiStmtBase attribute*), 1105
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase attribute*), 1104
stmt_comments (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator attribute*), 1129
stmt_comments (*angr.angrdb.models.DbStructuredCode attribute*), 1024
stmt_handlers (*angr.ailment.AILBlockWalker attribute*), 1365
stmt_idx (*angr.analyses.cfg.cfg_fast.FunctionEdge attribute*), 985
stmt_idx (*angr.analyses.decompiler.clinic.DataRefDesc attribute*), 1057
stmt_idx (*angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter attribute*), 1110
stmt_idx (*angr.analyses.decompiler.region_simplifiers.expr_folding.SimplifierAILCallback attribute*), 1110
stmt_idx (*angr.analyses.disassembly.SootStatement property*), 1295
stmt_idx (*angr.code_location.AILCodeLocation attribute*), 867
stmt_idx (*angr.code_location.CodeLocation attribute*), 866
stmt_idx (*angr.engines.light.engine.SimEngineLight attribute*), 1171
stmt_idx (*angr.engines.light.SimEngineLight attribute*), 1168
stmt_idx (*angr.errors.SimError attribute*), 1358
stmt_idx (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 771
stmt_idx (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 753
stmt_idx (*angr.knowledge_plugins.xrefs.XRef attribute*), 861
stmt_idx (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 863
stmt_idx (*angr.SimError attribute*), 178
stmt_idx (*angr.state_plugins.unicorn_engine.VEXStmtDetails attribute*), 393
stmt_observe() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 1217
stmt_observe() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 1200
stmt_observe() (*angr.analyses.ReachingDefinitionsAnalysis method*), 915
stmts (*angr.ailment.expression.MultiStatementExpression attribute*), 1389
stmts (*angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression attribute*), 1143
stmts (*angr.lifter.sootmem.SootBlockNode attribute*), 1324
stmts_to_instrument (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessor attribute*), 1000
stmts_to_instrument (*angr.analyses.vfg.VFG attribute*), 1256
stmts_used (*angr.engines.vex.lifter.VEXLifter property*), 609
StochasticBeamSearch (*class in angr.exploration_techniques*), 564
StochasticSearch (*class in angr.exploration_techniques.stochastic*), 584
STOP (*class in angr.state_plugins.unicorn_engine*), 394
stop() (*angr.distributed.Server method*), 1362
stop() (*angr.distributed.server.Server method*), 1362
stop() (*angr.Server method*), 173
STOP_ERROR (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_EXECNONE (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_HLT (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
stop_message (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
STOP_NODECODE (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_NORMAL (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_NOSTART (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
stop_reason (*angr.state_plugins.unicorn_engine.StopDetails attribute*), 395
STOP_SEGFAULT (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_STOPPOINT (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_SYMBOLIC_BLOCK_EXIT_CONDITION (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_SYMBOLIC_BLOCK_EXIT_TARGET (*angr.state_plugins.unicorn_engine.STOP*

attribute), 394
STOP_SYMBOLIC_PC (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_SYMBOLIC_READ_ADDR (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_SYMBOLIC_READ_SYMBOLIC_TRACKING_DISABLED (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_SYMBOLIC_WRITE_ADDR (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_SYSCALL (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_SYSCALL_ARM (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
STOP_UNKNOWN_MEMORY_WRITE_SIZE (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
STOP_UNSUPPORTED_EXPR_GETI (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
STOP_UNSUPPORTED_EXPR_UNKNOWN (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
STOP_UNSUPPORTED_STMT_CAS (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_UNSUPPORTED_STMT_DIRTY (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
STOP_UNSUPPORTED_STMT_LLSC (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_UNSUPPORTED_STMT_LOADG (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_UNSUPPORTED_STMT_PUTI (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_UNSUPPORTED_STMT_STOREG (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_UNSUPPORTED_STMT_UNKNOWN (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
STOP_VEX_LIFT_FAILED (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_X86_CPUID (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
STOP_ZERO_DIV (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
STOP_ZEROPAGE (*angr.state_plugins.unicorn_engine.STOP attribute*), 394
StopDetails (*class in angr.state_plugins.unicorn_engine*), 395
stopped (*angr.distributed.Server property*), 1362
stopped (*angr.distributed.server.Server property*), 1363
stopped (*angr.Server property*), 173
storage (*angr.state_plugins.sim_action.SimActionData property*), 640
storage (*angr.state_plugins.SimActionData property*), 264
Store (*class in angr.ailment.statement*), 1399
Store (*class in angr.analyses.typehoon.typevars*), 1271
Store() (*angr.ailment.converter_vex.VEXStmtConverter static method*), 1375
store() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 1245
store() (*angr.sim_type.SimCppClass method*), 706
store() (*angr.sim_type.SimStruct method*), 702
store() (*angr.sim_type.SimType method*), 687
store() (*angr.sim_type.SimTypeArray method*), 696
store() (*angr.sim_type.SimTypeBitfield method*), 705
store() (*angr.sim_type.SimTypeBool method*), 693
store() (*angr.sim_type.SimTypeChar method*), 692
store() (*angr.sim_type.SimTypeEnum method*), 704
store() (*angr.sim_type.SimTypeFloat method*), 700
store() (*angr.sim_type.SimTypeNum method*), 690
store() (*angr.sim_type.SimTypeNumOffset method*), 707
store() (*angr.sim_type.SimTypeReg method*), 689
store() (*angr.sim_type.SimTypeWideChar method*), 693
store() (*angr.sim_type.SimTypeWString method*), 698
store() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 422
store() (*angr.state_plugins.light_registers.SimLightRegisters method*), 375
store() (*angr.state_plugins.SimDebugVariable method*), 267
store() (*angr.state_plugins.SimLightRegisters method*), 281
store() (*angr.state_plugins.SimMemView method*), 285
store() (*angr.state_plugins.view.SimMemView method*), 431
store() (*angr.storage.memory_mixins.actions_mixin.ActionsMixinHigh method*), 493
store() (*angr.storage.memory_mixins.actions_mixin.ActionsMixinLow method*), 493
store() (*angr.storage.memory_mixins.ActionsMixinHigh method*), 456
store() (*angr.storage.memory_mixins.ActionsMixinLow method*), 456
store() (*angr.storage.memory_mixins.address_concretization_mixin.Addr method*), 497
store() (*angr.storage.memory_mixins.AddressConcretizationMixin method*), 458

store() (*angr.storage.memory_mixins.bvv_conversion_mixin.BVVConversionMixin method*), 491
store() (*angr.storage.memory_mixins.conditional_store_mixin.ConditionalStoreMixin method*), 497
store() (*angr.storage.memory_mixins.ConditionalMixin method*), 459
store() (*angr.storage.memory_mixins.convenient_mappings_mixin.ConvenientMappingsMixin method*), 499
store() (*angr.storage.memory_mixins.ConvenientMappingsMixin method*), 459
store() (*angr.storage.memory_mixins.DataNormalizationMixin method*), 460
store() (*angr.storage.memory_mixins.dirty_addrs_mixin.DirtyAddrsMixin method*), 495
store() (*angr.storage.memory_mixins.DirtyAddrsMixin method*), 461
store() (*angr.storage.memory_mixins.HistoryTrackingMixin method*), 462
store() (*angr.storage.memory_mixins.InspectMixin method*), 463
store() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 464
store() (*angr.storage.memory_mixins.KeyValueMemoryMixin method*), 466
store() (*angr.storage.memory_mixins.ListPage method*), 467
store() (*angr.storage.memory_mixins.MemoryRegionMetaMixin method*), 472
store() (*angr.storage.memory_mixins.MVListPage method*), 469
store() (*angr.storage.memory_mixins.name_resolution_mixin.NameResolutionMixin method*), 490
store() (*angr.storage.memory_mixins.NameResolutionMixin method*), 474
store() (*angr.storage.memory_mixins.paged_memory.pages.cache_mixin.CachingMixin method*), 504
store() (*angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.HistoryTrackingMixin method*), 519
store() (*angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.HistoryTrackingMixin method*), 510
store() (*angr.storage.memory_mixins.paged_memory.pages.ultra.ultra.UltraPage method*), 521
store() (*angr.storage.memory_mixins.paged_memory.pages.ultra.UltraPage method*), 511
store() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 500
store() (*angr.storage.memory_mixins.paged_memory.pages.MVListPage method*), 513
store() (*angr.storage.memory_mixins.paged_memory.pages.cache_mixin.CachingMixin method*), 523
store() (*angr.storage.memory_mixins.paged_memory.pages.ListPage method*), 517
store() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 474
store() (*angr.storage.memory_mixins.regioned_memory.MemoryRegionM method*), 525
store() (*angr.storage.memory_mixins.regioned_memory.region_meta_mix method*), 537
store() (*angr.storage.memory_mixins.regioned_memory.regioned_memor method*), 531
store() (*angr.storage.memory_mixins.regioned_memory.RegionedMemor method*), 528
store() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 481
store() (*angr.storage.memory_mixins.simple_interface_mixin.SimpleInter method*), 493
store() (*angr.storage.memory_mixins.SimpleInterfaceMixin method*), 483
store() (*angr.storage.memory_mixins.simplification_mixin.SimplificationI method*), 498
store() (*angr.storage.memory_mixins.SimplificationMixin method*), 483
store() (*angr.storage.memory_mixins.size_resolution_mixin.SizeConcreti method*), 495
store() (*angr.storage.memory_mixins.size_resolution_mixin.SizeNormaliz method*), 494
store() (*angr.storage.memory_mixins.SizeConcretizationMixin method*), 484
store() (*angr.storage.memory_mixins.SizeNormalizationMixin method*), 484
store() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMix method*), 541
store() (*angr.storage.memory_mixins.SlottedMemoryMixin method*), 485
store() (*angr.storage.memory_mixins.UltraPage method*), 487
store() (*angr.storage.memory_mixins.underconstrained_mixin.Undercons method*), 493
store() (*angr.storage.memory_mixins.UnderconstrainedMixin method*), 489
store() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 498
store() (*angr.storage.memory_mixins.UnwrapperMixin method*), 490
store() (*angr.storage.ListPage method*), 872
store() (*angr.vaults.Vault method*), 873
store() (*angr.vaults.VaultDirShelf method*), 873
store_array_element() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 464
store_array_elements() (*angr.storage.memory_mixins.JavaVmMemoryMixin method*), 464
store_cmp() (*angr.state_plugins.scratch.SimStateScratch method*), 388
store_tmp() (*angr.state_plugins.SimStateScratch method*), 318
store_variable() (*angr.analyses.decompiler.optimization_passes.engine method*), 1101

stored_objects (*angr.keyed_region.RegionObject attribute*), 868
StoredObject (*class in angr.keyed_region*), 867
StoreG() (*angr.ailment.converter_vex.VEXStmtConverter static method*), 1375
StoreHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptables*), 1001
StoreStatementFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 1113
str_to_c_str() (*angr.analyses.decompiler.structured_codegen.base.StructuredCodeGenerator static method*), 1142
Strict (*angr.exploration_techniques.tracer.TracingMode attribute*), 575
strict_block_end (*angr.engines.pcode.lifter.Lifter attribute*), 610
strict_block_end (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
STRING (*angr.analyses.decompiler.structured_codegen.base.StructuredCodeGenerator attribute*), 1128
String (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 1309
String (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
String (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 755
string (*angr.procedures.stubs.format_parser.FormatSpecifier attribute*), 647
string (*angr.state_plugins.debug_variables.SimDebugVariable property*), 421
string (*angr.state_plugins.SimDebugVariable property*), 266
string_escape() (*in module angr.analyses.reassembler*), 1297
string_references() (*angr.knowledge_plugins.Function method*), 719
string_references() (*angr.knowledge_plugins.functions.Function method*), 780
string_references() (*angr.knowledge_plugins.functions.function.Function method*), 804
StringLiteral (*class in angr.ailment.expression*), 1394
StringProxiNode (*class in angr.analyses.proximity_graph*), 1310
STRONGREF_STATE (*angr.SimStatePlugin attribute*), 211
STRONGREF_STATE (*angr.state_plugins.history.SimStateHistory attribute*), 375
STRONGREF_STATE (*angr.state_plugins.plugin.SimStatePlugin attribute*), 329
STRONGREF_STATE (*angr.state_plugins.SimStateHistory attribute*), 301
STRONGREF_STATE (*angr.state_plugins.SimStatePlugin attribute*), 314
struct (*angr.sim_type.SimStructValue property*), 702
struct (*angr.state_plugins.SimMemView attribute*), 284
struct (*angr.state_plugins.view.SimMemView attribute*), 430
Struct (*class in angr.ailment.expression*), 1394
Struct (*class in angr.analyses.typehoon.typeconsts*), 1279
struct_name() (*angr.analyses.typehoon.translator.TypeTranslator method*), 1266
struct_name_to_idx (*angr.analyses.typehoon.translator.TypeTranslator attribute*), 1266
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCCdecl attribute*), 668
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 670
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCMicrosoftCdecl attribute*), 668
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCMicrosoftThiscall attribute*), 668
struct_type (*angr.analyses.decompiler.structured_codegen.c.CStructField attribute*), 1138
StructMode (*class in angr.state_plugins*), 326
StructMode (*class in angr.state_plugins.view*), 431
structs (*angr.analyses.typehoon.translator.TypeTranslator attribute*), 1266
structured_code (*angr.angrdb.models.DbKnowledgeBase attribute*), 1022
structured_node_has_multi_predecessors() (*in module angr.analyses.decompiler.utils*), 1153
structured_node_is_simple_return() (*in module angr.analyses.decompiler.utils*), 1153
structured_node_is_simple_return_strict() (*in module angr.analyses.decompiler.utils*), 1153
StructuredCodeGenerator (*in module angr.analyses.decompiler*), 1052
StructuredCodeManager (*class in angr.knowledge_plugins*), 734
StructuredCodeManager (*class in angr.knowledge_plugins.structured_code*), 822
StructuredCodeManagerSerializer (*class in angr.angrdb.serializers.structured_code*), 1031
structurer_class_from_name() (*in module angr.analyses.decompiler.structuring*), 1035
StructurerBase (*class in angr.analyses.decompiler.structuring.structurer_base*), 1036
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ConstPropOp attribute*), 1074

STRUCTURING (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator attribute*), 1093
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator attribute*), 1096
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator attribute*), 1086
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator attribute*), 1086
StructuringOptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 1096
StubStasher (*class in angr.exploration_techniques*), 564
Sub (*angr.engines.light.ArithmeticExpression attribute*), 1167
Sub (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
Sub (*class in angr.analyses.typehoon.typevars*), 1268
sub_graph (*angr.analyses.data_dep.data_dependency_analysis.DataDependencyGraphAnalysis property*), 1312
sub_graph (*angr.analyses.data_dep.DataDependencyGraphAnalysis property*), 1317
sub_graph (*angr.analyses.DataDependencyGraphAnalysis property*), 900
sub_type (*angr.analyses.typehoon.typevars.Subtype attribute*), 1267
subgraph() (*angr.knowledge_plugins.Function method*), 722
subgraph() (*angr.knowledge_plugins.functions.Function method*), 783
subgraph() (*angr.knowledge_plugins.functions.function.Function method*), 807
subgraph_between_nodes() (*in module angr.utils.graph*), 1347
subject (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 1217
subject (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 1201
subject (*angr.analyses.ReachingDefinitionsAnalysis property*), 916
Subject (*class in angr.analyses.reaching_definitions.subject*), 1238
SubjectType (*class in angr.analyses.reaching_definitions.subject*), 1237
SubN (*class in angr.analyses.typehoon.typevars*), 1272
subscribe_actions() (*angr.state_plugins.history.SimStateHistory method*), 377
subscribe_actions() (*angr.state_plugins.SimStateHistory method*), 303
Subtype (*class in angr.analyses.typehoon.typevars*), 1267
SUCCESS (*angr.analysis.BaseOptimizationPass attribute*), 996
successor_blocks (*angr.analyses.decompiler.optimization_passes.StructuringOptimizationPass attribute*), 1249
successor_func() (*angr.annocfg.AnnotatedCFG method*), 1122
successors (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 1071
successors (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
successors (*angr.knowledge_plugins.cfg.CFGNode property*), 750
successors() (*angr.analyses.forward_analysis.CallGraphVisitor method*), 931
successors() (*angr.analyses.forward_analysis.FunctionGraphVisitor method*), 933
successors() (*angr.analyses.forward_analysis.LoopVisitor method*), 936
successors() (*angr.analyses.forward_analysis.SingleNodeGraphVisitor method*), 934
successors() (*angr.analyses.forward_analysis.visitors.call_graph.CallGr method*), 941
successors() (*angr.analyses.forward_analysis.visitors.CallGraphVisitor method*), 937
successors() (*angr.analyses.forward_analysis.visitors.function_graph.Fu method*), 942
successors() (*angr.analyses.forward_analysis.visitors.FunctionGraphVis method*), 938
successors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisito method*), 942
successors() (*angr.analyses.forward_analysis.visitors.loop.LoopVisitor method*), 944
successors() (*angr.analyses.forward_analysis.visitors.LoopVisitor method*), 939
successors() (*angr.analyses.forward_analysis.visitors.single_node_graph method*), 945
successors() (*angr.analyses.forward_analysis.visitors.SingleNodeGraphV method*), 940
successors() (*angr.codenode.CodeNode method*), 1323
successors() (*angr.exploration_techniques.Bucketizer method*), 553
successors() (*angr.exploration_techniques.bucketizer.Bucketizer method*), 586
successors() (*angr.exploration_techniques.ExplorationTechnique method*), 556
successors() (*angr.exploration_techniques.local_loop_seer.LocalLoopSe method*), 583
successors() (*angr.exploration_techniques.LocalLoopSeer method*), 559
successors() (*angr.exploration_techniques.loop_seer.LoopSeer method*), 582
successors() (*angr.exploration_techniques.LoopSeer

method), 560
successors() (angr.exploration_techniques.Oppologist method), 561
successors() (angr.exploration_techniques.oppologist.Oppologist method), 581
successors() (angr.exploration_techniques.Slicecutor method), 563
successors() (angr.exploration_techniques.slicecutor.Slicecutor method), 579
successors() (angr.ExplorationTechnique method), 163
successors() (angr.factory.AngrObjectFactory method), 228
successors() (angr.sim_manager.SimulationManager method), 548
successors() (angr.SimulationManager method), 218
successors_and_jumpkinds() (angr.knowledge_plugins.cfg.cfg_node.CFGNode method), 767
successors_and_jumpkinds() (angr.knowledge_plugins.cfg.CFGNode method), 751
SuccessorsEngine (class in angr.engines), 593
SuccessorsEngine (class in angr.engines.successors), 596
Suggestions (class in angr.exploration_techniques), 564
Suggestions (class in angr.exploration_techniques.suggestions), 586
super_type (angr.analyses.typehoon.typevars.Subtype attribute), 1267
SuperFastpathMixin (class in angr.engines), 594
SuperFastpathMixin (class in angr.engines.vex), 600
SUPPORTS_CONCRETE_LOAD (angr.storage.memory_mixins.paged_memory.pagesize_mixins.PageSizeMixin attribute), 504
SUPPORTS_CONCRETE_LOAD (angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin attribute), 522
SUPPORTS_CONCRETE_LOAD (angr.storage.memory_mixins.paged_memory.page_ll.LinkedPage attribute), 516
SUPPORTS_CONCRETE_LOAD (angr.storage.memory_mixins.PagedMemoryMixin attribute), 474
SUPPORTS_CONCRETE_LOAD (angr.storage.memory_mixins.UltraPage attribute), 487
switch (angr.analyses.decompiler.structured_codegen.c.CSwitchCase attribute), 1135
switch_case_entry_node_has_common_successor_case_1() (angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer static method), 1039
switch_case_entry_node_has_common_successor_case_1() (angr.analyses.decompiler.structuring.PhoenixStructurer static method), 1034
switch_case_entry_node_has_common_successor_case_2() (angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer static method), 1039
switch_case_entry_node_has_common_successor_case_2() (angr.analyses.decompiler.structuring.PhoenixStructurer static method), 1034
switch_case_known_heads (angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer attribute), 1038
switch_extract_bitwiseand_jumptable_info() (in module angr.analyses.decompiler.utils), 1150
switch_extract_cmp_bounds() (in module angr.analyses.decompiler.utils), 1150
switch_extract_cmp_bounds_from_condition() (in module angr.analyses.decompiler.utils), 1150
switch_extract_switch_expr_from_jump_target() (in module angr.analyses.decompiler.utils), 1150
SwitchCaseRegion (class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl 1116
SwitchClusterFinder (class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl 1116
SwitchClusterReplacer (class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl 1117
SwitchDefaultCaseDuplicator (class in angr.analyses.decompiler.optimization_passes), 1089
SwitchExpressionSimplifierMixin (class in angr.analyses.decompiler.region_simplifiers.switch_expr_simplifi 1119
SwitchReusedEntryRewriter (class in angr.analyses.decompiler.optimization_passes), 1089
symbol (angr.analyses.reaching_definitions.function_handler.FunctionCall attribute), 1224
symbol (angr.analyses.reaching_definitions.FunctionCallData attribute), 1183
symbol (angr.knowledge_plugins.Function property), 720
symbol (angr.knowledge_plugins.functions.Function property), 780
symbol (angr.knowledge_plugins.functions.function.Function property), 805
symbol_hooked_by() (angr.Project method), 171
symbol_hooked_by() (angr.project.Project method), 226

symbol_name (*angr.analyses.reassembler.NotypeLabel property*), 1298
symbol_name (*angr.analyses.reassembler.ObjectLabel property*), 1298
symbolic (*angr.analyses.reaching_definitions.MemoryLocation property*), 1197
symbolic (*angr.engines.light.data.RegisterOffset property*), 1167
symbolic (*angr.engines.light.RegisterOffset property*), 1167
symbolic (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation property*), 841
SYMBOLIC (*angr.simos.windows.SecurityCookieInit attribute*), 1338
symbolic (*angr.state_plugins.sim_action_object.SimActionObject property*), 641
symbolic (*angr.state_plugins.SimActionObject property*), 265
symbolic (*angr.storage.memory_object.SimMemoryObject property*), 451
symbolic (*angr.storage.SimMemoryObject property*), 426
symbolic() (*angr.state_plugins.SimSolver method*), 296
symbolic() (*angr.state_plugins.solver.SimSolver method*), 370
symbolic_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
symbolic_expr (*angr.state_plugins.InspectAttrs attribute*), 254
symbolic_name (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
symbolic_name (*angr.state_plugins.InspectAttrs attribute*), 254
symbolic_size (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
symbolic_size (*angr.state_plugins.InspectAttrs attribute*), 254
symbolic_stop_reasons (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
SYMBOLIC_VARIABLE (*angr.state_plugins.inspect.EventType attribute*), 332
symbolic_vex_stmts (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 394
symbolic_vex_stmts_count (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 394
SymbolicMergerMixin (*class in angr.storage.memory_mixins*), 487
SymbolicMergerMixin (*class in angr.storage.memory_mixins.symbolic_merger_mixin*), 493
symbolize() (*angr.analyses.Reassembler method*), 918
symbolize() (*angr.analyses.reassembler.Reassembler method*), 1304
symbolized (*angr.analyses.reassembler.Operand property*), 1299
SymbolManager (*class in angr.analyses.reassembler*), 1298
sympy_expr_to_claripy_ast() (*angr.analyses.decompiler.condition_processor.ConditionProcess static method*), 1063
syscall (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 988
syscall (*angr.analyses.cfg.cfg_fast.FunctionCallEdge attribute*), 986
syscall (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 769
syscall (*angr.knowledge_plugins.cfg.CFGENode attribute*), 741
SYSCALL (*angr.state_plugins.inspect.EventType attribute*), 332
syscall() (*angr.SimOS method*), 196
syscall() (*angr.simos.SimOS method*), 1329
syscall() (*angr.simos.simos.SimOS method*), 1334
syscall() (*angr.simos.SimUserland method*), 1331
syscall() (*angr.simos.SimWindows method*), 1332
syscall() (*angr.simos.userland.SimUserland method*), 1337
syscall() (*angr.simos.windows.SimWindows method*), 1339
syscall_abi() (*angr.SimOS method*), 196
syscall_abi() (*angr.simos.linux.SimLinux method*), 1336
syscall_abi() (*angr.simos.SimLinux method*), 1327
syscall_abi() (*angr.simos.SimOS method*), 1329
syscall_abi() (*angr.simos.simos.SimOS method*), 1334
syscall_abi() (*angr.simos.SimUserland method*), 1331
syscall_abi() (*angr.simos.userland.SimUserland method*), 1337
syscall_cc() (*angr.SimOS method*), 197
syscall_cc() (*angr.simos.SimOS method*), 1329
syscall_cc() (*angr.simos.simos.SimOS method*), 1334
syscall_cc() (*angr.simos.SimUserland method*), 1331
syscall_cc() (*angr.simos.userland.SimUserland method*), 1337
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 676
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 675
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 678
SYSCALL_ERRNO_START

(*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 677
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCSyscall attribute*), 670
syscall_from_addr() (*angr.SimOS method*), 197
syscall_from_addr() (*angr.simos.SimOS method*), 1329
syscall_from_addr() (*angr.simos.simos.SimOS method*), 1335
syscall_from_addr() (*angr.simos.SimUserland method*), 1331
syscall_from_addr() (*angr.simos.SimWindows method*), 1332
syscall_from_addr() (*angr.simos.userland.SimUserland method*), 1337
syscall_from_addr() (*angr.simos.windows.SimWindows method*), 1339
syscall_from_number() (*angr.SimOS method*), 197
syscall_from_number() (*angr.simos.SimOS method*), 1330
syscall_from_number() (*angr.simos.simos.SimOS method*), 1335
syscall_from_number() (*angr.simos.SimUserland method*), 1331
syscall_from_number() (*angr.simos.userland.SimUserland method*), 1338
syscall_hook() (*in module angr.state_plugins.trace_additions*), 382
syscall_name (*angr.knowledge_plugins.cfg.cfg_node.CFGENode property*), 769
syscall_name (*angr.knowledge_plugins.cfg.CFGENode property*), 741
syscall_name (*angr.state_plugins.inspect.InspectAttrs attribute*), 336
syscall_name (*angr.state_plugins.InspectAttrs attribute*), 255
syscall_num() (*angr.calling_conventions.SimCCAArch64LinuxSyscall static method*), 674
syscall_num() (*angr.calling_conventions.SimCCAMD64LinuxSyscall static method*), 672
syscall_num() (*angr.calling_conventions.SimCCAMD64WindowsSyscall static method*), 672
syscall_num() (*angr.calling_conventions.SimCCARMLinuxSyscall static method*), 673
syscall_num() (*angr.calling_conventions.SimCCN64LinuxSyscall static method*), 676
syscall_num() (*angr.calling_conventions.SimCCO32LinuxSyscall static method*), 675
syscall_num() (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall static method*), 678
syscall_num() (*angr.calling_conventions.SimCCPowerPCLinuxSyscall static method*), 677
syscall_num() (*angr.calling_conventions.SimCCRISCV64LinuxSyscall static method*), 675
syscall_num() (*angr.calling_conventions.SimCCS390XLinuxSyscall static method*), 679
syscall_num() (*angr.calling_conventions.SimCCSyscall static method*), 670
syscall_num() (*angr.calling_conventions.SimCCX86LinuxSyscall static method*), 670
syscall_num() (*angr.calling_conventions.SimCCX86WindowsSyscall static method*), 671
SyscallNode (*class in angr.codenode*), 1324
SyscallResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1012

T

tag (*angr.analyses.cfg.cfg_base.CFGBase attribute*), 981
tag (*angr.analyses.cfg.cfg_emulated.CFGEmulated attribute*), 977
tag (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 990
tag (*angr.analyses.cfg.CFGBase attribute*), 959
tag (*angr.analyses.cfg.CFGEmulated attribute*), 962
tag (*angr.analyses.cfg.CFGFast attribute*), 967
tag (*angr.analyses.CFGEmulated attribute*), 885
tag (*angr.analyses.CFGFast attribute*), 890
tag (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode attribute*), 1262
Tag (*class in angr.knowledge_plugins.key_definitions.tag*), 857
TagDict (*class in angr.ailment.tagged_object*), 1407
TaggedObject (*class in angr.ailment.tagged_object*), 1408
tags (*angr.ailment.tagged_object.TaggedObject attribute*), 1408
tags (*angr.analyses.decompiler.structured_codegen.c.CConstruct attribute*), 1131
tags (*angr.analyses.decompiler.structured_codegen.c.CStatement attribute*), 1132
tags (*angr.analyses.reaching_definitions.Definition attribute*), 1182
tags (*angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute*), 1222
tags (*angr.knowledge_plugins.Function attribute*), 717
tags (*angr.knowledge_plugins.functions.Function attribute*), 777
tags (*angr.knowledge_plugins.functions.function.Function attribute*), 802
tags (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 810
tags (*angr.knowledge_plugins.key_definitions.Definition attribute*), 823

tags (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 843
TagSlicer (*class in angr.analyses.decompiler.optimization_passes*), 1090
TAINT_ENTITY_MEM (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 392
TAINT_ENTITY_NONE (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 392
TAINT_ENTITY_REG (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 392
TAINT_ENTITY_TMP (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 392
TaintEntityEnum (*class in angr.state_plugins.unicorn_engine*), 392
taken (*angr.exploration_techniques.spiller_db.PickledState attribute*), 574
tally() (*angr.sim_state_options.SimStateOptions method*), 246
target (*angr.ailment.expression.Call attribute*), 1393
target (*angr.ailment.statement.Jump attribute*), 1400
target (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 1241
target (*angr.analyses.cfg_slice_to_sink.CFGSliceToSink property*), 1239
target (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.Case attribute*), 1100
target (*angr.analyses.decompiler.structured_codegen.c.CGoto attribute*), 1137
target (*angr.analyses.reaching_definitions.call_trace.CallTrace attribute*), 1212
target (*angr.analyses.reaching_definitions.function_handler.FunctionCallRelationships attribute*), 1226
target (*angr.analyses.reaching_definitions.FunctionCallRelationships attribute*), 1185
target (*angr.analyses.reaching_definitions.GuardUse attribute*), 1189
target (*angr.analyses.typehoon.simple_solver.RecursiveRefNode attribute*), 1260
target (*angr.knowledge_plugins.key_definitions.atoms.GuardUse attribute*), 838
target_idx (*angr.ailment.statement.Jump attribute*), 1400
target_idx (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.Case attribute*), 1100
target_idx (*angr.analyses.decompiler.structured_codegen.c.CGoto attribute*), 1137
tc2simtype() (*angr.analyses.typehoon.translator.TypeTranslator method*), 1266
teardown_callsite() (*angr.calling_conventions.SimCC method*), 666
teardown_callsite() (*angr.SimCC method*), 177
TechniqueBuilder (*class in angr.exploration_techniques*), 565
TechniqueBuilder (*class in angr.exploration_techniques.tech_builder*), 585
tell() (*angr.SimFileDescriptor method*), 183
tell() (*angr.SimFileDescriptorDuplex method*), 185
tell() (*angr.storage.file.SimFileDescriptor method*), 444
tell() (*angr.storage.file.SimFileDescriptorBase method*), 443
tell() (*angr.storage.file.SimFileDescriptorDuplex method*), 447
TemporaryNode (*class in angr.utils.graph*), 1348
temps (*angr.slicer.SimLightState attribute*), 1321
terminate_execution() (*angr.engines.soot.engine.SootMixin static method*), 602
terminate_execution() (*angr.engines.soot.SootMixin static method*), 602
terminate_execution() (*angr.engines.SootMixin static method*), 593
terminate_execution() (*angr.Project method*), 172
terminate_execution() (*angr.project.Project method*), 227
test_path() (*angr.analyses.Pathfinder method*), 911
test_reducibility() (*angr.analyses.decompiler.region_identifier.RegionIdentifier method*), 1108
test_reducibility() (*angr.analyses.decompiler.RegionIdentifier method*), 1050
test_unsupported_overlap() (*angr.knowledge_plugins.debug_variables.DebugVariable method*), 821
text (*angr.analyses.decompiler.structured_codegen.c.CArrayTypeLength attribute*), 1144
TGraph (*in module angr.analyses.decompiler.region_identifier*), 1107
Threading (*class in angr.exploration_techniques*), 565
Threading (*class in angr.exploration_techniques.threading*), 574
THUMB (*angr.analyses.cfg.cfg_fast.ARMDecodingMode attribute*), 984
thumb (*angr.block.Block attribute*), 160
thumb (*angr.block.Block attribute*), 233
thumb (*angr.block.DisassemblerBlock attribute*), 232
thumb (*angr.codenode.CodeNode attribute*), 1323
thumb (*angr.engines.pcode.lifter.PcodeDisassemblerBlock attribute*), 606
thumb (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 767
thumb (*angr.knowledge_plugins.cfg.CFGNode property*), 750
thumb (*angr.sim_state.SimState property*), 242
thumb (*angr.SimState property*), 211

tidy_data_references() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 762
tidy_data_references() (*angr.knowledge_plugins.cfg.CFGModel method*), 748
timed_function() (*in module angr.state_plugins.solver*), 361
Timeout (*class in angr.exploration_techniques*), 565
Timeout (*class in angr.exploration_techniques.timeout*), 569
timestamp (*angr.exploration_techniques.spiller_db.PickledState attribute*), 574
timethis() (*in module angr.utils*), 1344
timethis() (*in module angr.utils.timing*), 1353
TMP (*angr.ailment.expression.VirtualVariableCategory attribute*), 1379
Tmp (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 1313
Tmp (*angr.analyses.data_dep.DepNodeTypes attribute*), 1318
tmp (*angr.analyses.propagator.vex_vars.VEXTmp attribute*), 1176
TMP (*angr.analyses.reaching_definitions.AtomKind attribute*), 1181
TMP (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 836
TMP (*angr.state_plugins.sim_action.SimAction attribute*), 638
TMP (*angr.state_plugins.SimAction attribute*), 263
Tmp (*class in angr.ailment*), 1372
Tmp (*class in angr.ailment.expression*), 1377
Tmp (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 999
Tmp (*class in angr.analyses.reaching_definitions*), 1211
Tmp (*class in angr.knowledge_plugins.key_definitions.atoms*), 839
tmp() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
tmp_deps (*angr.state_plugins.sim_action.SimAction property*), 638
tmp_deps (*angr.state_plugins.sim_action.SimActionData property*), 640
tmp_deps (*angr.state_plugins.sim_action_object.SimActionObject attribute*), 640
tmp_deps (*angr.state_plugins.SimAction property*), 263
tmp_deps (*angr.state_plugins.SimActionData property*), 264
tmp_deps (*angr.state_plugins.SimActionObject attribute*), 265
tmp_expr() (*angr.state_plugins.scratch.SimStateScratch method*), 388
tmp_expr() (*angr.state_plugins.SimStateScratch method*), 318
tmp_id (*angr.sim_variable.SimTemporaryVariable attribute*), 682
tmp_idx (*angr.ailment.expression.Tmp attribute*), 1377
tmp_idx (*angr.ailment.expression.VirtualVariable property*), 1380
tmp_idx (*angr.ailment.Tmp attribute*), 1372
tmp_idx (*angr.analyses.reaching_definitions.Tmp attribute*), 1212
tmp_idx (*angr.knowledge_plugins.key_definitions.atoms.Tmp attribute*), 839
tmp_idx (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 840
tmp_idx (*angr.knowledge_plugins.key_definitions.definition.DefinitionMat attribute*), 842
TMP_READ (*angr.state_plugins.inspect.EventType attribute*), 331
tmp_read_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
tmp_read_expr (*angr.state_plugins.InspectAttrs attribute*), 254
tmp_read_num (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
tmp_read_num (*angr.state_plugins.InspectAttrs attribute*), 254
tmp_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
tmp_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitions property*), 1233
tmp_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 1206
tmp_uses (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDe attribute*), 848
tmp_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 825
TMP_WRITE (*angr.state_plugins.inspect.EventType attribute*), 331
tmp_write_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
tmp_write_expr (*angr.state_plugins.InspectAttrs attribute*), 254
tmp_write_num (*angr.state_plugins.inspect.InspectAttrs attribute*), 335
tmp_write_num (*angr.state_plugins.InspectAttrs attribute*), 254
TmpDepNode (*class in angr.analyses.data_dep*), 1318
TmpDepNode (*class in angr.analyses.data_dep.dep_nodes*), 1315
tmps (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
tmps (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 1233
tmps (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 1206
tmps (*angr.engines.light.engine.SimEngineLight at-*

tribute), 1171
tmps (*angr.engines.light.SimEngineLight attribute*), 1168
tmps (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 848
tmps (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 825
TNode (*in module angr.analyses.decompiler.region_identifier*), 1107
to_acyclic_graph() (*in module angr.utils.graph*), 1347
to_ail_supergraph() (*in module angr.analyses.decompiler.utils*), 1151
to_bits (*angr.ailment.expression.Convert attribute*), 1383
to_bits (*angr.ailment.expression.Reinterpret attribute*), 1383
to_bits (*angr.analyses.typehoon.typevars.ConvertTo attribute*), 1272
to_bits (*angr.analyses.typehoon.typevars.ReinterpretAs attribute*), 1272
to_claripy() (*angr.state_plugins.sim_action_object.SimActionObject method*), 640
to_claripy() (*angr.state_plugins.SimActionObject method*), 265
to_codenode() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 768
to_codenode() (*angr.knowledge_plugins.cfg.CFGNode method*), 751
to_json() (*angr.angrdb.serializers.structured_code.ConstantMergerMixin static method*), 1031
to_json() (*angr.knowledge_plugins.functions.function.FunctionInfo method*), 800
to_json() (*angr.knowledge_plugins.functions.function_parser.CallingConventionSerializer static method*), 809
to_json() (*angr.procedures.definitions.SimTypeCollection method*), 649
to_json() (*angr.sim_type.SimCppClass method*), 706
to_json() (*angr.sim_type.SimStruct method*), 701
to_json() (*angr.sim_type.SimType method*), 687
to_json() (*angr.sim_type.SimTypeFunction method*), 698
to_json() (*angr.sim_type.SimTypeInt method*), 690
to_json() (*angr.sim_type.SimTypePointer method*), 694
to_json() (*angr.sim_type.SimTypeRef method*), 708
to_outside (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute*), 986
to_size (*angr.analyses.decompiler.expression_narrower.ExpressionNarrowerLight attribute*), 1069
to_string() (*angr.knowledge_plugins.xrefs.xref_types.XRefTypeLabel static method*), 864
to_string() (*angr.knowledge_plugins.xrefs.XRefType static method*), 863
to_type (*angr.ailment.expression.Convert attribute*), 1383
to_type (*angr.ailment.expression.Reinterpret attribute*), 1383
to_type (*angr.analyses.typehoon.typevars.ReinterpretAs attribute*), 1272
to_valueset() (*angr.storage.memory_mixins.regioned_memory.region_d method*), 533
TOLOWER_LOC_ARRAY (*angr.state_plugins.libc.SimStateLibc attribute*), 345
TOLOWER_LOC_ARRAY (*angr.state_plugins.SimStateLibc attribute*), 309
top (*angr.state_plugins.CallStack property*), 250
top (*angr.state_plugins.callstack.CallStack property*), 373
Top (*class in angr.analyses.propagator.values*), 1175
top() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 1190
top() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsSta method*), 1231
top() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 1205
top() (*angr.analyses.variable_recovery.variable_recovery_base.VariableR static method*), 1249
top() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinit static method*), 848
top() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 825
TopMergerMixin (*class in angr.storage.memory_mixins*), 487
TopMergerMixin (*class in angr.storage.memory_mixins.top_merger_mixin*), 503
TopType (*class in angr.analyses.typehoon.typeconsts*), 1274
total_count (*angr.knowledge_plugins.functions.function_manager.Spillin property*), 794
total_function_count (*angr.knowledge_plugins.FunctionManager property*), 730
total_function_count (*angr.knowledge_plugins.functions.function_manager.FunctionM property*), 799
total_function_count (*angr.knowledge_plugins.functions.FunctionManager property*), 790
TOUPPER_LOC_ARRAY (*angr.state_plugins.libc.SimStateLibc attribute*), 345
TOUPPER_LOC_ARRAY (*angr.state_plugins.SimStateLibc attribute*), 309
traceback (*angr.knowledge_plugins.cfg.cfg_node.CFGNodeCreationFailu attribute*), 766
traceback (*angr.sim_manager.ErrorRecord attribute*), 551
traceflags (*angr.engines.pcode.lifter.Lifter attribute*), 610

traceflags (*angr.engines.pcode.lifter.PcodeLifter attribute*), 613
Tracer (*class in angr.exploration_techniques*), 566
Tracer (*class in angr.exploration_techniques.tracer*), 575
TracerDesyncError, 575
TracerEnvironmentError, 221, 1357
TracingMode (*class in angr.exploration_techniques.tracer*), 575
TRACK_STACK_POINTERS (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 1057
track_tmps (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 1190
track_tmps (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 848
track_tmps (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 824
TrackActionsMixin (*class in angr.engines*), 594
TrackActionsMixin (*class in angr.engines.vex*), 600
transform_graph_from_ssa() (*angr.analyses.Decompiler method*), 902
transform_graph_from_ssa() (*angr.analyses.decompiler.Decompiler method*), 1048
transform_graph_from_ssa() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1067
transform_seqnode_from_ssa() (*angr.analyses.Decompiler method*), 902
transform_seqnode_from_ssa() (*angr.analyses.decompiler.Decompiler method*), 1048
transform_seqnode_from_ssa() (*angr.analyses.decompiler.decompiler.Decompiler method*), 1067
transition_graph (*angr.knowledge_plugins.Function attribute*), 717
transition_graph (*angr.knowledge_plugins.functions.Function attribute*), 777
transition_graph (*angr.knowledge_plugins.functions.function.Function attribute*), 801
transition_graph (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 810
transition_graph_ex() (*angr.knowledge_plugins.Function method*), 722
transition_graph_ex() (*angr.knowledge_plugins.functions.Function method*), 782
transition_graph_ex() (*angr.knowledge_plugins.functions.function.Function method*), 807
transitions (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 1241
transitions (*angr.analyses.cfg_slice_to_sink.CFGSliceToSink property*), 1239
transitions_as_tuples (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 1241
transitions_as_tuples (*angr.analyses.cfg_slice_to_sink.CFGSliceToSink property*), 1239
transitive_closure() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 1218
translate() (*angr.calling_conventions.AllocHelper method*), 658
translated_definitions (*angr.analyses.typehoon.translator.TypeTranslator attribute*), 1265
translated_simtypes (*angr.analyses.typehoon.translator.TypeTranslator attribute*), 1265
TRANSMIT_RECORD (*class in angr.state_plugins.unicorn_engine*), 392
traverse_in_order() (*in module angr.analyses.decompiler.ailgraph_walker*), 1054
TreeIter (*class in angr.state_plugins.history*), 378
trim() (*angr.state_plugins.history.SimStateHistory method*), 376
trim() (*angr.state_plugins.SimStateHistory method*), 302
Triop() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
triop_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1173
triop_handler() (*angr.engines.light.SimEngineLightVEX static method*), 1169
true_target (*angr.ailment.statement.ConditionalJump attribute*), 1401
true_target_idx (*angr.ailment.statement.ConditionalJump attribute*), 1401
Truncation (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransformation attribute*), 998
try_unpack_const() (*angr.engines.light.ArithmeticExpression static method*), 1167
try_unpack_const() (*angr.engines.light.data.ArithmeticExpression static method*), 1166
ty (*angr.procedures.stubs.format_parser.FormatSpecifier attribute*), 647
ty (*angr.sim_type.TypeRef property*), 688
tyenv (*angr.engines.light.engine.SimEngineLightVEX attribute*), 1172
tyenv (*angr.engines.light.SimEngineLightVEX attribute*), 1169
tyenv (*angr.engines.pcode.lifter.IRSB property*), 609
type (*angr.cle.backends.tagged_object.TagDict attribute*),

1408
type (*angr.analyses.data_dep.BaseDepNode property*), 1316
type (*angr.analyses.data_dep.dep_nodes.BaseDepNode property*), 1314
type (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp property*), 1141
type (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 1142
type (*angr.analyses.decompiler.structured_codegen.c.CDirtyExpression property*), 1144
type (*angr.analyses.decompiler.structured_codegen.c.CDirtyStatement property*), 1137
type (*angr.analyses.decompiler.structured_codegen.c.CExpression property*), 1133
type (*angr.analyses.decompiler.structured_codegen.c.CFakeVariable property*), 1139
type (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall property*), 1137
type (*angr.analyses.decompiler.structured_codegen.c.CIndexedVariable property*), 1139
type (*angr.analyses.decompiler.structured_codegen.c.CITE property*), 1143
type (*angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression property*), 1143
type (*angr.analyses.decompiler.structured_codegen.c.CRegister property*), 1142
type (*angr.analyses.decompiler.structured_codegen.c.CStructureField property*), 1138
type (*angr.analyses.decompiler.structured_codegen.c.CTypeCast property*), 1141
type (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp property*), 1140
type (*angr.analyses.decompiler.structured_codegen.c.CVariable property*), 1139
type (*angr.analyses.decompiler.structured_codegen.c.CVariableField property*), 1140
type (*angr.analyses.decompiler.structured_codegen.c.CVEXCallExpression property*), 1143
type (*angr.analyses.reaching_definitions.subject.Subject property*), 1238
type (*angr.angrdb.models.DbComment attribute*), 1025
type (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 771
type (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 753
type (*angr.knowledge_plugins.xrefs.XRef attribute*), 861
type (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 863
type (*angr.sim_type.TypeRef property*), 688
type_ (*angr.analyses.typehoon.typevars.Existence attribute*), 1267
type_0 (*angr.analyses.typehoon.typevars.Add attribute*), 1267
type_0 (*angr.analyses.typehoon.typevars.Sub attribute*), 1268
type_1 (*angr.analyses.typehoon.typevars.Add attribute*), 1268
type_1 (*angr.analyses.typehoon.typevars.Sub attribute*), 1268
type_a (*angr.analyses.typehoon.typevars.Equivalence attribute*), 1266
type_b (*angr.analyses.typehoon.typevars.Equivalence attribute*), 1266
type_constraints (*angr.analyses.decompiler.decompilation_cache.Deco attribute*), 1064
type_constraints (*angr.analyses.variable_recovery.engine_base.RichR attribute*), 1257
type_equals() (*in module angr.analyses.decompiler.structured_codegen.c*), 1130
TYPE_FP (*angr.ailment.expression.Convert attribute*), 1382
TYPE_FP (*angr.ailment.expression.ConvertType attribute*), 1382
type_hints (*angr.analyses.decompiler.peephole_optimizations.base.Peeph attribute*), 1107
type_hints (*angr.analyses.decompiler.peephole_optimizations.base.Peeph attribute*), 1106
type_hints (*angr.analyses.decompiler.peephole_optimizations.base.Peeph attribute*), 1105
TYPE_INT (*angr.ailment.expression.Convert attribute*), 1382
TYPE_INT (*angr.ailment.expression.ConvertType attribute*), 1382
type_parser_singleton() (*in module angr.sim_type*), 709
type_r (*angr.analyses.typehoon.typevars.Add attribute*), 1268
type_r (*angr.analyses.typehoon.typevars.Sub attribute*), 1268
type_string (*angr.knowledge_plugins.xrefs.XRef property*), 861
type_string (*angr.knowledge_plugins.xrefs.xref.XRef property*), 864
type_to_c_repr_chunks() (*in module angr.analyses.decompiler.structured_codegen.c*), 1131
type_var (*angr.analyses.typehoon.typevars.DerivedTypeVariable attribute*), 1270
TypeConstant (*class in angr.analyses.typehoon.typeconsts*), 1274
TypeConstraint (*class in angr.analyses.typehoon.typevars*), 1266
typehoon (*angr.analyses.decompiler.clinic.Clinic attribute*), 1061
Typehoon (*class in angr.analyses*), 923
Typehoon (*class in angr.analyses.typehoon*), 1281

Typehoon (*class in angr.analyses.typehoon.typehoon*), 1272
TypeRef (*class in angr.sim_type*), 688
types (*angr.knowledge_base.KnowledgeBase attribute*), 711
types (*angr.KnowledgeBase attribute*), 164
types (*angr.sim_state_options.StateOption attribute*), 243
types (*angr.state_plugins.SimMemView attribute*), 282
types (*angr.state_plugins.view.SimMemView attribute*), 428
TypesStore (*class in angr.knowledge_plugins*), 735
TypesStore (*class in angr.knowledge_plugins.types*), 772
TypeTranslator (*class in angr.analyses.typehoon.translator*), 1265
typevar (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode attribute*), 1262
typevar (*angr.analyses.typehoon.simple_solver.SketchNode attribute*), 1259
typevar (*angr.analyses.variable_recovery.engine_base.RichR attribute*), 1257
typevar_to_variable() (*angr.analyses.typehoon.typevars.TypeVariables method*), 1271
TypeVariable (*class in angr.analyses.typehoon.typevars*), 1269
TypeVariableManager (*class in angr.analyses.typehoon.typevars*), 1268
TypeVariableReference (*class in angr.analyses.typehoon.typeconsts*), 1281
TypeVariables (*class in angr.analyses.typehoon.typevars*), 1270

U

UberEngine (*class in angr.engines*), 594
UberEnginePcode (*class in angr.engines*), 594
uc (*angr.state_plugins.Unicorn property*), 328
uc (*angr.state_plugins.unicorn_engine.Unicorn property*), 398
UC_CONFIG (*angr.state_plugins.Unicorn attribute*), 327
UC_CONFIG (*angr.state_plugins.unicorn_engine.Unicorn attribute*), 397
UInt16 (*class in angr.analyses.typehoon.typeconsts*), 1276
UInt32 (*class in angr.analyses.typehoon.typeconsts*), 1276
UInt64 (*class in angr.analyses.typehoon.typeconsts*), 1276
UInt8 (*class in angr.analyses.typehoon.typeconsts*), 1275
UltraPage (*class in angr.storage.memory_mixins*), 487
UltraPage (*class in angr.storage.memory_mixins.paged_memory.pages*), 516
UltraPage (*class in angr.storage.memory_mixins.paged_memory.pages.ult*, 522
UltraPagesMixin (*class in angr.storage.memory_mixins*), 489
UltraPagesMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mix*, 508
UnaryOp (*class in angr.ailment*), 1372
UnaryOp (*class in angr.ailment.expression*), 1381
uncache_region() (*angr.state_plugins.Unicorn method*), 328
uncache_region() (*angr.state_plugins.unicorn_engine.Unicorn method*), 398
unconstrained (*angr.sim_manager.SimulationManager attribute*), 545
unconstrained (*angr.SimulationManager attribute*), 215
Unconstrained() (*angr.state_plugins.SimSolver method*), 288
Unconstrained() (*angr.state_plugins.solver.SimSolver method*), 362
Undefined (*class in angr.knowledge_plugins.key_definitions.undefined*), 858
UnderconstrainedMixin (*class in angr.storage.memory_mixins*), 489
UnderconstrainedMixin (*class in angr.storage.memory_mixins.underconstrained_mixin*), 492
unfreeze() (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrack method*), 1244
unhook() (*angr.Project method*), 171
unhook() (*angr.project.Project method*), 225
unhook_symbol() (*angr.Project method*), 172
unhook_symbol() (*angr.project.Project method*), 226
UnicodeString (*angr.knowledge_plugins.cfg.memory_data.MemoryDataS attribute*), 764
UnicodeString (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 755
Unicorn (*class in angr.state_plugins*), 326
Unicorn (*class in angr.state_plugins.unicorn_engine*), 396
unified_local_vars (*angr.analyses.decompiler.structured_codegen.c.CF attribute*), 1132
unified_variable (*angr.analyses.decompiler.structured_codegen.c.CVar attribute*), 1139
unified_variable() (*angr.knowledge_plugins.variables.variable_manag method*), 818
unify_arch_name() (*in module angr.calling_conventions*), 680
unify_variables() (*angr.knowledge_plugins.variables.variable_manage method*), 818
uninit_read_base (*angr.analyses.cfg.indirect_jump_resolvers.jumptable. attribute*), 998
uninitialized (*angr.ailment.tagged_object.TagDict at-*

tribute), 1408
UninitReadMeta (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 998
union() (*angr.state_plugins.sim_action_object.SimActionObject method*), 641
union() (*angr.state_plugins.SimActionObject method*), 265
unique() (*angr.state_plugins.SimSolver method*), 296
unique() (*angr.state_plugins.solver.SimSolver method*), 369
unique_type_name() (*angr.knowledge_plugins.types.TypesStore method*), 773
unique_type_name() (*angr.knowledge_plugins.TypesStore method*), 735
UniqueSearch (*class in angr.exploration_techniques*), 567
UniqueSearch (*class in angr.exploration_techniques.unique*), 584
Uniwrapper (*class in angr.state_plugins.unicorn_engine*), 396
UNKNOWN (*angr.ailment.expression.VirtualVariableCategory attribute*), 1379
UNKNOWN (*angr.analyses.decompiler.ail_simplifier.DefEqRelation attribute*), 1053
Unknown (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 1309
UNKNOWN (*angr.analyses.typehoon.simple_solver.ConstraintGraphTag attribute*), 1261
Unknown (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 769
Unknown (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 754
Unknown (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 764
Unknown (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 755
UNKNOWN (*angr.sim_type.PointerDisposition attribute*), 694
Unknown (*class in angr.analyses.cfg.cfb*), 972
unknown_returning_func_addrs() (*angr.knowledge_plugins.FunctionManager method*), 728
unknown_returning_func_addrs() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 798
unknown_returning_func_addrs() (*angr.knowledge_plugins.functions.FunctionManager method*), 789
UnknownProxiNode (*class in angr.analyses.proximity_graph*), 1311
UnknownSize (*class in angr.knowledge_plugins.key_definitions.unknown_size*), 859
UnknownSizeTag (*class in angr.knowledge_plugins.key_definitions.tag*), 858
unlinks (*angr.state_plugins.filesystem.SimFilesystem property*), 356
unlinks (*angr.state_plugins.SimFilesystem property*), 268
unmap_by_address() (*angr.storage.memory_mixins.regioned_memory.reg method*), 534
unmap_region() (*angr.storage.memory_mixins.address_concretization_mi method*), 497
unmap_region() (*angr.storage.memory_mixins.AddressConcretizationMix method*), 458
unmap_region() (*angr.storage.memory_mixins.paged_memory.paged_mem method*), 506
unmap_region() (*angr.storage.memory_mixins.PagedMemoryMixin method*), 476
unmatched_blocks (*angr.analyses.bindiff.FunctionDiff property*), 949
unmatched_functions (*angr.analyses.BinDiff property*), 883
unmatched_functions (*angr.analyses.bindiff.BinDiff property*), 950
UnmatchedStatementsException, 947
unmount() (*angr.state_plugins.filesystem.SimFilesystem method*), 357
unmount() (*angr.state_plugins.SimFilesystem method*), 270
Unop() (*angr.ailment.converter_vex.VEXExprConverter static method*), 1374
unop_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1172
unop_handler() (*angr.engines.light.SimEngineLightVEX static method*), 1169
unoptimized_graph (*angr.analyses.decompiler.clinic.Clinic attribute*), 1060
unqualified_name() (*angr.sim_type.NamedTypeMixin method*), 688
unreachable_history() (*angr.state_hierarchy.StateHierarchy method*), 552
unreachable_history() (*angr.StateHierarchy method*), 221
unreachable_state() (*angr.state_hierarchy.StateHierarchy method*), 552
unreachable_state() (*angr.StateHierarchy method*), 221
unresolvables (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 980
unresolvables (*angr.analyses.cfg.CFGEmulated property*), 966
unresolvables (*angr.analyses.CFGEmulated property*), 888

unresolved_indirect_jumps (*angr.knowledge_base.KnowledgeBase property*), 712
unresolved_indirect_jumps (*angr.KnowledgeBase property*), 164
unroll_loops() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 979
unroll_loops() (*angr.analyses.cfg.CFGEmulated method*), 964
unroll_loops() (*angr.analyses.CFGEmulated method*), 887
unsat (*angr.sim_manager.SimulationManager attribute*), 545
unsat (*angr.SimulationManager attribute*), 215
unsat_core() (*angr.state_plugins.SimSolver method*), 292
unsat_core() (*angr.state_plugins.solver.SimSolver method*), 366
unset_stack_address_mapping() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 532
unset_stack_address_mapping() (*angr.storage.memory_mixins.regioned_memory.RegionedMemoryMixin method*), 530
unset_stack_address_mapping() (*angr.storage.memory_mixins.RegionedMemoryMixin method*), 482
unsigned_int_type() (*in module angr.analyses.typehoon.typeconsts*), 1281
UnsignedExtension (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessorInfo attribute*), 998
Unspecified (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 764
Unspecified (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 755
unstash() (*angr.sim_manager.SimulationManager method*), 549
unstash() (*angr.SimulationManager method*), 219
unsupported_reasons (*angr.state_plugins.unicorn_engine.STOP attribute*), 395
UnsupportedCCallError, 221, 1359
UnsupportedDirtyError, 221, 1359
UnsupportedIRExprError, 222, 1359
UnsupportedIROpError, 222, 1359
UnsupportedIRStmtError, 222, 1359
UnsupportedNodeTypeError, 222, 1361
UnsupportedSyscallError (*in module angr*), 222
UnsupportedSyscallError (*in module angr.errors*), 1360
UnwrapperMixin (*class in angr.storage.memory_mixins*), 489
UnwrapperMixin (*class in angr.storage.memory_mixins.unwrapper_mixin*), 498
update() (*angr.knowledge_plugins.PropagationManager method*), 733
update() (*angr.knowledge_plugins.propagations.PropagationManager method*), 773
update() (*angr.procedures.definitions.SimLibrary method*), 650
update() (*angr.procedures.definitions.SimSyscallLibrary method*), 655
update() (*angr.sim_state_options.SimStateOptions method*), 245
update() (*angr.storage.memory_mixins.regioned_memory.region_meta_m method*), 536
update_dbinfo() (*angr.angrdb.AngrDB method*), 1019
update_dbinfo() (*angr.angrdb.db.AngrDB method*), 1020
update_func_block_count() (*angr.knowledge_plugins.Function method*), 721
update_func_block_count() (*angr.knowledge_plugins.functions.Function method*), 781
update_func_block_count() (*angr.knowledge_plugins.functions.function.Function method*), 805
update_graph_with_super_edits() (*angr.analyses.decompiler.optimization_passes.CodeMotionOptim static method*), 1073
update_labels() (*in module angr.analyses.decompiler.utils*), 1153
update_resolved_addrs() (*angr.knowledge_plugins.indirect_jumps.IndirectJumps method*), 775
update_resolved_addrs() (*angr.knowledge_plugins.IndirectJumps method*), 730
update_switch_case_list() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl*, 1117
update_variable_types() (*angr.analyses.Typehoon method*), 924
update_variable_types() (*angr.analyses.typehoon.Typehoon method*), 1282
update_variable_types() (*angr.analyses.typehoon.typehoon.Typehoon method*), 1273
upper_bound (*angr.analyses.typehoon.simple_solver.SketchNode attribute*), 1260
use_exprs (*angr.analyses.decompiler.expression_narrower.ExprNarrowing attribute*), 1069
use_plugin_preset() (*angr.misc.plugins.PluginHub method*), 235
use_technique() (*angr.sim_manager.SimulationManager*

method), 546
use_technique() (angr.SimulationManager method), 216
USER (angr.knowledge_plugins.functions.function.PrototypeSource attribute), 800
UsercallArgSession (class in angr.calling_conventions), 663
uses (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionUseFinder attribute), 1112
Uses (class in angr.knowledge_plugins.key_definitions), 834
Uses (class in angr.knowledge_plugins.key_definitions.uses), 859
uses_by_codeloc (angr.analyses.reaching_definitions.LiveDefinitions attribute), 1190
uses_by_codeloc (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property), 1233
uses_by_codeloc (angr.analyses.reaching_definitions.ReachingDefinitionsState property), 1206
uses_by_codeloc (angr.knowledge_plugins.key_definitions.LiveDefinitions attribute), 848
uses_by_codeloc (angr.knowledge_plugins.key_definitions.rd_state.ReachingDefinitionsState attribute), 825

V

va_arg() (angr.sim_procedure.SimProcedure method), 645
va_arg() (angr.SimProcedure method), 205
val (angr.analyses.stack_pointer_tracker.Constant attribute), 1243
val0 (angr.analyses.stack_pointer_tracker.Eq attribute), 1244
val1 (angr.analyses.stack_pointer_tracker.Eq attribute), 1244
validate() (angr.sim_type.SimTypeBitfield method), 705
value (angr.ailment.Const attribute), 1369
value (angr.ailment.expression.Const attribute), 1377
value (angr.ailment.expression.Insert attribute), 1392
value (angr.analyses.data_dep.BaseDepNode attribute), 1316
value (angr.analyses.data_dep.dep_nodes.BaseDepNode attribute), 1314
value (angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.Case attribute), 1100
value (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.ConditionalRegion attribute), 1116
value (angr.analyses.decompiler.structured_codegen.c.CConstant attribute), 1142
value (angr.analyses.reaching_definitions.ConstantSrc attribute), 1181
value (angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute), 1222
value (angr.angrdb.models.DbInformation attribute), 1021
value (angr.knowledge_plugins.key_definitions.atoms.ConstantSrc attribute), 839
value (angr.knowledge_plugins.key_definitions.heap_address.HeapAddress property), 845
value (angr.sim_variable.SimConstantVariable attribute), 681
value (angr.state_plugins.unicorn_engine.MemoryValue attribute), 393
value (angr.state_plugins.unicorn_engine.RegisterValue attribute), 393
Value (class in angr.analyses.disassembly), 1295
value_float (angr.ailment.Const property), 1369
value_float (angr.ailment.expression.Const property), 1377
value_int (angr.ailment.Const property), 1369
value_int (angr.ailment.expression.Const property), 1377
value_tuple() (angr.analyses.data_dep.BaseDepNode method), 1316
value_tuple() (angr.analyses.data_dep.dep_nodes.BaseDepNode method), 1314
values (angr.analyses.disassembly.MemoryOperand attribute), 1295
values() (angr.knowledge_plugins.FunctionManager method), 726
values() (angr.knowledge_plugins.functions.function_manager.FunctionM method), 795
values() (angr.knowledge_plugins.functions.function_manager.SpillingFu method), 794
values() (angr.knowledge_plugins.functions.FunctionManager method), 786
values() (angr.knowledge_plugins.patches.PatchManager method), 738
values() (angr.knowledge_plugins.PatchManager method), 732
values() (angr.state_plugins.globals.SimStateGlobals method), 387
values() (angr.state_plugins.SimStateGlobals method), 300
values() (angr.storage.memory_mixins.paged_memory.pages.multi_value. method), 502
var2condnodes (angr.analyses.decompiler.region_simplifiers.switch_cluste attribute), 1117
var2switches (angr.analyses.decompiler.region_simplifiers.switch_cluster attribute), 1117
var_collections (angr.angrdb.models.DbKnowledgeBase attribute), 1022
var_to_typevar (angr.analyses.decompiler.decompilation_cache.Decomp attribute), 1065
VarDepNode (class in angr.analyses.data_dep), 1319
VarDepNode (class in angr.analyses.data_dep.dep_nodes), 1315

variable (*angr.ailment.BinaryOp attribute*), 1367
variable (*angr.ailment.expression.Atom attribute*), 1376
variable (*angr.ailment.expression.BasePointerOffset attribute*), 1390
variable (*angr.ailment.expression.BinaryOp attribute*), 1385
variable (*angr.ailment.expression.ComboRegister attribute*), 1378
variable (*angr.ailment.expression.ITE attribute*), 1387
variable (*angr.ailment.expression.Load attribute*), 1386
variable (*angr.ailment.expression.UnaryOp attribute*), 1381
variable (*angr.ailment.statement.Store attribute*), 1399
variable (*angr.ailment.UnaryOp attribute*), 1373
variable (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterRegion attribute*), 1116
variable (*angr.analyses.decompiler.region_simplifiers.switch_cases_simplifier.SwitchCasesRegion attribute*), 1116
variable (*angr.analyses.decompiler.structured_codegen.c.CVariable attribute*), 1139
Variable (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 1309
variable (*angr.analyses.variable_recovery.engine_base.RichR attribute*), 1257
variable (*angr.knowledge_plugins.key_definitions.definition.DefinitionMatchPredicate attribute*), 842
variable (*angr.knowledge_plugins.variables.variable_access.VariableAccessSort attribute*), 812
variable_hash (*angr.analyses.decompiler.optimization_passes.ldiv_opt.LDivOptReplacer attribute*), 1099
variable_list_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 1132
variable_manager (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 1132
variable_manager (*angr.analyses.variable_recovery.variable_recovery.VariableRecoveryBase property*), 1250
variable_manager (*angr.knowledge_plugins.key_definitions.definition.DefinitionMatchPredicate attribute*), 842
variable_offset (*angr.ailment.BinaryOp attribute*), 1367
variable_offset (*angr.ailment.expression.Atom attribute*), 1376
variable_offset (*angr.ailment.expression.BasePointerOffset attribute*), 1390
variable_offset (*angr.ailment.expression.BinaryOp attribute*), 1385
variable_offset (*angr.ailment.expression.ComboRegister attribute*), 1379
variable_offset (*angr.ailment.expression.ITE attribute*), 1387
variable_offset (*angr.ailment.expression.Load attribute*), 1386
variable_offset (*angr.ailment.expression.UnaryOp attribute*), 1381
variable_offset (*angr.ailment.UnaryOp attribute*), 1373
variable_to_types (*angr.knowledge_plugins.variables.variable_manager.VariableManager attribute*), 813
variable_type (*angr.analyses.decompiler.structured_codegen.c.CVariable attribute*), 1139
VariableAccess (*class in angr.knowledge_plugins.variables.variable_access*), 812
VariableAccessSort (*class in angr.knowledge_plugins.variables.variable_access*), 812
VariableAnnotation (*class in angr.analyses.variable_recovery.variable_recovery_base*), 1247
VariableManager (*class in angr.knowledge_plugins*), 735
VariableManager (*class in angr.knowledge_plugins.variables*), 811
VariableManager (*class in angr.knowledge_plugins.variables.variable_manager*), 819
VariableManagerInternal (*class in angr.knowledge_plugins.variables.variable_manager*), 813
VariableManagerSerializer (*class in angr.angrdb.serializers.variables*), 1030
VariableProxiNodeSimplifier.Case (*class in angr.analyses.proximity_graph*), 1310
VariableRecovery (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
VariableRecovery (*class in angr.analyses*), 925
VariableRecovery (*class in angr.analyses.variable_recovery*), 1257
VariableRecoveryBase (*class in angr.analyses.variable_recovery.variable_recovery*), 1255
VariableRecoveryBase (*class in angr.analyses.variable_recovery.variable_recovery_base*), 1247
VariableRecoveryFast (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
VariableRecoveryFast (*class in angr.analyses*), 925
VariableRecoveryFast (*class in angr.analyses.variable_recovery*), 1258
VariableRecoveryFast (*class in angr.analyses.variable_recovery.variable_recovery_fast*), 1252
VariableRecoveryFastState (*class in angr.analyses.variable_recovery.variable_recovery_fast*), 1251
VariableRecoveryState (*class in*

angr.analyses.variable_recovery.variable_recovery), 1254
VariableRecoveryStateBase (*class in angr.analyses.variable_recovery.variable_recovery_base*), 1248
VARIABLES (*angr.analyses.complete_calling_conventions.CallingConvention.AnalysisMode attribute*), 953
variables (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase property*), 1250
variables (*angr.knowledge_base.KnowledgeBase attribute*), 711
variables (*angr.KnowledgeBase attribute*), 164
variables (*angr.state_plugins.sim_action_object.SimActionObject property*), 641
variables (*angr.state_plugins.SimActionObject property*), 265
variables (*angr.storage.memory_object.SimMemoryObject property*), 451
variables (*angr.storage.SimMemoryObject property*), 425
variables() (*angr.state_plugins.SimSolver method*), 296
variables() (*angr.state_plugins.solver.SimSolver method*), 370
variables_in_use (*angr.analyses.decompiler.structured_codegen.CFunction attribute*), 1132
variables_unify() (*angr.analyses.decompiler.CStructuredCodeGenerator method*), 1042
variables_unify() (*angr.analyses.decompiler.structured_codegen.base.StructuredCodeGenerator method*), 1147
variables_unify() (*angr.analyses.decompiler.structured_codegen.CStructuredCodeGenerator method*), 1123
VariableSourceAnnotation (*class in angr.analyses.variable_recovery.annotations*), 1246
VariableType (*class in angr.knowledge_plugins.variables*), 811
VariableType (*class in angr.knowledge_plugins.variables.variable_manager*), 813
variance (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode attribute*), 1262
variance (*angr.analyses.typehoon.typevars.BaseLabel property*), 1271
variance (*angr.analyses.typehoon.typevars.Store property*), 1271
variant (*angr.ailment.expression.Let attribute*), 1396
varid (*angr.ailment.expression.VirtualVariable attribute*), 1379
varid (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable attribute*), 840
Vault (*class in angr.vaults*), 871
VaultDict (*class in angr.vaults*), 872
VaultDir (*class in angr.vaults*), 872
VaultDirShelf (*class in angr.vaults*), 872
VaultPickler (*class in angr.vaults*), 871
VaultShelf (*class in angr.vaults*), 872
VaultUnpickler (*class in angr.vaults*), 871
vector_count (*angr.ailment.BinaryOp attribute*), 1367
vector_count (*angr.ailment.expression.BinaryOp attribute*), 1385
vector_size (*angr.ailment.BinaryOp attribute*), 1367
vector_size (*angr.ailment.expression.BinaryOp attribute*), 1385
VendorPreset (*class in angr.misc.plugins*), 238
verbose_op (*angr.ailment.BinaryOp property*), 1367
verbose_op (*angr.ailment.expression.BinaryOp property*), 1385
verbose_op (*angr.ailment.expression.Call property*), 1393
verbose_op (*angr.ailment.expression.DirtyExpression property*), 1388
verbose_op (*angr.ailment.expression.FunctionLikeMacro property*), 1397
verbose_op (*angr.ailment.expression.Op property*), 1381
verbose_op (*angr.ailment.expression.Phi property*), 1380
verbose_op (*angr.ailment.expression.VEXCCallExpression property*), 1389
VerificationAnalysis (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
Veritesting (*class in angr.analyses.veritesting*), 926
Veritesting (*class in angr.analyses.veritesting*), 1284
Veritesting (*class in angr.exploration_techniques*), 568
Veritesting (*class in angr.exploration_techniques.veritesting*), 574
VeritestingError, 1284
version (*angr.analyses.flirt.FlirtSignatureParsed attribute*), 1164
VERSION (*angr.angrdb.AngrDB attribute*), 1018
VERSION (*angr.angrdb.db.AngrDB attribute*), 1020
vex (*angr.block.BlockNode property*), 160
vex (*angr.block.Block property*), 233
vex_block_addr (*angr.ailment.tagged_object.TagDict attribute*), 1408
VEX_LIFT (*angr.state_plugins.inspect.EventType attribute*), 331
vex_lift_addr (*angr.state_plugins.inspect.InspectAttrs attribute*), 334
vex_lift_addr (*angr.state_plugins.InspectAttrs attribute*), 253
vex_lift_buff (*angr.state_plugins.inspect.InspectAttrs attribute*), 334
vex_lift_buff (*angr.state_plugins.InspectAttrs attribute*), 253

vex_lift_size (*angr.state_plugins.inspect.InspectAttrs attribute*), 334
vex_lift_size (*angr.state_plugins.InspectAttrs attribute*), 253
vex_nostmt (*angr.Block property*), 161
vex_nostmt (*angr.block.Block property*), 233
vex_stmt_idx (*angr.ailment.tagged_object.TagDict attribute*), 1408
VEXCCallExpression (*class in angr.ailment.expression*), 1388
VEXExprConverter (*class in angr.ailment.converter_vex*), 1374
VEXIRSBConverter (*class in angr.ailment*), 1373
VEXIRSBConverter (*class in angr.ailment.converter_vex*), 1375
VEXIRSBScanner (*class in angr.analyses.variable_recovery.irsb_scanner*), 1257
VEXLifter (*class in angr.engines.vex*), 600
VEXMemVar (*class in angr.analyses.propagator.vex_vars*), 1175
VEXMixin (*class in angr.engines.vex*), 601
VEXReg (*class in angr.analyses.propagator.vex_vars*), 1176
VEXResilienceMixin (*class in angr.engines.vex*), 601
VEXSlicingMixin (*class in angr.engines.vex*), 601
VEXStmtConverter (*class in angr.ailment.converter_vex*), 1375
VEXStmtDetails (*class in angr.state_plugins.unicorn_engine*), 393
VEXTmp (*class in angr.analyses.propagator.vex_vars*), 1176
VEXVariable (*class in angr.analyses.propagator.vex_vars*), 1175
VFG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
VFG (*class in angr.analyses*), 878
VFG (*class in angr.analyses.vfg*), 1288
VFG_HANDLE_SUCCESSOR (*angr.state_plugins.inspect.EventType attribute*), 332
VFG_WIDEN_STATE (*angr.state_plugins.inspect.EventType attribute*), 332
VFGJob (*class in angr.analyses.vfg*), 1285
VFGNode (*class in angr.analyses.vfg*), 1288
Virtual (*angr.analyses.disassembly.SootExpressionInvoke attribute*), 1294
VirtualVariable (*class in angr.ailment.expression*), 1379
VirtualVariable (*class in angr.knowledge_plugins.key_definitions.atoms*), 840
VirtualVariableCategory (*class in angr.ailment.expression*), 1379
visited_blocks (*angr.analyses.reaching_definitions.function_handler.Fu attribute*), 1224
visited_blocks (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 1183
visited_blocks (*angr.analyses.reaching_definitions.reaching_definitions. property*), 1216
visited_blocks (*angr.analyses.reaching_definitions.ReachingDefinitions. property*), 1200
visited_blocks (*angr.analyses.ReachingDefinitionsAnalysis property*), 914
visitor (*angr.analyses.reaching_definitions.subject.Subject property*), 1238
VSA_DDG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 928
VSA_DDG (*class in angr.analyses*), 880
VSA_DDG (*class in angr.analyses.vsa_ddg*), 1291
Vtable (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 769
Vtable (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 754
Vtable (*class in angr.analyses.vtable*), 1291
VtableFinder (*class in angr.analyses*), 927
VtableFinder (*class in angr.analyses.vtable*), 1291
vvar_id (*angr.analyses.decompiler.structured_codegen.c.CVariable attribute*), 1139
vvar_region (*angr.analyses.variable_recovery.engine_base.SimEngineVR attribute*), 1257
vvar_to_vvar (*angr.analyses.decompiler.clinic.Clinic attribute*), 1060
vvar_type_hints (*angr.analyses.variable_recovery.engine_base.SimEngi attribute*), 1257

W

walk() (*angr.ailment.AILBlockWalker method*), 1365
walk() (*angr.analyses.decompiler.ailgraph_walker.AILGraphWalker method*), 1054
walk() (*angr.analyses.decompiler.expression_narrower.ExpressionNarrow method*), 1070
walk() (*angr.analyses.decompiler.region_walker.RegionWalker method*), 1119
walk() (*angr.analyses.decompiler.sequence_walker.SequenceWalker method*), 1120
walk_expression() (*angr.ailment.AILBlockWalker method*), 1365
walk_node() (*angr.analyses.decompiler.optimization_passes.ite_expr_con method*), 1098
walk_node() (*angr.analyses.decompiler.region_walker.RegionWalker method*), 1119
walk_statement() (*angr.ailment.AILBlockWalker method*), 1365
was_combo_reg (*angr.ailment.expression.VirtualVariable property*), 1379
was_parameter (*angr.ailment.expression.VirtualVariable property*), 1379

was_parameter (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 840
was_reg (*angr.ailment.expression.VirtualVariable property*), 1379
was_reg (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 840
was_stack (*angr.ailment.expression.VirtualVariable property*), 1379
was_stack (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 840
was_tmp (*angr.ailment.expression.VirtualVariable property*), 1379
was_tmp (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 840
WeakAssignment (*class in angr.ailment.statement*), 1398
When (*class in angr.state_plugins.inspect*), 339
whitelist (*angr.analyses.veritesting.CallTracingFilter attribute*), 1284
whitelist_edges (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructuring attribute*), 1038
widen() (*angr.sim_state.SimState method*), 242
widen() (*angr.SimFile method*), 180
widen() (*angr.SimFileDescriptor method*), 184
widen() (*angr.SimFileDescriptorDuplex method*), 187
widen() (*angr.SimHeapBrk method*), 191
widen() (*angr.SimHeapPTMalloc method*), 193
widen() (*angr.SimPackets method*), 199
widen() (*angr.SimState method*), 210
widen() (*angr.SimStatePlugin method*), 212
widen() (*angr.state_plugins.CallStack method*), 249
widen() (*angr.state_plugins.callstack.CallStack method*), 373
widen() (*angr.state_plugins.cgc.SimStateCGC method*), 381
widen() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 360
widen() (*angr.state_plugins.filesystem.SimFilesystem method*), 357
widen() (*angr.state_plugins.globals.SimStateGlobals method*), 386
widen() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 413
widen() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 420
widen() (*angr.state_plugins.heap.SimHeapBrk method*), 408
widen() (*angr.state_plugins.heap.SimHeapPTMalloc method*), 411
widen() (*angr.state_plugins.history.SimStateHistory method*), 376
widen() (*angr.state_plugins.inspect.SimInspector method*), 343
widen() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 402
widen() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 404
widen() (*angr.state_plugins.libc.SimStateLibc method*), 346
widen() (*angr.state_plugins.log.SimStateLog method*), 371
widen() (*angr.state_plugins.loop_data.SimStateLoopData method*), 400
widen() (*angr.state_plugins.plugin.SimStatePlugin method*), 330
widen() (*angr.state_plugins.posix.PosixDevFS method*), 348
widen() (*angr.state_plugins.posix.PosixProcFS method*), 350
widen() (*angr.state_plugins.posix.SimSystemPosix method*), 354
widen() (*angr.state_plugins.PosixDevFS method*), 260
widen() (*angr.state_plugins.PosixProcFS method*), 262
widen() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 391
widen() (*angr.state_plugins.scratch.SimStateScratch method*), 390
widen() (*angr.state_plugins.sim_action_object.SimActionObject method*), 641
widen() (*angr.state_plugins.SimActionObject method*), 265
widen() (*angr.state_plugins.SimFilesystem method*), 269
widen() (*angr.state_plugins.SimHeapBrk method*), 272
widen() (*angr.state_plugins.SimHeapPTMalloc method*), 275
widen() (*angr.state_plugins.SimInspector method*), 279
widen() (*angr.state_plugins.SimJavaVmClassloader method*), 280
widen() (*angr.state_plugins.SimMemView method*), 285
widen() (*angr.state_plugins.SimRegNameView method*), 287
widen() (*angr.state_plugins.SimSolver method*), 290
widen() (*angr.state_plugins.SimStateCGC method*), 298
widen() (*angr.state_plugins.SimStateEdgeHitmap method*), 299
widen() (*angr.state_plugins.SimStateGlobals method*), 300
widen() (*angr.state_plugins.SimStateHistory method*), 301
widen() (*angr.state_plugins.SimStateIcicle method*), 305
widen() (*angr.state_plugins.SimStateJNIReferences method*), 307
widen() (*angr.state_plugins.SimStateLibc method*), 310
widen() (*angr.state_plugins.SimStateLog method*), 312
widen() (*angr.state_plugins.SimStateLoopData method*), 313
widen() (*angr.state_plugins.SimStatePlugin method*), 315
widen() (*angr.state_plugins.SimStatePreconstrainer

method), 317
widen() (angr.state_plugins.SimStateScratch method), 319
widen() (angr.state_plugins.SimSystemPosix method), 324
widen() (angr.state_plugins.solver.SimSolver method), 364
widen() (angr.state_plugins.trace_additions.ChallRespInfo method), 383
widen() (angr.state_plugins.trace_additions.ZenPlugin method), 385
widen() (angr.state_plugins.Unicorn method), 328
widen() (angr.state_plugins.unicorn_engine.Unicorn method), 398
widen() (angr.state_plugins.view.SimMemView method), 431
widen() (angr.state_plugins.view.SimRegNameView method), 427
widen() (angr.storage.file.SimFile method), 436
widen() (angr.storage.file.SimFileDescriptor method), 446
widen() (angr.storage.file.SimFileDescriptorDuplex method), 448
widen() (angr.storage.file.SimPackets method), 440
widen() (angr.storage.file.SimPacketsSlots method), 450
widen() (angr.storage.memory_mixins.JavaVmMemoryMixin method), 466
widen() (angr.storage.memory_mixins.MemoryRegionMetaMixin method), 473
widen() (angr.storage.memory_mixins.regioned_memory.MemoryRegionMetaMixin method), 526
widen() (angr.storage.memory_mixins.regioned_memory.RegionedMemoryMixin method), 538
widen() (angr.storage.SimFile method), 425
widened_jobs (angr.analyses.forward_analysis.job_info.JobInfo property), 937
widened_state (angr.analyses.vfg.VFGNode attribute), 1288
width (angr.analyses.data_dep.dep_nodes.MemDepNode property), 1315
width (angr.analyses.data_dep.MemDepNode property), 1318
width() (angr.analyses.disassembly.DisassemblyPiece method), 1293
WinStackCanarySimplifier (class in angr.analyses.decompiler.optimization_passes), 1090
with_arch() (angr.sim_type.SimType method), 687
with_arch() (angr.sim_type.TypeRef method), 688
with_condition (angr.sim_state.SimState property), 242
with_condition (angr.SimState property), 211
with_type() (angr.state_plugins.debug_variables.SimDebugVariables method), 421
with_type() (angr.state_plugins.SimDebugVariable method), 267
with_type() (angr.state_plugins.SimMemView method), 284
with_type() (angr.state_plugins.view.SimMemView method), 430
work() (angr.analyses.complete_calling_conventions.CompleteCallingConvention method), 954
work() (angr.analyses.CompleteCallingConventionsAnalysis method), 898
Worker (class in angr.distributed.worker), 1363
WRITE (angr.knowledge_plugins.variables.variable_access.VariableAccessSize attribute), 812
Write (angr.knowledge_plugins.xrefs.xref_types.XRefType attribute), 864
Write (angr.knowledge_plugins.xrefs.XRefType attribute), 863
WRITE (angr.state_plugins.sim_action.SimActionData attribute), 639
WRITE (angr.state_plugins.SimActionData attribute), 263
write() (angr.SimFile method), 179
write() (angr.SimFileBase method), 182
write() (angr.SimFileStream method), 188
write() (angr.SimPackets method), 198
write() (angr.SimPacketsStream method), 200
write() (angr.storage.file.SimFile method), 435
write() (angr.storage.file.SimFileBase method), 433
write() (angr.storage.file.SimFileDescriptorBase method), 442
write() (angr.storage.file.SimFileStream method), 437
write() (angr.storage.file.SimPackets method), 439
write() (angr.storage.file.SimPacketsSlots method), 449
write() (angr.storage.file.SimPacketsStream method), 441
write() (angr.storage.SimFile method), 424
write_data() (angr.SimFileDescriptor method), 183
write_data() (angr.SimFileDescriptorDuplex method), 185
write_data() (angr.storage.file.SimFileDescriptor method), 444
write_data() (angr.storage.file.SimFileDescriptorBase method), 443
write_data() (angr.storage.file.SimFileDescriptorDuplex method), 447
write_msr() (angr.state_plugins.Unicorn method), 329
write_msr() (angr.state_plugins.unicorn_engine.Unicorn method), 399
write_pos (angr.SimFileDescriptor property), 183
write_pos (angr.SimFileDescriptorDuplex property), 186
write_pos (angr.storage.file.SimFileDescriptor property), 445
write_pos (angr.storage.file.SimFileDescriptorBase property), 443

write_pos (*angr.storage.file.SimFileDescriptorDuplex property*), 447
write_size (*angr.ailment.tagged_object.TagDict attribute*), 1408
write_storage (*angr.SimFileDescriptor property*), 183
write_storage (*angr.SimFileDescriptorDuplex property*), 186
write_storage (*angr.storage.file.SimFileDescriptor property*), 445
write_storage (*angr.storage.file.SimFileDescriptorBase property*), 443
write_storage (*angr.storage.file.SimFileDescriptorDuplex property*), 447
write_to() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 814
WrTmp() (*angr.ailment.converter_vex.VEXStmtConverter static method*), 1375

X

X86ElfPicPltResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1013
X86ElfPicPltResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt*), 997
X86GccGetPcSimplifier (*class in angr.analyses.decompiler.optimization_passes*), 1091
X86GccGetPcSimplifier (*class in angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier*), 1104
X86PeIatResolver (*class in angr.analyses.cfg.indirect_jump_resolvers*), 1014
X86PeIatResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat*), 995
Xor (*angr.engines.light.ArithmeticExpression attribute*), 1167
Xor (*angr.engines.light.data.ArithmeticExpression attribute*), 1166
XRef (*class in angr.knowledge_plugins.xrefs*), 861
XRef (*class in angr.knowledge_plugins.xrefs.xref*), 863
XrefDict (*class in angr.knowledge_plugins.xrefs.xref_manager*), 864
XRefManager (*class in angr.knowledge_plugins*), 736
XRefManager (*class in angr.knowledge_plugins.xrefs*), 862
XRefManager (*class in angr.knowledge_plugins.xrefs.xref_manager*), 864
XRefs (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 929
xrefs (*angr.angrdb.models.DbKnowledgeBase attribute*), 1022
xrefs (*angr.knowledge_base.KnowledgeBase attribute*), 711
xrefs (*angr.knowledge_plugins.Function property*), 718
xrefs (*angr.knowledge_plugins.functions.Function property*), 778
xrefs (*angr.knowledge_plugins.functions.function.Function property*), 803
xrefs (*angr.KnowledgeBase attribute*), 164
XRefsAnalysis (*class in angr.analyses*), 927
XRefsAnalysis (*class in angr.analyses.xrefs*), 1308
XRefsSerializer (*class in angr.angrdb.serializers.xrefs*), 1030
XRefType (*class in angr.knowledge_plugins.xrefs*), 863
XRefType (*class in angr.knowledge_plugins.xrefs.xref_types*), 864

Z

zen_hook() (*in module angr.state_plugins.trace_additions*), 384
zen_memory_write() (*in module angr.state_plugins.trace_additions*), 384
zen_register_write() (*in module angr.state_plugins.trace_additions*), 384
ZenPlugin (*class in angr.state_plugins.trace_additions*), 384