angr

The angr Project

Jan 02, 2024

CONTENTS

1 Introduction **3**
- 1.1 Getting Support . . . 3
- 1.2 Citing angr . . . 4
- 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 **15**
- 3.1 Core Concepts . . . 15
- 3.2 Loading a Binary . . . 20
- 3.3 Symbolic Expressions and Constraint Solving . . . 26
- 3.4 Machine State - memory, registers, and so on . . . 32
- 3.5 Simulation Managers . . . 38
- 3.6 Simulation and Instrumentation . . . 42
- 3.7 Analyses . . . 48
- 3.8 Symbolic Execution . . . 49
- 3.9 A final word of advice . . . 49

4 Build-in Analyses **51**
- 4.1 Control-flow Graph Recovery (CFG) . . . 51
- 4.2 Backward Slicing . . . 56
- 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? . . . 69
- 5.4 Optimization considerations . . . 72
- 5.5 Working with File System, Sockets, and Pipes . . . 74
- 5.6 Intermediate Representation . . . 79
- 5.7 Working with Data and Conventions . . . 83
- 5.8 Solver Engine . . . 86
- 5.9 Symbolic memory addressing . . . 91
- 5.10 Java Support . . . 92
- 5.11 Symbion: Interleaving symbolic and concrete execution . . . 93

5.12 Debug variable resolution . . . 95
5.13 Variable visibility . . . 96

6 Extending angr **99**
6.1 Hooks and SimProcedures . . . 99
6.2 State Plugins . . . 103
6.3 Extending the Environment Model . . . 107
6.4 Writing Analyses . . . 110
6.5 Scripting angr management . . . 112

7 angr examples **115**
7.1 Introduction . . . 115
7.2 Reversing . . . 115
7.3 Vulnerability Discovery . . . 118
7.4 Exploitation . . . 119

8 Frequently Asked Questions **121**
8.1 Why is it named angr? . . . 121
8.2 How should "angr" be stylized? . . . 121
8.3 Why isn't symbolic execution doing the thing I want? . . . 121
8.4 How can I get diagnostic information about what angr is doing? . . . 122
8.5 Why is angr so slow? . . . 122
8.6 How do I find bugs using angr? . . . 122
8.7 Why did you choose VEX instead of another IR (such as LLVM, REIL, BAP, etc)? . . . 122
8.8 Why are some ARM addresses off-by-one? . . . 123
8.9 How do I serialize angr objects? . . . 123
8.10 What does `UnsupportedIROpError("floating point support disabled")` mean? . . . 123
8.11 Why is angr's CFG different from IDA's? . . . 124
8.12 Why do I get incorrect register values when reading from a state during a SimInspect breakpoint? . . . 124

9 Appendix **125**
9.1 Cheatsheet . . . 125
9.2 List of Claripy Operations . . . 130
9.3 List of State Options . . . 131
9.4 CTF Challenge Examples . . . 134
9.5 Changelog . . . 138
9.6 Migrating to angr 9.1 . . . 150
9.7 Migrating to angr 8 . . . 150
9.8 Migrating to angr 7 . . . 153

10 API Reference **157**
10.1 Project . . . 212
10.2 Plugin Ecosystem . . . 222
10.3 Program State . . . 224
10.4 Storage . . . 309
10.5 Memory Mixins . . . 336
10.6 Concretization Strategies . . . 379
10.7 Simulation Manager . . . 382
10.8 Exploration Techniques . . . 390
10.9 Simulation Engines . . . 427
10.10 Simulation Logging . . . 466
10.11 Procedures . . . 469
10.12 Calling Conventions and Types . . . 483
10.13 Knowledge Base . . . 520
10.14 Serialization . . . 616

10.15 Analysis . . . 619
10.16 SimOS . . . 875
10.17 Function Signature Matching . . . 883
10.18 Utils . . . 884
10.19 Errors . . . 894
10.20 Distributed analysis . . . 900

11 Indices and tables **903**

Python Module Index **905**

Index **911**

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
- ailment - angr's high-level intermediate language
- cle - Many-platform binary loader
- claripy - Solver abstraction layer

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 ask via:

- the slack channel: angr.slack.com, for which you can get an account here.
- opening 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},
  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 3.8+, and must be installed into a Python environment before it can be used.

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

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
```

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`, and a C compiler. On Ubuntu, these can be installed with:

```
sudo apt-get install python3-dev build-essential
```

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

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

2.1.3 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 instanciated 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 needs a *lot* of work. Here is a non-exhaustive list of what is currently missing in angr-management:

- A navigator toolbar showing content in a program's memory space, just like IDA Pro's navigator toolbar.
- A text-based disassembly view of the program.
- Better view showing details in program states during path exploration, including modifiable register view, memory view, file descriptor view, etc.
- A GUI for cross referencing.

Exposing 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 sematic 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>

>>> 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
```

(continues on next page)

(continued from previous page)

```
<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 = state.solver.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 = state.solver.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
# 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.get_any_node
>>> entry_node = cfg.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]>,
 <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
```

(continues on next page)

(continued from previous page)

```
# 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
<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]>

>>> 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 accessable 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>,
 '_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.

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.

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* <*Gotchas when using angr*>.

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.

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 = state.solver.BVV(1, 64)
>>> one
 <BV64 0x1>
>>> one_hundred = state.solver.BVV(100, 64)
>>> one_hundred
 <BV64 0x64>

# create a 27-bit bitvector with concrete value 9
>>> weird_nine = state.solver.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 compliment signed integer semantics.

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

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

(continues on next page)

(continued from previous page)

```
>>> 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 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
<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>
```

(continues on next page)

(continued from previous page)

```
>>> 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 = state.solver.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 = state.solver.FPV(3.2, state.solver.fp.FSORT_DOUBLE)
>>> a
```

(continues on next page)

(continued from previous page)

```
<FP64 FPV(3.2, DOUBLE)>

>>> b = state.solver.FPS('b', state.solver.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 (`solver.fpAdd` for example) with a rounding mode (one of `solver.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 0x4009999999999999a>
>>> b.raw_to_bv()
<BV64 fpToIEEEBV(FPS('FP_b_0_64', DOUBLE))>

>>> state.solver.BVV(0, 64).raw_to_fp()
<FP64 FPV(0.0, DOUBLE)>
>>> state.solver.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(state.solver.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(state.solver.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'

>>> 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, s.solver.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 bizzare 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 unconst	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
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 rea
mem_write	mem_write_address	BP_BEFORE or BP_AFTER	The address at which memory is
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 writ
mem_write	mem_write_condition	BP_BEFORE or BP_AFTER	The condition of the memory wr
reg_read	reg_read_offset	BP_BEFORE or BP_AFTER	The offset of the register being re
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.
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 w
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 writ
reg_write	reg_write_condition	BP_BEFORE or BP_AFTER	The condition of the register writ
tmp_read	tmp_read_num	BP_BEFORE or BP_AFTER	The number of the temp being re
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 tem
expr	expr	BP_BEFORE or BP_AFTER	The IR expression.
expr	expr_result	BP_AFTER	The value (e.g. AST) which the e

Event type	Attribute name	Attribute availability	Attribute meaning
statement	statement	BP_BEFORE or BP_AFTER	The index of the IR statement (in
instruction	instruction	BP_BEFORE or BP_AFTER	The address of the native instruct
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
call	function_address	BP_BEFORE or BP_AFTER	The name of the function being c
exit	exit_target	BP_BEFORE or BP_AFTER	The expression representing the t
exit	exit_guard	BP_BEFORE or BP_AFTER	The expression representing the g
exit	exit_jumpkind	BP_BEFORE or BP_AFTER	The expression representing the k
symbolic_variable	symbolic_name	BP_AFTER	The name of the symbolic variab
symbolic_variable	symbolic_size	BP_AFTER	The size of the symbolic variable
symbolic_variable	symbolic_expr	BP_AFTER	The expression representing the v
address_concretization	address_concretization_strategy	BP_BEFORE or BP_AFTER	The SimConcretizationStrategy b
address_concretization	address_concretization_action	BP_BEFORE or BP_AFTER	The SimAction object being used
address_concretization	address_concretization_memory	BP_BEFORE or BP_AFTER	The SimMemory object on which
address_concretization	address_concretization_expr	BP_BEFORE or BP_AFTER	The AST representing the memo
address_concretization	address_concretization_add_constraints	BP_BEFORE or BP_AFTER	Whether or not constraints shoul
address_concretization	address_concretization_result	BP_AFTER	The list of resolved memory addr
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 simproce
simprocedure	simprocedure	BP_BEFORE or BP_AFTER	The actual SimProcedure object.
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 h
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.
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 defini

These attributes can be accessed as members of `state.inspect` 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.mem_read_expr, 'from', state.inspect.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
```

(continues on next page)

(continued from previous page)

```
>>> 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
>>> def cond(state):
...     return state.eval(state.regs.rax, cast_to=str) == 'AAAA' and 0x8004 in state.
↪inspect.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 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 allows the program to go from a point A to a point B, by resolving the constraints.

If you've read this far, you can see how the components of angr work together to make this possible. Read on to learn about how to make the leap from tools to results.

Todo: A real introduction to the concept of symbolic execution.

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 slack 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.graph)
>>> print("It has %d nodes and %d edges" % (len(cfg.graph.nodes()), len(cfg.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.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.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.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 peforms 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

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

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_back	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:
```

(continues on next page)

(continued from previous page)

```
# - 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.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) ])

# 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

User-friendly Representation

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

4.2.3 Implementation Choices

4.2.4 Limitations

Completeness

Soundness

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()
```

(continues on next page)

(continued from previous page)

```
>>> 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
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,
```

(continues on next page)

(continued from previous page)

```
↪ 0x7fffffffffefffff], 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,
↪ 0x7fffffffffefffff], 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):
        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 unsuprisingly 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 inheritence 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,
   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)

>>> 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)
[(<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 accessable 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 supplamented 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
```

(continues on next page)

(continued from previous page)

```
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>
```

(continues on next page)

(continued from previous page)

```
>>> 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:

Name	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
```

(continues on next page)

(continued from previous page)

```
>>> 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()`.

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

Enabling Java support requires few more steps than typical angr installation. Assuming you installed angr-dev, activate the virtualenv and run:

```
pip install -e ./claripy[cvc4-solver]
./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 Symbion: Interleaving symbolic and concrete execution

Let's suppose you want to symbolically analyze a specific function of a program, but there is a huge initialization step that you want to skip because it is not necessary for your analysis, or cannot properly be emulated by angr. For example, maybe your program is running on an embedded system and you have access to a debug interface, but you can't easily replicate the hardware in a simulated environment.

This is the perfect scenario for `Symbion`, our interleaved execution technique!

We implemented a built-in system that let users define a `ConcreteTarget` that is used to "import" a concrete state of the target program from an external source into `angr`. Once the state is imported you can make parts of the state symbolic, use symbolic execution on this state, run your analyses, and finally concretize the symbolic parts and resume concrete execution in the external environment. By iterating this process it is possible to implement run-time and interactive advanced symbolic analyses that are backed up by the real program's execution!

Isn't that cool?

5.11.1 How to install

To use this technique you'll need an implementation of a `ConcreteTarget` (effectively, an object that is going to be the "glue" between angr and the external process.) We ship a default one (the AvatarGDBConcreteTarget, which control an instance of a program being debugged under GDB) in the following repo https://github.com/angr/angr-targets.

Assuming you installed angr-dev, activate the virtualenv and run:

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

Now you're ready to go!

5.11.2 Gists

Once you have created an entry state, instantiated a `SimulationManager`, and specified a list of *stop_points* using the `Symbion` interface we are going to resume the concrete process execution.

```
# Instantiating the ConcreteTarget
avatar_gdb = AvatarGDBConcreteTarget(avatar2.archs.x86.X86_64,
                                     GDB_SERVER_IP, GDB_SERVER_PORT)

# Creating the Project
p = angr.Project(binary_x64, concrete_target=avatar_gdb,
                             use_sim_procedures=True)

# Getting an entry_state
entry_state = p.factory.entry_state()

# Forget about these options as for now, will explain later.
entry_state.options.add(angr.options.SYMBION_SYNC_CLE)
entry_state.options.add(angr.options.SYMBION_KEEP_STUBS_ON_SYNC)

# Use Symbion!
simgr.use_technique(angr.exploration_techniques.Symbion(find=[0x85b853])
```

When one of your stop_points (effectively a breakpoint) is hit, we give control to `angr`. A new plugin called *concrete* is in charge of synchronizing the concrete state of the program inside a new `SimState`.

Roughly, synchronization does the following:

- All the registers' values (NOT marked with concrete=False in the respective arch file in archinfo) are copied inside the new SimState.
- The underlying memory backend is hooked in a way that all the further memory accesses triggered during symbolic execution are redirected to the concrete process.
- If the project is initialized with SimProcedure (use_sim_procedures=True) we are going to re-hook the external functions' addresses with a `SimProcedure` if we happen to have it, otherwise with a `SimProcedure` stub (you can control this decision by using the Options SYMBION_KEEP_STUBS_ON_SYNC). Conversely, the real code of the function is executed inside angr (Warning: do that at your own risk!)

Once this process is completed, you can play with your new `SimState` backed by the concrete process stopped at that particular stop_point.

5.11.3 Options

The way we synchronize the concrete process inside angr is customizable by 2 state options:

- **SYMBION_SYNC_CLE**: this option controls the synchronization of the memory mapping of the program inside angr. When the project is created, the memory mapping inside angr is different from the one inside the concrete process (this will change as soon as Symbion will be fully compatible with archr). If you want the process mapping to be fully synchronized with the one of the concrete process, set this option to the SimState before initializing the SimulationManager (Note that this is going to happen at the first synchronization of the concrete process inside angr, NOT before)

  ```
  entry_state.options.add(angr.options.SYMBION_SYNC_CLE)
  simgr = project.factory.simgr(state)
  ```

- **SYMBION_KEEP_STUBS_ON_SYNC**: this option controls how we re-hook external functions with SimProcedures. If the project has been initialized to use SimProcedures (use_sim_procedures=True), we are going to re-hook external functions with SimProcedures (if we have that particular implementation) or with a generic stub. If you want to execute SimProcedures for functions for which we have an available implementation and a generic stub SimProcedure for the ones we have not, set this option to the SimState before initializing the SimulationManager. In the other case, we are going to execute the real code for the external functions that miss a SimProcedure (no generic stub is going to be used).

```
entry_state.options.add(angr.options.SYMBION_KEEP_STUBS_ON_SYNC)
simgr = project.factory.simgr(state)
```

5.11.4 Example

You can find more information about this technique and a complete example in our blog post: https://angr.io/blog/angr_symbion/. For more technical details a public paper will be available soon, or, ping @degrigis on our angr Slack channel.

5.12 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.

5.12.1 Setting up

To use it you need binary that is compiled with dwarf debuging 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()
```

5.12.2 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){
```

(continues on next page)

(continued from previous page)

```
   int a = 10;
   int* b = &a;
   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.

5.13 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]
```

(continues on next page)

(continued from previous page)

```
# 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:
```

(continues on next page)

(continued from previous page)

```
            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 intializers 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 implementaion 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'

>>> 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 instanciated 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']` or `angr.SIM_LIBRARIES['ld-linux.so']` 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].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 implemementation 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'].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!

6.5 Scripting angr management

Warning: Please note that the documentation and the API for angr management are highly in-flux. You will need to spend time reading the source code. Grep is your friend. If you have questions, please ask in the angr slack.

If you build something which uses an API and you want to make sure it doesn't break, you can contribute a testcase for the API!

This codebase is absolutely filled to the brim with one-off hacks. If you see some code and think, "hm, that doesn't seem like an extensible or best-practices way to code that", you're probably right. Cleaning up angr management's code is a top priority for us, so if you have some ideas to fix these sorts of issues, please let us know, either in an issue or a pull request!

6.5.1 The console, and the basic objects

angr management opens with an IPython console ready for input. This console has in its namespace several objects which are important for manipulating angr management and its data.

- First, the `main_window`. This is the `QMainWindow` instance for the application. It contains basic functions that correspond to top-level buttons, such as loading a binary.
- Next, the `workspace`. This is a light object which coordinates the UI elements and manages the tabbed environment. You can use it to access any analysis-related GUI element, such as the disassembly view.
- Finally, the `instance`. This is angr management's data model. It contains mechanisms for synchronizing components on shared data sources, as well as logic for creating long-running jobs.

`workspace` is also available as an attribute on `main_window` and `instance` is available as an attribute on `workspace`. If you are programming in a namespace where none of these objects are available, you can import the `angrmanagment.logic.GlobalInfo` object, which contains a reference to `main_window`.

6.5.2 The ObjectContainer

angr management uses a class called ObjectContainer to implement a pub-sub model and synchronize changing object references. Let's use `instance.project` as an example. This is an ObjectContainer that contains the current project. You can use it in every way that you would normally use a project - you can access `project.factory`, `project.kb`, etc. However, it also has two very important features that are helpful for building UIs.

First, the pub-sub model. You can subscribe to changes to this object by calling `instance.project.am_subscribe(callback)`. Then, you can notify listeners of changes by calling `instance.project.am_event()`. Note that events are NEVER automatically triggered - you must call `am_event` in order to trigger the callbacks. One useful feature of this model is that you can provide arbitrary keyword arguments to `am_event`, and they will be passed on to each callback. This means that you should always have your callbacks take `**kwargs` in order to account for unknown parameters. This feature is particularly useful to prevent feedback loops - if you ever find yourself in a situation where you need to broadcast an event from your callback, you can add an argument that you can use as a flag not to recurse any further.

Next, object reference mutability. Let's say you have a widget that displays information about the project. Following the principle of least access, you should only provide as much information as is necessary to do the job - in this case, just the project object. If you provide the basic project object, this will cause issues when a new project is loaded. Notably, there will be a dangling reference held to the original project, preventing it from being garbage collected, and the widget will not update, continuing to show the old project's information. Now, if you provide the project's ObjectContainer, a new project can be created and inserted into the container and the reference will instantly be available to your widget. If you ever wanted to load a new project yourself, all you have to do is assign to `instance.project.am_obj` and then send off an event. Combined with the event publication model, this provides an efficient way to build responsive UIs that follow the principle of least access.

One important way that you can't use the object container the same way that you would a normal object is that `is None` will obviously not work. To resolve this, you can use `instance.project.am_none` - this will be True when no project is loaded.

One interesting feature of the ObjectContainer is that they can nest. If you have a container which contains a container which contains an object, any events sent to the inner container will also be sent to subscribers to the outer container. This allows patterns such as the list of SimStates actually containing a list of ObjectContainers which contain states, and the "current state" container actually contains one of these containers. The result of this is that UI elements can either subscribe to the current state, no matter

A full list of standard ObjectContainers that can be found in the instance __init__ method. There are more containers floating around for synchronizing on non-global elements - for example, the current state of the disassembly view is synchronized through its InfoDock object. Given a disassembly view instance, you can subscribe to, for example, its current selected instructions through `view.infodock.selected_insns`.

6.5.3 Manipulating UI elements

The `workspace` contains methods to manipulate UI elements. Notably, you can manipulate all open tabs with the workspace.view_manager reference. Additionally, you can pass any sort of object you like to `workspace.viz()` and it will attempt to visualize the object in the current window.

6.5.4 Writing plugins

angr management has a very flexible plugin framework. A plugin is a Python file containing a subclass of `angrmanagement.plugins.BasePlugin`. Plugin files will be automatically loaded from the `plugins` module of angr management, and also from `~/.local/share/angr-management/plugins`. These paths are configurable through the program configuration, but at the time of writing, this is not exposed in the UI.

The best way to see the tools you can use while building a plugin is to read the plugin base class source code. Any method or attribute can be overridden from a base class and will be automatically called on relevant events.

6.5.5 Writing tests

Look at the existing tests for examples. Generally, you can test UI components by creating the component and driving input to it via QTest. You can create a headless MainWindow instance by passing `show=False` to its constructor - this will also get you access to a workspace and an instance.

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: unusal 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
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 **:ref:`install instructions <Installing angr>`_**.

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 Managment 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 recieved (useful for debugging scripts that are running forever)

```
import signal
def killmyself():
    os.system('kill %d' % os.getpid())
def sigint_handler(signum, frame):
    print 'Stopping Execution for Debug. If you want to kill the programm 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␣
↪dissassembly
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):
    print 'Read', state.inspect.mem_read_expr, 'from', state.inspect.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)	`solver.And(x == y, x > 0)`
Or	Logical Or (on boolean expressions)	`solver.Or(x == y, y < 10)`
Not	Logical Not (on a boolean expression)	`solver.Not(x == y)` is the same as `x != y`
If	An If-then-else	Choose the maximum of two expressions: `solver.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
ABSTRACT_MEMORY	Use `SimAbstractMemory` to model memory as discrete regions
ABSTRACT_SOLVER	Allow splitting constraint sets during simplification
ACTION_DEPS	Track dependencies in SimActions
APPROXIMATE_GUARDS	Use VSA when evaluating guard conditions
APPROXIMATE_MEMORY_INDICES	Use VSA when evaluating memory indices
APPROXIMATE_MEMORY_SIZES	Use VSA when evaluating memory load/store sizes
APPROXIMATE_SATISFIABILITY	Use VSA when evaluating state satisfiability
AST_DEPS	Enables dependency tracking for all claripy ASTs
AUTO_REFS	An internal option used to track dependencies in SimProcedures
AVOID_MULTIVALUED_READS	Return a symbolic value without touching memory for any read that has a symbolic a
AVOID_MULTIVALUED_WRITES	Do not perfrom any write that has a symbolic address
BEST_EFFORT_MEMORY_STORING	Handle huge writes of symbolic size by pretending they are actually smaller
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
BREAK_SIRSTMT_END	Debug: trigger a breakpoint at the end of each IR statement
BREAK_SIRSTMT_START	Debug: trigger a breakpoint at the start of each IR statement
BYPASS_ERRORED_IRCCALL	Treat clean helpers that fail with errors as returning unconstrained symbolic values
BYPASS_ERRORED_IROP	Treat operations that fail with errors as returning unconstrained symbolic values
BYPASS_UNSUPPORTED_IRCCALL	Treat unsupported clean helpers as returning unconstrained symbolic values
BYPASS_UNSUPPORTED_IRDIRTY	Treat unsupported dirty helpers as returning unconstrained symbolic values
BYPASS_UNSUPPORTED_IREXPR	Treat unsupported IR expressions as returning unconstrained symbolic values
BYPASS_UNSUPPORTED_IROP	Treat unsupported operations as returning unconstrained symbolic values
BYPASS_UNSUPPORTED_IRSTMT	Treat unsupported IR statements as returning unconstrained symbolic values
BYPASS_UNSUPPORTED_SYSCALL	Treat unsupported syscalls as returning unconstrained symbolic values
BYPASS_VERITESTING_EXCEPTIONS	Discard emulation errors during veritesting
CACHELESS_SOLVER	enable `SolverCacheless`
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_FDS	CGC: always report “data available” in fdwait
CGC_NO_SYMBOLIC_RECEIVE_LENGTH	CGC: always read the maximum amount of data requested in the receive syscall
COMPOSITE_SOLVER	Enable `SolverComposite` for independent constraint set optimization
CONCRETIZE	Concretize all symbolic expressions encountered during emulation
CONCRETIZE_SYMBOLIC_FILE_READ_SIZES	Concreteize the sizes of file reads
CONCRETIZE_SYMBOLIC_WRITE_SIZES	Concretize the sizes of symbolic writes to memory
CONSERVATIVE_READ_STRATEGY	Do not use SimConcretizationStrategyAny for reads; in case of read address concreti
CONSERVATIVE_WRITE_STRATEGY	Do not use SimConcretizationStrategyAny for writes; in case of write address concre
CONSTRAINT_TRACKING_IN_SOLVER	Set `track=True` for making claripy Solvers; enable use of `unsat_core`
COW_STATES	Copy states instead of mutating the initial state directly
DOWNSIZE_Z3	Downsize the claripy solver whenever possible to save memory
DO_CCALLS	Perform IR clean calls
DO_GETS	Perform IR register reads
DO_LOADS	Perform IR memory loads
DO_OPS	Perform IR computation operations
DO_PUTS	Perform IR register writes
DO_RET_EMULATION	For each `Ijk_Call` successor, add a corresponding `Ijk_FakeRet` successor
DO_STORES	Perform IR memory stores
EFFICIENT_STATE_MERGING	Keep in memory any state that might be a common ancestor in a merge

Option name	Description
ENABLE_NX	When in conjunction with STRICT_PAGE_ACCESS, raise a SimSegfaultException on e
EXCEPTION_HANDLING	Ask all SimExceptions raised during execution to be handled by the SimOS
FAST_MEMORY	Use SimFastMemory for memory storage
FAST_REGISTERS	Use SimFastMemory for register storage
INITIALIZE_ZERO_REGISTERS	Treat the initial value of registers as zero instead of unconstrained symbolic
KEEP_IP_SYMBOLIC	Don't try to concretize successor states with symbolic instruction pointers
KEEP_MEMORY_READS_DISCRETE	In abstract memory, handle failed loads by returning a DCIS?
LAZY_SOLVES	Don't check satisfiability until absolutely necessary
MEMORY_SYMBOLIC_BYTES_MAP	Maintain a mapping of symbolic variable to which memory address it "really" corres
NO_SYMBOLIC_JUMP_RESOLUTION	Do not attempt to flatten symbolic-ip successors into discrete targets
NO_SYMBOLIC_SYSCALL_RESOLUTION	Do not attempt to flatten symbolic-syscall-number successors into discrete targets
OPTIMIZE_IR	Use LibVEX's optimization
REGION_MAPPING	Maintain a mapping of symbolic variable to which memory region it corresponds to,
REPLACEMENT_SOLVER	Enable SolverReplacement
REVERSE_MEMORY_HASH_MAP	Maintain a mapping from AST hash to which addresses it is present in
REVERSE_MEMORY_NAME_MAP	Maintain a mapping from symbolic variable name to which addresses it is present in,
SIMPLIFY_CONSTRAINTS	Run added constraints through z3's simplifcation
SIMPLIFY_EXIT_GUARD	Run branch guards through z3's simplification
SIMPLIFY_EXIT_STATE	Perform simplification on all successor states generated
SIMPLIFY_EXIT_TARGET	Run jump/call/branch targets through z3's simplification
SIMPLIFY_EXPRS	Run the results of IR expressions through z3's simplification
SIMPLIFY_MEMORY_READS	Run the results of memory reads through z3's simplification
SIMPLIFY_MEMORY_WRITES	Run values stored to memory through z3's simplification
SIMPLIFY_REGISTER_READS	Run values read from registers through z3's simplification
SIMPLIFY_REGISTER_WRITES	Run values written to registers through z3's simplification
SIMPLIFY_RETS	Run values returned from SimProcedures through z3's simplification
STRICT_PAGE_ACCESS	Raise a SimSegfaultException when attempting to interact with memory in a way not
SUPER_FASTPATH	Only execute the last four instructions of each block
SUPPORT_FLOATING_POINT	When disabled, throw an UnsupportedIROpError when encountering floating point o
SYMBOLIC	Enable constraint solving?
SYMBOLIC_INITIAL_VALUES	make state.solver.Unconstrained return a symbolic value instead of zero
SYMBOLIC_TEMPS	Treat each IR temporary as a symbolic variable; treat stores to them as constraint add
SYMBOLIC_WRITE_ADDRESSES	Allow writes with symbolic addresses to be processed by concretization strategies; w
TRACK_CONSTRAINTS	When disabled, don't keep any constraints added to the state
TRACK_CONSTRAINT_ACTIONS	Keep a SimAction for each constraint added
TRACK_JMP_ACTIONS	Keep a SimAction for each jump or branch
TRACK_MEMORY_ACTIONS	Keep a SimAction for each memory read and write
TRACK_MEMORY_MAPPING	Keep track of which pages are mapped into memory and which are not
TRACK_OP_ACTIONS	Keep a SimAction for each IR operation
TRACK_REGISTER_ACTIONS	Keep a SimAction for each register read and write
TRACK_SOLVER_VARIABLES	Maintain a listing of all the variables in all the constraints in the solver
TRACK_TMP_ACTIONS	Keep a SimAction for each temporary variable read and write
TRUE_RET_EMULATION_GUARD	With DO_RET_EMULATION, add fake returns with guard condition true instead of false
UNDER_CONSTRAINED_SYMEXEC	Enable under-constrained symbolic execution
UNICORN	Use unicorn engine to execute symbolically when data is concrete
UNICORN_AGGRESSIVE_CONCRETIZATION	Concretize any register variable unicorn tries to access
UNICORN_HANDLE_TRANSMIT_SYSCALL	CGC: handle the transmit syscall without leaving unicorn
UNICORN_SYM_REGS_SUPPORT	Attempt to stay in unicorn even in the presence of symbolic registers by checking that
UNICORN_THRESHOLD_CONCRETIZATION	Concretize variables if they prevent unicorn from executing too often
UNICORN_TRACK_BBL_ADDRS	Keep state.history.bbl_addrs up to date when using unicorn

Option name	Description
UNICORN_TRACK_STACK_POINTERS	Track a list of the stack pointer's value at each block in state.scratch.stack_poi
UNICORN_ZEROPAGE_GUARD	Prevent unicorn from mapping the zero page into memory
UNINITIALIZED_ACCESS_AWARENESS	Broken/unused?
UNSUPPORTED_BYPASS_ZERO_DEFAULT	When using the resilience options, return zero instead of an unconstrained symbol
USE_SIMPLIFIED_CCALLS	Use a "simplified" set of ccalls optimized for specific cases
USE_SYSTEM_TIMES	In library functions and syscalls and hardware instructions accessing clock data, retri
VALIDATE_APPROXIMATIONS	Debug: When performing approximations, ensure that the approximation is sound by
ZERO_FILL_UNCONSTRAINED_MEMORY	Make the value of memory read from an uninitialized address zero instead of an unco

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 instanciated 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, conrol 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 instrution traces without installing another repostory! (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 substancially, 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 instanciated 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 constructued 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 equivilant 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 substancially 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.**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

The following parameters are optional:

Parameters

- **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** – The SimCC to use for this procedure
- **sim_kwargs** – Additional keyword arguments to be passed to run()
- **is_function** – Whether this procedure emulates a function

The following class variables should be set if necessary when implementing a new SimProcedure:

Variables

- **NO_RET** – Set this to true if control flow will never return from this function
- **DYNAMIC_RET** – Set this to true if whether the control flow returns from this function or not depends on the context (e.g., libc's error() call). Must implement dynamic_returns() method.
- **ADDS_EXITS** – Set this to true if you do any control flow other than returning
- **IS_FUNCTION** – Does this procedure simulate a function? True by default

- **ARGS_MISMATCH** – Does this procedure have a different list of arguments than what is provided in the function specification? This may happen when we manually extract arguments in the run() method of a SimProcedure. False by default.
- **local_vars** – If you use `self.call()`, set this to a list of all the local variable names in your class. They will be restored on return.

The following instance variables are available when working with simprocedures from the inside or the outside:

Variables

- **project** – The associated angr project
- **arch** – The associated architecture
- **addr** – The linear address at which the procedure is executing
- **cc** – The calling convention in use for engaging with the ABI
- **canonical** – The canonical version of this SimProcedure. Procedures are deepcopied for many reasons, including to be able to store state related to a specific run and to be able to hook continuations.
- **kwargs** – Any extra keyword arguments used to construct the procedure; will be passed to `run`
- **display_name** – See the eponymous parameter
- **library_name** – See the eponymous parameter
- **abi** – If this is a syscall simprocedure, which ABI are we using to map the syscall numbers?
- **symbolic_return** – See the eponymous parameter
- **syscall_number** – If this procedure is a syscall, the number will be populated here.
- **returns** – See eponymous parameter and NO_RET cvar
- **is_syscall** – See eponymous parameter
- **is_function** – See eponymous parameter and cvar
- **is_stub** – See eponymous parameter
- **is_continuation** – Whether this procedure is the original or a continuation resulting from `self.call()`
- **continuations** – A mapping from name to each known continuation
- **run_func** – The name of the function implementing the procedure. "run" by default, but different in continuations.
- **num_args** – The number of arguments to the procedure. If not provided in the parameter, extracted from the definition of `self.run`

The following instance variables are only used in a copy of the procedure that is actually executing on a state:

Variables

- **state** – The SimState we should be mutating to perform the procedure
- **successors** – The SimSuccessors associated with the current step
- **arguments** – The function arguments, deserialized from the state
- **arg_session** – The ArgSession that was used to parse arguments out of the state, in case you need it for varargs

- **use_state_arguments** – Whether we're using arguments extracted from the state or manually provided
- **ret_to** – The current return address
- **ret_expr** – The computed return value
- **call_ret_expr** – The return value from having used `self.call()`
- **inhibit_autoret** – Whether we should avoid automatically adding an exit for returning once the run function ends
- **arg_session** – The ArgSession object that was used to extract the runtime argument values. Useful for if you want to extract variadic args.

__init__(*project=None*, *cc=None*, *prototype=None*, *symbolic_return=None*, *returns=None*, *is_syscall=False*, *is_stub=False*, *num_args=None*, *display_name=None*, *library_name=None*, *is_function=None*, ***kwargs*)

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!

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 construtor

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.

ty_ptr(*ty*)

property is_java

property argument_types

property return_type

class angr.BP(*when='before'*, *enabled=None*, *condition=None*, *action=None*, ***kwargs*)

Bases: object

A breakpoint.

__init__(*when='before'*, *enabled=None*, *condition=None*, *action=None*, ***kwargs*)

check(*state*, *when*)

Checks state *state* to see if the breakpoint should fire.

Parameters

- **state** – The state.
- **when** – Whether the check is happening before or after the event.

Returns

A boolean representing whether the checkpoint should fire.

fire(*state*)

Trigger the breakpoint.

Parameters

state – The state.

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__()

set_state(*state*)

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

set_strongref_state(*state*)

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 instanciate 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.

static memo(*f*)

A decorator function you should apply to copy

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 state.solver.ite_cases which will help with constructing arbitrarily large merged ASTs. Use it like self.bar = self.state.solver.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 plugin

Returns

True if the state plugin is actually widened.

Return type

bool

classmethod register_default(*name*, *xtr=None*)

init_state()

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

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*, ***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.
- **arch** (*Arch*) –
- **load_options** (*Dict[str, Any] | None*) –
- **selfmodifying_code** (*bool*) –
- **support_selfmodifying_code** (*bool | None*) –

The following parameters are optional.

Parameters

- **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*) –

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*) –

__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*, ***kwargs*)

Parameters

- **load_options** (*Dict[str, Any] | None*) –
- **selfmodifying_code** (*bool*) –
- **support_selfmodifying_code** (*bool | None*) –

arch: *Arch*

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** (`Optional[bool]`) – 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
`Optional[SimProcedure]`

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** (Optional[bool]) – 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
Optional[*SimProcedure*]

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().

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** (Union[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.**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*)

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** (*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*) –

__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*)

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.
- **max_predecessors** (*int*) –

Returns
None

property slice

dbg_repr(*arch=None*)

class angr.**SimOS**(*project*, *name=None*)

Bases: object

A class describing OS/arch-level configuration.

Parameters
project (angr.Project) –

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

Parameters
project (Project) –

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*)

syscall_abi(*state*)

Return type
str

syscall_cc(*state*)

Return type
Optional[*SimCCSyscall*]

is_syscall_addr(*addr*)

syscall_from_addr(*addr*, *allow_unsupported=True*)

syscall_from_number(*number*, *allow_unsupported=True*, *abi=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.**Block**(*addr*, *project=None*, *arch=None*, *size=None*, *byte_string=None*, *vex=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*, *initial_regs=None*)

Bases: *Serializable*

Represents a basic block in a binary or a program.

BLOCK_MAX_SIZE = 4096

__init__(*addr*, *project=None*, *arch=None*, *size=None*, *byte_string=None*, *vex=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*, *initial_regs=None*)

arch

thumb

addr

size

pp(***kwargs*)

set_initial_regs()

static reset_initial_regs()

property vex: *IRSB*

property vex_nostmt

property disassembly: *DisassemblerBlock*

Provide a disassembly object using whatever disassembler is available

property capstone

property codenode

property bytes

property instructions

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*)

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.**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

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 ErroredState 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 aproximation. 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.**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 = []

named_errors = {}

angr.**register_analysis**(*cls*, *name*)

class angr.**ExplorationTechnique**

Bases: object

An otiegnqwvk is a set of hooks for a simulation manager that assists in the implementation of new techniques in symbolic exploration.

TODO: choose actual name for the functionality (techniques? strategies?)

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.**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)

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*

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

Parameters

- **project** (angr.Project) – The project instance.
- **arch** (*archinfo.Arch* | *str*) – The architecture of the state.

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*)

property plugins

property se

Deprecated alias for *solver*

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

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

T = ~T

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

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*)

prepare_callsite(*retval*, *args*, *cc='wtf'*)

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

angr.**default_cc**(*arch*, *platform='Linux'*, *language=None*, *syscall=False*, ***kwargs*)

Return the default calling convention for a given architecture, platform, and language combination.

Parameters

- **arch** (str) – The architecture name.
- **platform** (Optional[str]) – The platform name (e.g., "Linux" or "Win32").
- **language** (Optional[str]) – The programming language name (e.g., "go").
- **syscall** (bool) – Return syscall convention (True), or normal calling convention (False, default).

Return type

Optional[Type[*SimCC*]]

Returns

A default calling convention class if we can find one for the architecture, platform, and language combination, or None if nothing fits.

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

Bases: object

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

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 (*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

RETURN_VAL: *SimFunctionArgument* = None

OVERFLOW_RETURN_VAL: Optional[*SimFunctionArgument*] = None

FP_RETURN_VAL: Optional[*SimFunctionArgument*] = None

ARCH = None

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

cc

fp_iter

int_iter

both_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.

Parameters
ret_ty (SimType | *None*) –

return_in_implicit_outparam(*ty*)

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*)

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'*)

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[*SimFunctionArgument*]) – 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.
- **platform** (*str*) –

Return type
Optional[*SimCC*]

Returns
A calling convention instance, or None if none of the SimCC subclasses seems to fit the arguments provided.

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 :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.**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=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

`state: angr.SimState`

`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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: **angr.SimState**

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

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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

state: angr.SimState

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

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()

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*

structured_code: *StructuredCodeManager*

defs: *KeyDefinitionManager*

cfgs: *CFGManager*

types: *TypesStore*

propagations: PropagationManager

xrefs: *XRefManager*

__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*)

K = ~K

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: :rtype: `Optional`[`TypeVar`(K, bound= *KnowledgeBasePlugin*)] :return: Instance of the requested plugin class or null if it is not a known plugin

Parameters
requested_plugin_cls (*Type[K]*) –

Return type
K | None

request_knowledge(*requested_plugin_cls*)

Return type
TypeVar(K, bound= *KnowledgeBasePlugin*)

Parameters
requested_plugin_cls (*Type[K]*) –

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** (Union[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*, ***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.
- **arch** (*Arch*) –
- **load_options** (*Dict[str, Any] | None*) –
- **selfmodifying_code** (*bool*) –
- **support_selfmodifying_code** (*bool | None*) –

The following parameters are optional.

Parameters

- **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*) –

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*) –

__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*, **kwargs*)

Parameters

- **load_options** (*Dict[str, Any] | None*) –
- **selfmodifying_code** (*bool*) –
- **support_selfmodifying_code** (*bool | None*) –

arch: Arch

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** (Optional[bool]) – 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
Optional[*SimProcedure*]

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** (Optional[bool]) – 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
Optional[*SimProcedure*]

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().

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

Bases: object

This factory provides access to important analysis elements.

Parameters
default_engine (*Type[SimEngine] | None*) –

__init__(*project*, *default_engine=None*)

Parameters
default_engine (*Type[SimEngine] | None*) –

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.

The following parameters are optional.

Parameters

- **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 sequencial allocations happen at increasing addresses.

simulation_manager(*thing=None*, ***kwargs*)

Constructs a new simulation manager.

Parameters

- **thing** (Union[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
angr.sim_manager.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*)

A Callable is a representation of a function in the binary that can be interacted with like a native python function.

Parameters

- **addr** – The address of the function to use
- **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

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*, *vex=None*, *thumb=False*, *backup_state=None*, *extra_stop_points=None*, *opt_level=None*, *num_inst=None*, *traceflags=0*, *insn_bytes=None*, *insn_text=None*, *strict_block_end=None*, *collect_data_refs=False*, *cross_insn_opt=True*, *load_from_ro_regions=False*, *initial_regs=None*)

fresh_block(*addr*, *size*, *backup_state=None*)

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

Bases: object

Helper class to represent a block of dissassembled 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.**Block**(*addr*, *project=None*, *arch=None*, *size=None*, *byte_string=None*, *vex=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*, *initial_regs=None*)

Bases: *Serializable*

Represents a basic block in a binary or a program.

BLOCK_MAX_SIZE = 4096

__init__(*addr*, *project=None*, *arch=None*, *size=None*, *byte_string=None*, *vex=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*, *initial_regs=None*)

arch

thumb

addr

size

pp(***kwargs*)

set_initial_regs()

static reset_initial_regs()

property vex: *IRSB*

property vex_nostmt

property disassembly: *DisassemblerBlock*

Provide a disassembly object using whatever disassembler is available

property capstone

property codenode

property bytes

property instructions

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*)

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

Bases: object

Represents a Soot IR basic block.

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

property soot

property size

property codenode

10.2 Plugin Ecosystem

class angr.misc.plugins.**PluginHub**

Bases: Generic[P]

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__()

classmethod register_default(*name*, *plugin_cls*, *preset='default'*)

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.

property plugin_preset

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.

discard_plugin_preset()

Discard the current active preset. Will release any active plugins that could have come from the old preset.

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.

register_plugin(*name*, *plugin*)

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

Parameters
name (*str*) –

release_plugin(*name*)

Deactivate and remove the plugin with name `name`.

class angr.misc.plugins.**PluginPreset**

Bases: object

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__()

activate(*hub*)

This method is called when the preset becomes active on a hub.

deactivate(*hub*)

This method is called when the preset is discarded from the hub.

add_default_plugin(*name*, *plugin_cls*)

Add a plugin to the preset.

list_default_plugins()

Return a list of the names of available default plugins.

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.

class angr.misc.plugins.**PluginVendor**

Bases: `Generic[P]`, *PluginHub*[P]

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 instanciating them.

release_plugin(*name*)

Deactivate and remove the plugin with name `name`.

register_plugin(*name*, *plugin*)

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

class angr.misc.plugins.**VendorPreset**

Bases: *PluginPreset*

A specialized preset class for use with the PluginVendor.

10.3 Program State

angr.sim_state.**arch_overrideable**(*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*

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

Parameters

- **project** (`angr.Project`) – The project instance.
- **arch** (*archinfo.Arch* | *str*) – The architecture of the state.

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*)

property plugins

property se

Deprecated alias for *solver*

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

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

T = ~T

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

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*)

prepare_callsite(*retval*, *args*, *cc='wtf'*)

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': <0 ABSTRACT_MEMORY[bool]>, 'ABSTRACT_SOLVER': <0
ABSTRACT_SOLVER[bool]>, 'ACTION_DEPS': <0 ACTION_DEPS[bool]>, 'ADD_AUTO_REFS': <0
ADD_AUTO_REFS[bool]>, 'ALLOW_SEND_FAILURES': <0 ALLOW_SEND_FAILURES[bool]>,
'ALL_FILES_EXIST': <0 ALL_FILES_EXIST[bool]>, 'ANY_FILE_MIGHT_EXIST': <0
ANY_FILE_MIGHT_EXIST[bool]>, 'APPROXIMATE_FIRST': <0 APPROXIMATE_FIRST[bool]>,
'APPROXIMATE_GUARDS': <0 APPROXIMATE_GUARDS[bool]>, 'APPROXIMATE_MEMORY_INDICES': <0
APPROXIMATE_MEMORY_INDICES[bool]>, 'APPROXIMATE_MEMORY_SIZES': <0
APPROXIMATE_MEMORY_SIZES[bool]>, 'APPROXIMATE_SATISFIABILITY': <0
APPROXIMATE_SATISFIABILITY[bool]>, 'AST_DEPS': <0 AST_DEPS[bool]>, 'AUTO_REFS': <0
AUTO_REFS[bool]>, 'AVOID_MULTIVALUED_READS': <0 AVOID_MULTIVALUED_READS[bool]>,
'AVOID_MULTIVALUED_WRITES': <0 AVOID_MULTIVALUED_WRITES[bool]>,
'BEST_EFFORT_MEMORY_STORING': <0 BEST_EFFORT_MEMORY_STORING[bool]>,
'BYPASS_ERRORED_IRCCALL': <0 BYPASS_ERRORED_IRCCALL[bool]>, 'BYPASS_ERRORED_IROP':
<0 BYPASS_ERRORED_IROP[bool]>, 'BYPASS_ERRORED_IRSTMT': <0
BYPASS_ERRORED_IRSTMT[bool]>, 'BYPASS_UNSUPPORTED_IRCCALL': <0
BYPASS_UNSUPPORTED_IRCCALL[bool]>, 'BYPASS_UNSUPPORTED_IRDIRTY': <0
BYPASS_UNSUPPORTED_IRDIRTY[bool]>, 'BYPASS_UNSUPPORTED_IREXPR': <0
BYPASS_UNSUPPORTED_IREXPR[bool]>, 'BYPASS_UNSUPPORTED_IROP': <0
BYPASS_UNSUPPORTED_IROP[bool]>, 'BYPASS_UNSUPPORTED_IRSTMT': <0
BYPASS_UNSUPPORTED_IRSTMT[bool]>, 'BYPASS_UNSUPPORTED_SYSCALL': <0
BYPASS_UNSUPPORTED_SYSCALL[bool]>, 'BYPASS_VERITESTING_EXCEPTIONS': <0
BYPASS_VERITESTING_EXCEPTIONS[bool]>, 'CACHELESS_SOLVER': <0
CACHELESS_SOLVER[bool]>, 'CALLLESS': <0 CALLLESS[bool]>, 'CGC_ENFORCE_FD': <0
CGC_ENFORCE_FD[bool]>, 'CGC_NON_BLOCKING_FDS': <0 CGC_NON_BLOCKING_FDS[bool]>,
'CGC_NO_SYMBOLIC_RECEIVE_LENGTH': <0 CGC_NO_SYMBOLIC_RECEIVE_LENGTH[bool]>,
'COMPOSITE_SOLVER': <0 COMPOSITE_SOLVER[bool]>, 'CONCRETIZE': <0 CONCRETIZE[bool]>,
'CONCRETIZE_SYMBOLIC_FILE_READ_SIZES': <0
CONCRETIZE_SYMBOLIC_FILE_READ_SIZES[bool]>, 'CONCRETIZE_SYMBOLIC_WRITE_SIZES': <0
CONCRETIZE_SYMBOLIC_WRITE_SIZES[bool]>, 'CONSERVATIVE_READ_STRATEGY': <0
CONSERVATIVE_READ_STRATEGY[bool]>, 'CONSERVATIVE_WRITE_STRATEGY': <0
CONSERVATIVE_WRITE_STRATEGY[bool]>, 'CONSTRAINT_TRACKING_IN_SOLVER': <0
CONSTRAINT_TRACKING_IN_SOLVER[bool]>, 'COPY_STATES': <0 COPY_STATES[bool]>,
'CPUID_SYMBOLIC': <0 CPUID_SYMBOLIC[bool]>, 'DOWNSIZE_Z3':  <0 DOWNSIZE_Z3[bool]>,
'DO_CCALLS': <0 DO_CCALLS[bool]>, 'DO_RET_EMULATION': <0 DO_RET_EMULATION[bool]>,
'EFFICIENT_STATE_MERGING': <0 EFFICIENT_STATE_MERGING[bool]>, 'ENABLE_NX': <0
ENABLE_NX[bool]>, 'EXCEPTION_HANDLING': <0 EXCEPTION_HANDLING[bool]>,
'EXTENDED_IROP_SUPPORT': <0 EXTENDED_IROP_SUPPORT[bool]>, 'FAST_MEMORY': <0
FAST_MEMORY[bool]>, 'FAST_REGISTERS': <0 FAST_REGISTERS[bool]>, 'FILES_HAVE_EOF': <0
FILES_HAVE_EOF[bool]>, 'HYBRID_SOLVER': <0 HYBRID_SOLVER[bool]>,
'JAVA_IDENTIFY_GETTER_SETTER': <0 JAVA_IDENTIFY_GETTER_SETTER[bool]>,
'JAVA_TRACK_ATTRIBUTES': <0 JAVA_TRACK_ATTRIBUTES[bool]>, 'KEEP_IP_SYMBOLIC': <0
KEEP_IP_SYMBOLIC[bool]>, 'KEEP_MEMORY_READS_DISCRETE': <0
KEEP_MEMORY_READS_DISCRETE[bool]>, 'LAZY_SOLVES': <0 LAZY_SOLVES[bool]>,
'MEMORY_CHUNK_INDIVIDUAL_READS': <0 MEMORY_CHUNK_INDIVIDUAL_READS[bool]>,
'MEMORY_FIND_STRICT_SIZE_LIMIT': <0 MEMORY_FIND_STRICT_SIZE_LIMIT[bool]>,
'MEMORY_SYMBOLIC_BYTES_MAP': <0 MEMORY_SYMBOLIC_BYTES_MAP[bool]>,
'NO_CROSS_INSN_OPT': <0 NO_CROSS_INSN_OPT[bool]>, 'NO_IP_CONCRETIZATION': <0
NO_IP_CONCRETIZATION[bool]>, 'NO_SYMBOLIC_JUMP_RESOLUTION': <0
NO_SYMBOLIC_JUMP_RESOLUTION[bool]>, 'NO_SYMBOLIC_SYSCALL_RESOLUTION': <0
NO_SYMBOLIC_SYSCALL_RESOLUTION[bool]>, 'OPTIMIZE_IR': <0 OPTIMIZE_IR[bool]>,
'PRODUCE_ZERODIV_SUCCESSORS': <0 PRODUCE_ZERODIV_SUCCESSORS[bool]>,
'REGION_MAPPING': <0 REGION_MAPPING[bool]>, 'REPLACEMENT_SOLVER': <0
REPLACEMENT_SOLVER[bool]>, 'REVERSE_MEMORY_HASH_MAP': <0
REVERSE_MEMORY_HASH_MAP[bool]>, 'REVERSE_MEMORY_NAME_MAP': <0
REVERSE_MEMORY_NAME_MAP[bool]>, 'SHORT_READS': <0 SHORT_READS[bool]>,
'SIMPLIFY_CONSTRAINTS': <0 SIMPLIFY_CONSTRAINTS[bool]>, 'SIMPLIFY_EXIT_GUARD': <0
SIMPLIFY_EXIT_GUARD[bool]>, 'SIMPLIFY_EXIT_STATE': <0 SIMPLIFY_EXIT_STATE[bool]>,
'SIMPLIFY_EXIT_TARGET': <0 SIMPLIFY_EXIT_TARGET[bool]>, 'SIMPLIFY_EXPRS': <0
SIMPLIFY_EXPRS[bool]>, 'SIMPLIFY_MEMORY_READS': <0 SIMPLIFY_MEMORY_READS[bool]>,
'SIMPLIFY_MEMORY_WRITES': <0 SIMPLIFY_MEMORY_WRITES[bool]>,
```

__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

`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__()

state: *SimState*

set_state(*state*)

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

set_strongref_state(*state*)

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 instanciate 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.

static memo(*f*)

A decorator function you should apply to `copy`

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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

classmethod register_default(*name*, *xtr=None*)

init_state()

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

class angr.state_plugins.inspect.**BP**(*when='before'*, *enabled=None*, *condition=None*, *action=None*, ***kwargs*)

Bases: object

A breakpoint.

__init__(*when='before'*, *enabled=None*, *condition=None*, *action=None*, ***kwargs*)

check(*state*, *when*)

Checks state *state* to see if the breakpoint should fire.

Parameters

- **state** – The state.
- **when** – Whether the check is happening before or after the event.

Returns
A boolean representing whether the checkpoint should fire.

fire(*state*)

Trigger the breakpoint.

Parameters
state – 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.

make_breakpoint(*event_type*, **args*, ***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.

b(*event_type*, **args*, ***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.

add_breakpoint(*event_type*, *bp*)

Adds a breakpoint which would trigger on *event_type*.

Parameters

- **event_type** – The event type to trigger on
- **bp** – The breakpoint

Returns
The created breakpoint.

remove_breakpoint(*event_type*, *bp=None*, *filter_func=None*)

Removes a breakpoint.

Parameters

- **bp** – The breakpoint to remove.
- **filter_func** – A filter function to specify whether each breakpoint should be removed or not.

copy(*memo=None*, ***kwargs*)

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 instanciate 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()

Remove previously stored attributes from 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` to `self.state.inspect`
>>> 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.attr0
>>> new_attr1 = self.state._inspect.attr1
>>> # Remove them from SimInspect
>>> 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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)

`state: angr.SimState`

`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'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x03
', b'\x02 ', b'\x02 ', b'\x02 ', b'\x02 ', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x01`', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8',
b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x08\xd5', b'\x08\xd5', b'\x08\xd5', b'\x08\xd5', b'\x08\xd5', b'\x08\xd5',
b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5',
b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5',
b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5',
b'\x08\xc5', b'\x08\xc5', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x08\xd6', b'\x08\xd6', b'\x08\xd6', b'\x08\xd6',
b'\x08\xd6', b'\x08\xd6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6',
b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6',
b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6',
b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x02\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
```

```
TOLOWER_LOC_ARRAY = [128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173,
174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224,
225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241,
242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 4294967295, 0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 97, 98,
99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,
116, 117, 118, 119, 120, 121, 122, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101,
102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118,
119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135,
136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,
153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186,
187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203,
204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237,
238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254,
255]

TOUPPER_LOC_ARRAY = [128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173,
174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224,
225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241,
242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 4294967295, 0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 123, 124, 125, 126, 127,
128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144,
145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178,
179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212,
213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229,
230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
247, 248, 249, 250, 251, 252, 253, 254, 255]
```

__init__()

copy(*memo=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

property errno

ret_errno(*val*)

state: angr.SimState

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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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 instanciate 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.

state: **angr.SimState**

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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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 instanciate 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.

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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.

state: **angr.SimState**

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

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns

True if the state plugin is actually widened.

Return type

bool

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

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*, ***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. accessable 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=None*, *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. accessable 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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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*)

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*)

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*)

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*

state: angr.SimState

eval_atmost(*e*, *n*, *cast_to=None*, ***kwargs*)

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

Parameters

- **e** – the expression to get a solution for
- **n** – the inclusive upper limit on the number of solutions
- **cast_to** – desired type of resulting values
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

Raises

- *SimUnsatError* – if no solution could be found satisfying the given constraints
- *SimValueError* – if more than *n* solutions were found to satisfy the given constraints

Returns
The solutions for *e*

eval_atleast(*e*, *n*, *cast_to=None*, ***kwargs*)

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

Parameters

- **e** – the expression to get a solution for
- **n** – the inclusive lower limit on the number of solutions
- **cast_to** – desired type of resulting values
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

Raises

- ***SimUnsatError*** – if no solution could be found satisfying the given constraints
- ***SimValueError*** – if less than *n* solutions were found to satisfy the given constraints

Returns
The solutions for *e*

eval_exact(*e*, *n*, *cast_to=None*, ***kwargs*)

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

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

clear()

state: **angr.SimState**

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 (CallStack | *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 (CallStack | *None*) –

state: **angr.SimState**

copy(*memo=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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. :param tuple stack_suffix: The stack suffix. :return: A string representation :rtype: str

`property top`

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.

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

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=None*, ***kwargs*)

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 instanciate 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*)

state: angr.SimState

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*)

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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns

True if the state plugin is actually widened.

Return type

bool

copy(*memo=None*, ***kwargs*)

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 instanciate 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

property actions

property jumpkinds

property jump_guards

property jump_targets

property jump_sources

property descriptions

property bbl_addrs

property ins_addrs

property stack_actions

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()

state: angr.SimState

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. :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=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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.

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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*)

state: *SimState*

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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*)

state: *SimState*

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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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*)

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

state: angr.SimState

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*)

state: angr.SimState

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=None*, *tmp_deps=None*, *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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

copy(*memo=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

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: Uc

__init__(*arch*, *cache_key*, *thumb=False*)

hook_add(*htype*, *callback*, *user_data=None*, *begin=1*, *end=0*, *arg1=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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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

state: **angr.SimState**

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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

copy(*memo=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

class angr.state_plugins.concrete.**Concrete**(*segment_registers_initialized=False*, *segment_registers_callback_initialized=False*, *whitelist=None*, *fs_register_bp=None*, *already_sync_objects_addresses=None*)

Bases: *SimStatePlugin*

__init__(*segment_registers_initialized=False*, *segment_registers_callback_initialized=False*, *whitelist=None*, *fs_register_bp=None*, *already_sync_objects_addresses=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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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)

sync()

Handle the switch between the concrete execution and angr. This method takes care of: 1- Synchronize registers. 2- Set a concrete target to the memory backer so the memory reads are redirected in the concrete process memory. 3- If possible restore the SimProcedures with the real addresses inside the concrete process. 4- Set an inspect point to sync the segments register as soon as they are read during the symbolic execution. 5- Flush all the pages loaded until now.

Returns

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns

True if the state plugin is actually widened.

Return type
bool

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: **angr.SimState**

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 instanciate 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

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: angr.SimState

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 "foward" 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()

state: angr.SimState

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

state: **angr.SimState**

angr.state_plugins.heap.heap_ptmalloc.**silence_logger**()

angr.state_plugins.heap.heap_ptmalloc.**unsilence_logger**(*level*)

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

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 varibles, 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

state: angr.SimState

10.4 Storage

class angr.state_plugins.view.**SimRegNameView**

Bases: *SimStatePlugin*

copy(*memo=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

get(*reg_name*)

state: angr.SimState

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 accesing 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) –

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

set_state(*state*)

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

```
types = {'CharT': char, '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, '__clock_t': uint32_t, '__dev_t':
uint64_t, '__gid_t': unsigned int, '__ino64_t': unsigned long long, '__ino_t':
unsigned long, '__int128': int128_t, '__int256': int256_t, '__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, 'crypt_data': struct crypt_data,
'dirent': struct dirent, 'dirent64': struct dirent64, 'double': double,
'drand48_data': struct <anon>, 'dword': uint32_t, 'error_t': int, 'exit_status':
struct exit_status, 'float': float, 'fstab': struct fstab, '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, 'mntent': struct mntent, '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>,
'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, 'sembuf': struct sembuf, 'servent':
struct servent, 'sgttyb': struct sgttyb, 'short': short, 'short int': short,
'sigevent': struct sigevent, '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, 'sigstack': struct sigstack, 'sigval': union sigval { sival_int int;
sival_ptr void*; }, 'size_t': size_t, 'sockaddr': struct sockaddr, 'sockaddr_in':
struct sockaddr_in, 'speed_t': long, 'ssize': size_t, 'ssize_t': size_t, 'stat':
struct stat, 'stat64': struct stat64, 'string': string_t, 'struct iovec': struct
iovec, '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, 'vtimes': struct vtimes, 'wchar_t':
short, 'winsize': struct winsize, 'word': uint16_t, 'wstring': wstring_t}
```

state: *angr.SimState = 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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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=None*, ***kwargs*)

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 instanciate 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.

`state: angr.SimState`

`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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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

`state: angr.SimState`

`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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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

state: angr.SimState

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.

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

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

length

endness

size()

property variables

property cache_key

property symbolic

property last_addr

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*

__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.storage.pcap.**PCAP**(*path*, *ip_port_tup*, *init=True*)

Bases: object

__init__(*path*, *ip_port_tup*, *init=True*)

initialize(*path*)

recv(*length*)

copy()

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 determins if this strategy can handle resolving the provided AST.
- **exact** – A flag (default: True) that determines if the convenience resolution functions provided by this class use exact or approximate resolution.

concretize(*memory*, *addr*, ***kwargs*)

Concretizes the address into a list of values. If this strategy cannot handle this address, returns None.

copy()

Returns a copy of the strategy, if there is data that should be kept separate between states. If not, returns self.

merge(*others*)

Merges this strategy with others (if there is data that should be kept separate between states. If not, is a no-op.

10.5 Memory Mixins

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

Bases: *SimStatePlugin*

SUPPORTS_CONCRETE_LOAD = False

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

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 instanciate 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 category

reg, mem, or file.

Type

Return the category of this SimMemory instance. It can be one of the three following categories

property variable_key_prefix

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

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

store(*addr*, *data*, ***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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

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*, ***kwargs*)

Set SUPPORTS_CONCRETE_LOAD to True and implement concrete_load if reading concrete bytes is faster in this memory model.

Parameters

- **addr** – The address to load from.
- **size** – Size of the memory read.
- **writing** –

Return type
memoryview

Returns
A memoryview into the loaded bytes.

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

replace_all(*old*, *new*)

Parameters

- **old** (*BV*) –
- **new** (*BV*) –

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

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

Parameters

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

Returns
None

state: angr.SimState

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

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

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

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

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

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

state: angr.SimState

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

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

state: angr.SimState

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

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

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

Bases: *KeyValueMemoryMixin*

state: angr.SimState

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*

state: angr.SimState

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.

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

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

state: angr.SimState

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.

find(*addr*, *data*, *max_search*, *default=None*, *endness=None*, *chunk_size=None*, *max_symbolic_bytes=None*, *condition=None*, *char_size=1*, ***kwargs*)

state: **angr.SimState**

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

Bases: *MemoryMixin*

state: **angr.SimState**

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 instanciate 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.

state: **angr.SimState**

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 instanciate 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.

state: **angr.SimState**

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.

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

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

state: angr.SimState

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

Bases: *MemoryMixin*

hex_dump(*start*, *size*, *word_size=4*, *words_per_row=4*, *endianness='Iend_BE'*, *symbolic_char='?'*, *unprintable_char='.'*, *solve=False*, *extra_constraints=None*, *inspect=False*, *disable_actions=True*)

Returns a hex dump as a string. The solver, if enabled, is called once for every byte potentially making this function very slow. It is meant to be used mainly as a "visualization" for debugging.

Warning: May read and display more bytes than *size* due to rounding. Particularly, if size is less than, or not a multiple of word_size*words_per_line.

Parameters

- **start** – starting address from which to print
- **size** – number of bytes to display
- **word_size** – number of bytes to group together as one space-delimited unit
- **words_per_row** – number of words to display per row of output
- **endianness** – endianness to use when displaying each word (ASCII representation is unchanged)
- **symbolic_char** – the character to display when a byte is symbolic and has multiple solutions
- **unprintable_char** – the character to display when a byte is not printable
- **solve** – whether or not to attempt to solve (warning: can be very slow)
- **extra_constraints** – extra constraints to pass to the solver is solve is True
- **inspect** – whether or not to trigger SimInspect breakpoints for the memory load
- **disable_actions** – whether or not to disable SimActions for the memory load

Returns

hex dump as a string

state: angr.SimState

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

Bases: *MemoryMixin*

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

copy(*memo=None*, ***kwargs*)

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 instanciate 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*, ***kwargs*)

store(*addr*, *data*, ***kwargs*)

state: **angr.SimState**

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

Bases: *MemoryMixin*

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

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

state: **angr.SimState**

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

Bases: *MemoryMixin*

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

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

state: **angr.SimState**

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

Bases: *MemoryMixin*

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

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

Parameters
action (*SimActionData | None*) –

state: **angr.SimState**

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

Bases: *MemoryMixin*

state: **angr.SimState**

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

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

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

state: **angr.SimState**

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 instanciate 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*)

state: angr.SimState

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

Bases: *MemoryMixin*

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

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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*, **args*, ***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.

state: angr.SimState

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

Bases: *MemoryMixin*

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

load(*addr*, *size=None*, *condition=None*, *endness=None*, *inspect=True*, ***kwargs*)

state: angr.SimState

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

Bases: *MemoryMixin*

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

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

state: angr.SimState

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 instanciate 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.

state: angr.SimState

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

Bases: *MemoryMixin*

store(*addr*, *data*, ***kwargs*)

state: angr.SimState

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.

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

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

find(*addr*, *what*, *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

state: **angr.SimState**

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 instanciate 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.

state: **angr.SimState**

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, thus allowing weak updates.

Each store() may take a value or multiple values, and a "weak" parameter to specify if this store is a weak update or not. 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 instanciate 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, Union[SimMemoryObject, SimLabeledMemoryObject]]]`

store(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *cooperate=False*, *weak=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,
```

(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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 | 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** (MVListPage) –
- **page_addr** (*int | None*) –

content_gen(*index*)

state: angr.SimState

class angr.storage.memory_mixins.paged_memory.pages.multi_values.**MultiValues**(*v=None*, *offset_to_values=None*)

Bases: object

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 (*Bits | MultiValues | None | Dict[int, Set[Bits]]*) –

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

Parameters

v (*Bits | MultiValues | None | Dict[int, Set[Bits]]*) –

add_value(*offset*, *value*)

Return type

None

Parameters

- **offset** (*int*) –
- **value** (*Bits*) –

one_value(*strip_annotations=False*)

Return type
Optional[Bits]

Parameters
strip_annotations (*bool*) –

merge(*mv*)

Return type
MultiValues

Parameters
mv (MultiValues) –

keys()

Return type
Set[int]

values()

Return type
Iterator[Set[Bits]]

items()

Return type
Iterator[Tuple[int, Set[Bits]]]

count()

Return type
int

extract(*offset*, *length*, *endness*)

Return type
MultiValues

Parameters

- **offset** (*int*) –
- **length** (*int*) –
- **endness** (*str*) –

concat(*other*)

Return type
MultiValues

Parameters
other (MultiValues | *Bits* | *bytes*) –

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 instanciate 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.

state: angr.SimState

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

Bases: *MemoryMixin*

__init__(**args*, *element_limit=5*, *annotation_limit=256*, *top_func=None*, *phi_maker=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 instanciate 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.

state: angr.SimState

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*

A bottom-level storage mechanism. Dispatches reads to individual pages, the type of which is the PAGE_TYPE class variable.

SUPPORTS_CONCRETE_LOAD = True

PAGE_TYPE: Type[TypeVar(PageType, bound= *PageBase*)] = None

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

copy(*memo=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.ite_cases(zip(conditions[1:], [o.bar for o in others]), self.bar)`

Parameters

- **others** (Iterable[*PagedMemoryMixin*]) – 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

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** –

Returns

A memoryview into the loaded bytes.

changed_bytes(*other*)

Return type
Set[int]

changed_pages(*other*)

Return type
Dict[int, Optional[Set[int]]]

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*) –

state: angr.SimState

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*

state: angr.SimState

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=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

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

Bases: *LabeledPagesMixin*, *ListPagesMixin*

state: angr.SimState

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

Bases: *LabeledPagesMixin*, *MVListPagesMixin*

state: **angr.SimState**

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

Bases: *PagedMemoryMixin*

PAGE_TYPE

alias of *UltraPage*

state: **angr.SimState**

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=None*, ***kwargs*)

Bases: *PagedMemoryMixin*

Parameters
cle_memory_backer (*None | Loader | 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 instanciate 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.

state: **angr.SimState**

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

Bases: *ClemoryBackerMixin*

Parameters
cle_memory_backer (*None | Loader | Clemory*) –

state: **angr.SimState**

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

Bases: *PagedMemoryMixin*

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

state: angr.SimState

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 instanciate 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 instanciate 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

state: angr.SimState

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`.

state: angr.SimState

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

state: angr.SimState

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 instanciate 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

state: **angr.SimState**

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

Bases: *MemoryMixin*

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

__init__(*permissions=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 instanciate 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

state: **angr.SimState**

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 instanciate 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
Optional[Set[int]]

all_bytes_changed_in_history()

Return type
Set[int]

state: **angr.SimState**

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.

set_state(*state*)

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

state: **angr.SimState**

class angr.storage.memory_mixins.paged_memory.pages.cooperation.**CooperationBase**

Bases: object

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*

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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*) –

state: angr.SimState

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 = 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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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*, ***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]

state: angr.SimState

replace_all_with_offsets(*offsets*, *old*, *new*, *memory=None*)

Parameters

- **offsets** (*Iterable[int]*) –
- **old** (*BV*) –
- **new** (*BV*) –

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().

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*) –
- **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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.ite_cases(zip(conditions[1:], [o.bar for o in others]), self.bar)`

Parameters

- **others** (Iterable[*RegionedMemoryMixin*]) – 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** (Optional[int]) – 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*) –

state: angr.SimState

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** (Optional[int]) – 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=None*, ***kwargs*)

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*

property category

reg, mem, or file.

Type
Return the category of this SimMemory instance. It can be one of the three following categories

state: angr.SimState

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.

find(*addr*, *data*, *max_search*, *default=None*, *endness=None*, *chunk_size=None*, *max_symbolic_bytes=None*, *condition=None*, *char_size=1*, ***kwargs*)

state: angr.SimState

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** (*StridedInterval*) –

clear()

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=None*, ***kwargs*)

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 instanciate 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*, *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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns
True if the state plugin is actually widened.

Return type
bool

dbg_print(*indent=0*)

Print out debugging information

state: **angr.SimState**

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

Bases: *MemoryMixin*

state: **angr.SimState**

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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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

state: angr.SimState

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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*)

state: angr.SimState

class angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory_mixin.**TypedVariable**(*type_*, *value*)

Bases: object

__init__(*type_*, *value*)

type

value

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

Bases: *MemoryMixin*

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

load(*key*, *none_if_missing=False*, ***kwargs*)

store(*key*, *data*, *type_=None*, ***kwargs*)

copy(*memo=None*, ***kwargs*)

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 instanciate 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.

state: angr.SimState

class angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin.**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*

__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=None*, ***kwargs*)

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 instanciate 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 `state.solver.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = self.state.solver.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 plugin

Returns

True if the state plugin is actually widened.

Return type

bool

state: **angr.SimState**

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 determins 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 determins 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 determins 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 determins 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 determins 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 determins 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 determins 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 determins 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 determins 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

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 ErroredState 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 aproximation. 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** – The state that encountered an error, at the point in time just before the erroring step began.
- **error** – The error that was thrown.
- **traceback** – The traceback for the error that was thrown.

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

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.**ExplorationTechnique**

Bases: object

An otiegnqwvk is a set of hooks for a simulation manager that assists in the implementation of new techniques in symbolic exploration.

TODO: choose actual name for the functionality (techniques? strategies?)

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.**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.**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.**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.**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 – 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.**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.**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*)

successors(*simgr*, *state*, *engine=None*, ***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.**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.**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.**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.**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.**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.`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.**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.**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.**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.**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.**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.**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.**Symbion**(*find=None*, *memory_concretize=None*, *register_concretize=None*, *timeout=0*, *find_stash='found'*)

Bases: *ExplorationTechnique*

The Symbion exploration technique uses the SimEngineConcrete available to step a SimState.

Parameters

- **find** – address or list of addresses that we want to reach, these will be translated into breakpoints inside the concrete process using the ConcreteTarget interface provided by the user inside the SimEngineConcrete.
- **memory_concretize** – list of tuples (address, symbolic variable) that are going to be written in the concrete process memory.
- **register_concretize** – list of tuples (reg_name, symbolic variable) that are going to be written
- **timeout** – how long we should wait the concrete target to reach the breakpoint

__init__(*find=None*, *memory_concretize=None*, *register_concretize=None*, *timeout=0*, *find_stash='found'*)

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*) –

step_state(*simgr*, **args*, ***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) –

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.**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**

Bases: *ExplorationTechnique*

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

__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.**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.**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.**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.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*)

successors(*simgr*, *state*, *engine=None*, ***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.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 – 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.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.symbion.**Symbion**(*find=None*, *memory_concretize=None*, *register_concretize=None*, *timeout=0*, *find_stash='found'*)

Bases: *ExplorationTechnique*

The Symbion exploration technique uses the SimEngineConcrete available to step a SimState.

Parameters

- **find** – address or list of addresses that we want to reach, these will be translated into breakpoints inside the concrete process using the ConcreteTarget interface provided by the user inside the SimEngineConcrete.
- **memory_concretize** – list of tuples (address, symbolic variable) that are going to be written in the concrete process memory.
- **register_concretize** – list of tuples (reg_name, symbolic variable) that are going to be written
- **timeout** – how long we should wait the concrete target to reach the breakpoint

__init__(*find=None*, *memory_concretize=None*, *register_concretize=None*, *timeout=0*, *find_stash='found'*)

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*) –

step_state(*simgr*, **args*, ***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) –

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.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.

__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) –

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

Bases: *SimEngineFailure*, *SimEngineSyscall*, *HooksMixin*, *SimEngineUnicorn*, SuperFastpathMixin, TrackActionsMixin, SimInspectMixin, HeavyResilienceMixin, *SootMixin*, HeavyVEXMixin, *TLSMixin*

irsb

state

stmt_idx

successors: Optional[SimSuccessors]

tmps

class angr.engines.**UberEnginePcode**(**args*, ***kwargs*)

Bases: *SimEngineFailure*, *SimEngineSyscall*, *HooksMixin*, *HeavyPcodeMixin*

class angr.engines.engine.**SimEngineBase**(*project=None*, ***kwargs*)

Bases: object

Even more basey of a base class for SimEngine. Used as a base by mixins which want access to the project but for which having method *process* (contained in *SimEngine*) doesn't make sense

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

class angr.engines.engine.**SimEngine**(*project=None*, ***kwargs*)

Bases: *SimEngineBase*

A SimEngine is a class which understands how to perform execution on a state. This is a base class.

abstract process(*state*, ***kwargs*)

The main entry point for an engine. Should take a state and return a result.

Parameters

state – The state to proceed from

Returns
The result. Whatever you want ;)

class angr.engines.engine.**TLSMixin**(**args*, ***kwargs*)

Bases: object

Mix this class into any class that defines __tls to make all of the attributes named in that list into thread-local properties.

MAGIC MAGIC MAGIC

class angr.engines.engine.**TLSProperty**(*name*)

Bases: object

__init__(*name*)

class angr.engines.engine.**SuccessorsMixin**(**args*, ***kwargs*)

Bases: *SimEngine*

A mixin for SimEngine which implements `process` to perform common operations related to symbolic execution and dispatches to a `process_successors` method to fill a SimSuccessors object with the results.

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

process(*state*, **args*, ***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** – 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

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** – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

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** – A string identifying the type of engine that produced these successors
- **processed** (*bool*) – Whether or not the processing succeeded
- **description** (*str*) – A textual description of the execution step

The successor states produced by this run are categorized into several lists:

Variables

- **artifacts** (*dict*) – Any analysis byproducts (for example, an IRSB) that were produced during execution
- **successors** – The "normal" successors. IP may be symbolic, but must have reasonable number of solutions
- **unsat_successors** – Any successor which is unsatisfiable after its guard condition is added.
- **all_successors** – successors + unsat_successors
- **flat_successors** – 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** – Any state for which during the flattening process we find too many solutions.

A more detailed description of the successor lists may be found here: https://docs.angr.io/core-concepts/simulation#simsuccessors

__init__(*addr*, *initial_state*)

classmethod failure()

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

Bases: *ProcedureMixin*, *SuccessorsMixin*

A SimEngine that you may use if you only care about processing SimProcedures. *Requires* the procedure kwarg to be passed to process.

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

Bases: *SuccessorsMixin*, *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

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

Bases: *SuccessorsMixin*, *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.

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

Bases: *SuccessorsMixin*, *ProcedureMixin*

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.soot.engine.**SootMixin**(**args*, ***kwargs*)

Bases: *SuccessorsMixin*, *ProcedureMixin*

Execution engine based on Soot.

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

Bases: *SuccessorsMixin*

Concrete execution in the Unicorn Engine, a fork of qemu.

Responds to the following parameters in the step stack:

- step: How many basic blocks we want to execute
- extra_stop_points: A collection of addresses at which execution should halt

__init__(**args*, ***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.

class angr.engines.concrete.**SimEngineConcrete**(*project*)

Bases: *SuccessorsMixin*

Concrete execution using a concrete target provided by the user.

__init__(*project*)

process_successors(*successors*, *extra_stop_points=None*, *memory_concretize=None*, *register_concretize=None*, *timeout=0*, **args*, ***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.

to_engine(*state*, *extra_stop_points*, *memory_concretize*, *register_concretize*, *timeout*)

Handle the concrete execution of the process This method takes care of: 1- Set the breakpoints on the addresses provided by the user 2- Concretize the symbolic variables and perform the write inside the concrete process 3- Continue the program execution.

Parameters

- **state** – The state with which to execute
- **extra_stop_points** – list of a addresses where to stop the concrete execution and return to the simulated one
- **memory_concretize** – list of tuples (address, symbolic variable) that are going to be written in the concrete process memory.
- **register_concretize** – list of tuples (reg_name, symbolic variable) that are going to be written
- **timeout** – how long we should wait the concrete target to reach the breakpoint

Returns

None

static check_concrete_target_methods(*concrete_target*)

Check if the concrete target methods return the correct type of data :return: True if the concrete target is compliant

class angr.engines.pcode.engine.**HeavyPcodeMixin**(**args*, ***kwargs*)

Bases: *SuccessorsMixin*, *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_text=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_text** (*str | 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: Optional[int]

jumpkind: str

class angr.engines.pcode.lifter.**PcodeDisassemblerBlock**(*addr*, *insns*, *thumb*, *arch*)

Bases: *DisassemblerBlock*

Helper class to represent a block of dissassembled target architecture instructions

addr

insns

thumb

arch

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** (Union[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** (Optional[int]) – The maximum number of instructions to lift. (See note below)
- **max_bytes** (Optional[int]) – The maximum number of bytes to use.

- **num_inst** (Optional[int]) – Replaces max_inst if max_inst is None. If set to None as well, no instruction limit is used.
- **num_bytes** (Optional[int]) – 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: Optional[*BehaviorFactory*]

data_refs: Sequence

default_exit_target: Optional

jumpkind: Optional[str]

next: Optional[int]

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

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 statements: *Iterable*

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: Union[str, bytes, None]

bytes_offset: Optional[int]

opt_level: int

traceflags: Optional[int]

allow_arch_optimizations: Optional[bool]

strict_block_end: Optional[bool]

collect_data_refs: bool

max_inst: Optional[int]

max_bytes: Optional[int]

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** (Union[str, bytes, None]) – The bytes to lift as either a python string of bytes or a cffi buffer object.
- **max_bytes** (Optional[int]) – The maximum number of bytes to lift. If set to None, no byte limit is used.
- **max_inst** (Optional[int]) – 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 (*Arch*) –

__init__(*arch*)

Parameters
arch (*Arch*) –

context: Context

behaviors: *BehaviorFactory*

lift(*irsb*, *baseaddr*, *data*, *bytes_offset=0*, *max_bytes=None*, *max_inst=None*)

Return type
None

Parameters

- **irsb** (IRSB) –
- **baseaddr** (*int*) –
- **data** (*bytes | bytearray*) –
- **bytes_offset** (*int*) –
- **max_bytes** (*int | None*) –
- **max_inst** (*int | None*) –

class angr.engines.pcode.lifter.**PcodeLifter**(*arch*, *addr*)

Bases: *Lifter*

Handles calling into pypcode to lift a block

Parameters

- **arch** (*Arch*) –
- **addr** (*int*) –

data: Union[str, bytes, None]

bytes_offset: Optional[int]

opt_level: int

traceflags: Optional[int]

allow_arch_optimizations: Optional[bool]

strict_block_end: Optional[bool]

collect_data_refs: bool

max_inst: Optional[int]

max_bytes: Optional[int]

skip_stmts: bool

irsb: *IRSB*

arch: Arch

addr: int

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*, *use_cache=None*, *cache_size=50000*, *default_opt_level=1*, *selfmodifying_code=None*, *single_step=False*, *default_strict_block_end=False*, ***kwargs*)

Bases: *SimEngineBase*

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*, *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*)

Temporary compatibility interface for integration with block code.

Parameters

- **addr** (*int | None*) –
- **state** (*SimState | None*) –
- **clemory** (*Clemory | 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*) –

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*)

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** (Optional[*SimState*]) – A state to use as a data source.
- **clemory** (Optional[Clemory]) – A cle.memory.Clemory object to use as a data source.
- **addr** (Optional[int]) – The address at which to start the block.
- **thumb** (bool) – Whether the block should be lifted in ARM's THUMB mode.
- **opt_level** (Optional[int]) – Unused for P-Code lifter
- **insn_bytes** (Optional[bytes]) – A string of bytes to use as a data source.
- **size** (Optional[int]) – The maximum size of the block, in bytes.
- **num_inst** (Optional[int]) – The maximum number of instructions.
- **traceflags** (int) – Unused by P-Code lifter
- **strict_block_end** (Optional[bool]) – 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*) –

class angr.engines.pcode.emulate.**PcodeEmulatorMixin**(**args*, ***kwargs*)

Bases: *SimEngineBase*

Mixin for p-code execution.

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

handle_pcode_block(*irsb*)

Execute a single IRSB.

Parameters

irsb (*IRSB*) – Block to be executed.

Return type

None

angr.engines.pcode.behavior.**make_bv_sizes_equal**(*bv1*, *bv2*)

Makes two BVs equal in length through sign extension.

Return type
Tuple[BV, BV]

Parameters

- **bv1** (*BV*) –
- **bv2** (*BV*) –

class angr.engines.pcode.behavior.**OpBehavior**(*opcode*, *is_unary*, *is_special=False*)

Bases: object

Base class for all operation behaviors.

Parameters

- **opcode** (*int*) –
- **is_unary** (*bool*) –
- **is_special** (*bool*) –

__init__(*opcode*, *is_unary*, *is_special=False*)

Parameters

- **opcode** (*int*) –
- **is_unary** (*bool*) –
- **is_special** (*bool*) –

Return type
None

opcode: int

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatEqual**

Bases: *OpBehavior*

Behavior for the FLOAT_EQUAL operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatNotEqual**

Bases: *OpBehavior*

Behavior for the FLOAT_NOTEQUAL operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatLess**

Bases: *OpBehavior*

Behavior for the FLOAT_LESS operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatLessEqual**

Bases: *OpBehavior*

Behavior for the FLOAT_LESSEQUAL operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatNan**

Bases: *OpBehavior*

Behavior for the FLOAT_NAN operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatAdd**

Bases: *OpBehavior*

Behavior for the FLOAT_ADD operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatDiv**

Bases: *OpBehavior*

Behavior for the FLOAT_DIV operation.

__init__()

opcode: int

is_unary: bool

is_special: *bool*

class angr.engines.pcode.behavior.**OpBehaviorFloatMult**

Bases: *OpBehavior*

Behavior for the FLOAT_MULT operation.

__init__()

opcode: *int*

is_unary: *bool*

is_special: *bool*

class angr.engines.pcode.behavior.**OpBehaviorFloatSub**

Bases: *OpBehavior*

Behavior for the FLOAT_SUB operation.

__init__()

opcode: *int*

is_unary: *bool*

is_special: *bool*

class angr.engines.pcode.behavior.**OpBehaviorFloatNeg**

Bases: *OpBehavior*

Behavior for the FLOAT_NEG operation.

__init__()

opcode: *int*

is_unary: *bool*

is_special: *bool*

class angr.engines.pcode.behavior.**OpBehaviorFloatAbs**

Bases: *OpBehavior*

Behavior for the FLOAT_ABS operation.

__init__()

opcode: *int*

is_unary: *bool*

is_special: *bool*

class angr.engines.pcode.behavior.**OpBehaviorFloatSqrt**

Bases: *OpBehavior*

Behavior for the FLOAT_SQRT operation.

__init__()

opcode: *int*

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatInt2Float**

Bases: *OpBehavior*

Behavior for the FLOAT_INT2FLOAT operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatFloat2Float**

Bases: *OpBehavior*

Behavior for the FLOAT_FLOAT2FLOAT operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatTrunc**

Bases: *OpBehavior*

Behavior for the FLOAT_TRUNC operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatCeil**

Bases: *OpBehavior*

Behavior for the FLOAT_CEIL operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatFloor**

Bases: *OpBehavior*

Behavior for the FLOAT_FLOOR operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorFloatRound**

Bases: *OpBehavior*

Behavior for the FLOAT_ROUND operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

class angr.engines.pcode.behavior.**OpBehaviorPiece**

Bases: *OpBehavior*

Behavior for the PIECE operation.

__init__()

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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*) –

opcode: int

is_unary: bool

is_special: bool

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 (*int*) –

class angr.engines.pcode.cc.**SimCCM68k**(*arch*)

Bases: *SimCC*

Default CC for M68k

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = []

FP_ARG_REGS: List[str] = []

STACKARG_SP_DIFF = 4

RETURN_VAL: *SimFunctionArgument* = <d0>

RETURN_ADDR: *SimFunctionArgument* = [0x0]

class angr.engines.pcode.cc.**SimCCRISCV**(*arch*)

Bases: *SimCC*

Default CC for RISCV

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']

RETURN_ADDR: *SimFunctionArgument* = <ra>

RETURN_VAL: *SimFunctionArgument* = <a0>

class angr.engines.pcode.cc.SimCCSPARC(*arch*)

Bases: *SimCC*

Default CC for SPARC

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['o0', 'o1']

RETURN_VAL: *SimFunctionArgument* = <o0>

RETURN_ADDR: *SimFunctionArgument* = <o7>

class angr.engines.pcode.cc.SimCCSH4(*arch*)

Bases: *SimCC*

Default CC for SH4

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['r4', 'r5']

RETURN_VAL: *SimFunctionArgument* = <r0>

RETURN_ADDR: *SimFunctionArgument* = <pr>

class angr.engines.pcode.cc.SimCCPARISC(*arch*)

Bases: *SimCC*

Default CC for PARISC

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['r26', 'r25']

RETURN_VAL: *SimFunctionArgument* = <r28>

RETURN_ADDR: *SimFunctionArgument* = <rp>

class angr.engines.pcode.cc.SimCCPowerPC(*arch*)

Bases: *SimCC*

Default CC for PowerPC

Parameters
arch (*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* = <lr>

RETURN_VAL: *SimFunctionArgument* = <r3>

class angr.engines.pcode.cc.**SimCCXtensa**(*arch*)

Bases: *SimCC*

Default CC for Xtensa

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['i2', 'i3', 'i4', 'i5', 'i6', 'i7']

FP_ARG_REGS: List[str] = []

RETURN_ADDR: *SimFunctionArgument* = <a0>

RETURN_VAL: *SimFunctionArgument* = <o2>

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_stripping_op**(*f*, **args*, ***kwargs*)

angr.state_plugins.sim_action_object.**ast_preserving_op**(*f*, **args*, ***kwargs*)

angr.state_plugins.sim_action_object.**ast_stripping_decorator**(*f*)

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

Bases: object

A SimActionObject tracks an AST and its dependencies.

__init__(*ast*, *reg_deps=None*, *tmp_deps=None*, *deps=None*, *state=None*)

to_claripy()

copy()

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

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

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

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

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

angr.state_plugins.sim_action_object.**make_methods**()

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
- **project** (Project) –
- **cc** (SimCC) –
- **prototype** (SimTypeFunction) –

The following parameters are optional:

Parameters

- **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** – 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** (Project) –
- **prototype** (SimTypeFunction) –

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** (Project) –
- **cc** (SimCC) –
- **prototype** (SimTypeFunction) –

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** (Project) –
- **cc** (SimCC) –

- **prototype** (SimTypeFunction) –

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** – 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** (Project) –
- **cc** (SimCC) –
- **prototype** (SimTypeFunction) –

__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*)

project: *Project*

arch: Arch

cc: *SimCC*

prototype: *SimTypeFunction*

state: *SimState*

arg_session: Union[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!

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 construtor

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.

ty_ptr(*ty*)

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**(*string*, *length_spec*, *pad_chr*, *size*, *signed*)

Bases: object

Describes a format specifier within a format string.

__init__(*string*, *length_spec*, *pad_chr*, *size*, *signed*)

string

size

signed

length_spec

pad_chr

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** (Project) –
- **cc** (SimCC) –
- **prototype** (SimTypeFunction) –

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

Returns
a FormatString object

state: SimState

project: angr.Project

arch: archinfo.arch.Arch

cc: angr.SimCC

prototype: angr.sim_type.SimTypeFunction

arg_session: Union[None, ArgSession, int]

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.

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'>}

state: SimState

project: angr.Project

arch: archinfo.arch.Arch

cc: angr.SimCC

prototype: angr.sim_type.SimTypeFunction

arg_session: Union[None, ArgSession, int]

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__()

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

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** – The prototype of the function as a SimTypeFunction

set_prototypes(*protos*)

Set the prototypes of many functions

Parameters
protos – Dictionary mapping function names to SimTypeFunction objects

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.

Returns
A tuple of (function name, function prototype)

Return type
tuple

add(*name*, *proc_cls*, ***kwargs*)

Add a function implementation fo 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*)

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.

Return type
Optional[*SimTypeFunction*]

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

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*)

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.

Return type
Optional[*SimTypeFunction*]

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

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*)

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.

Return type
Optional[*SimTypeFunction*]

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_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*)

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

__init__(*size*, *is_fp=False*)

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

List[Union[*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** (*str*) –
- **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*)

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

__init__(*stack_offset*, *size*, *is_fp=False*)

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*

An argument which spans multiple storage locations. Locations should be given least-significant first.

__init__(*locations*, *is_fp=False*)

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_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)`

__init__(*ptr_loc*, *main_loc*)

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 (*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

RETURN_VAL: *SimFunctionArgument* = None

OVERFLOW_RETURN_VAL: Optional[*SimFunctionArgument*] = None

FP_RETURN_VAL: Optional[*SimFunctionArgument*] = None

ARCH = None

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

cc

fp_iter

int_iter

both_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.

Parameters
ret_ty (SimType | *None*) –

return_in_implicit_outparam(*ty*)

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*)

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'*)

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[*SimFunctionArgument*]) – 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.
- **platform** (str) –

Return type
Optional[SimCC]

Returns
A calling convention instance, or None if none of the SimCC subclasses seems to fit the arguments provided.

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 :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*)

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 (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 = <eax>

OVERFLOW_RETURN_VAL: Optional[*SimFunctionArgument*] = <edx>

FP_RETURN_VAL: Optional[*SimFunctionArgument*] = <st0>

RETURN_ADDR: *SimFunctionArgument* = [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 (*Arch*) –

STRUCT_RETURN_THRESHOLD = 64

class angr.calling_conventions.SimCCStdcall(*arch*)
Bases: *SimCCMicrosoftCdecl*

Parameters
arch (*Arch*) –

CALLEE_CLEANUP = True

class angr.calling_conventions.SimCCMicrosoftFastcall(*arch*)
Bases: *SimCC*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['ecx', 'edx']

STACKARG_SP_DIFF = 4

RETURN_VAL: *SimFunctionArgument* = <eax>

RETURN_ADDR: *SimFunctionArgument* = [0x0]

ARCH
alias of ArchX86

class angr.calling_conventions.MicrosoftAMD64ArgSession(*cc*)
Bases: object

__init__(*cc*)

class angr.calling_conventions.SimCCMicrosoftAMD64(*arch*)
Bases: *SimCC*

Parameters
arch (*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* = <rax>

OVERFLOW_RETURN_VAL: Optional[*SimFunctionArgument*] = <rdx>

FP_RETURN_VAL: Optional[*SimFunctionArgument*] = <xmm0>

RETURN_ADDR: *SimFunctionArgument* = [0x0]

ARCH

alias of ArchAMD64

STACK_ALIGNMENT = 16

ArgSession

alias of *MicrosoftAMD64ArgSession*

next_arg(*session*, *arg_type*)

return_in_implicit_outparam(*ty*)

class angr.calling_conventions.**SimCCSyscall**(*arch*)

Bases: *SimCC*

The base class of all syscall CCs.

Parameters
arch (*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 (*Arch*) –

ARG_REGS: List[str] = ['ebx', 'ecx', 'edx', 'esi', 'edi', 'ebp']

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <eax>

RETURN_ADDR: *SimFunctionArgument* = <ip_at_syscall>

ARCH

alias of ArchX86

static syscall_num(*state*)

class angr.calling_conventions.**SimCCX86WindowsSyscall**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = []

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <eax>

RETURN_ADDR: *SimFunctionArgument* = <ip_at_syscall>

ARCH

alias of ArchX86

static syscall_num(*state*)

class angr.calling_conventions.**SimCCSystemVAMD64**(*arch*)

Bases: *SimCC*

Parameters
arch (*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* = [0x0]

RETURN_VAL: *SimFunctionArgument* = <rax>

OVERFLOW_RETURN_VAL: Optional[*SimFunctionArgument*] = <rdx>

FP_RETURN_VAL: Optional[*SimFunctionArgument*] = <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 (*Arch*) –

ARG_REGS: List[str] = ['rdi', 'rsi', 'rdx', 'r10', 'r8', 'r9']

RETURN_VAL: *SimFunctionArgument* = <rax>

RETURN_ADDR: *SimFunctionArgument* = <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 (*Arch*) –

ARG_REGS: List[str] = []

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <rax>

RETURN_ADDR: *SimFunctionArgument* = <ip_at_syscall>

ARCH

alias of ArchAMD64

static syscall_num(*state*)

class angr.calling_conventions.**SimCCARM**(*arch*)

Bases: *SimCC*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['r0', 'r1', 'r2', 'r3']

FP_ARG_REGS: List[str] = []

CALLER_SAVED_REGS: List[str] = []

RETURN_ADDR: *SimFunctionArgument* = <lr>

RETURN_VAL: *SimFunctionArgument* = <r0>

OVERFLOW_RETURN_VAL: Optional[*SimFunctionArgument*] = <r1>

ARCH

alias of ArchARM

next_arg(*session*, *arg_type*)

class angr.calling_conventions.**SimCCARMHF**(*arch*)

Bases: *SimCCARM*

Parameters
arch (*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: Optional[*SimFunctionArgument*] = <s0>

CALLER_SAVED_REGS: List[str] = []

RETURN_ADDR: *SimFunctionArgument* = <lr>

RETURN_VAL: *SimFunctionArgument* = <r0>

ARCH

alias of ArchARMHF

next_arg(*session*, *arg_type*)

class angr.calling_conventions.**SimCCARMLinuxSyscall**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['r0', 'r1', 'r2', 'r3']

FP_ARG_REGS: List[str] = []

RETURN_ADDR: *SimFunctionArgument* = <ip_at_syscall>

RETURN_VAL: *SimFunctionArgument* = <r0>

ARCH

alias of ArchARM

static syscall_num(*state*)

class angr.calling_conventions.**SimCCAArch64**(*arch*)

Bases: *SimCC*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7']

FP_ARG_REGS: List[str] = []

RETURN_ADDR: *SimFunctionArgument* = <lr>

RETURN_VAL: *SimFunctionArgument* = <x0>

ARCH

alias of ArchAArch64

class angr.calling_conventions.**SimCCAArch64LinuxSyscall**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7']

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <x0>

RETURN_ADDR: *SimFunctionArgument* = <ip_at_syscall>

ARCH

alias of ArchAArch64

static syscall_num(*state*)

class angr.calling_conventions.**SimCCRISCV64LinuxSyscall**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <a0>

RETURN_ADDR: *SimFunctionArgument* = <ip_at_syscall>

ARCH

alias of ArchRISCV64

static syscall_num(*state*)

class angr.calling_conventions.**SimCCO32**(*arch*)

Bases: *SimCC*

Parameters
arch (*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* = <ra>

RETURN_VAL: *SimFunctionArgument* = <v0>

OVERFLOW_RETURN_VAL: Optional[*SimFunctionArgument*] = <v1>

ARCH
alias of ArchMIPS32

next_arg(*session*, *arg_type*)

class angr.calling_conventions.**SimCCO32LinuxSyscall**(*arch*)
Bases: *SimCCSyscall*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['a0', 'a1', 'a2', 'a3']

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <v0>

RETURN_ADDR: *SimFunctionArgument* = <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 (*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* = <ra>

RETURN_VAL: *SimFunctionArgument* = <v0>

ARCH
alias of ArchMIPS64

angr.calling_conventions.**SimCCO64**
alias of *SimCCN64*

class angr.calling_conventions.**SimCCN64LinuxSyscall**(*arch*)
Bases: *SimCCSyscall*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <v0>

RETURN_ADDR: *SimFunctionArgument* = <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 (*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* = <lr>

RETURN_VAL: *SimFunctionArgument* = <r3>

ARCH

alias of ArchPPC32

class angr.calling_conventions.**SimCCPowerPCLinuxSyscall**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9', 'r10']

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <r3>

RETURN_ADDR: *SimFunctionArgument* = <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 (*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* = <lr>

RETURN_VAL: *SimFunctionArgument* = <r3>

ARCH
alias of ArchPPC64

class angr.calling_conventions.**SimCCPowerPC64LinuxSyscall**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9', 'r10']

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <r3>

RETURN_ADDR: *SimFunctionArgument* = <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 (*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 (*Arch*) –

class angr.calling_conventions.**SimCCS390X**(*arch*)

Bases: *SimCC*

Parameters
arch (*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* = <r14>

RETURN_VAL: *SimFunctionArgument* = <r2>

ARCH

alias of ArchS390X

class angr.calling_conventions.**SimCCS390XLinuxSyscall**(*arch*)

Bases: *SimCCSyscall*

Parameters
arch (*Arch*) –

ARG_REGS: List[str] = ['r2', 'r3', 'r4', 'r5', 'r6', 'r7']

FP_ARG_REGS: List[str] = []

RETURN_VAL: *SimFunctionArgument* = <r2>

RETURN_ADDR: *SimFunctionArgument* = <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*, ***kwargs*)
Return the default calling convention for a given architecture, platform, and language combination.

Parameters
- **arch** (str) – The architecture name.
- **platform** (Optional[str]) – The platform name (e.g., "Linux" or "Win32").
- **language** (Optional[str]) – The programming language name (e.g., "go").
- **syscall** (bool) – Return syscall convention (True), or normal calling convention (False, default).

Return type
Optional[Type[*SimCC*]]

Returns
A default calling convention class if we can find one for the architecture, platform, and language combination, or None 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*)

class angr.sim_variable.**SimVariable**(*ident=None*, *name=None*, *region=None*, *category=None*, *size=None*)

Bases: *Serializable*

Parameters

- **region** (*int | None*) –
- **size** (*int | None*) –

__init__(*ident=None*, *name=None*, *region=None*, *category=None*, *size=None*)

Parameters

- **ident** – A unique identifier provided by user or the program. Usually a string.
- **name** (*str*) – Name of this variable.
- **region** (*int | None*) –
- **size** (*int | None*) –

ident

name

region: Optional[int]

category: Optional[str]

renamed

candidate_names

size

copy()

loc_repr(*arch*)

The representation that shows up in a GUI

Parameters

arch (*Arch*) –

property is_function_argument

class angr.sim_variable.**SimConstantVariable**(*ident=None*, *value=None*, *region=None*, *size=None*)

Bases: *SimVariable*

Parameters

region (*int | None*) –

__init__(*ident=None*, *value=None*, *region=None*, *size=None*)

Parameters

- **ident** – A unique identifier provided by user or the program. Usually a string.
- **name** (*str*) – Name of this variable.

value

loc_repr(*arch*)

The representation that shows up in a GUI

copy()

Return type
SimConstantVariable

class angr.sim_variable.**SimTemporaryVariable**(*tmp_id*, *size=None*)

Bases: *SimVariable*

__init__(*tmp_id*, *size=None*)

Parameters

- **ident** – A unique identifier provided by user or the program. Usually a string.
- **name** (*str*) – Name of this variable.

tmp_id

loc_repr(*arch*)

The representation that shows up in a GUI

copy()

Return type
SimTemporaryVariable

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*

Parameters

- **region** (*int | None*) –
- **category** (*str | None*) –

__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: *int*

property bits

loc_repr(*arch*)

The representation that shows up in a GUI

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.**SimMemoryVariable**(*addr*, *size*, *ident=None*, *name=None*, *region=None*, *category=None*)

Bases: *SimVariable*

Parameters

- **region** (*int | None*) –
- **category** (*str | None*) –

__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.

addr

loc_repr(*arch*)

The representation that shows up in a GUI

property bits

copy()

Return type
SimMemoryVariable

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='sp'*, *base_addr=None*, *ident=None*, *name=None*, *region=None*, *category=None*)

Bases: *SimMemoryVariable*

Parameters

- **region** (*int | None*) –
- **category** (*str | None*) –

__init__(*offset*, *size*, *base='sp'*, *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.

base

offset

base_addr

loc_repr(*arch*)

The representation that shows up in a GUI

copy()

Return type
SimStackVariable

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.**SimVariableSet**

Bases: MutableSet

A collection of SimVariables.

__init__()

add(*item*)

Add an element.

add_register_variable(*reg_var*)

add_memory_variable(*mem_var*)

discard(*item*)

Remove an element. Do not raise an exception if absent.

discard_register_variable(*reg_var*)

discard_memory_variable(*mem_var*)

add_memory_variables(*addrs*, *size*)

copy()

complement(*other*)

Calculate the complement of *self* and *other*.

Parameters
other – Another SimVariableSet instance.

Returns
The complement result.

contains_register_variable(*reg_var*)

contains_memory_variable(*mem_var*)

class angr.sim_type.**SimType**(*label=None*)

Bases: object

SimType exists to track type information for SimProcedures.

base = True

__init__(*label=None*)

Parameters
label – the type label.

property size

The size of the type in bits.

property alignment

The alignment of the type in bytes.

with_arch(*arch*)

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

copy()

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.

class angr.sim_type.**TypeRef**(*name*, *ty*)

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.

__init__(*name*, *ty*)

Parameters
label – the type label.

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.

property alignment

The alignment of the type in bytes.

with_arch(*arch*)

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

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*)

Bases: *SimType*

SimTypeBottom basically represents a type error.

__init__(*label=None*)

Parameters
label – the type label.

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

copy()

class angr.sim_type.**SimTypeTop**(*size=None*, *label=None*)

Bases: *SimType*

SimTypeTop represents any type (mostly used with a pointer for void*).

__init__(*size=None*, *label=None*)

Parameters
label – the type label.

copy()

class angr.sim_type.**SimTypeReg**(*size*, *label=None*)

Bases: *SimType*

SimTypeReg is the base type for all types that are register-sized.

__init__(*size*, *label=None*)

Parameters

- **label** – the type label.
- **size** – the size of the type (e.g. 32bit, 8bit, etc.).

extract(*state*, *addr*, *concrete=False*)

store(*state*, *addr*, *value*)

copy()

class angr.sim_type.**SimTypeNum**(*size*, *signed=True*, *label=None*)

Bases: *SimType*

SimTypeNum is a numeric type of arbitrary length

__init__(*size*, *signed=True*, *label=None*)

Parameters

- **size** – The size of the integer, in bits
- **signed** – Whether the integer is signed or not
- **label** – A label for the type

extract(*state*, *addr*, *concrete=False*)

store(*state*, *addr*, *value*)

copy()

class angr.sim_type.**SimTypeInt**(*signed=True*, *label=None*)

Bases: *SimTypeReg*

SimTypeInt is a type that specifies a signed or unsigned C integer.

__init__(*signed=True*, *label=None*)

Parameters

- **signed** – True if signed, False if unsigned
- **label** – The type label

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

property size

The size of the type in bits.

extract(*state*, *addr*, *concrete=False*)

copy()

class angr.sim_type.**SimTypeShort**(*signed=True*, *label=None*)

Bases: *SimTypeInt*

class angr.sim_type.**SimTypeLong**(*signed=True*, *label=None*)

Bases: *SimTypeInt*

class angr.sim_type.**SimTypeLongLong**(*signed=True*, *label=None*)

Bases: *SimTypeInt*

class angr.sim_type.**SimTypeChar**(*signed=True*, *label=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.

__init__(*signed=True*, *label=None*)

Parameters

label – the type label.

store(*state*, *addr*, *value*)

extract(*state*, *addr*, *concrete=False*)

copy()

class angr.sim_type.**SimTypeWideChar**(*signed=True*, *label=None*)

Bases: *SimTypeReg*

SimTypeWideChar is a type that specifies a wide character (a UTF-16 character).

__init__(*signed=True*, *label=None*)

Parameters
label – the type label.

store(*state*, *addr*, *value*)

extract(*state*, *addr*, *concrete=False*)

copy()

class angr.sim_type.**SimTypeBool**(*signed=True*, *label=None*)

Bases: *SimTypeChar*

store(*state*, *addr*, *value*)

extract(*state*, *addr*, *concrete=False*)

class angr.sim_type.**SimTypeFd**(*label=None*)

Bases: *SimTypeReg*

SimTypeFd is a type that specifies a file descriptor.

__init__(*label=None*)

Parameters
label – the type label

copy()

class angr.sim_type.**SimTypePointer**(*pts_to*, *label=None*, *offset=0*)

Bases: *SimTypeReg*

SimTypePointer is a type that specifies a pointer to some other type.

__init__(*pts_to*, *label=None*, *offset=0*)

Parameters

- **label** – The type label.
- **pts_to** – The type to which this pointer points.

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

make(*pts_to*)

property size
The size of the type in bits.

copy()

class angr.sim_type.**SimTypeReference**(*refs*, *label=None*)

Bases: *SimTypeReg*

SimTypeReference is a type that specifies a reference to some other type.

__init__(*refs*, *label=None*)

Parameters

- **label** – the type label.
- **size** – the size of the type (e.g. 32bit, 8bit, etc.).

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

make(*refs*)

property size

The size of the type in bits.

copy()

class angr.sim_type.**SimTypeArray**(*elem_type*, *length=None*, *label=None*)

Bases: *SimType*

SimTypeArray is a type that specifies a series of data laid out in sequence.

__init__(*elem_type*, *length=None*, *label=None*)

Parameters

- **label** – The type label.
- **elem_type** – The type of each element in the array.
- **length** – An expression of the length of the array, if known.

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

property size

The size of the type in bits.

property alignment

The alignment of the type in bytes.

copy()

extract(*state*, *addr*, *concrete=False*)

store(*state*, *addr*, *values*)

angr.sim_type.**SimTypeFixedSizeArray**

alias of *SimTypeArray*

class angr.sim_type.**SimTypeString**(*length=None*, *label=None*, *name=None*)

Bases: *NamedTypeMixin*, *SimTypeArray*

SimTypeString is a type that represents a C-style string, i.e. a NUL-terminated array of bytes.

Parameters

name (*str | None*) –

__init__(*length=None*, *label=None*, *name=None*)

Parameters

- **label** – The type label.
- **length** – An expression of the length of the string, if known.

- **name** (*str | None*) –

extract(*state*, *addr*, *concrete=False*)

Parameters
state (SimState) –

property size

The size of the type in bits.

property alignment

The alignment of the type in bytes.

copy()

class angr.sim_type.**SimTypeWString**(*length=None*, *label=None*, *name=None*)

Bases: *NamedTypeMixin*, *SimTypeArray*

A wide-character null-terminated string, where each character is 2 bytes.

Parameters
name (*str | None*) –

__init__(*length=None*, *label=None*, *name=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.
- **name** (*str | None*) –

extract(*state*, *addr*, *concrete=False*)

property size

The size of the type in bits.

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** (*List[SimType]*) –
- **returnty** (SimType | *None*) –

base = False

__init__(*args*, *returnty*, *label=None*, *arg_names=None*, *variadic=False*)

Parameters

- **label** – The type label

- **args** (List[*SimType*]) – A tuple of types representing the arguments to the function
- **returnty** (Optional[*SimType*]) – The return type of the function, or none for void
- **variadic** – Whether the function accepts varargs

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

property size

The size of the type in bits.

copy()

class angr.sim_type.**SimTypeCppFunction**(*args*, *returnty*, *label=None*, *arg_names=None*, *ctor=False*, *dtor=False*)

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

- **args** (*List[SimType]*) –
- **returnty** (*SimType | None*) –
- **arg_names** (*Tuple[str]*) –
- **ctor** (*bool*) –
- **dtor** (*bool*) –

__init__(*args*, *returnty*, *label=None*, *arg_names=None*, *ctor=False*, *dtor=False*)

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** (*Tuple[str] | None*) –
- **ctor** (*bool*) –
- **dtor** (*bool*) –

copy()

args: List[*SimType*]

returnty: Optional[*SimType*]

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.

copy()

class angr.sim_type.**SimTypeFloat**(*size=32*)

Bases: *SimTypeReg*

An IEEE754 single-precision floating point number

__init__(*size=32*)

Parameters

- **label** – the type label.
- **size** – the size of the type (e.g. 32bit, 8bit, etc.).

sort = FLOAT

signed = True

extract(*state*, *addr*, *concrete=False*)

store(*state*, *addr*, *value*)

copy()

class angr.sim_type.**SimTypeDouble**(*align_double=True*)

Bases: *SimTypeFloat*

An IEEE754 double-precision floating point number

__init__(*align_double=True*)

Parameters

- **label** – the type label.
- **size** – the size of the type (e.g. 32bit, 8bit, etc.).

sort = DOUBLE

property alignment

The alignment of the type in bytes.

copy()

class angr.sim_type.SimStruct(*fields*, *name=None*, *pack=False*, *align=None*)

Bases: *NamedTypeMixin*, *SimType*

Parameters
fields (*Dict[str, SimType] | OrderedDict*) –

__init__(*fields*, *name=None*, *pack=False*, *align=None*)

Parameters

- **label** – the type label.
- **fields** (*Dict[str, SimType] | OrderedDict*) –

property packed

property offsets: Dict[str, int]

extract(*state*, *addr*, *concrete=False*)

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

property size

The size of the type in bits.

property alignment

The alignment of the type in bytes.

store(*state*, *addr*, *value*)

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*)

Bases: *NamedTypeMixin*, *SimType*

fields = ('members', 'name')

__init__(*members*, *name=None*, *label=None*)

Parameters

- **members** – The members of the union, as a mapping name -> type
- **name** – The name of the union

property size

The size of the type in bits.

property alignment

The alignment of the type in bytes.

extract(*state*, *addr*, *concrete=False*)

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

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.**SimCppClass**(*members=None*, *function_members=None*, *vtable_ptrs=None*, *name=None*, *pack=False*, *align=None*)

Bases: *SimStruct*

Parameters

- **members** (*Dict[str, SimType] | None*) –
- **function_members** (*Dict[str, SimTypeCppFunction] | None*) –
- **name** (*str | None*) –
- **pack** (*bool*) –

__init__(*members=None*, *function_members=None*, *vtable_ptrs=None*, *name=None*, *pack=False*, *align=None*)

Parameters

- **label** – the type label.
- **members** (*Dict[str, SimType] | None*) –
- **function_members** (*Dict[str, SimTypeCppFunction] | None*) –
- **name** (*str | None*) –
- **pack** (*bool*) –

property members

extract(*state*, *addr*, *concrete=False*)

store(*state*, *addr*, *value*)

copy()

class angr.sim_type.**SimCppClassValue**(*class_type*, *values*)

Bases: object

A SimCppClass type paired with some real values

__init__(*class_type*, *values*)

copy()

class angr.sim_type.**SimTypeNumOffset**(*size*, *signed=True*, *label=None*, *offset=0*)

Bases: *SimTypeNum*

like SimTypeNum, but supports an offset of 1 to 7 to a byte aligned address to allow structs with bitfields

__init__(*size*, *signed=True*, *label=None*, *offset=0*)

Parameters

- **size** – The size of the integer, in bits
- **signed** – Whether the integer is signed or not
- **label** – A label for the type

extract(*state*, *addr*, *concrete=False*)

Parameters
state (SimState) –

store(*state*, *addr*, *value*)

copy()

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.**do_preprocess**(*defn*, *include_path=()*)

Run a string through the C preprocessor that ships with pycparser but is weirdly inaccessible?

angr.sim_type.**parse_signature**(*defn*, *preprocess=True*, *predefined_types=None*, *arch=None*)

Parse a single function prototype and return its type

angr.sim_type.**parse_defns**(*defn*, *preprocess=True*, *predefined_types=None*, *arch=None*)

Parse a series of C definitions, returns a mapping from variable name to variable type object

angr.sim_type.**parse_types**(*defn*, *preprocess=True*, *predefined_types=None*, *arch=None*)

Parse a series of C definitions, returns a mapping from type name to type object

angr.sim_type.**parse_file**(*defn*, *preprocess=True*, *predefined_types=None*, *arch=None*)

Parse a series of C definitions, returns a tuple of two type mappings, one for variable definitions and one for type definitions.

Parameters
predefined_types (*Dict[Any, SimType] | None*) –

angr.sim_type.**type_parser_singleton**()

Return type
Optional[CParser]

angr.sim_type.**parse_type**(*defn*, *preprocess=True*, *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*)

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*) –

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*)

Bases: object

Callable is a representation of a function in the binary that can be interacted with like a native python function.

If you set perform_merge=True (the default), the result will be returned to you, and you can get the result state with callable.result_state.

Otherwise, you can get the resulting simulation manager at callable.result_path_group.

__init__(*project*, *addr*, *prototype=None*, *concrete_only=False*, *perform_merge=True*, *base_state=None*, *toc=None*, *cc=None*, *add_options=None*, *remove_options=None*)

Parameters

- **project** – The project to operate on
- **addr** – The address of the function to use

The following parameters are optional:

Parameters

- **prototype** – 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** – The state from which to do these runs
- **toc** – The address of the table of contents for ppc64
- **cc** – The SimCC to use for a calling convention

set_base_state(*state*)

Swap out the state you'd like to use to perform the call :type state: :param state: The state to use to perform the call

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.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*

structured_code: *StructuredCodeManager*

defs: *KeyDefinitionManager*

cfgs: *CFGManager*

types: *TypesStore*

propagations: PropagationManager

xrefs: *XRefManager*

__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*)

K = ~K

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: :rtype: Optional[TypeVar(K, bound= *KnowledgeBasePlugin*)] :return: Instance of the requested plugin class or null if it is not a known plugin

Parameters
requested_plugin_cls (*Type[K]*) –

Return type
K | None

request_knowledge(*requested_plugin_cls*)

Return type
TypeVar(K, bound= *KnowledgeBasePlugin*)

Parameters
requested_plugin_cls (*Type[K]*) –

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

Bases: object

copy()

static register_default(*name*, *cls*)

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*, *manual=False*)

Return type
None

Parameters

- **callsite_block_addr** (*int*) –
- **cc** (SimCC) –
- **prototype** (SimTypeFunction) –
- **manual** (*bool*) –

get_cc(*callsite_block_addr*)

Return type
Optional[*SimCC*]

Parameters
callsite_block_addr (*int*) –

get_prototype(*callsite_block_addr*)

Return type
Optional[*SimTypeFunction*]

Parameters
callsite_block_addr (*int*) –

get_prototype_type(*callsite_block_addr*)

Return type
Optional[bool]

Parameters
callsite_block_addr (*int*) –

has_prototype(*callsite_block_addr*)

Return type
bool

Parameters
callsite_block_addr (*int*) –

copy()

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'

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: Optional[str]

max_size: Optional[int]

pointer_addr: Optional[int]

content: Optional[bytes]

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.**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.

__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__.

addr

size

simprocedure_name

no_ret

function_address

thumb

byte_string: Optional[bytes]

is_syscall

instruction_addrs

irsb

soot_block

has_return

block_id: Union[angr.analyses.cfg.cfg_job_base.BlockID, int]

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

syscall_name

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.

Parameters

- **block_id** (angr.analyses.cfg.cfg_job_base.BlockID | *int*) –
- **byte_string** (*bytes* | *None*) –

__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

return_target

syscall

property callstack_key

property creation_failed

downsize()
Drop saved states.

copy()

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*

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

jumptable_addr

jumptable_size

jumptable_entry_size

jumptable_entries

type

class angr.knowledge_plugins.cfg.**IndirectJumpType**

Bases: object

Jumptable_AddressLoadedFromMemory = 0

Jumptable_AddressComputed = 1

Vtable = 3

Unknown = 255

class angr.knowledge_plugins.cfg.**CFGModel**(*ident*, *cfg_manager=None*, *is_arm=False*)

Bases: *Serializable*

This class describes a Control Flow Graph for a specific range of code.

__init__(*ident*, *cfg_manager=None*, *is_arm=False*)

ident

is_arm

graph

jump_tables: Dict[int, *IndirectJump*]

memory_data: Dict[int, *MemoryData*]

insn_addr_to_memory_data: Dict[int, *MemoryData*]

normalized

edges_to_repair

property project

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

get_node(*block_id*)

Get a single node from node key.

Parameters
block_id (BlockID) – Block ID of the node.

Returns
The CFGNode

Return type
CFGNode

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** (Optional[bool]) – 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 targer 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 _nodes_by_addr dict.

Return type
Optional[*CFGNode*]

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** (Optional[bool]) – 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*]

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** (Optional[str]) – 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.
- **jumpkind** –

Returns
A list of successors

Return type
list

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

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

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** (Optional[*MemoryDataSort*]) – Type of the memory data
- **data_size** (Optional[int]) – 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*)

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** (Optional[List[int]]) – A list of addresses of memory data, or None if tidying all known memory data entries.
- **exec_mem_regions** (Optional[List[Tuple[int, int]]]) – A list of start and end addresses of executable memory regions.
- **seg_list** (Optional[*SegmentList*]) – The segment list that CFGFast uses during CFG recovery.

- **data_type_guessing_handlers** (Optional[List[Callable]]) – 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.
- **xrefs** (XRefManager | *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** (Optional[*KnowledgeBase*]) – 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** (Optional[*KnowledgeBase*]) – Knowledge base to search for functions in.

Return type
Optional[*Function*]

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** (Optional[*KnowledgeBase*]) – Knowledge base to search for functions in.

Return type
None

class angr.knowledge_plugins.cfg.**CFGManager**(*kb*)

Bases: *KnowledgeBasePlugin*

__init__(*kb*)

new_model(*prefix*)

copy()

get_most_accurate()

Return type
Optional[*CFGModel*]

Returns
The most accurate CFG present in the CFGManager, or None if it does not hold any.

class angr.knowledge_plugins.cfg.cfg_model.**CFGModel**(*ident*, *cfg_manager=None*, *is_arm=False*)

Bases: *Serializable*

This class describes a Control Flow Graph for a specific range of code.

__init__(*ident*, *cfg_manager=None*, *is_arm=False*)

ident

is_arm

graph

jump_tables: Dict[int, *IndirectJump*]

memory_data: Dict[int, *MemoryData*]

insn_addr_to_memory_data: Dict[int, *MemoryData*]

normalized

edges_to_repair

property project

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

get_node(*block_id*)

Get a single node from node key.

Parameters
block_id (BlockID) – Block ID of the node.

Returns
The CFGNode

Return type
CFGNode

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** (Optional[bool]) – 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 targer 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 _nodes_by_addr dict.

Return type
Optional[*CFGNode*]

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** (`Optional[bool]`) – 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]`

`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** (`Optional[str]`) – 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.
- **jumpkind** –

Returns
A list of successors

Return type
list

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

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

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** (Optional[*MemoryDataSort*]) – Type of the memory data
- **data_size** (Optional[int]) – 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*)

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** (Optional[List[int]]) – A list of addresses of memory data, or None if tidying all known memory data entries.
- **exec_mem_regions** (Optional[List[Tuple[int, int]]]) – A list of start and end addresses of executable memory regions.
- **seg_list** (Optional[*SegmentList*]) – The segment list that CFGFast uses during CFG recovery.
- **data_type_guessing_handlers** (Optional[List[Callable]]) – 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.
- **xrefs** (XRefManager | *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** (Optional[*KnowledgeBase*]) – 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** (Optional[*KnowledgeBase*]) – Knowledge base to search for functions in.

Return type
Optional[*Function*]

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** (Optional[*KnowledgeBase*]) – 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'

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: Optional[str]

max_size: Optional[int]

pointer_addr: Optional[int]

content: Optional[bytes]

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*

__init__(*kb*)

new_model(*prefix*)

copy()

get_most_accurate()

Return type
Optional[*CFGModel*]

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** (angr.analyses.cfg.cfg_job_base.BlockID | *int*) –
- **byte_string** (*bytes* | *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__.

addr

size

simprocedure_name

no_ret

function_address

thumb

byte_string: Optional[bytes]

is_syscall

instruction_addrs

irsb

soot_block

has_return

block_id: Union[angr.analyses.cfg.cfg_job_base.BlockID, int]

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

syscall_name

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.

Parameters

- **block_id** (angr.analyses.cfg.cfg_job_base.BlockID | *int*) –

- **byte_string** (*bytes | None*) –

__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

return_target

syscall

property callstack_key

property creation_failed

downsize()

Drop saved states.

copy()

class angr.knowledge_plugins.cfg.indirect_jump.**IndirectJumpType**

Bases: object

Jumptable_AddressLoadedFromMemory = 0

Jumptable_AddressComputed = 1

Vtable = 3

Unknown = 255

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*

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

jumptable_addr

jumptable_size

jumptable_entry_size

jumptable_entries

type

class angr.knowledge_plugins.types.**TypesStore**(*kb*)

Bases: *KnowledgeBasePlugin*, UserDict

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()

iter_own()

Iterate over all the names which are stored in this object - i.e. values() without ALL_TYPES

rename(*old*, *new*)

unique_type_name()

Return type
str

class angr.knowledge_plugins.comments.**Comments**(*kb*)

Bases: *KnowledgeBasePlugin*, dict

__init__(*kb*)

copy() → a shallow copy of D

class angr.knowledge_plugins.data.**Data**(*kb*)

Bases: *KnowledgeBasePlugin*

__init__(*kb*)

copy()

class angr.knowledge_plugins.indirect_jumps.**IndirectJumps**(*kb*)

Bases: *KnowledgeBasePlugin*, dict

__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*)

Get a label as string for a given address Same as .labels[x]

lookup(*name*)

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_manager.**FunctionDict**(*backref*, **args*, ***kwargs*)

Bases: SortedDict

FunctionDict is a dict where the keys are function starting addresses and map to the associated Function.

__init__(*backref*, **args*, ***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
```

get(*addr*)

Return the value for key if key is in the dictionary, else default.

floor_addr(*addr*)

ceiling_addr(*addr*)

class angr.knowledge_plugins.functions.function_manager.**FunctionManager**(*kb*)

Bases: *KnowledgeBasePlugin*, Mapping

This is a function boundaries management tool. It takes in intermediate results during CFG generation, and manages a function map of the binary.

__init__(*kb*)

copy()

clear()

get_by_addr(*addr*)

Return type
Function

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_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.

Returns
A Function instance, or None if there is no other function after *addr*.

Return type
Function or None

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

function(*addr=None*, *name=None*, *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** (*int*) – Address of the function.
- **name** (*str*) – 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 or None*) – True to find the PLT stub, False to find a non-PLT stub, None to disable this restriction.

Returns
The Function instance, or None if the function is not found and create is False.

Return type
Function or None

dbg_draw(*prefix='dbg_function_'*)

rebuild_callgraph()

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*)

Bases: *Serializable*

A representation of a function and various information about it.

Parameters

- **is_simprocedure** (*bool | None*) –
- **is_plt** (*bool | None*) –

__init__(*function_manager*, *addr*, *name=None*, *syscall=None*, *is_simprocedure=None*, *binary_name=None*, *is_plt=None*, *returning=None*, *alignment=False*)

Function constructor. If the optional parameters are not provided, they will be automatically determined upon the creation of a Function object.

Parameters

- **addr** – The address of the function.
- **is_simprocedure** (*bool | None*) –
- **is_plt** (*bool | None*) –

The following parameters are optional.

Parameters

- **name** (*str*) – The name of the function.
- **syscall** (*bool*) – Whether this function is a syscall or not.
- **is_simprocedure** (*bool*) – Whether this function is a SimProcedure or not.
- **binary_name** (*str*) – Name of the binary where this function is.
- **is_plt** (*bool*) – 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.

transition_graph

normalized

addr

startpoint

is_alignment

bp_on_stack

retaddr_on_stack

sp_delta

prototype: Optional[*SimTypeFunction*]

is_prototype_guessed: bool

prepared_registers

prepared_stack_variables

registers_read_afterwards

info

tags

ran_cca

is_syscall

is_simprocedure

is_plt

is_default_name

from_signature

binary_name

calling_convention: Optional[*SimCC*]

property alignment

property name

property project

property returning

property blocks

An iterator of all local blocks in the current function.

Returns
angr.lifter.Block instances.

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

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*) – The size of the block. This is optional. If not provided, angr will load
- **byte_string** (Optional[bytes]) –

Returns

get_block_size(*addr*)

Return type
Optional[int]

Parameters
addr (*int*) –

property nodes: Iterable[*CodeNode*]

get_node(*addr*)

Return type
Block

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

the function's pseudocode

Type
return

add_jumpout_site(*node*)

Add a custom jumpout site.

Parameters
node – 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 – The address of the basic block that control flow leaves the current function after a call.

Returns
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** (Optional[str]) – 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.

property demangled_name

apply_definition(*definition*, *calling_convention=None*)

Return type
None

Parameters

- **definition** (*str*) –
- **calling_convention** (SimCC | *Type*[SimCC] | *None*) –

functions_called()

Return type
Set[*Function*]

Returns
The set of all functions that can be reached from the function represented by self.

copy()

pp(***kwargs*)

Pretty-print the function disassembly.

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*, *all_func_addrs=None*)

Parameters
cmsg – The data to instanciate the <Function> from.

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

addr

is_syscall

is_plt

is_simprocedure

binary_name

bp_on_stack

retaddr_on_stack

sp_delta

calling_convention: Optional[SimCC]

prototype: Optional[SimTypeFunction]

prepared_registers

prepared_stack_variables

registers_read_afterwards

startpoint

info

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

is_default_name

from_signature

is_alignment

is_prototype_guessed: bool

ran_cca

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: Optional[int]

atom_hash: Optional[int]

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
cls

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".

__init__(*manager*, *func_addr=None*)

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
cls

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) –

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]]

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]]

Parameters
block_idx (*int | None*) –

find_variables_by_stack_offset(*offset*)

Return type
Set[*SimVariable*]

Parameters
offset (*int*) –

find_variables_by_register(*reg*)

Return type
Set[*SimVariable*]

Parameters
reg (*str | int*) –

get_variable_accesses(*variable*, *same_name=False*)

Return type
List[*VariableAccess*]

Parameters

- **variable** (SimVariable) –
- **same_name** (*bool*) –

get_variables(*sort=None*, *collapse_same_ident=False*)

Get a list of variables.

Parameters

- **sort** (Optional[Literal['stack', 'reg']]) – Sort of the variable to get.
- **collapse_same_ident** – Whether variables of the same identifier should be collapsed or not.

Return type
List[Union[*SimStackVariable*, *SimRegisterVariable*]]

Returns
A list of variables.

get_unified_variables(*sort=None*)

Get a list of unified variables.

Parameters
sort (Optional[Literal['stack', 'reg']]) – Sort of the variable to get.

Return type
List[Union[*SimStackVariable*, *SimRegisterVariable*]]

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*)

Assign default names to all unified variables.

Parameters

- **labels** – Known labels in the binary.
- **arg_names** (Optional[List[str]]) – Known argument names.
- **reset** (bool) – Reset all variable names or not.

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
Optional[*SimType*]

remove_types()

unify_variables()

Map SSA variables to a unified variable. Fill in self._unified_variables.

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
Optional[*SimVariable*]

Returns
The unified variable, or None if there is no such SSA variable.

class angr.knowledge_plugins.variables.variable_manager.**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.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 varibles 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** (Optional[ELF]) – Optional, when only one elf object should be considered (e.g. p.loader.main_object)
- **cu** (Optional[CompilationUnit]) – Optional, when only one compilation unit should be considered

class angr.knowledge_plugins.structured_code.manager.**StructuredCodeManager**(*kb*)

Bases: *KnowledgeBasePlugin*

__init__(*kb*)

discard(*key*)

available_flavors(*item*)

copy()

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
Optional[*LiveDefinitions*]

Parameters

- **ins_addr** (*int* | CodeLocation) –
- **kind** (ObservationPointType) –

get_observation_by_node(*node_addr*, *kind*, *node_idx=None*)

Return type
Optional[*LiveDefinitions*]

Parameters

- **node_addr** (*int* | CodeLocation) –
- **kind** (ObservationPointType) –
- **node_idx** (*int* | *None*) –

get_observation_by_stmt(*arg1*, *arg2*, *arg3=None*, ***, *block_idx=None*)

get_observation_by_exit(*node_addr*, *stmt_idx*, *src_node_idx=None*)

Return type
Optional[*LiveDefinitions*]

Parameters

- **node_addr** (*int*) –
- **stmt_idx** (*int*) –
- **src_node_idx** (*int* | *None*) –

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*)

Bases: object

A LiveDefinitions instance contains definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

Parameters

- **arch** (*Arch*) –
- **track_tmps** (*bool*) –

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*)

Parameters

- **arch** (*Arch*) –
- **track_tmps** (*bool*) –

project: Optional[Project]

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*]]

property register_definitions

property stack_definitions

property memory_definitions

property heap_definitions

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
Optional[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
Optional[int]

Parameters
addr (*Base*) –

static annotate_with_def(*symvar*, *definition*)

Parameters

- **symvar** (BV) –
- **definition** (*Definition*) –

Return type
BV

Returns

static extract_defs(*symvar*)

Return type
Generator[*Definition*, None, None]

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*, None, None]

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
Optional[int]

get_stack_address(*offset*)

Return type
Optional[int]

Parameters
offset (*Base*) –

stack_offset_to_stack_addr(*offset*)

Return type
int

merge(**others*)

Return type
Tuple[*LiveDefinitions*, bool]

Parameters
others (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
Optional[*MultiValues*]

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 | None*) –

get_definitions(*thing*)

Return type
Set[*Definition*[*Atom*]]

Parameters
thing (*Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]] | MultiValues*) –

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
Optional[*MultiValues*]

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_definitions_from_atoms(***kwargs*)

get_value_from_definition(***kwargs*)

get_one_value_from_definition(***kwargs*)

get_concrete_value_from_definition(***kwargs*)

get_value_from_atom(***kwargs*)

get_one_value_from_atom(***kwargs*)

get_concrete_value_from_atom(***kwargs*)

get_values(*spec*)

Return type
Optional[*MultiValues*]

Parameters
spec (Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]]) –

get_one_value(*spec*, *strip_annotations=False*)

Return type
Optional[BV]

Parameters

- **spec** (Atom | Definition | Iterable[Atom] | Iterable[Definition[Atom]]) –
- **strip_annotations** (*bool*) –

get_concrete_value(*spec*, *cast_to=<class 'int'>*)

Return type
Union[int, bytes, None]

Parameters

- **spec** (Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]]) –
- **cast_to** (*Type[int]* | *Type[bytes]*) –

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*)

static is_heap_address(*addr*)

Return type
bool

Parameters
addr (*Base*) –

static get_heap_offset(*addr*)

Return type
Optional[int]

Parameters
addr (*Base*) –

heap_address(*offset*)

Return type
BV

Parameters
offset (*int* | HeapAddress) –

class angr.knowledge_plugins.key_definitions.**DerefSize**(*value*)

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.**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** (Optional[Any]) – 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*, Optional[Any]]]

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** (Optional[Any]) – 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
Union[Set[*Definition*], Set[Tuple[*Definition*, Optional[Any]]]]

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
Union[Set[*Definition*], Set[Tuple[*Definition*, Optional[Any]]]]

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.**Definition**(*atom*, *codeloc*, *dummy=False*, *tags=None*)

Bases: Generic[A]

An atom definition.

Variables

- **atom** – The atom being defined.
- **codeloc** – Where this definition is created in the original binary code.
- **dummy** – Tell whether the definition should be considered dummy or not. During simplification by AILment, definitions marked as dummy will not be removed.
- **tags** – A set of tags containing information about the definition gathered during analyses.

__init__(*atom*, *codeloc*, *dummy=False*, *tags=None*)

Parameters

- **atom** (*A*) –
- **codeloc** (CodeLocation) –
- **dummy** (*bool*) –
- **tags** (*Set[Tag] | None*) –

atom: TypeVar(A, bound= *Atom*)

codeloc: *CodeLocation*

dummy: bool

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.atoms.**AtomKind**(*value*)

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*)

Instanciate an *Atom* from a given argument.

Parameters

- **argument** (*SimFunctionArgument*) – The argument to create a new atom from.
- **registers** – A mapping representing the registers of a given architecture.
- **full_reg** – Whether to return an atom indicating the entire register if the argument only specifies a slice of the register.
- **sp** (Optional[int]) – The current stack offset. Optional. Only used when argument is a SimStackArg.

- **arch** (*Arch*) –

Return type
Union[*Register*, *MemoryLocation*]

static reg(*thing*, *size=None*, *arch=None*)

Create a Register atom.

Parameters

- **thing** (Union[str, RegisterOffset]) – The register offset (e.g., project.arch.registers["rax"][0]) or the register name (e.g., "rax").
- **size** (Optional[int]) – Size of the register atom. Must be provided when creating the atom using a register offset.
- **arch** (Optional[Arch]) – 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** (Union[str, RegisterOffset]) – The register offset (e.g., project.arch.registers["rax"][0]) or the register name (e.g., "rax").
- **size** (Optional[int]) – Size of the register atom. Must be provided when creating the atom using a register offset.
- **arch** (Optional[Arch]) – 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** (Union[*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** (Optional[str]) – 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** (Union[*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** (Optional[str]) – 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*) –
- **size** (*int*) –
- **arch** (*Arch | None*) –

__init__(*reg_offset*, *size*, *arch=None*)

Parameters

- **size** (int) – The size of the atom in bytes
- **reg_offset** (*RegisterOffset*) –
- **arch** (*Arch | None*) –

reg_offset

arch

property name: str

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** (*str | None*) –

__init__(*addr*, *size*, *endness=None*)

Parameters

- **addr** (*int*) – The address of the beginning memory location slice.
- **size** (*int*) – The size of the represented memory location, in bytes.
- **endness** (*str | None*) –

addr: Union[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**(*value*)

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: Union[*AtomKind*, Type[*Atom*], None] = None

bbl_addr: Optional[int] = None

ins_addr: Optional[int] = None

variable: Optional[*SimVariable*] = None

variable_manager: Union[*VariableManagerInternal*, None, Literal[False]] = None

stack_offset: Optional[int] = None

reg_name: Union[str, int, None] = None

heap_offset: Optional[int] = None

global_addr: Optional[int] = None

tmp_idx: Optional[int] = None

const_val: Optional[int] = None

extern: Optional[bool] = 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[A]

An atom definition.

Variables

- **atom** – The atom being defined.
- **codeloc** – Where this definition is created in the original binary code.
- **dummy** – Tell whether the definition should be considered dummy or not. During simplification by AILment, definitions marked as dummy will not be removed.
- **tags** – A set of tags containing information about the definition gathered during analyses.

__init__(*atom*, *codeloc*, *dummy=False*, *tags=None*)

Parameters

- **atom** (*A*) –
- **codeloc** (CodeLocation) –
- **dummy** (*bool*) –
- **tags** (*Set[Tag] | None*) –

atom: *TypeVar(A, bound= Atom)*

codeloc: *CodeLocation*

dummy: *bool*

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[Base]]*) –

__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** (Union[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) –

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_addr*, *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**(*value*)

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*)

Bases: object

A LiveDefinitions instance contains definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

Parameters

- **arch** (*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]]*) –

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*)

Parameters

- **arch** (*Arch*) –
- **track_tmps** (*bool*) –

project: Optional[Project]

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*]]

property register_definitions

property stack_definitions

property memory_definitions

property heap_definitions

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
Optional[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
Optional[int]

Parameters
addr (*Base*) –

static annotate_with_def(*symvar*, *definition*)

Parameters

- **symvar** (BV) –
- **definition** (*Definition*) –

Return type
BV

Returns

static **extract_defs**(*symvar*)

Return type
Generator[*Definition*, None, None]

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*, None, None]

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
Optional[int]

get_stack_address(*offset*)

Return type
Optional[int]

Parameters
offset (*Base*) –

stack_offset_to_stack_addr(*offset*)

Return type
int

merge(**others*)

Return type
Tuple[*LiveDefinitions*, bool]

Parameters
others (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
Optional[*MultiValues*]

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 | None*) –

get_definitions(*thing*)

Return type
Set[*Definition*[*Atom*]]

Parameters
thing (Atom | Definition[Atom] | *Iterable*[Atom] | *Iterable*[Definition[Atom]] | MultiValues) –

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
Optional[*MultiValues*]

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_definitions_from_atoms(**kwargs*)

get_value_from_definition(**kwargs*)

get_one_value_from_definition(**kwargs*)

get_concrete_value_from_definition(**kwargs*)

get_value_from_atom(**kwargs*)

get_one_value_from_atom(***kwargs*)

get_concrete_value_from_atom(***kwargs*)

get_values(*spec*)

Return type
Optional[*MultiValues*]

Parameters
spec (Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]]) –

get_one_value(*spec*, *strip_annotations=False*)

Return type
Optional[BV]

Parameters

- **spec** (Atom | Definition | Iterable[Atom] | Iterable[Definition[Atom]]) –
- **strip_annotations** (*bool*) –

get_concrete_value(*spec*, *cast_to=<class 'int'>*)

Return type
Union[int, bytes, None]

Parameters

- **spec** (Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]]) –
- **cast_to** (*Type[int]* | *Type[bytes]*) –

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*)

static is_heap_address(*addr*)

Return type
bool

Parameters
addr (*Base*) –

static get_heap_offset(*addr*)

Return type
Optional[int]

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
Optional[*LiveDefinitions*]

Parameters

- **ins_addr** (*int* | CodeLocation) –
- **kind** (ObservationPointType) –

get_observation_by_node(*node_addr*, *kind*, *node_idx=None*)

Return type
Optional[*LiveDefinitions*]

Parameters

- **node_addr** (*int* | CodeLocation) –
- **kind** (ObservationPointType) –
- **node_idx** (*int* | *None*) –

get_observation_by_stmt(*arg1*, *arg2*, *arg3=None*, ***, *block_idx=None*)

get_observation_by_exit(*node_addr*, *stmt_idx*, *src_node_idx=None*)

Return type
Optional[*LiveDefinitions*]

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 | None*) –

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*) –
- **metadata** (*object*) –

__init__(*function=None*, *metadata=None*)

Parameters

- **function** (*int | None*) –
- **metadata** (*object | None*) –

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*) –
- **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*) –
- **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*) –
- **metadata** (*object*) –

class angr.knowledge_plugins.key_definitions.tag.**ReturnValueTag**(*function=None*, *metadata=None*)

Bases: *FunctionTag*

A tag for a definiton of a return value of a function.

Parameters

- **function** (*int*) –
- **metadata** (*object*) –

class angr.knowledge_plugins.key_definitions.tag.**InitialValueTag**(*metadata=None*)

Bases: *Tag*

A tag for a definiton of an initial value

Parameters

metadata (*object*) –

class angr.knowledge_plugins.key_definitions.tag.**UnknownSizeTag**(*metadata=None*)

Bases: *Tag*

A tag for a definiton 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** (Optional[Any]) – 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*, Optional[Any]]]

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** (Optional[Any]) – 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
Union[Set[*Definition*], Set[Tuple[*Definition*, Optional[Any]]]]

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
Union[Set[*Definition*], Set[Tuple[*Definition*, Optional[Any]]]]

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

angr.knowledge_plugins.sync.sync_controller.**import_binsync**()

angr.knowledge_plugins.sync.sync_controller.**make_state**(*f*)

Build a writeable State instance and pass to *f* as the *state* kwarg if the *state* kwarg is None. Function *f* should have have at least two kwargs, *user* and *state*.

angr.knowledge_plugins.sync.sync_controller.**make_ro_state**(*f*)

Build a read-only State instance and pass to *f* as the *state* kwarg if the *state* kwarg is None. Function *f* should have have at least two kwargs, *user* and *state*.

angr.knowledge_plugins.sync.sync_controller.**init_checker**(*f*)

class angr.knowledge_plugins.sync.sync_controller.**SyncController**(*kb*)

Bases: *KnowledgeBasePlugin*

SyncController interfaces with a binsync client to push changes upwards and pull changes downwards.

Variables
client (*binsync.Client*) – The binsync client.

__init__(*kb*)

connect(*user*, *path*, *bin_hash=''*, *init_repo=False*, *ssh_agent_pid=None*, *ssh_auth_sock=None*, *remote_url=None*)

property connected

commit()

update()

copy()

pull()

property has_remote

users()

status()

tally(*users=None*)

push_function(*func*, *user=None*, *state=None*)

Push a function upwards.

Parameters
func (Function) – The angr Function object to push upwards.

Returns
True if updates are made. False otherwise.

Return type
bool

push_comment(*addr*, *comment*, *decompiled=False*, *user=None*, *state=None*)

push_comments(*comments*, *user=None*, *state=None*)

Push a bunch of comments upwards.

Parameters
comments (*list*) – A list of BinSync Comments

Returns
bool

push_stack_variables(*stack_variables*, *var_manager*, *user=None*, *state=None*)

Parameters

- **stack_variables** (List[*SimStackVariable*]) –
- **var_manager** (*VariableManagerInternal*) –

Returns

push_stack_variable(*func_addr*, *offset*, *name*, *type_*, *size_*, *user=None*, *state=None*)

pull_function(*addr*, *user=None*, *state=None*)

Pull a function downwards.

Parameters

- **addr** (*int*) – Address of the function.
- **user** (*str*) – Name of the user.

Returns
The binsync.data.Function object if pulling succeeds, or None if pulling fails.

Return type
binsync.data.Function | None

pull_comment(*addr*, *user=None*, *state=None*)

Pull a comment downwards.

Parameters

- **addr** (*int*) – Address of the comment.
- **user** (*str*) – Name of the user.

Returns
a Comment object from BinSync, or None

Return type
binsync.data.Comment | None

pull_comments(*func_addr*, *user=None*, *state=None*)

Pull comments downwards.

Parameters

- **start_addr** (*int*) – Where we want to pull comments.
- **end_addr** (*int*) – Where we want to stop pulling comments (exclusive).

Returns
An iterator.

Return type
Iterable

pull_patches(*user=None*, *state=None*)

Pull patches.

Parameters
user (*str*) – Name of the user to patches from.

Returns
An iterator

Return type
Iterable

pull_stack_variables(*func_addr*, *user=None*, *state=None*)

Pull stack variables from a function.

@param func_addr: Function address to pull from @param user: @param state: @return:

get_func_addr_from_addr(*addr*)

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*) –
- **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*) –
- **dst** (*int | None*) –

ins_addr: Optional[int]

insn_op_idx: Optional[int]

block_addr: Optional[int]

stmt_idx: Optional[int]

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

copy()

insn_op_type

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.**XRefManager**(*kb*)

Bases: *KnowledgeBasePlugin*, *Serializable*

__init__(*kb*)

copy()

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: object

Stands for a specific program point by specifying basic block address and statement ID (for IRSBs), or SimProcedure name (for SimProcedures).

Parameters

- **block_addr** (*int*) –
- **stmt_idx** (*int | None*) –
- **ins_addr** (*int | None*) –
- **context** (*Any*) –
- **block_idx** (*int*) –

__init__(*block_addr*, *stmt_idx*, *sim_procedure=None*, *ins_addr=None*, *context=None*, *block_idx=None*, ***kwargs*)

Constructor.

Parameters

- **block_addr** (int) – Address of the block
- **stmt_idx** (Optional[int]) – 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** (Optional[int]) – The instruction address.
- **context** (Optional[Any]) – A tuple that represents the context of this CodeLocation in contextful 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: int

stmt_idx: Optional[int]

sim_procedure

ins_addr: Optional[int]

context: Optional[Tuple[int]]

block_idx

info: Optional[Dict]

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 contextful 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.keyed_region.**StoredObject**(*start*, *obj*, *size*)

Bases: object

__init__(*start*, *obj*, *size*)

start

obj

size: Union[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

angr.analyses.**register_analysis**(*cls*, *name*)

class angr.analyses.analysis.**AnalysisLogEntry**(*message*, *exc_info=False*)

Bases: object

__init__(*message*, *exc_info=False*)

class angr.analyses.analysis.**AnalysesHub**(*project*)

Bases: *PluginVendor*[A]

This class contains functions for all the registered and runnable analyses,

__init__(*project*)

reload_analyses(***kwargs*)

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[A]

__init__(*project*, *analysis_cls*)

Parameters

- **project** (Project) –
- **analysis_cls** (*Type[A]*) –

prep(*fail_fast=False*, *kb=None*, *progress_callback=None*, *show_progressbar=False*)

Return type
Type[TypeVar(A, bound= Analysis)]

Parameters

- **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 = []

named_errors = {}

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[AnalysisState, NodeType, JobType, JobKey]

This is my very first attempt to build a static forward analysis framework that can serve as the base of multiple static analyses in angr, including CFG analysis, VFG analysis, DDG, etc.

In short, ForwardAnalysis performs a forward data-flow analysis by traversing a graph, compute on abstract values, and store results in abstract states. The user can specify what graph to traverse, how a graph should be traversed, how abstract values and abstract states are defined, etc.

ForwardAnalysis has a few options to toggle, making it suitable to be the base class of several different styles of forward data-flow analysis implementations.

ForwardAnalysis supports a special mode when no graph is available for traversal (for example, when a CFG is being initialized and constructed, no other graph can be used). In that case, the graph traversal functionality is disabled, and the optimal graph traversal order is not guaranteed. The user can provide a job sorting method to sort the jobs in queue and optimize traversal order.

Feel free to discuss with me (Fish) if you have any suggestions or complaints.

__init__(*order_jobs=False*, *allow_merging=False*, *allow_widening=False*, *status_callback=None*, *graph_visitor=None*)

Constructor

Parameters

- **order_jobs** (*bool*) – If all jobs should be ordered or not.
- **allow_merging** (*bool*) – If job merging is allowed.
- **allow_widening** (*bool*) – If job widening is allowed.
- **graph_visitor** (GraphVisitor *or None*) – A graph visitor to provide successors.
- **status_callback** (*Callable[[Type[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.job_info.**JobInfo**(*key*, *job*)

Bases: Generic[JobType, JobKey]

Stores information of each job.

__init__(*key*, *job*)

Parameters

- **key** (*JobKey*) –
- **job** (*JobType*) –

property job: JobType

Get the latest available job.

Returns
The latest available job.

property merged_jobs

property widened_jobs

add_job(*job*, *merged=False*, *widened=False*)

Appended a new job to this JobInfo node. :type job: :param job: The new job to append. :param bool merged: Whether it is a merged job or not. :param bool widened: Whether it is a widened job or not.

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 overriden, 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[NodeType]

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

`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 (*iterable*) – 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 overriden, 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

nodes_iter(***kwargs*)

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

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.

project: Project

kb: KnowledgeBase

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*, *enable_advanced_backward_slicing=False*, *cfg_a=None*, *cfg_b=None*)

Bases: *Analysis*

This class computes the a diff between two binaries represented by angr Projects

__init__(*other_project*, *enable_advanced_backward_slicing=False*, *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

project: Project

kb: KnowledgeBase

class angr.analyses.boyscout.**BoyScout**(*cookiesize=1*)

Bases: *Analysis*

Try to determine the architecture and endieness of a binary blob

__init__(*cookiesize=1*)

project: Project

kb: KnowledgeBase

class angr.analyses.calling_convention.**CallSiteFact**(*return_value_used*)

Bases: object

Store facts about each call site.

__init__(*return_value_used*)

class angr.analyses.calling_convention.**UpdateArgumentsOption**

Bases: object

Enums for controlling the argument updating behavior in _adjust_cc.

DoNotUpdate = 0

AlwaysUpdate = 1

UpdateWhenCCHasNoArgs = 2

class angr.analyses.calling_convention.**CallingConventionAnalysis**(*func*, *cfg=None*, *analyze_callsites=False*, *caller_func_addr=None*, *callsite_block_addr=None*, *callsite_insn_addr=None*, *func_graph=None*)

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** – The recovered calling convention for the function.

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*) –

__init__(*func*, *cfg=None*, *analyze_callsites=False*, *caller_func_addr=None*, *callsite_block_addr=None*, *callsite_insn_addr=None*, *func_graph=None*)

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*) –

project: Project

kb: KnowledgeBase

is_va_start_amd64(*func*)

Return type
Tuple[bool, Optional[int]]

Parameters
func (Function) –

class angr.analyses.complete_calling_conventions.**CompleteCallingConventionsAnalysis**(*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*)

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

- **cfg** (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** (*Iterable[int] | None*) –
- **skip_other_funcs** (*bool*) –
- **auto_start** (*bool*) –
- **func_graphs** (*Dict[int, networkx.DiGraph] | None*) –

__init__(*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*)

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** (Optional[*CFGModel*]) – 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** (Optional[int]) – 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** (Optional[int]) – 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.
- **cc_callback** (*Callable | None*) –
- **prioritize_func_addrs** (*Iterable[int] | None*) –
- **skip_other_funcs** (*bool*) –
- **auto_start** (*bool*) –
- **func_graphs** (*Dict[int, DiGraph] | None*) –

work()

project: Project

kb: KnowledgeBase

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*)

project: Project

kb: KnowledgeBase

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** (Optional[Callable[[int, Any], 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

project: Project

kb: KnowledgeBase

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 compability 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.
- **elf_eh_frame** (*bool*) – Retrieve function starts (and maybe sizes later) from the .eh_frame of ELF 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).
- **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** (*dict*) – 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.

Extra parameters that angr.Analysis takes:

Parameters

- **progress_callback** – Specify a callback function to get the progress during CFG recovery.
- **show_progressbar** (*bool*) – Should CFGFast show a progressbar during CFG recovery or not.

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 intentially 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: Optional[str] = 'CFGEmulated'

__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 contorl 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`

`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: Optional[str] = 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

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_predecessors(***kwargs*)

get_successors(***kwargs*)

get_successors_and_jumpkind(***kwargs*)

get_all_predecessors(***kwargs*)

get_all_successors(***kwargs*)

get_node(***kwargs*)

get_any_node(***kwargs*)

get_all_nodes(***kwargs*)

nodes(***kwargs*)

nodes_iter(***kwargs*)

get_loop_back_edges()

get_branching_nodes(***kwargs*)

get_exit_stmt_idx(***kwargs*)

property graph: networkx.DiGraph[*CFGNode*]

remove_edge(*block_from*, *block_to*)

is_thumb_addr(*addr*)

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 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

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**(*functions*, *deregister_job_callback*)

Bases: object

A collection of pending jobs during CFG recovery.

__init__(*functions*, *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*)

src_func_addr

stmt_idx

ins_addr

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**(*value*)

Bases: Enum

Defines the type of work of a CFGJob

NORMAL = 0

FUNCTION_PROLOGUE = 1

COMPLETE_SCANNING = 2

IFUNC_HINTS = 3

DATAREF_HINTS = 4

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=True*, *resolve_indirect_jumps=True*, *force_segment=False*, *force_smart_scan=True*, *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*, *elf_eh_frame=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=512*, *nodecode_threshold=0.3*, *nodecode_step=16483*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *start=None*, *end=None*, *collect_data_references=None*, *extra_cross_references=None*, ***extra_arch_options*)

Bases: *ForwardAnalysis*[*CFGNode*, *CFGNode*, *CFGJob*, int], *CFGBase*

We find functions inside the given binary, and build a control-flow graph in very fast manners: instead of simulating program executions, keeping track of states, and performing expensive data-flow analysis, CFGFast will only perform light-weight analyses combined with some heuristics, and with some strong assumptions.

In order to identify as many functions as possible, and as accurate as possible, the following operation sequence is followed:

Active scanning

- If the binary has "function symbols" (TODO: this term is not accurate enough), they are starting points of the code scanning
- If the binary does not have any "function symbol", we will first perform a function prologue scanning on the entire binary, and start from those places that look like function beginnings
- Otherwise, the binary's entry point will be the starting point for scanning

Passive scanning

- After all active scans are done, we will go through the whole image and scan all code pieces

Due to the nature of those techniques that are used here, a base address is often not required to use this analysis routine. However, with a correct base address, CFG recovery will almost always yield a much better result. A custom analysis, called GirlScout, is specifically made to recover the base address of a binary blob. After the base address is determined, you may want to reload the binary with the new base address by creating a new Project object, and then re-recover the CFG.

```
PRINTABLES = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\
'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r'

SPECIAL_THUNKS = {'AMD64':
{b'\xe8\x07\x00\x00\x00\xf3\x90\x0f\xae\xe8\xeb\xf9H\x89\x04$\xc3':  ('jmp', 'rax'),
b'\xe8\x07\x00\x00\x00\xf3\x90\x0f\xae\xe8\xeb\xf9H\x8dd$\x08\xc3':  ('ret',)}}

tag:  Optional[str] = 'CFGFast'
```

__init__(*binary=None*, *objects=None*, *regions=None*, *pickle_intermediate_results=False*, *symbols=True*, *function_prologues=True*, *resolve_indirect_jumps=True*, *force_segment=False*, *force_smart_scan=True*, *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*, *elf_eh_frame=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=512*, *nodecode_threshold=0.3*, *nodecode_step=16483*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *start=None*, *end=None*, *collect_data_references=None*, *extra_cross_references=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*) – 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.
- **elf_eh_frame** (*bool*) – Retrieve function starts (and maybe sizes later) from the .eh_frame of ELF 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** (Optional[bool]) – 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** (Optional[bool]) – 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).
- **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** (*dict*) – 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.

Extra parameters that angr.Analysis takes:

Parameters

- **progress_callback** – Specify a callback function to get the progress during CFG recovery.
- **show_progressbar** (*bool*) – Should CFGFast show a progressbar during CFG recovery or not.
- **indirect_calls_always_return** (*bool | None*) –
- **jumptable_resolver_resolves_calls** (*bool | None*) –

Returns

None

property graph

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

copy()

indirect_jumps: Dict[int, IndirectJump]

project: Project

kb: KnowledgeBase

output()

generate_code_cover(***kwargs*)

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': {'pattern_match_ifuncs': (<class 'bool'>, True), 'ret_jumpkind_heuristics': (<class 'bool'>, True), 'switch_mode_on_nodecode': (<class 'bool'>, False)}, 'ARMEL': {'pattern_match_ifuncs': (<class 'bool'>, True), 'ret_jumpkind_heuristics': (<class 'bool'>, True), 'switch_mode_on_nodecode': (<class 'bool'>, True)}, 'ARMHF': {'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.**BlockID**(*addr*, *callsite_tuples*, *jump_type*)

Bases: object

A context-sensitive key for a SimRun object.

__init__(*addr*, *callsite_tuples*, *jump_type*)

callsite_repr()

static new(*addr*, *callstack_suffix*, *jumpkind*)

property func_addr

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 the indirect jump in ARM ELF binaries where all internal function calls are performed in the following manner:

```
ldr r3, [pc+#0x124]  ; load a constant from the constant_pool
blx r3
```

__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.

__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.**OverwriteTmpValueCallback**(*gp_value*)

Bases: object

Overwrites temporary values during resolution

__init__(*gp_value*)

overwrite_tmp_value(*state*)

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.**AddressTransferringTypes**

Bases: object

Types of address transfer.

Assignment = 0

SignedExtension = 1

UnsignedExtension = 2

Truncation = 3

Or1 = 4

ShiftLeft = 5

ShiftRight = 6

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.

__init__(*stmt_loc*, *stmt*, *tmp*, *base_addr=None*, *tmp_1=None*)

property base_addr_available

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**ConstantValueManager**(*project*, *kb*, *func*)

Bases: object

Manages the loading of registers who hold constant values.

Parameters
func (Function) –

__init__(*project*, *kb*, *func*)

Parameters
func (Function) –

project

kb

func

mapping

reg_read_callback(*state*)

Parameters
state (SimState) –

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**JumpTableProcessorState**(*arch*)

Bases: object

The state used in JumpTableProcessor.

__init__(*arch*)

arch

is_jumptable

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: *SimEngineLightVEXMixin*, *SimEngineLight*

Implements a simple and stupid data dependency tracking for stack and register variables.

Also determines which statements to instrument during static execution of the slice later. For example, the following example is not uncommon in non-optimized binaries:

```
    mov  [rbp+var_54], 1
loc_4051a6:
    cmp  [rbp+var_54], 6
    ja   loc_405412 (default)
loc_4051b0:
    mov  eax, [rbp+var_54]
    mov  rax, qword [rax*8+0x223a01]
    jmp  rax
```

We want to instrument the first instruction and replace the constant 1 with a symbolic variable, otherwise we will not be able to recover all jump targets later in block 0x4051b0.

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*, *value*)

Bases: object

Hook for register init.

__init__(*reg_offset*, *reg_bits*, *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*)

Bases: *IndirectJumpResolver*

Resolve an indirect jump by running a constant propagation on the entire function and check if the indirect jump can be resolved to a constant value. This resolver must be run after all other more specific resolvers.

__init__(*project*)

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.cfg_fast_soot.**CFGFastSoot**(*support_jni=False*, ***kwargs*)

Bases: *CFGFast*

__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.
- **elf_eh_frame** (*bool*) – Retrieve function starts (and maybe sizes later) from the .eh_frame of ELF 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).
- **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** (*dict*) – 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.

Extra parameters that angr.Analysis takes:

Parameters

- **progress_callback** – Specify a callback function to get the progress during CFG recovery.
- **show_progressbar** (*bool*) – Should CFGFast show a progressbar during CFG recovery or not.

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

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

indirect_jumps: Dict[int, IndirectJump]

project: Project

kb: KnowledgeBase

class angr.analyses.cfg.segment_list.**Segment**(*start*, *end*, *sort*)

Bases: object

Representing a memory block. This is not the "Segment" in ELF memory model

__init__(*start*, *end*, *sort*)

Parameters

- **start** (*int*) – Start address.
- **end** (*int*) – End address.
- **sort** (*str*) – Type of the segment, can be code, data, etc.

Returns
None

start

end

sort

property size

Calculate the size of the Segment.

Returns
Size of the Segment.

Return type
int

`copy()`

Make a copy of the Segment.

Returns
A copy of the Segment instance.

Return type
angr.analyses.cfg_fast.Segment

`class angr.analyses.cfg.segment_list.SegmentList`

Bases: `object`

SegmentList describes a series of segmented memory blocks. You may query whether an address belongs to any of the blocks or not, and obtain the exact block(segment) that the address belongs to.

`__init__()`

`search(addr)`

Checks which segment that the address *addr* should belong to, and, returns the offset of that segment. Note that the address may not actually belong to the block.

Parameters
addr (int) – The address to search

Return type
int

Returns
The offset of the segment.

`next_free_pos(address)`

Returns the next free position with respect to an address, including that address itself

Parameters
address – The address to begin the search with (including itself)

Returns
The next free position

`next_pos_with_sort_not_in(address, sorts, max_distance=None)`

Returns the address of the next occupied block whose sort is not one of the specified ones.

Parameters

- **address** (*int*) – The address to begin the search with (including itself).
- **sorts** – A collection of sort strings.
- **max_distance** – The maximum distance between *address* and the next position. Search will stop after we come across an occupied position that is beyond *address* + max_distance. This check will be disabled if *max_distance* is set to None.

Returns
The next occupied position whose sort is not one of the specified ones, or None if no such position exists.

Return type
int or None

`is_occupied`(*address*)

Check if an address belongs to any segment

Parameters
address – The address to check

Returns
True if this address belongs to a segment, False otherwise

`occupied_by_sort`(*address*)

Check if an address belongs to any segment, and if yes, returns the sort of the segment

Parameters
address (`int`) – The address to check

Return type
`Optional`[`str`]

Returns
Sort of the segment that occupies this address

`occupied_by`(*address*)

Check if an address belongs to any segment, and if yes, returns the beginning, the size, and the sort of the segment.

Parameters
address (`int`) – The address to check

Return type
`Optional`[`Tuple`[`int`, `int`, `str`]]

`occupy`(*address*, *size*, *sort*)

Include a block, specified by (address, size), in this segment list.

Parameters

- **address** (*int*) – The starting address of the block.
- **size** (*int*) – Size of the block.
- **sort** (*str*) – Type of the block.

Returns
None

`release`(*address*, *size*)

Remove a block, specified by (address, size), in this segment list.

Parameters

- **address** (`int`) – The starting address of the block.
- **size** (`int`) – Size of the block.

Return type
`None`

`copy`()

Make a copy of the SegmentList.

Returns
A copy of the SegmentList instance.

Return type
angr.analyses.cfg_fast.SegmentList

property occupied_size

The sum of sizes of all blocks

Returns
An integer

property has_blocks

Returns if this segment list has any block or not. !is_empty

Returns
True if it's not empty, False otherwise

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

project: Project

kb: KnowledgeBase

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.

project: Project

kb: KnowledgeBase

class angr.angrdb.db.**AngrDB**(*project=None*)

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*)

static 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

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

__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.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.**LoaderSerializer**

Bases: object

Serialize/unserialize a CLE Loader object into/from an angr DB.

backend2name = {<class 'cle.backends.blob.Blob'>: 'blob', <class 'cle.backends.elf.elf.ELF'>: 'elf', <class 'cle.backends.elf.elfcore.ELFCore'>: 'elfcore', <class 'cle.backends.cgc.cgc.CGC'>: 'cgc', <class 'cle.backends.cgc.backedcgc.BackedCGC'>: 'backedcgc', <class 'cle.backends.coff.Coff'>: 'COFF', <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.static_archive.StaticArchive'>: 'AR', <class 'cle.backends.te.TE'>: 'te', <class 'cle.backends.uefi_firmware.UefiFirmware'>: 'uefi', <class 'cle.backends.xbe.XBE'>: 'xbe'}

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.**StructuredCodeManagerSerializer**

Bases: object

Serialize/unserialize 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*, *improve_structurer=True*, ***kwargs*)

Bases: *Analysis*

Recursively structure a region and all of its subregions.

Parameters

- **func** (*Function | None*) –
- **structurer_cls** (*Type | None*) –

__init__(*region*, *cond_proc=None*, *func=None*, *structurer_cls=None*, *improve_structurer=True*, ***kwargs*)

Parameters

- **func** (*Function | None*) –
- **structurer_cls** (*Type | None*) –

project: Project

kb: KnowledgeBase

angr.analyses.decompiler.structuring.**structurer_class_from_name**(*name*)

Return type
Optional[Type]

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*) –

exception angr.analyses.decompiler.structuring.structurer_nodes.**EmptyBlockNotice**

Bases: Exception

class angr.analyses.decompiler.structuring.structurer_nodes.**MultiNode**(*nodes*, *addr=None*, *idx=None*)

Bases: object

__init__(*nodes*, *addr=None*, *idx=None*)

nodes

addr

idx

copy()

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structuring.structurer_nodes.**BaseNode**

Bases: object

static test_empty_node(*node*)

static test_empty_condition_node(*cond_node*)

addr: Optional[int]

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structuring.structurer_nodes.**SequenceNode**(*addr*, *nodes=None*)

Bases: *BaseNode*

Parameters
addr (*int | None*) –

__init__(*addr*, *nodes=None*)

Parameters
addr (*int | None*) –

addr: Optional[int]

nodes

add_node(*node*)

insert_node(*pos*, *node*)

remove_node(*node*)

node_position(*node*)

copy()

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structuring.structurer_nodes.**CodeNode**(*node*, *reaching_condition*)

Bases: BaseNode

__init__(*node*, *reaching_condition*)

node

reaching_condition

property addr

property idx

dbg_repr(*indent=0*)

copy()

class angr.analyses.decompiler.structuring.structurer_nodes.**ConditionNode**(*addr*, *reaching_condition*, *condition*, *true_node*, *false_node=None*)

Bases: BaseNode

Parameters
addr (*int | None*) –

__init__(*addr*, *reaching_condition*, *condition*, *true_node*, *false_node=None*)

addr: Optional[int]

reaching_condition

condition

true_node

false_node

dbg_repr(*indent=0*)

node

class angr.analyses.decompiler.structuring.structurer_nodes.**CascadingConditionNode**(*addr*, *condition_and_nodes*, *else_node=None*)

Bases: BaseNode

Parameters

- **addr** (*int | None*) –
- **condition_and_nodes** (*List[Tuple[Any, BaseNode]]*) –
- **else_node** (BaseNode) –

__init__(*addr*, *condition_and_nodes*, *else_node=None*)

Parameters

- **condition_and_nodes** (*List[Tuple[Any, BaseNode]]*) –
- **else_node** (*BaseNode | None*) –

addr: Optional[int]

condition_and_nodes

else_node

class angr.analyses.decompiler.structuring.structurer_nodes.**LoopNode**(*sort*, *condition*, *sequence_node*, *addr=None*, *continue_addr=None*, *initializer=None*, *iterator=None*)

Bases: *BaseNode*

Parameters

addr (*int | None*) –

__init__(*sort*, *condition*, *sequence_node*, *addr=None*, *continue_addr=None*, *initializer=None*, *iterator=None*)

sort

condition

sequence_node

initializer

iterator

copy()

property addr

property continue_addr

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structuring.structurer_nodes.**BreakNode**(*addr*, *target*)

Bases: *BaseNode*

Parameters

addr (*int | None*) –

__init__(*addr*, *target*)

addr: Optional[int]

target

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structuring.structurer_nodes.**ContinueNode**(*addr*, *target*)

Bases: *BaseNode*

Parameters
addr (*int | None*) –

__init__(*addr*, *target*)

addr: Optional[int]

target

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structuring.structurer_nodes.**ConditionalBreakNode**(*addr*, *condition*, *target*)

Bases: *BreakNode*

Parameters
addr (*int | None*) –

__init__(*addr*, *condition*, *target*)

condition

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structuring.structurer_nodes.**SwitchCaseNode**(*switch_expr*, *cases*, *default_node*, *addr=None*)

Bases: *BaseNode*

Parameters

- **cases** (*OrderedDict[int | Tuple[int, ...], SequenceNode]*) –
- **addr** (*int | None*) –

__init__(*switch_expr*, *cases*, *default_node*, *addr=None*)

Parameters
cases (*OrderedDict[int | Tuple[int, ...], SequenceNode]*) –

switch_expr

cases: OrderedDict[Union[int, Tuple[int, ...]], *SequenceNode*]

default_node

addr: Optional[int]

class angr.analyses.decompiler.structuring.structurer_nodes.**IncompleteSwitchCaseNode**(*addr*, *head*, *cases*)

Bases: *BaseNode*

Describes an incomplete set of switch-case nodes. Usually an intermediate result. Should always be restructured into a SwitchCaseNode by the end of structuring. Only used in Phoenix structurer.

Parameters

- **addr** (*int | None*) –
- **cases** (*List*) –

__init__(*addr*, *head*, *cases*)

Parameters

cases (*List*) –

addr: Optional[int]

head

cases: List

class angr.analyses.decompiler.structuring.structurer_nodes.**IncompleteSwitchCaseHeadStatement**(**args*, ***kwargs*)

Bases: Statement

Describes a switch-case head. This is only created by LoweredSwitchSimplifier.

__init__(*idx*, *switch_variable*, *case_addrs*, ***kwargs*)

switch_variable

case_addrs: List[Tuple[Block, Union[int, str], int, Optional[int], int]]

addr

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*, *improve_structurer=True*, ***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*) –

NAME: str = None

__init__(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, *improve_structurer=True*, ***kwargs*)

Parameters

- **func** (Function | None) –
- **case_entry_to_switch_head** (Dict[int, int] | None) –

static 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) –
- **new_node** (BaseNode | Block) –

static is_a_jump_target(*stmt*, *addr*)

Return type
bool

Parameters

- **stmt** (ConditionalJump | Jump) –
- **addr** (int) –

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**(*value*)

Bases: str, Enum

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.**PhoenixStructurer**(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, *improve_structurer=True*, *use_multistmtexprs=MultiStmtExprMode.MA*, ***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*) –

NAME: str = 'phoenix'

__init__(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, *improve_structurer=True*, *use_multistmtexprs=MultiStmtExprMode.MAX_ONE_CALL*, ***kwargs*)

Parameters

- **func** (*Function | None*) –
- **case_entry_to_switch_head** (*Dict[int, int] | None*) –
- **use_multistmtexprs** (*MultiStmtExprMode*) –

static dump_graph(*graph*, *path*)

Return type
None

Parameters

- **graph** (*DiGraph*) –
- **path** (*str*) –

project: Project

kb: KnowledgeBase

exception angr.analyses.decompiler.ail_simplifier.**HasCallNotification**

Bases: Exception

Notifies the existence of a call statement.

class angr.analyses.decompiler.ail_simplifier.**AILBlockTempCollector**(***kwargs*)

Bases: AILBlockWalker

Collects any temporaries used in a block.

__init__(***kwargs*)

class angr.analyses.decompiler.ail_simplifier.**AILSimplifier**(*func*, *func_graph=None*, *remove_dead_memdefs=False*, *stack_arg_offsets=None*, *unify_variables=False*, *ail_manager=None*, *gp=None*, *narrow_expressions=False*, *only_consts=False*, *fold_callexprs_into_conditions=False*, *use_callee_saved_regs_at_return=True*)

Bases: *Analysis*

Perform function-level simplifications.

Parameters

- **stack_arg_offsets** (*Set[Tuple[int, int]] | None*) –
- **ail_manager** (*Manager | None*) –
- **gp** (*int | None*) –

__init__(*func*, *func_graph=None*, *remove_dead_memdefs=False*, *stack_arg_offsets=None*, *unify_variables=False*, *ail_manager=None*, *gp=None*, *narrow_expressions=False*, *only_consts=False*, *fold_callexprs_into_conditions=False*, *use_callee_saved_regs_at_return=True*)

Parameters

- **stack_arg_offsets** (*Set[Tuple[int, int]] | None*) –
- **ail_manager** (*Manager | None*) –
- **gp** (*int | None*) –

project: Project

kb: KnowledgeBase

exception angr.analyses.decompiler.ailgraph_walker.**RemoveNodeNotice**

Bases: Exception

class angr.analyses.decompiler.ailgraph_walker.**AILGraphWalker**(*graph*, *handler*, *replace_nodes=False*)

Bases: object

Walks an AIL graph and optionally replaces each node with a new node.

Parameters
replace_nodes (*bool*) –

__init__(*graph*, *handler*, *replace_nodes=False*)

Parameters
replace_nodes (*bool*) –

walk()

class angr.analyses.decompiler.block_simplifier.**HasCallExprWalker**

Bases: AILBlockWalkerBase

Test if an expression contains a call expression inside.

__init__()

class angr.analyses.decompiler.block_simplifier.**BlockSimplifier**(*block*, *func_addr=None*, *remove_dead_memdefs=False*, *stack_pointer_tracker=None*, *peephole_optimizations=None*, *stack_arg_offsets=None*, *cached_reaching_definitions=None*, *cached_propagator=None*)

Bases: *Analysis*

Simplify an AIL block.

Parameters

- **block** (Block | *None*) –

- **func_addr** (*int | None*) –
- **peephole_optimizations** (*Iterable[Type[PeepholeOptimizationStmtBase] | Type[PeepholeOptimizationExprBase]] | None*) –
- **stack_arg_offsets** (*Set[Tuple[int, int]] | None*) –

__init__(*block*, *func_addr=None*, *remove_dead_memdefs=False*, *stack_pointer_tracker=None*, *peephole_optimizations=None*, *stack_arg_offsets=None*, *cached_reaching_definitions=None*, *cached_propagator=None*)

Parameters

- **block** (Optional[Block]) – The AIL block to simplify. Setting it to None to skip calling self._analyze(), which is useful in test cases.
- **func_addr** (*int | None*) –
- **peephole_optimizations** (*Iterable[Type[PeepholeOptimizationStmtBase] | Type[PeepholeOptimizationExprBase]] | None*) –
- **stack_arg_offsets** (*Set[Tuple[int, int]] | None*) –

project: Project

kb: KnowledgeBase

class angr.analyses.decompiler.callsite_maker.**CallSiteMaker**(*block*, *reaching_definitions=None*, *stack_pointer_tracker=None*, *ail_manager=None*)

Bases: *Analysis*

Add calling convention, declaration, and args to a call site.

__init__(*block*, *reaching_definitions=None*, *stack_pointer_tracker=None*, *ail_manager=None*)

project: Project

kb: KnowledgeBase

class angr.analyses.decompiler.ccall_rewriters.rewriter_base.**CCallRewriterBase**(*ccall*, *arch*)

Bases: object

The base class for CCall rewriters.

Parameters
ccall (*VEXCCallExpression*) –

__init__(*ccall*, *arch*)

Parameters
ccall (*VEXCCallExpression*) –

arch

result: Optional[Expression]

class angr.analyses.decompiler.ccall_rewriters.amd64_ccalls.**AMD64CCallRewriter**(*ccall*, *arch*)

Bases: *CCallRewriterBase*

Implements ccall rewriter for AMD64.

Parameters
ccall (*VEXCCallExpression*) –

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.**Clinic**(*func*, *remove_dead_memdefs=False*, *exception_edges=False*, *sp_tracker_track_memory=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*)

Bases: *Analysis*

A Clinic deals with AILments.

Parameters

- **peephole_optimizations** (*Iterable[Type[PeepholeOptimizationStmtBase] | Type[PeepholeOptimizationExprBase]] | None*) –
- **must_struct** (*Set[str] | None*) –
- **cache** (DecompilationCache | *None*) –

__init__(*func*, *remove_dead_memdefs=False*, *exception_edges=False*, *sp_tracker_track_memory=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*)

Parameters

- **peephole_optimizations** (*Iterable[Type[PeepholeOptimizationStmtBase] | Type[PeepholeOptimizationExprBase]] | None*) –
- **must_struct** (*Set[str] | None*) –
- **cache** (DecompilationCache | *None*) –

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

copy_graph()

Copy AIL Graph.

Return type
DiGraph

Returns
A copy of the AIl graph.

parse_variable_addr(*addr*)

Return type
Optional[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*) –

project: Project

kb: KnowledgeBase

class angr.analyses.decompiler.condition_processor.**ConditionProcessor**(*arch*, *condition_mapping=None*)

Bases: object

Convert between claripy AST and AIL expressions. Also calculates reaching conditions of all nodes on a graph.

__init__(*arch*, *condition_mapping=None*)

clear()

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*)

Parameters
case_entry_to_switch_head (*Dict[int, int] | None*) –

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[Optional[Statement]]

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*)

Return type
Bool

Parameters
nobool (*bool*) –

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*)

static simplify_condition_deprecated(*cond*)

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: object

Describes a decompilation option.

Parameters

- **candidate_values** (*List | None*) –
- **convert** (*Callable | 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

- **candidate_values** (*List | None*) –
- **convert** (*Callable | None*) –

angr.analyses.decompiler.decompilation_options.**O**

alias of *DecompilationOption*

angr.analyses.decompiler.decompilation_options.**get_structurer_option**()

Return type
Optional[*DecompilationOption*]

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*)

addr

type_constraints: Optional[Set]

var_to_typevar: Optional[Dict]

codegen: Optional[*BaseStructuredCodeGenerator*]

clinic: Optional[*Clinic*]

ite_exprs: Optional[Set[Tuple[int, Any]]]

binop_operators: Optional[Dict[OpDescriptor, str]]

property local_types

class angr.analyses.decompiler.decompiler.**Decompiler**(*func*, *cfg=None*, *options=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*, *update_memory_data=True*)

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*) –
- **peephole_optimizations** (*Iterable[Type[PeepholeOptimizationStmtBase]* | *Type[PeepholeOptimizationExprBase]]* | *None*) –
- **vars_must_struct** (*Set[str]* | *None*) –
- **update_memory_data** (*bool*) –

__init__(*func*, *cfg=None*, *options=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*, *update_memory_data=True*)

Parameters

- **func** (Function | *str* | *int*) –
- **cfg** (CFGFast | CFGModel | *None*) –
- **peephole_optimizations** (*Iterable[Type[PeepholeOptimizationStmtBase]* | *Type[PeepholeOptimizationExprBase]]* | *None*) –
- **vars_must_struct** (*Set[str]* | *None*) –
- **update_memory_data** (*bool*) –

reflow_variable_types(*type_constraints*, *var_to_typevar*, *codegen*)

Re-run type inference on an existing variable recovery result, then rerun codegen to generate new results.

Returns

Parameters

- **type_constraints** (*Set*) –
- **var_to_typevar** (*Dict*) –

find_data_references_and_update_memory_data(*seq_node*)

Parameters

seq_node (SequenceNode) –

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.

project: Project

kb: KnowledgeBase

class angr.analyses.decompiler.empty_node_remover.**EmptyNodeRemover**(*node*, *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
claripy_ast_conditions (*bool*) –

__init__(*node*, *claripy_ast_conditions=True*)

Parameters
claripy_ast_conditions (*bool*) –

class angr.analyses.decompiler.expression_narrower.**ExpressionNarrowingWalker**(*target_expr*)

Bases: AILBlockWalkerBase

Walks a statement or an expression and extracts the operations that are applied on the given expression.

For example, for target expression rax, *(rax & 0xff) + 0x1* means the following operations are applied on *rax*: rax & 0xff (rax & 0xff) + 0x1

The previous expression is always used in the succeeding expression.

Parameters
target_expr (*Expression*) –

__init__(*target_expr*)

Parameters
target_expr (*Expression*) –

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.

Parameters

- **successors** (*Set | None*) –
- **graph_with_successors** (*DiGraph | None*) –
- **full_graph** (*DiGraph | None*) –

- **cyclic_ancestor** (*bool*) –

__init__(*head*, *graph*, *successors*, *graph_with_successors*, *cyclic*, *full_graph*, *cyclic_ancestor=False*)

Parameters

- **successors** (*Set | None*) –
- **graph_with_successors** (*DiGraph | None*) –
- **full_graph** (*DiGraph | None*) –
- **cyclic_ancestor** (*bool*) –

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*)

angr.analyses.decompiler.optimization_passes.**get_optimization_passes**(*arch*, *platform*)

angr.analyses.decompiler.optimization_passes.**get_default_optimization_passes**(*arch*, *platform*)

Parameters

- **arch** (*Arch | str*) –
- **platform** (*str | None*) –

angr.analyses.decompiler.optimization_passes.**register_optimization_pass**(*opt_pass*, *enable_by_default*)

Parameters

enable_by_default (*bool*) –

class angr.analyses.decompiler.optimization_passes.const_derefs.**BlockWalker**(*project*)

Bases: AILBlockWalker

Parameters

project (Project) –

__init__(*project*)

Parameters

project (Project) –

walk(*block*)

Parameters

block (*Block*) –

class angr.analyses.decompiler.optimization_passes.const_derefs.**ConstantDereferencesSimplifier**(*func*, ***kwargs*)

Bases: *OptimizationPass*

Makes the following simplifications:

```
*(*(const_addr))  ==>  *(value) iff  *const_addr == value
```

ARCHES = None

PLATFORMS = None

STAGE: int = 1

NAME = 'Simplify constant dereferences'

DESCRIPTION = 'Makes the following simplifications::\n\n *(*(const_addr)) ==>
***(value) iff *const_addr == value'**

__init__(*func*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.eager_returns.**EagerReturnsSimplifier**(*func*, *blocks_by_addr=No*, *blocks_by_addr_an*, *graph=None*, *node_idx_start=0*, *max_level=2*, *min_indegree=2*, *max_calls_in_regio*, *reach-ing_definitions=Nor*, ***kwargs*)

Bases: *OptimizationPass*

Some compilers (if not all) generate only one returning block for a function regardless of how many returns there are in the source code. This oftentimes result in irreducible graphs and reduce the readability of the decompiled code. This optimization pass will make the function return eagerly by duplicating the return site of a function multiple times and assigning one copy of the return site to each of its sources when certain thresholds are met.

Note that this simplifier may reduce the readability of the generated code in certain cases, especially if the graph is already reducible without applying this simplifier.

Variables

- **max_level** (*int*) – Number of times that we repeat the process of making returns eager.
- **min_indegree** (*int*) – The minimum in-degree of the return site to be duplicated.
- **node_idx** – The next node index. Each duplicated return site gets assigned a unique index, otherwise those duplicates will be considered as the same block in the graph because they have the same hash.

ARCHES = None

PLATFORMS = None

STAGE: int = 0

NAME = 'Duplicate return blocks to reduce goto statements'

DESCRIPTION = 'Some compilers (if not all) generate only one returning block for a function regardless of how many returns there\nare in the source code. This oftentimes result in irreducible graphs and reduce the readability of the decompiled\ncode. This optimization pass will make the function return eagerly by duplicating the return site of a function\nmultiple times and assigning one copy of the return site to each of its sources when certain thresholds are met.\n\nNote that this simplifier may reduce the readability of the generated code in certain cases, especially if the graph\nis already reducible without applying this simplifier.'

__init__(*func*, *blocks_by_addr=None*, *blocks_by_addr_and_idx=None*, *graph=None*, *node_idx_start=0*, *max_level=2*, *min_indegree=2*, *max_calls_in_regions=2*, *reaching_definitions=None*, ***kwargs*)

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**(*value*)

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

AFTER_SINGLE_BLOCK_SIMPLIFICATION = 1

AFTER_MAKING_CALLSITES = 2

AFTER_GLOBAL_SIMPLIFICATION = 3

AFTER_VARIABLE_RECOVERY = 4

BEFORE_REGION_IDENTIFICATION = 5

DURING_REGION_IDENTIFICATION = 6

AFTER_STRUCTURING = 7

class angr.analyses.decompiler.optimization_passes.optimization_pass.**BaseOptimizationPass**(*func*)

Bases: object

The base class for any optimization pass.

ARCHES = []

PLATFORMS = []

STAGE: int = None

STRUCTURING: Optional[str] = None

NAME = 'N/A'

DESCRIPTION = 'N/A'

__init__(*func*)

property project

property kb

analyze()

class angr.analyses.decompiler.optimization_passes.optimization_pass.**OptimizationPass**(*func*, *blocks_by_addr=None*, *blocks_by_addr_and_i*, *graph=None*, *variable_kb=None*, *region_identifier=None*, *reaching_definitions=None*, ***kwargs*)

Bases: *BaseOptimizationPass*

The base class for any function-level graph optimization pass.

__init__(*func*, *blocks_by_addr=None*, *blocks_by_addr_and_idx=None*, *graph=None*, *variable_kb=None*, *region_identifier=None*, *reaching_definitions=None*, ***kwargs*)

property blocks_by_addr: `Dict[int, Set[Block]]`

property blocks_by_addr_and_idx: `Dict[Tuple[int, int | None], Block]`

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*, *seq=None*, ***kwargs*)

Bases: *BaseOptimizationPass*

The base class for any sequence node optimization pass.

ARCHES = []

PLATFORMS = []

STAGE: int = None

__init__(*func*, *seq=None*, ***kwargs*)

angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.**s2u**(*s*, *bits*)

class angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.**StackCanarySimplifier**(*func*, ***kwarg*

Bases: *OptimizationPass*

Removes stack canary checks from decompilation results.

ARCHES = ['X86', 'AMD64']

PLATFORMS = ['cgc', 'linux']

STAGE: int = 3

NAME = 'Simplify stack canaries'

DESCRIPTION = 'Removes stack canary checks from decompilation results.'

__init__(*func*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.**BasePointerSaveSimplifier**(*fu*
***

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 = ['cgc', 'linux']

STAGE: int = 3

NAME = 'Simplify base pointer saving'

DESCRIPTION = 'Removes the effects of base pointer stack storage at function invocation and restoring at function return.'

__init__(*func*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.div_simplifier.**DivSimplifierAILEngine**

Bases: *SimplifierAILEngine*

An AIL pass for the div simplifier

class angr.analyses.decompiler.optimization_passes.div_simplifier.**DivSimplifier**(*func*, ***kwargs*)

Bases: *OptimizationPass*

Simplifies various division optimizations back to "div".

ARCHES = ['X86', 'AMD64', 'ARMCortexM', 'ARMHF', 'ARMEL']

PLATFORMS = None

STAGE: int = 3

NAME = 'Simplify arithmetic division'

DESCRIPTION = 'Simplifies various division optimizations back to "div".'

__init__(*func*, ***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: AILBlockWalker

Replace expressions.

__init__(*block_addr*, *target_expr*, *callback*)

class angr.analyses.decompiler.optimization_passes.ite_expr_converter.**ITEExprConverter**(*func*, *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: int = 6

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__(*func*, *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: AILBlockWalkerBase

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**(*fun*
blo
blo
gra
***k*

Bases: *OptimizationPass*

Recognize and simplify lowered switch-case constructs.

ARCHES = ['AMD64']

PLATFORMS = ['linux', 'windows']

STAGE: int = 5

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.'

STRUCTURING: Optional[str] = ['phoenix']

__init__(*func*, *blocks_by_addr=None*, *blocks_by_addr_and_idx=None*, *graph=None*, ***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.multi_simplifier.**MultiSimplifierAILEngine**

Bases: *SimplifierAILEngine*

An AIL pass for the multi simplifier

class angr.analyses.decompiler.optimization_passes.multi_simplifier.**MultiSimplifier**(*func*, ***kwargs*)

Bases: *OptimizationPass*

Implements several different arithmetic optimizations.

ARCHES = ['X86', 'AMD64']

PLATFORMS = ['linux', 'windows']

STAGE: int = 3

NAME = 'Simplify various arithmetic expressions'

DESCRIPTION = 'Implements several different arithmetic optimizations.'

__init__(*func*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.mod_simplifier.**ModSimplifierAILEngine**

Bases: *SimplifierAILEngine*

class angr.analyses.decompiler.optimization_passes.mod_simplifier.**ModSimplifier**(*func*, ***kwargs*)

Bases: *OptimizationPass*

Simplifies optimized forms of modulo computation back to "mod".

ARCHES = ['X86', 'AMD64', 'ARMCortexM', 'ARMHF', 'ARMEL']

PLATFORMS = ['linux', 'windows']

STAGE: int = 3

NAME = 'Simplify optimized mod forms'

DESCRIPTION = 'Simplifies optimized forms of modulo computation back to "mod".'

__init__(*func*, ***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*)

get_variable(*old*)

remove_variable(*old*)

filter_variables(*atom*)

class angr.analyses.decompiler.optimization_passes.engine_base.**SimplifierAILEngine**

Bases: *SimEngineLightAILMixin*, *SimEngineLight*

Essentially implements a peephole optimization engine for AIL statements (because we do not perform memory or register loads).

__init__()

process(*state*, **args*, ***kwargs*)

The main entry point for an engine. Should take a state and return a result.

Parameters
state – The state to proceed from

Returns
The result. Whatever you want ;)

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.

__init__(*desc*)

class angr.analyses.decompiler.optimization_passes.expr_op_swapper.**ExpressionReplacer**(*block_addr*, *target_expr_predicate*, *callback*)

Bases: AILBlockWalker

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**(*func*, *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: int = 7

NAME = 'Swap operands of expressions as requested'

DESCRIPTION = 'Swap operands (and the operator accordingly) in a BinOp expression.'

__init__(*func*, *binop_operators=None*, ***kwargs*)

Parameters

binop_operators (*Dict[OpDescriptor, str] | None*) –

angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.**s2u**(*s*, *bits*)

class angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.**RegisterSaveAreaSimpli**

Bases: *OptimizationPass*

Optimizes away register spilling effects, including callee-saved registers.

ARCHES = None

PLATFORMS = None

STAGE: int = 1

NAME = 'Simplify register save areas'

DESCRIPTION = 'Optimizes away register spilling effects, including callee-saved registers.'

__init__(*func*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.**RetAddrSaveSimplifier**(*func*, ***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: int = 3

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__(*func*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.**X86GccGetPcSimplifier**(*func*, ***kwa*

Bases: *OptimizationPass*

Simplifies __x86.get_pc_thunk calls.

ARCHES = ['X86']

PLATFORMS = ['linux']

STAGE: int = 1

NAME = 'Simplify getpc()'

DESCRIPTION = 'Simplifies __x86.get_pc_thunk calls.'

__init__(*func*, ***kwargs*)

class angr.analyses.decompiler.peephole_optimizations.base.**PeepholeOptimizationStmtBase**(*project*, *kb*, *func_addr=None*)

Bases: object

The base class for all peephole optimizations that are applied on AIL statements.

Parameters

- **project** (Project | None) –
- **kb** (KnowledgeBase | None) –
- **func_addr** (int | None) –

NAME = 'Peephole Optimization - Statement'

DESCRIPTION = 'Peephole Optimization - Statement'

stmt_classes = None

__init__(*project*, *kb*, *func_addr=None*)

Parameters

- **project** (Project | None) –
- **kb** (KnowledgeBase | None) –
- **func_addr** (int | None) –

project: Optional[*Project*]

kb: Optional[*KnowledgeBase*]

func_addr: Optional[int]

optimize(*stmt*, *stmt_idx=None*, *block=None*, ***kwargs*)

Parameters

stmt_idx (int | None) –

class angr.analyses.decompiler.peephole_optimizations.base.**PeepholeOptimizationMultiStmtBase**(*project*, *kb*, *func_addr=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) –
- **func_addr** (int | None) –

NAME = 'Peephole Optimization - Multi-statement'

DESCRIPTION = 'Peephole Optimization - Multi-statement'

stmt_classes = None

__init__(*project*, *kb*, *func_addr=None*)

Parameters

- **project** (Project | None) –
- **kb** (KnowledgeBase | None) –
- **func_addr** (int | None) –

project: Optional[*Project*]

kb: Optional[*KnowledgeBase*]

func_addr: Optional[int]

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*, *func_addr=None*)

Bases: object

The base class for all peephole optimizations that are applied on AIL expressions.

Parameters

- **project** (Project | None) –
- **kb** (KnowledgeBase | None) –
- **func_addr** (*int* | None) –

NAME = 'Peephole Optimization - Expression'

DESCRIPTION = 'Peephole Optimization - Expression'

expr_classes = None

__init__(*project*, *kb*, *func_addr=None*)

Parameters

- **project** (Project | None) –
- **kb** (KnowledgeBase | None) –
- **func_addr** (*int* | None) –

project: Optional[*Project*]

kb: Optional[*KnowledgeBase*]

func_addr: Optional[int]

optimize(*expr*, ***kwargs*)

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*)

class angr.analyses.decompiler.region_identifier.**RegionIdentifier**(*func*, *cond_proc=None*, *graph=None*, *largest_successor_tree_outside_loop=True*, *force_loop_single_exit=True*, *complete_successors=False*)

Bases: *Analysis*

Identifies regions within a function.

__init__(*func*, *cond_proc=None*, *graph=None*, *largest_successor_tree_outside_loop=True*, *force_loop_single_exit=True*, *complete_successors=False*)

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

project: Project

kb: KnowledgeBase

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 { … }*.

__init__(*node*)

class angr.analyses.decompiler.region_simplifiers.cascading_ifs.**CascadingIfsRemover**(*node*)

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*)

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*)

Bases: *LocationBase*

__init__(*block_addr*, *block_idx*, *stmt_idx*)

block_addr

block_idx

stmt_idx

copy()

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionLocation**(*block_addr*, *block_idx*, *stmt_idx*, *expr_idx*)

Bases: *LocationBase*

__init__(*block_addr*, *block_idx*, *stmt_idx*, *expr_idx*)

block_addr

block_idx

stmt_idx

expr_idx

statement_location()

Return type
StatementLocation

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ConditionLocation**(*cond_node_addr*, *case_idx=None*)

Bases: *LocationBase*

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*

__init__(*node_addr*)

node_addr

class angr.analyses.decompiler.region_simplifiers.expr_folding.**MultiStatementExpressionAssignmentFinder**

Bases: AILBlockWalker

Process statements in MultiStatementExpression objects and find assignments.

__init__(*stmt_handler*)

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionUseFinder**

Bases: AILBlockWalker

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[SimVariable, Set[Tuple[Expression, Optional[ExpressionLocation]]]]

has_load

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionCounter**(*node*, *variable_manager*)

Bases: *SequenceWalker*

Find all expressions that are assigned once and only used once.

__init__(*node*, *variable_manager*)

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionReplacer**(*assignments*, *uses*, *variable_manager*)

Bases: AILBlockWalker

Parameters

- **assignments** (*Dict*) –
- **uses** (*Dict*) –

__init__(*assignments*, *uses*, *variable_manager*)

Parameters

- **assignments** (*Dict*) –
- **uses** (*Dict*) –

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionFolder**(*assignments*, *uses*, *node*, *variable_manager*)

Bases: *SequenceWalker*

Parameters

- **assignments** (*Dict*) –
- **uses** (*Dict*) –

__init__(*assignments*, *uses*, *node*, *variable_manager*)

Parameters

- **assignments** (*Dict*) –
- **uses** (*Dict*) –

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]]*) –

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*, *variable_kb=None*, *simplify_switches=True*, *simplify_ifelse=True*)

Bases: *Analysis*

Simplifies a given region.

Parameters

- **simplify_switches** (*bool*) –
- **simplify_ifelse** (*bool*) –

__init__(*func*, *region*, *variable_kb=None*, *simplify_switches=True*, *simplify_ifelse=True*)

Parameters

- **simplify_switches** (*bool*) –
- **simplify_ifelse** (*bool*) –

project: Project

kb: KnowledgeBase

class angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**CmpOp**(*value*)

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 | *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*)

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*, *var2condi*
func-
tions)

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.

Returns
None

angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**simplify_lowered_switches_core**(*reg*
var
co
fun
tio

Return type
bool

Parameters
region (SequenceNode) –

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*

Bases: *SequenceWalker*

Identifies switch expressions that adds or minuses a constant, removes the constant from the switch expression, and adjust all case expressions accordingly.

__init__(*node*)

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.base.**PositionMappingElement**(*start*, *length*, *obj*)

Bases: object

__init__(*start*, *length*, *obj*)

start: int

length: int

obj

class angr.analyses.decompiler.structured_codegen.base.**PositionMapping**

Bases: object

DUPLICATION_CHECK = True

__init__()

items()

add_mapping(*start_pos*, *length*, *obj*)

get_node(*pos*)

Parameters
pos (*int*) –

get_element(*pos*)

Return type
Optional[*PositionMappingElement*]

Parameters
pos (*int*) –

class angr.analyses.decompiler.structured_codegen.base.**InstructionMappingElement**(*ins_addr*, *posmap_pos*)

Bases: object

__init__(*ins_addr*, *posmap_pos*)

ins_addr: int

posmap_pos: int

class angr.analyses.decompiler.structured_codegen.base.**InstructionMapping**

Bases: object

__init__()

items()

add_mapping(*ins_addr*, *posmap_pos*)

get_nearest_pos(*ins_addr*)

Return type
Optional[int]

Parameters
ins_addr (*int*) –

class angr.analyses.decompiler.structured_codegen.base.**BaseStructuredCodeGenerator**(*flavor=None*)

Bases: object

__init__(*flavor=None*)

reapply_options(*options*)

regenerate_text()

Return type
None

reload_variable_types()

Return type
None

angr.analyses.decompiler.structured_codegen.c.**unpack_typeref**(*ty*)

angr.analyses.decompiler.structured_codegen.c.**unpack_pointer**(*ty*)

Return type
Optional[*SimType*]

angr.analyses.decompiler.structured_codegen.c.**unpack_array**(*ty*)

Return type
Optional[*SimType*]

angr.analyses.decompiler.structured_codegen.c.**squash_array_reference**(*ty*)

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
Optional[*SimType*]

Parameters

- **value** (*int*) –
- **project** (Project) –

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*)

Bases: object

Represents a program construct in C. Acts as the base class for all other representation constructions.

__init__(*codegen*)

codegen: StructuredCodeGenerator

c_repr(*indent=0*, *pos_to_node=None*, *pos_to_addr=None*, *addr_to_pos=None*)

Creates the C representation of the code and displays it by constructing a large string. This function is called by each program function that needs to be decompiled. The map_pos_to_node and map_pos_to_addr act as position maps for the location of each variable and statement to be tracked for later GUI operations. The map_pos_to_addr also contains expressions that are nested inside of statements.

c_repr_chunks(*indent=0*, *asexpr=False*)

static indent_str(*indent=0*)

class angr.analyses.decompiler.structured_codegen.c.**CFunction**(*addr*, *name*, *functy*, *arg_list*, *statements*, *variables_in_use*, *variable_manager*, *demangled_name=None*, *show_demangled_name=True*, ***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*, ***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

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*)

class angr.analyses.decompiler.structured_codegen.c.**CStatement**(*codegen*)

Bases: *CConstruct*

Represents a statement in C.

Parameters
codegen (*StructuredCodeGenerator*) –

class angr.analyses.decompiler.structured_codegen.c.**CExpression**(*collapsed=False*, ***kwargs*)

Bases: *CConstruct*

Base class for C expressions.

__init__(*collapsed=False*, ***kwargs*)

collapsed

property type

set_type(*v*)

class angr.analyses.decompiler.structured_codegen.c.**CStatements**(*statements*, ***kwargs*)

Bases: *CStatement*

Represents a sequence of statements in C.

__init__(*statements*, ***kwargs*)

statements

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**(*codegen*)

Bases: *CStatement*

Represents a loop in C.

Parameters
codegen (*StructuredCodeGenerator*) –

class angr.analyses.decompiler.structured_codegen.c.**CWhileLoop**(*condition*, *body*, *tags=None*, ***kwargs*)

Bases: *CLoop*

Represents a while loop in C.

__init__(*condition*, *body*, *tags=None*, ***kwargs*)

condition

body

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CDoWhileLoop**(*condition*, *body*, *tags=None*, ***kwargs*)

Bases: *CLoop*

Represents a do-while loop in C.

__init__(*condition*, *body*, *tags=None*, ***kwargs*)

condition

body

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CForLoop**(*initializer*, *condition*, *iterator*, *body*, *tags=None*, ***kwargs*)

Bases: *CStatement*

Represents a for-loop in C.

__init__(*initializer*, *condition*, *iterator*, *body*, *tags=None*, ***kwargs*)

initializer

condition

iterator

body

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CIfElse**(*condition_and_nodes*, *else_node=None*, *simplify_else_scope=False*, *cstyle_ifs=True*, *tags=None*, ***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*, *tags=None*, ***kwargs*)

Parameters
condition_and_nodes (*List[Tuple[CExpression, CStatement | None]]*) –

condition_and_nodes

else_node

simplify_else_scope

cstyle_ifs

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CIfBreak**(*condition*, *cstyle_ifs=True*, *tags=None*, ***kwargs*)

Bases: *CStatement*

Represents an if-break statement in C.

__init__(*condition*, *cstyle_ifs=True*, *tags=None*, ***kwargs*)

condition

cstyle_ifs

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CBreak**(*tags=None*, ***kwargs*)

Bases: *CStatement*

Represents a break statement in C.

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

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CContinue**(*tags=None*, ***kwargs*)

Bases: *CStatement*

Represents a continue statement in C.

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

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CSwitchCase**(*switch*, *cases*, *default*, *tags=None*, ***kwargs*)

Bases: *CStatement*

Represents a switch-case statement in C.

__init__(*switch*, *cases*, *default*, *tags=None*, ***kwargs*)

switch

cases: List[Tuple[Union[int, Tuple[int]], *CStatements*]]

default

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CAssignment**(*lhs*, *rhs*, *tags=None*, ***kwargs*)

Bases: *CStatement*

a = b

__init__(*lhs*, *rhs*, *tags=None*, ***kwargs*)

lhs

rhs

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CFunctionCall**(*callee_target*, *callee_func*, *args*, *returning=True*, *ret_expr=None*, *tags=None*, *is_expr=False*, *show_demangled_name=True*, ***kwargs*)

Bases: *CStatement*, *CExpression*

func(arg0, arg1)

Variables

- **callee_func** (Function) – The function getting called.
- **is_expr** – True if the return value of the function is written to ret_expr; Essentially, ret_expr = call().

Parameters
is_expr (*bool*) –

__init__(*callee_target*, *callee_func*, *args*, *returning=True*, *ret_expr=None*, *tags=None*, *is_expr=False*, *show_demangled_name=True*, ***kwargs*)

Parameters
is_expr (*bool*) –

callee_target

callee_func: Optional[Function]

args

returning

ret_expr

tags

is_expr

show_demangled_name

property prototype: *SimTypeFunction* | None

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

Parameters

- **indent** – Number of whitespace indentation characters.
- **asexpr** (bool) – True if this call is used as an expression (which means we will skip the generation of semicolons and newlines at the end of the call).

class angr.analyses.decompiler.structured_codegen.c.**CReturn**(*retval*, *tags=None*, ***kwargs*)

Bases: *CStatement*

__init__(*retval*, *tags=None*, ***kwargs*)

retval

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CGoto**(*target*, *target_idx*, *tags=None*, ***kwargs*)

Bases: *CStatement*

__init__(*target*, *target_idx*, *tags=None*, ***kwargs*)

target: Union[int, *CExpression*]

target_idx

tags

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.**CLabel**(*name*, *ins_addr*, *block_idx*, *tags=None*, ***kwargs*)

Bases: *CStatement*

Represents a label in C code.

Parameters

- **name** (*str*) –
- **ins_addr** (*int*) –
- **block_idx** (*int | None*) –

__init__(*name*, *ins_addr*, *block_idx*, *tags=None*, ***kwargs*)

Parameters

- **name** (*str*) –
- **ins_addr** (*int*) –
- **block_idx** (*int | None*) –

name

ins_addr

block_idx

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CStructField**(*struct_type*, *offset*, *field*, *tags=None*, ***kwargs*)

Bases: *CExpression*

Parameters

struct_type (SimStruct) –

__init__(*struct_type*, *offset*, *field*, *tags=None*, ***kwargs*)

Parameters

struct_type (SimStruct) –

struct_type

offset

field

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CFakeVariable**(*name*, *ty*, *tags=None*, ***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*, *tags=None*, ***kwargs*)

Parameters

- **name** (*str*) –
- **ty** (SimType) –

name

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CVariable**(*variable*, *unified_variable=None*, *variable_type=None*, *tags=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*, *tags=None*, ***kwargs*)

Parameters

variable (SimVariable) –

variable: *SimVariable*

unified_variable: Optional[*SimVariable*]

variable_type: *SimType*

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CIndexedVariable**(*variable*, *index*, *variable_type=None*, *tags=None*, ***kwargs*)

Bases: *CExpression*

Represent a variable (an array) that is indexed.

Parameters

- **variable** (CExpression) –
- **index** (CExpression) –

__init__(*variable*, *index*, *variable_type=None*, *tags=None*, ***kwargs*)

Parameters

- **variable** (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*, *tags=None*, ***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*, *tags=None*, ***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*, *tags=None*, ***kwargs*)

Bases: *CExpression*

Unary operations.

Parameters
operand (CExpression) –

__init__(*op*, *operand*, *tags=None*, ***kwargs*)

Parameters
operand (CExpression) –

op

operand

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CBinaryOp**(*op*, *lhs*, *rhs*, *tags=None*, ***kwargs*)

Bases: *CExpression*

Binary operations.

Parameters
tags (*dict | None*) –

__init__(*op*, *lhs*, *rhs*, *tags=None*, ***kwargs*)

Parameters
tags (*dict | None*) –

op

lhs

rhs

tags

common_type

static compute_common_type(*op*, *lhs_ty*, *rhs_ty*)

Return type
SimType

Parameters

- **op** (*str*) –
- **lhs_ty** (SimType) –
- **rhs_ty** (SimType) –

property type

property op_precedence

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CTypeCast**(*src_type*, *dst_type*, *expr*, *tags=None*, ***kwargs*)

Bases: *CExpression*

Parameters

- **src_type** (*SimType | None*) –
- **dst_type** (*SimType*) –
- **expr** (*CExpression*) –

__init__(*src_type*, *dst_type*, *expr*, *tags=None*, ***kwargs*)

Parameters

- **src_type** (*SimType | None*) –
- **dst_type** (*SimType*) –
- **expr** (*CExpression*) –

src_type

dst_type

expr

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CConstant**(*value*, *type_*, *reference_values=None*, *tags=None*, ***kwargs*)

Bases: *CExpression*

Parameters

- **type_** (*SimType*) –
- **tags** (*Dict | None*) –

__init__(*value*, *type_*, *reference_values=None*, *tags=None*, ***kwargs*)

Parameters

- **type_** (*SimType*) –
- **tags** (*Dict | None*) –

value

reference_values

tags

property fmt

property fmt_hex

property fmt_neg

property fmt_char

property type

static str_to_c_str(*_str*, *prefix=''*)

Parameters
prefix (*str*) –

c_repr_chunks(*indent=0*, *asexpr=False*)

fmt_int(*value*)

Format an integer using the format setup of the current node.

Parameters
value (*int*) – The integer value to format.

Return type
str

Returns
The formatted string.

class angr.analyses.decompiler.structured_codegen.c.CRegister(*reg*, *tags=None*, ***kwargs*)

Bases: *CExpression*

__init__(*reg*, *tags=None*, ***kwargs*)

reg

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.CITE(*cond*, *iftrue*, *iffalse*, *tags=None*, ***kwargs*)

Bases: *CExpression*

__init__(*cond*, *iftrue*, *iffalse*, *tags=None*, ***kwargs*)

cond

iftrue

iffalse

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression(*stmts*, *expr*, *tags=None*, ***kwargs*)

Bases: *CExpression*

(stmt0, stmt1, stmt2, expr)

Parameters

- **stmts** (CStatements) –
- **expr** (CExpression) –

__init__(*stmts*, *expr*, *tags=None*, ***kwargs*)

Parameters

- **stmts** (CStatements) –
- **expr** (CExpression) –

stmts

expr

tags

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*, *ail_graph=None*, *simplify_else_scope=True*, *cstyle_ifs=True*)

Bases: *BaseStructuredCodeGenerator*, *Analysis*

Parameters

- **func_args** (*List[SimVariable] | None*) –
- **binop_depth_cutoff** (*int*) –

__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*, *ail_graph=None*, *simplify_else_scope=True*, *cstyle_ifs=True*)

Parameters

- **func_args** (*List[SimVariable] | None*) –
- **binop_depth_cutoff** (*int*) –

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(*cfunc*)

Return type
Tuple[str, *PositionMapping*, *PositionMapping*, *InstructionMapping*, Dict[Any, Set[Any]]]

Parameters
cfunc (CFunction) –

reload_variable_types()

Return type
None

default_simtype_from_size(*n*, *signed=True*)

Return type
SimType

Parameters

- **n** (*int*) –
- **signed** (*bool*) –

project: Project

kb: KnowledgeBase

class angr.analyses.decompiler.structured_codegen.c.**CStructuredCodeWalker**

Bases: object

classmethod handle(*obj*)

classmethod handle_default(*obj*)

classmethod handle_CFunction(*obj*)

classmethod handle_CStatements(*obj*)

classmethod handle_CWhileLoop(*obj*)

classmethod handle_CDoWhileLoop(*obj*)

classmethod handle_CForLoop(*obj*)

classmethod handle_CIfElse(*obj*)

classmethod handle_CIfBreak(*obj*)

classmethod handle_CSwitchCase(*obj*)

classmethod handle_CAssignment(*obj*)

classmethod handle_CFunctionCall(*obj*)

classmethod handle_CReturn(*obj*)

classmethod handle_CGoto(*obj*)

classmethod handle_CIndexedVariable(*obj*)

classmethod handle_CVariableField(*obj*)

classmethod handle_CUnaryOp(*obj*)

classmethod handle_CBinaryOp(*obj*)

classmethod handle_CTypeCast(*obj*)

classmethod 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) –

classmethod handle_CAssignment(*obj*)

classmethod handle_CFunctionCall(*obj*)

Parameters
obj (CFunctionCall) –

classmethod handle_CReturn(*obj*)

Parameters
obj (CReturn) –

classmethod handle_CBinaryOp(*obj*)

Parameters
obj (CBinaryOp) –

classmethod handle_CTypeCast(*obj*)

Parameters
obj (CTypeCast) –

class angr.analyses.decompiler.structured_codegen.c.**FieldReferenceCleanup**

Bases: *CStructuredCodeWalker*

classmethod 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.

classmethod handle_CBinaryOp(*obj*)

angr.analyses.decompiler.structured_codegen.c.**StructuredCodeGenerator**

alias of *CStructuredCodeGenerator*

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()

project: Project

kb: KnowledgeBase

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*) –

__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.**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) – The last statement of the switch-case header node.

Return type
Optional[Tuple[Any, int, int]]

Returns
A tuple of (comparison 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 | Tuple[int] | None*) –

angr.analyses.decompiler.utils.**to_ail_supergraph**(*transition_graph*)

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.**first_nonlabel_statement**(*block*)

Return type
Optional[Statement]

Parameters
block (*Block* | MultiNode) –

angr.analyses.decompiler.utils.**last_nonlabel_statement**(*block*)

Return type
Optional[Statement]

Parameters
block (*Block*) –

angr.analyses.decompiler.utils.**first_nonlabel_node**(*seq*)

Return type
Union[*BaseNode*, Block, None]

Parameters
seq (SequenceNode) –

angr.analyses.decompiler.utils.**remove_labels**(*graph*)

Parameters
graph (*DiGraph*) –

angr.analyses.decompiler.utils.**structured_node_is_simple_return**(*node*, *graph*)

Return type
bool

Parameters

- **node** (SequenceNode | MultiNode) –
- **graph** (*DiGraph*) –

Will check if a "simple return" is contained within the node a simple returns looks like this: if (cond) {

// simple return … return 0;

10.15.1 }

Returns true on any block ending in linear statements and a return.

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*)

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*)

angr.analyses.decompiler.utils.**decompile_functions**(*path*, *functions=None*, *structurer=None*, *catch_errors=False*)

Decompile a binary into a set of functions.

Parameters

- **path** – The path to the binary to decompile.
- **functions** – The functions to decompile. If None, all functions will be decompiled.
- **structurer** – The structuring algorithms to use.
- **catch_errors** – The structuring algorithms to use.

Return type
Optional[str]

Returns
The decompilation of all functions appended in order.

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 varaible 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

project: Project

kb: KnowledgeBase

class angr.analyses.flirt.**FlirtAnalysis**(*sig=None*)

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*) –

__init__(*sig=None*)

Parameters
sig (FlirtSignature | *str* | *None*) –

project: Project

kb: KnowledgeBase

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 '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.engine.**SimEngineLightMixin**(**args*, *logger=None*, ***kwargs*)

Bases: object

A mixin base class for engines meant to perform static analysis

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

static sp_offset(*bits*, *offset*)

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
Optional[int]

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.engine.**SimEngineLight**

Bases: *SimEngineLightMixin*, *SimEngine*

A full-featured engine base class, suitable for static analysis

__init__()

process(*state*, **args*, ***kwargs*)

The main entry point for an engine. Should take a state and return a result.

Parameters
state – The state to proceed from

Returns
The result. Whatever you want ;)

class angr.engines.light.engine.**SimEngineLightVEXMixin**(**args*, *logger=None*, ***kwargs*)

Bases: *SimEngineLightMixin*

A mixin for doing static analysis on VEX

class angr.engines.light.engine.**SimEngineLightAILMixin**(**args*, *logger=None*, ***kwargs*)

Bases: *SimEngineLightMixin*

A mixin for doing static analysis on AIL

angr.engines.light.engine.**SimEngineLightVEX**

alias of *SimEngineLightVEXMixin*

angr.engines.light.engine.**SimEngineLightAIL**

alias of *SimEngineLightAILMixin*

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

__init__(*addr*, *size*)

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.**SimEnginePropagatorBase**(*stack_pointer_tracker=None*, *project=None*, *propagate_tmps=True*, *arch=None*, *reaching_definitions=None*, *immediate_stmt_removal=False*, *bp_as_gpr=False*)

Bases: *SimEngineLight*

Parameters

- **reaching_definitions** (ReachingDefinitionsModel | *None*) –
- **immediate_stmt_removal** (*bool*) –
- **bp_as_gpr** (*bool*) –

__init__(*stack_pointer_tracker=None*, *project=None*, *propagate_tmps=True*, *arch=None*, *reaching_definitions=None*, *immediate_stmt_removal=False*, *bp_as_gpr=False*)

Parameters

- **reaching_definitions** (ReachingDefinitionsModel | *None*) –

- **immediate_stmt_removal** (*bool*) –
- **bp_as_gpr** (*bool*) –

process(*state*, **args*, ***kwargs*)

The main entry point for an engine. Should take a state and return a result.

Parameters
state – The state to proceed from

Returns
The result. Whatever you want ;)

class angr.analyses.propagator.engine_vex.**SimEnginePropagatorVEX**(*stack_pointer_tracker=None*, *project=None*, *propagate_tmps=True*, *arch=None*, *reaching_definitions=None*, *immediate_stmt_removal=False*, *bp_as_gpr=False*)

Bases: *TopCheckerMixin*, *SimEngineLightVEXMixin*, *SimEnginePropagatorBase*

Parameters

- **reaching_definitions** (*ReachingDefinitionsModel | None*) –
- **immediate_stmt_removal** (*bool*) –
- **bp_as_gpr** (*bool*) –

state: **PropagatorVEXState**

class angr.analyses.propagator.engine_ail.**SimEnginePropagatorAIL**(*stack_pointer_tracker=None*, *project=None*, *propagate_tmps=True*, *arch=None*, *reaching_definitions=None*, *immediate_stmt_removal=False*, *bp_as_gpr=False*)

Bases: *SimEngineLightAILMixin*, *SimEnginePropagatorBase*

The AIl engine for Propagator.

Parameters

- **reaching_definitions** (*ReachingDefinitionsModel | None*) –
- **immediate_stmt_removal** (*bool*) –
- **bp_as_gpr** (*bool*) –

state: **PropagatorAILState**

extract_offset_to_sp(*expr*)

Extract the offset to the original stack pointer.

Parameters

- **spoffset_expr** – The claripy AST to parse.
- **expr** (*Base | StackBaseOffset*) –

Return type
Optional[int]

Returns
The offset to the original stack pointer, or None if *spoffset_expr* is not a supported type of SpOffset expression.

is_using_outdated_def(*expr*, *expr_defat*, *current_loc*, *avoid=None*)

Return type
Tuple[bool, bool]

Parameters

- **expr** (*Expression*) –
- **expr_defat** (CodeLocation | *None*) –
- **current_loc** (CodeLocation) –
- **avoid** (*Expression* | *None*) –

should_force_replace(*stmt*, *new_expr*)

Determine if the expression should be replaced.

We always replace the expression if:

- the current statement is an indirect jump. this is to ensure the dynamically calculated jump targets are always using the originally defined expressions, which usually leads to better decompilation output.
- the current statement is a return to make void functions (even when we incorrectly determine that they return something) look better in general.
- the current statement has a shift-right operation and the source expression has a shift-right operation. this is to support the peephole optimizations for division and modulo.

Parameters

- **stmt** (Statement) –
- **new_expr** (Expression) –

Return type
bool

Returns

static has_tmpexpr(*expr*)

Return type
bool

Parameters
expr (*Expression*) –

class angr.analyses.propagator.outdated_definition_walker.**OutdatedDefinitionWalker**(*expr*, *expr_defat*, *current_loc*, *state*, *arch*, *avoid=None*, *extract_offset_to_sp=None*, *rda=None*)

Bases: AILBlockWalker

Walks an AIL expression to find outdated definitions.

Parameters

- **expr_defat** (CodeLocation) –
- **current_loc** (CodeLocation) –
- **state** (*PropagatorAILState*) –
- **arch** (*Arch*) –
- **avoid** (*Expression | None*) –
- **extract_offset_to_sp** (*Callable*) –
- **rda** (ReachingDefinitionsModel) –

__init__(*expr*, *expr_defat*, *current_loc*, *state*, *arch*, *avoid=None*, *extract_offset_to_sp=None*, *rda=None*)

Parameters

- **expr_defat** (CodeLocation) –
- **current_loc** (CodeLocation) –
- **state** (*PropagatorAILState*) –
- **arch** (*Arch*) –
- **avoid** (*Expression | None*) –
- **extract_offset_to_sp** (*Callable | None*) –
- **rda** (ReachingDefinitionsModel | *None*) –

class angr.analyses.propagator.tmpvar_finder.**TmpvarFinder**(*expr*)

Bases: AILBlockWalkerBase

Walks an AIL expression to find Tmp expressions.

Parameters

expr (*Expression*) –

__init__(*expr*)

Parameters

expr (*Expression*) –

class angr.analyses.propagator.propagator.**PropagatorAnalysis**(*func=None*, *block=None*, *func_graph=None*, *base_state=None*, *max_iterations=3*, *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*, *reaching_definitions=None*, *immediate_stmt_removal=False*, *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 supports both VEX and AIL. The VEX propagator only performs constant propagation. The AIL propagator performs both constant propagation and copy propagation of depth-N expressions.

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

__init__(*func=None*, *block=None*, *func_graph=None*, *base_state=None*, *max_iterations=3*, *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*, *reaching_definitions=None*, *immediate_stmt_removal=False*, *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*) –
- **reaching_definitions** (*ReachingDefinitionsModel | None*) –
- **immediate_stmt_removal** (*bool*) –
- **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

project: Project

kb: KnowledgeBase

class angr.analyses.propagator.top_checker_mixin.**TopCheckerMixin**(**args*, *logger=None*, ***kwargs*)

Bases: *SimEngineLightMixin*

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*)

Bases: object

A LiveDefinitions instance contains definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

Parameters

- **arch** (*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]]*) –

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*)

Parameters

- **arch** (*Arch*) –
- **track_tmps** (*bool*) –

project: Optional[Project]

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*]]

property register_definitions

property stack_definitions

property memory_definitions

property heap_definitions

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
Optional[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
Optional[int]

Parameters
addr (*Base*) –

static annotate_with_def(*symvar*, *definition*)

Parameters

- **symvar** (BV) –
- **definition** (*Definition*) –

Return type
BV

Returns

static extract_defs(*symvar*)

Return type
Generator[*Definition*, None, None]

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*, None, None]

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
Optional[int]

get_stack_address(*offset*)

Return type
Optional[int]

Parameters
offset (*Base*) –

stack_offset_to_stack_addr(*offset*)

Return type
int

merge(**others*)

Return type
Tuple[*LiveDefinitions*, bool]

Parameters
others (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
Optional[*MultiValues*]

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 | None*) –

get_definitions(*thing*)

Return type
Set[*Definition*[*Atom*]]

Parameters
thing (Atom | Definition[Atom] | *Iterable*[Atom] | *Iterable*[Definition[Atom]] | MultiValues) –

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
Optional[*MultiValues*]

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_definitions_from_atoms(***kwargs*)

get_value_from_definition(***kwargs*)

get_one_value_from_definition(***kwargs*)

get_concrete_value_from_definition(***kwargs*)

get_value_from_atom(***kwargs*)

get_one_value_from_atom(***kwargs*)

get_concrete_value_from_atom(***kwargs*)

get_values(*spec*)

Return type
Optional[*MultiValues*]

Parameters
spec (*Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]]*) –

get_one_value(*spec*, *strip_annotations=False*)

Return type
Optional[BV]

Parameters

- **spec** (Atom | Definition | *Iterable*[Atom] | *Iterable*[Definition[Atom]]) –
- **strip_annotations** (*bool*) –

get_concrete_value(*spec*, *cast_to=<class 'int'>*)

Return type
Union[int, bytes, None]

Parameters

- **spec** (Atom | Definition[Atom] | *Iterable*[Atom] | *Iterable*[Definition[Atom]]) –
- **cast_to** (*Type*[*int*] | *Type*[*bytes*]) –

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*)

static is_heap_address(*addr*)

Return type
bool

Parameters
addr (*Base*) –

static get_heap_offset(*addr*)

Return type
Optional[int]

Parameters
addr (*Base*) –

heap_address(*offset*)

Return type
BV

Parameters
offset (*int | HeapAddress*) –

class angr.analyses.reaching_definitions.**ObservationPointType**(*value*)

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.**AtomKind**(*value*)

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.**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*)

Instanciate an *Atom* from a given argument.

Parameters

- **argument** (*SimFunctionArgument*) – The argument to create a new atom from.
- **registers** – A mapping representing the registers of a given architecture.
- **full_reg** – Whether to return an atom indicating the entire register if the argument only specifies a slice of the register.
- **sp** (Optional[int]) – The current stack offset. Optional. Only used when argument is a SimStackArg.
- **arch** (*Arch*) –

Return type
Union[*Register*, *MemoryLocation*]

static reg(*thing*, *size=None*, *arch=None*)

Create a Register atom.

Parameters

- **thing** (Union[str, RegisterOffset]) – The register offset (e.g., project.arch.registers["rax"][0]) or the register name (e.g., "rax").

- **size** (Optional[int]) – Size of the register atom. Must be provided when creating the atom using a register offset.
- **arch** (Optional[Arch]) – 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** (Union[str, RegisterOffset]) – The register offset (e.g., project.arch.registers["rax"][0]) or the register name (e.g., "rax").
- **size** (Optional[int]) – Size of the register atom. Must be provided when creating the atom using a register offset.
- **arch** (Optional[Arch]) – 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** (Union[*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** (Optional[str]) – 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** (Union[*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** (Optional[str]) – Optional, either "Iend_LE" or "Iend_BE".

Return type
MemoryLocation

Returns
The MemoryLocation Atom object.

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*) –
- **size** (*int*) –
- **arch** (*Arch | None*) –

__init__(*reg_offset*, *size*, *arch=None*)

Parameters

- **size** (int) – The size of the atom in bytes
- **reg_offset** (*RegisterOffset*) –
- **arch** (*Arch | None*) –

reg_offset

arch

property name: str

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** (*str | None*) –

__init__(*addr*, *size*, *endness=None*)

Parameters

- **addr** (*int*) – The address of the beginning memory location slice.
- **size** (*int*) – The size of the represented memory location, in bytes.
- **endness** (*str | None*) –

addr: Union[*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.**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.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.**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[A]

An atom definition.

Variables

- **atom** – The atom being defined.
- **codeloc** – Where this definition is created in the original binary code.
- **dummy** – Tell whether the definition should be considered dummy or not. During simplification by AILment, definitions marked as dummy will not be removed.
- **tags** – A set of tags containing information about the definition gathered during analyses.

__init__(*atom*, *codeloc*, *dummy=False*, *tags=None*)

Parameters

- **atom** (*A*) –
- **codeloc** (CodeLocation) –
- **dummy** (*bool*) –
- **tags** (*Set[Tag] | None*) –

atom: TypeVar(A, bound= *Atom*)

codeloc: *CodeLocation*

dummy: bool

tags

property offset: int

property size: int

matches(***kwargs*)

Return whether this definition has certain characteristics.

Return type
bool

class angr.analyses.reaching_definitions.**ReachingDefinitionsAnalysis**(*subject=None*, *func_graph=None*, *max_iterations=3*, *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*)

Bases: *ForwardAnalysis*[*ReachingDefinitionsState*, NodeType, object, object], *Analysis*

ReachingDefinitionsAnalysis is a text-book implementation of a static data-flow analysis that works on either a function or a block. It supports both VEX and AIL. By registering observers to observation points, users may use this analysis to generate use-def chains, def-use chains, and reaching definitions, and perform other traditional data-flow analyses such as liveness analysis.

- I've always wanted to find a better name for this analysis. Now I gave up and decided to live with this name for the foreseeable future (until a better name is proposed by someone else).
- Aliasing is definitely a problem, and I forgot how aliasing is resolved in this implementation. I'll leave this as a post-graduation TODO.
- Some more documentation and examples would be nice.

__init__(*subject=None*, *func_graph=None*, *max_iterations=3*, *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*)

Parameters

- **subject** (Union[*Subject*, Block, *Block*, *Function*, str, None]) – 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*) – 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** (Optional[*ReachingDefinitionsState*]) – 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** (Optional[*FunctionHandler*]) – 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** (Union[*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)
- **state_initializer** (*RDAStateInitializer | None*) –
- **func_addr** (*int | None*) –

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(***kwargs*)

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** (Optional[int]) – 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** (Union[Statement, IRStmt]) – The statement.
- **block** (Union[*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** (Union[Statement, IRStmt]) –
- **block** (Union[*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

project: Project

kb: KnowledgeBase

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
Optional[*LiveDefinitions*]

Parameters

- **ins_addr** (*int* | CodeLocation) –
- **kind** (ObservationPointType) –

get_observation_by_node(*node_addr*, *kind*, *node_idx=None*)

Return type
Optional[*LiveDefinitions*]

Parameters

- **node_addr** (*int* | CodeLocation) –
- **kind** (ObservationPointType) –
- **node_idx** (*int* | *None*) –

get_observation_by_stmt(*arg1*, *arg2*, *arg3=None*, ***, *block_idx=None*)

get_observation_by_exit(*node_addr*, *stmt_idx*, *src_node_idx=None*)

Return type
Optional[*LiveDefinitions*]

Parameters

- **node_addr** (*int*) –
- **stmt_idx** (*int*) –
- **src_node_idx** (*int | None*) –

class angr.analyses.reaching_definitions.**ReachingDefinitionsState**(*codeloc*, *arch*, *subject*, *track_tmps=False*, *track_consts=False*, *analysis=None*, *rtoc_value=None*, *live_definitions=None*, *canonical_size=8*, *heap_allocator=None*, *environment=None*, *sp_adjusted=False*, *all_definitions=None*, *initializer=None*)

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** (Optional[*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** (Optional[*LiveDefinitions*]) –
- **canonical_size** (int) – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.
- **heap_allocator** (Optional[*HeapAllocator*]) – Mechanism to model the management of heap memory.
- **environment** (Optional[*Environment*]) – Representation of the environment of the analyzed program.
- **codeloc** (CodeLocation) –
- **arch** (*Arch*) –
- **track_consts** (*bool*) –
- **sp_adjusted** (*bool*) –

- **all_definitions** (*Set[Definition] | None*) –
- **initializer** (*RDAStateInitializer | None*) –

Variables
arch – The architecture targeted by the program.

__init__(*codeloc*, *arch*, *subject*, *track_tmps=False*, *track_consts=False*, *analysis=None*, *rtoc_value=None*, *live_definitions=None*, *canonical_size=8*, *heap_allocator=None*, *environment=None*, *sp_adjusted=False*, *all_definitions=None*, *initializer=None*)

Parameters

- **codeloc** (*CodeLocation*) –
- **arch** (*Arch*) –
- **subject** (*Subject*) –
- **track_tmps** (*bool*) –
- **track_consts** (*bool*) –
- **analysis** (*ReachingDefinitionsAnalysis | None*) –
- **live_definitions** (*LiveDefinitions | None*) –
- **canonical_size** (*int*) –
- **heap_allocator** (*HeapAllocator | None*) –
- **environment** (*Environment | None*) –
- **sp_adjusted** (*bool*) –
- **all_definitions** (*Set[Definition] | None*) –
- **initializer** (*RDAStateInitializer | None*) –

codeloc

arch: *Arch*

analysis

all_definitions: *Set[Definition]*

heap_allocator

codeloc_uses: *Set[Definition]*

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
Optional[int]

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
Optional[int]

Parameters
addr (*Base*) –

annotate_with_def(*symvar*, *definition*)

Parameters

- **symvar** (Base) –
- **definition** (*Definition*) –

Return type
Base

Returns

annotate_mv_with_def(*mv*, *definition*)

Return type
MultiValues

Parameters

- **mv** (MultiValues) –
- **definition** (Definition) –

extract_defs(*symvar*)

Return type
Iterator[*Definition*]

Parameters
symvar (*Base*) –

property tmps

property tmp_uses

property register_uses

property registers: *MultiValuedMemory*

property stack: *MultiValuedMemory*

property stack_uses

property heap: *MultiValuedMemory*

property heap_uses

property memory_uses

property memory: *MultiValuedMemory*

property uses_by_codeloc

get_sp()

Return type
int

get_stack_address(*offset*)

Return type
int

Parameters
offset (*Base*) –

property environment

property dep_graph

copy(*discard_tmpdefs=False*)

Return type
ReachingDefinitionsState

merge(**others*)

Return type
Tuple[*ReachingDefinitionsState*, bool]

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[Optional[*MultiValues*], Set[*Definition*]]

Parameters

- **atom** (Atom) –
- **data** (MultiValues) –
- **tags** (*Set[Tag] | None*) –
- **annotated** (*bool*) –
- **uses** (*Set[Definition] | 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) –
- **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]*) –
- **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]*) –
- **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]*) –
- **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]*) –
- **expr** (*Any | None*) –

add_memory_use_by_def(*definition*, *expr=None*)

Parameters

- **definition** (Definition) –
- **expr** (*Any | None*) –

add_memory_use_by_defs(*defs*, *expr=None*)

Parameters

- **defs** (*Iterable[Definition]*) –
- **expr** (*Any | None*) –

get_definitions(*atom*)

Return type
Set[*Definition*]

Parameters
atom (Atom | Definition | *Iterable[Atom]* | *Iterable[Definition]*) –

get_values(*spec*)

Return type
Optional[*MultiValues*]

Parameters
spec (Atom | Definition | *Iterable[Atom]*) –

get_one_value(*spec*, *strip_annotations=False*)

Return type
Optional[BV]

Parameters

- **spec** (Atom | Definition) –
- **strip_annotations** (*bool*) –

get_concrete_value(*spec*, *cast_to=<class 'int'>*)

Return type
Union[int, bytes, None]

Parameters

- **spec** (Atom | Definition[Atom] | Iterable[Atom]) –
- **cast_to** (Type[int] | Type[bytes]) –

mark_guard(*target*)

mark_const(*value*, *size*)

Parameters

- **value** (*int*) –
- **size** (*int*) –

downsize()

pointer_to_atoms(***kwargs*)

pointer_to_atom(***kwargs*)

deref(*pointer*, *size*, *endness=Endness.BE*)

Parameters

- **pointer** (MultiValues | Atom | Definition | Iterable[Atom] | Iterable[Definition] | int | BV | HeapAddress | SpOffset) –
- **size** (*int* | DerefSize) –
- **endness** (*str*) –

class angr.analyses.reaching_definitions.**FunctionHandler**(*interfunction_level=0*)

Bases: object

A mechanism for summarizing a function call's effect on a program for ReachingDefinitionsAnalysis.

Parameters
interfunction_level (*int*) –

__init__(*interfunction_level=0*)

Parameters
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.**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 | 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] | None) –
- **ret_atoms** (Set[Atom] | None) –
- **redefine_locals** (bool) –

- **visited_blocks** (*Set[int] | None*) –
- **effects** (*List[FunctionEffect]*) –
- **ret_values** (*MultiValues | 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: Optional[*MultiValues*]

address: Optional[int] = None

symbol: Optional[Symbol] = None

function: Optional[*Function*] = None

name: Optional[str] = None

cc: Optional[*SimCC*] = None

prototype: Optional[*SimTypeFunction*] = None

args_atoms: Optional[List[Set[*Atom*]]] = None

args_values: Optional[List[*MultiValues*]] = None

ret_atoms: Optional[Set[*Atom*]] = None

redefine_locals: bool = True

visited_blocks: Optional[Set[int]] = None

effects: List[*FunctionEffect*]

ret_values: Optional[*MultiValues*] = None

ret_values_deps: Optional[Set[*Definition*]] = 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 | *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] | None) –
- **ret_atoms** (Set[Atom] | None) –
- **redefine_locals** (bool) –
- **visited_blocks** (Set[int] | None) –
- **effects** (List[FunctionEffect]) –
- **ret_values** (MultiValues | 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

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*, *functions=None*, *function_handler=None*)

Bases: *SimEngineLightVEXMixin*, *SimEngineLight*

Implements the VEX execution engine for reaching definition analysis.

__init__(*project*, *functions=None*, *function_handler=None*)

process(*state*, **args*, *block=None*, *fail_fast=False*, *visited_blocks=None*, *dep_graph=None*, ***kwargs*)

The main entry point for an engine. Should take a state and return a result.

Parameters
state – The state to proceed from

Returns
The result. Whatever you want ;)

class angr.analyses.reaching_definitions.reaching_definitions.**ReachingDefinitionsAnalysis**(*subject=None*, *func_graph=None*, *max_iterations=3*, *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*)

Bases: *ForwardAnalysis*[*ReachingDefinitionsState*, NodeType, object, object], *Analysis*

ReachingDefinitionsAnalysis is a text-book implementation of a static data-flow analysis that works on either a function or a block. It supports both VEX and AIL. By registering observers to observation points, users may use this analysis to generate use-def chains, def-use chains, and reaching definitions, and perform other traditional data-flow analyses such as liveness analysis.

- I've always wanted to find a better name for this analysis. Now I gave up and decided to live with this name for the foreseeable future (until a better name is proposed by someone else).
- Aliasing is definitely a problem, and I forgot how aliasing is resolved in this implementation. I'll leave this as a post-graduation TODO.
- Some more documentation and examples would be nice.

__init__(*subject=None*, *func_graph=None*, *max_iterations=3*, *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*)

Parameters

- **subject** (Union[*Subject*, Block, *Block*, *Function*, str, None]) – 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*) – 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** (Optional[*ReachingDefinitionsState*]) – 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** (Optional[*FunctionHandler*]) – 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** (Union[*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)
- **state_initializer** (*RDAStateInitializer | None*) –
- **func_addr** (*int | None*) –

model: ReachingDefinitionsModel

function_calls: Dict[CodeLocation, FunctionCallRelationships]

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(***kwargs*)

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** (Optional[int]) – 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** (Union[Statement, IRStmt]) – The statement.
- **block** (Union[*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** (Union[Statement, IRStmt]) –
- **block** (Union[*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

project: Project

kb: KnowledgeBase

callsites_to(*target*)

Return type
Iterable[*FunctionCallRelationships*]

Parameters
target (*int* | *str* | Function) –

class angr.analyses.reaching_definitions.dep_graph.**FunctionCallRelationships**(*callsite*, *target*, *args_defns*, *other_input_defns*, *ret_defns*, *other_output_defns*)

Bases: object

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: Optional[int]

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.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>.

Parameters
graph (*networkx.DiGraph[Definition] | None*) –

__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
networkx.classes.reportviews.NodeView[*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[Union[Base, int]]) –
- **definition** (*Definition*) – The definition which has data that can contain concrete pointers.
- **cfg** (*CFGModel*) – The CFG, containing information about memory data.
- **loader** (Loader) –

find_definitions(***kwargs*)

Filter the definitions present in the graph based on various criteria. Parameters can be any valid keyword args to *DefinitionMatchPredicate*

Return type
List[*Definition*]

find_all_predecessors(*starts*, ***kwargs*)

Filter the ancestors of the given start node or nodes that match various criteria. Parameters can be any valid keyword args to *DefinitionMatchPredicate*

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
Optional[Tuple[*Definition*, ...]]

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 aforementionned.

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 chunck of <size> bytes.

Parameters
size (Union[int, *UnknownSize*]) – The requested size for the chunck, in number of bytes.

Return type
HeapAddress

Returns
The address of the chunck.

free(*address*)

Mark the chunck pointed by <address> as freed.

Parameters
address (Union[*Undefined*, *HeapAddress*]) – The address of the chunck 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: Optional[*Atom*]

sources: Set[*Atom*]

value: Optional[*MultiValues*] = None

sources_defns: Optional[Set[*Definition*]] = None

apply_at_callsite: bool = False

tags: Optional[Set[*Tag*]] = 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 | 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] | None) –
- **ret_atoms** (Set[Atom] | None) –
- **redefine_locals** (bool) –
- **visited_blocks** (Set[int] | None) –
- **effects** (List[FunctionEffect]) –
- **ret_values** (MultiValues | 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: Optional[*MultiValues*]

address: Optional[int] = None

symbol: Optional[Symbol] = None

function: Optional[*Function*] = None

name: Optional[str] = None

cc: Optional[*SimCC*] = None

prototype: Optional[*SimTypeFunction*] = None

args_atoms: Optional[List[Set[*Atom*]]] = None

args_values: Optional[List[*MultiValues*]] = None

ret_atoms: Optional[Set[*Atom*]] = None

redefine_locals: bool = True

visited_blocks: Optional[Set[int]] = None

effects: List[*FunctionEffect*]

ret_values: Optional[*MultiValues*] = None

ret_values_deps: Optional[Set[*Definition*]] = 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 | *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]* | *None*) –
- **ret_atoms** (*Set[Atom]* | *None*) –
- **redefine_locals** (*bool*) –
- **visited_blocks** (*Set[int]* | *None*) –
- **effects** (*List[FunctionEffect]*) –
- **ret_values** (MultiValues | *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__')*, *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.**FunctionHandler**(*interfunction_level=0*)

Bases: object

A mechanism for summarizing a function call's effect on a program for ReachingDefinitionsAnalysis.

Parameters
interfunction_level (*int*) –

__init__(*interfunction_level=0*)

Parameters
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*, *track_tmps=False*, *track_consts=False*, *analysis=None*, *rtoc_value=None*, *live_definitions=None*, *canonical_size=8*, *heap_allocator=None*, *environment=None*, *sp_adjusted=False*, *all_definitions=None*, *initializer=None*)

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** (Optional[*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** (Optional[*LiveDefinitions*]) –
- **canonical_size** (int) – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.
- **heap_allocator** (Optional[*HeapAllocator*]) – Mechanism to model the management of heap memory.
- **environment** (Optional[*Environment*]) – Representation of the environment of the analyzed program.
- **codeloc** (CodeLocation) –
- **arch** (*Arch*) –
- **track_consts** (*bool*) –
- **sp_adjusted** (*bool*) –
- **all_definitions** (*Set[Definition]*) –
- **initializer** (*RDAStateInitializer | None*) –

Variables

arch – The architecture targeted by the program.

__init__(*codeloc*, *arch*, *subject*, *track_tmps=False*, *track_consts=False*, *analysis=None*, *rtoc_value=None*, *live_definitions=None*, *canonical_size=8*, *heap_allocator=None*, *environment=None*, *sp_adjusted=False*, *all_definitions=None*, *initializer=None*)

Parameters

- **codeloc** (CodeLocation) –
- **arch** (*Arch*) –
- **subject** (Subject) –
- **track_tmps** (*bool*) –
- **track_consts** (*bool*) –
- **analysis** (ReachingDefinitionsAnalysis | *None*) –
- **live_definitions** (LiveDefinitions | *None*) –
- **canonical_size** (*int*) –
- **heap_allocator** (HeapAllocator | *None*) –
- **environment** (Environment | *None*) –
- **sp_adjusted** (*bool*) –
- **all_definitions** (*Set[Definition]* | *None*) –
- **initializer** (*RDAStateInitializer* | *None*) –

codeloc

arch: Arch

analysis

all_definitions: Set[*Definition*]

heap_allocator

codeloc_uses: Set[*Definition*]

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
Optional[int]

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
Optional[int]

Parameters
addr (*Base*) –

annotate_with_def(*symvar*, *definition*)

Parameters

- **symvar** (Base) –
- **definition** (*Definition*) –

Return type
Base

Returns

annotate_mv_with_def(*mv*, *definition*)

Return type
MultiValues

Parameters

- **mv** (MultiValues) –
- **definition** (Definition) –

extract_defs(*symvar*)

Return type
Iterator[*Definition*]

Parameters
symvar (*Base*) –

property tmps

property tmp_uses

property register_uses

property registers: *MultiValuedMemory*

property stack: *MultiValuedMemory*

property stack_uses

property heap: *MultiValuedMemory*

property heap_uses

property memory_uses

property memory: *MultiValuedMemory*

property uses_by_codeloc

get_sp()

Return type
int

get_stack_address(*offset*)

Return type
int

Parameters
offset (*Base*) –

property environment

property dep_graph

copy(*discard_tmpdefs=False*)

Return type
ReachingDefinitionsState

merge(**others*)

Return type
Tuple[*ReachingDefinitionsState*, bool]

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[Optional[*MultiValues*], Set[*Definition*]]

Parameters

- **atom** (Atom) –
- **data** (MultiValues) –
- **tags** (*Set[Tag] | None*) –
- **annotated** (*bool*) –
- **uses** (*Set[Definition] | 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) –
- **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]*) –
- **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]*) –
- **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]*) –
- **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]*) –
- **expr** (*Any | None*) –

add_memory_use_by_def(*definition*, *expr=None*)

Parameters

- **definition** (*Definition*) –
- **expr** (*Any | None*) –

add_memory_use_by_defs(*defs*, *expr=None*)

Parameters

- **defs** (*Iterable[Definition]*) –
- **expr** (*Any | None*) –

get_definitions(*atom*)

Return type
Set[*Definition*]

Parameters
atom (*Atom | Definition | Iterable[Atom] | Iterable[Definition]*) –

get_values(*spec*)

Return type
Optional[*MultiValues*]

Parameters
spec (*Atom | Definition | Iterable[Atom]*) –

get_one_value(*spec*, *strip_annotations=False*)

Return type
Optional[BV]

Parameters

- **spec** (*Atom | Definition*) –
- **strip_annotations** (*bool*) –

get_concrete_value(*spec*, *cast_to=<class 'int'>*)

Return type
Union[int, bytes, None]

Parameters

- **spec** (*Atom | Definition[Atom] | Iterable[Atom]*) –
- **cast_to** (*Type[int] | Type[bytes]*) –

mark_guard(*target*)

mark_const(*value*, *size*)

Parameters

- **value** (*int*) –
- **size** (*int*) –

downsize()

pointer_to_atoms(***kwargs*)

pointer_to_atom(***kwargs*)

deref(*pointer*, *size*, *endness=Endness.BE*)

Parameters

- **pointer** (MultiValues | Atom | Definition | *Iterable*[Atom] | *Iterable*[Definition] | *int* | *BV* | HeapAddress | SpOffset) –
- **size** (*int* | DerefSize) –
- **endness** (*str*) –

class angr.analyses.reaching_definitions.subject.**SubjectType**(*value*)

Bases: Enum

An enumeration.

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=None*, *stack_pointer_tracker=None*, *use_callee_saved_regs_at_return=True*, *bp_as_gpr=False*)

Bases: *SimEngineLightAILMixin*, *SimEngineLight*

Parameters

- **function_handler** (FunctionHandler | *None*) –
- **bp_as_gpr** (*bool*) –

arch: Arch

state: *ReachingDefinitionsState*

__init__(*project*, *function_handler=None*, *stack_pointer_tracker=None*, *use_callee_saved_regs_at_return=True*, *bp_as_gpr=False*)

Parameters

- **function_handler** (FunctionHandler | *None*) –
- **bp_as_gpr** (*bool*) –

process(*state*, **args*, *dep_graph=None*, *visited_blocks=None*, *block=None*, *fail_fast=False*, ***kwargs*)

The main entry point for an engine. Should take a state and return a result.

Parameters
state – The state to proceed from

Returns
The result. Whatever you want ;)

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 adresses.

Parameters

- **source** (int) – The source address.
- **destination** (int) – The destination address.
- **visited** (Optional[Set[Any]]) – 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.**FrozenStackPointerTrackerState**(*regs*, *memory*, *is_tracking_memory*)

Bases: object

Abstract state for StackPointerTracker analysis with registers and memory values being in frozensets.

__init__(*regs*, *memory*, *is_tracking_memory*)

regs

memory

is_tracking_memory

unfreeze()

merge(*other*)

class angr.analyses.stack_pointer_tracker.**StackPointerTrackerState**(*regs*, *memory*, *is_tracking_memory*)

Bases: object

Abstract state for StackPointerTracker analysis.

__init__(*regs*, *memory*, *is_tracking_memory*)

regs

memory

is_tracking_memory

give_up_on_memory_tracking()

store(*addr*, *val*)

load(*addr*)

get(*reg*)

put(*reg*, *val*)

copy()

freeze()

merge(*other*)

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*)

Bases: *Analysis*, *ForwardAnalysis*

Track the offset of stack pointer at the end of each basic block of a function.

__init__(*func*, *reg_offsets*, *block=None*, *track_memory=True*)

Parameters

- **func** (Function | *None*) –
- **reg_offsets** (*Set[int]*) –
- **block** (Block | *None*) –

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*)

project: Project

kb: KnowledgeBase

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*)

Bases: *Analysis*

The base class for VariableRecovery and VariableRecoveryFast.

Parameters
store_live_variables (*bool*) –

__init__(*func*, *max_iterations*, *store_live_variables*)

Parameters
store_live_variables (*bool*) –

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()

project: Project

kb: KnowledgeBase

class angr.analyses.variable_recovery.variable_recovery_base.**VariableRecoveryStateBase**(*block_addr*, *analysis*, *arch*, *func*, *stack_region=None*, *register_region=None*, *global_region=None*, *typevars=None*, *type_constraints=None*, *delayed_type_constraints=None*, *stack_offset_typevars=None*, *project=None*)

Bases: object

The base abstract state for variable recovery analysis.

__init__(*block_addr*, *analysis*, *arch*, *func*, *stack_region=None*, *register_region=None*, *global_region=None*, *typevars=None*, *type_constraints=None*, *delayed_type_constraints=None*, *stack_offset_typevars=None*, *project=None*)

static top(*bits*)

Return type
BV

static is_top(*thing*)

Return type
bool

static extract_variables(*expr*)

Return type
Generator[Tuple[int, Union[*SimVariable*, *SpOffset*]], None, None]

Parameters
expr (*Base*) –

static annotate_with_variables(*expr*, *addr_and_variables*)

Return type
Base

Parameters

- **expr** (*Base*) –
- **addr_and_variables** (*Iterable[Tuple[int, SimVariable | SpOffset]]*) –

stack_address(*offset*)

Return type
Base

Parameters
offset (*int*) –

static is_stack_address(*addr*)

Return type
bool

Parameters
addr (*Base*) –

is_global_variable_address(*addr*)

Return type
bool

Parameters
addr (*Base*) –

static extract_stack_offset_from_addr(*addr*)

Return type
Optional[Base]

Parameters
addr (*Base*) –

get_stack_offset(*addr*)

Return type
Optional[int]

Parameters
addr (*Base*) –

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

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*, *stack_region=None*, *register_region=None*, *global_region=None*, *typevars=None*, *type_constraints=No*
delayed_type_constrain
stack_offset_typevars
project=None, *ret_val_size=None*)

Bases: *VariableRecoveryStateBase*

The abstract state of variable recovery analysis.

Variables

- **stack_region** (KeyedRegion) – The stack store.
- **register_region** (KeyedRegion) – The register store.

__init__(*block_addr*, *analysis*, *arch*, *func*, *stack_region=None*, *register_region=None*, *global_region=None*, *typevars=None*, *type_constraints=None*, *delayed_type_constraints=None*, *stack_offset_typevars=None*, *project=None*, *ret_val_size=None*)

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*, *max_iterations=2*, *low_priority=False*, *track_sp=True*, *func_args=None*, *store_live_variables=False*, *unify_variables=True*)

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`.

__init__(*func*, *func_graph=None*, *max_iterations=2*, *low_priority=False*, *track_sp=True*, *func_args=None*, *store_live_variables=False*, *unify_variables=True*)

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*) –
- **max_iterations** (*int*) –
- **func_args** (*List[SimVariable]* | *None*) –

Returns
None

project: Project

kb: KnowledgeBase

class angr.analyses.variable_recovery.variable_recovery.**VariableRecoveryState**(*block_addr*, *analysis*, *arch*, *func*, *concrete_states*, *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.

__init__(*block_addr*, *analysis*, *arch*, *func*, *concrete_states*, *stack_region=None*, *register_region=None*)

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
VariableRecoveryState, and a boolean that indicates if any merge has happened.

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 varaibles. 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.

project: Project

kb: KnowledgeBase

class angr.analyses.variable_recovery.engine_ail.**SimEngineVRAIL**(**args*, *call_info=None*, ***kwargs*)

Bases: *SimEngineLightAILMixin*, *SimEngineVRBase*

The engine for variable recovery on AIL.

state: VariableRecoveryFastState

block: Block

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

class angr.analyses.variable_recovery.engine_vex.**SimEngineVRVEX**(**args*, *call_info=None*, ***kwargs*)

Bases: *SimEngineLightVEXMixin*, *SimEngineVRBase*

Implements the VEX engine for variable recovery analysis.

state: VariableRecoveryStateBase

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

class angr.analyses.variable_recovery.engine_base.**RichR**(*data*, *variable=None*, *typevar=None*, *type_constraints=None*)

Bases: object

A rich representation of calculation results. The variable recovery data domain.

Parameters

- **data** (*Base*) –
- **typevar** (TypeVariable | *None*) –

__init__(*data*, *variable=None*, *typevar=None*, *type_constraints=None*)

Parameters

- **data** (*Base*) –
- **typevar** (TypeVariable | *None*) –

data: Base

variable

typevar

type_constraints

property bits

class angr.analyses.variable_recovery.engine_base.**SimEngineVRBase**(*project*, *kb*)

Bases: *SimEngineLight*

The base class for variable recovery analyses. Contains methods for basic interactions with the state, like loading and storing data.

state: VariableRecoveryStateBase

__init__(*project*, *kb*)

property func_addr

process(*state*, **args*, ***kwargs*)

The main entry point for an engine. Should take a state and return a result.

Parameters
state – The state to proceed from

Returns
The result. Whatever you want ;)

class angr.analyses.variable_recovery.irsb_scanner.**VEXIRSBScanner**(**args*, ***kwargs*)

Bases: *SimEngineLightVEXMixin*

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*)

class angr.analyses.typehoon.lifter.**TypeLifter**(*bits*)

Bases: object

Lift SimTypes to type constants.

Parameters
bits (*int*) –

__init__(*bits*)

Parameters
bits (*int*) –

bits

lift(*ty*)

Parameters
ty (SimType) –

class angr.analyses.typehoon.simple_solver.**RecursiveType**(*typevar*, *offset*)

Bases: object

__init__(*typevar*, *offset*)

class angr.analyses.typehoon.simple_solver.**SimpleSolver**(*bits*, *constraints*)

Bases: object

SimpleSolver is, literally, a simple, unification-based type constraint solver.

Parameters
bits (*int*) –

__init__(*bits*, *constraints*)

Parameters
bits (*int*) –

solve()

determine()

class angr.analyses.typehoon.translator.**SimTypeTempRef**(*typevar*)

Bases: *SimType*

__init__(*typevar*)

Parameters
label – the type label.

c_repr()

class angr.analyses.typehoon.translator.**TypeTranslator**(*arch=None*)

Bases: object

Translate type variables to SimType equivalence.

__init__(*arch=None*)

struct_name()

tc2simtype(*tc*)

simtype2tc(*simtype*)

Return type
TypeConstant

Parameters
simtype (SimType) –

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*

__init__(*sub_type*, *super_type*)

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.**TypeVariable**(*idx=None*)

Bases: object

Parameters
idx (*int | None*) –

__init__(*idx=None*)

Parameters
idx (*int | None*) –

idx: int

pp_str(*mapping*)

Return type
str

Parameters
mapping (*Dict[TypeVariable, Any]*) –

class angr.analyses.typehoon.typevars.**DerivedTypeVariable**(*type_var*, *label*, *idx=None*)

Bases: *TypeVariable*

Parameters
idx (*int*) –

__init__(*type_var*, *label*, *idx=None*)

type_var

label

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*, *codeloc*, *typevar*)

Parameters

- **var** (SimVariable) –
- **typevar** (TypeVariable) –

get_type_variable(*var*, *codeloc*)

has_type_variable_for(*var*, *codeloc*)

Parameters
var (SimVariable) –

class angr.analyses.typehoon.typevars.**BaseLabel**

Bases: object

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*

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*)

Bases: *BaseLabel*

__init__(*bits*, *offset*)

bits

offset

class angr.analyses.typehoon.typevars.**IsArray**

Bases: *BaseLabel*

class angr.analyses.typehoon.typehoon.**Typehoon**(*constraints*, *ground_truth=None*, *var_mapping=None*, *must_struct=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*) –

__init__(*constraints*, *ground_truth=None*, *var_mapping=None*, *must_struct=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** (Optional[Dict[*SimVariable*, Set[*TypeVariable*]]]) –
- **must_struct** (Optional[Set[*TypeVariable*]]) –

update_variable_types(*func_addr*, *var_to_typevars*)

Parameters
func_addr (*int | str*) –

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

project: Project

kb: KnowledgeBase

All type constants used in type inference. They can be mapped, translated, or rewritten to C-style types.

class angr.analyses.typehoon.typeconsts.**TypeConstant**

Bases: object

SIZE = None

pp_str(*mapping*)

Return type
str

property size: int

class angr.analyses.typehoon.typeconsts.**TopType**

Bases: *TypeConstant*

class angr.analyses.typehoon.typeconsts.**BottomType**

Bases: *TypeConstant*

class angr.analyses.typehoon.typeconsts.**Int**

Bases: *TypeConstant*

class angr.analyses.typehoon.typeconsts.**Int1**

Bases: *Int*

SIZE = 1

class angr.analyses.typehoon.typeconsts.**Int8**

Bases: *Int*

SIZE = 1

class angr.analyses.typehoon.typeconsts.**Int16**

Bases: *Int*

SIZE = 2

class angr.analyses.typehoon.typeconsts.**Int32**

Bases: *Int*

SIZE = 4

class angr.analyses.typehoon.typeconsts.**Int64**

Bases: *Int*

SIZE = 8

class angr.analyses.typehoon.typeconsts.**Int128**

Bases: *Int*

SIZE = 16

class angr.analyses.typehoon.typeconsts.**FloatBase**

Bases: *TypeConstant*

class angr.analyses.typehoon.typeconsts.**Float**

Bases: *FloatBase*

SIZE = 4

class angr.analyses.typehoon.typeconsts.**Double**

Bases: *FloatBase*

SIZE = 8

class angr.analyses.typehoon.typeconsts.**Pointer**(*basetype*)

Bases: *TypeConstant*

__init__(*basetype*)

new(*basetype*)

class angr.analyses.typehoon.typeconsts.**Pointer32**(*basetype*)

Bases: *Pointer*, *Int32*

32-bit pointers.

__init__(*basetype*)

class angr.analyses.typehoon.typeconsts.**Pointer64**(*basetype*)

Bases: *Pointer*, *Int64*

64-bit pointers.

__init__(*basetype*)

class angr.analyses.typehoon.typeconsts.**Array**(*element*, *count=None*)

Bases: *TypeConstant*

__init__(*element*, *count=None*)

class angr.analyses.typehoon.typeconsts.**Struct**(*fields=None*)

Bases: *TypeConstant*

__init__(*fields=None*)

class angr.analyses.typehoon.typeconsts.**TypeVariableReference**(*typevar*)

Bases: *TypeConstant*

__init__(*typevar*)

angr.analyses.typehoon.typeconsts.**int_type**(*bits*)

Return type
Optional[*Int*]

Parameters
bits (*int*) –

angr.analyses.typehoon.typeconsts.**float_type**(*bits*)

Return type
Optional[*FloatBase*]

Parameters
bits (*int*) –

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

project: Project

kb: KnowledgeBase

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*)

project: Project

kb: KnowledgeBase

class angr.analyses.loop_analysis.**VariableTypes**

Bases: object

Iterator = 'Iterator'

HasNext = 'HasNext'

Next = 'Next'

class angr.analyses.loop_analysis.**AnnotatedVariable**(*variable*, *type_*)

Bases: object

__init__(*variable*, *type_*)

variable

type

class angr.analyses.loop_analysis.**Condition**(*op*, *val0*, *val1*)

Bases: object

Equal = '=='

NotEqual = '!='

__init__(*op*, *val0*, *val1*)

classmethod from_opstr(*opstr*)

class angr.analyses.loop_analysis.**SootBlockProcessor**(*state*, *block*, *loop*, *defuse*)

Bases: object

__init__(*state*, *block*, *loop*, *defuse*)

process()

class angr.analyses.loop_analysis.**LoopAnalysisState**(*block*)

Bases: object

__init__(*block*)

copy()

merge(*state*)

add_loop_exit_stmt(*stmt_idx*, *condition=None*)

class angr.analyses.loop_analysis.**LoopAnalysis**(*loop*, *defuse*)

Bases: *ForwardAnalysis*, *Analysis*

Analyze a loop and recover important information about the loop (e.g., invariants, induction variables) in a static manner.

__init__(*loop*, *defuse*)

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

project: Project

kb: KnowledgeBase

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.transmit.transmit'>, <class 'angr.procedures.posix.read.read'>, <class 'angr.procedures.libc.fgetc.fgetc'>, <class 'angr.procedures.libc.atoi.atoi'>, <class 'angr.procedures.cgc.receive.receive'>, <class 'angr.procedures.libc.strlen.strlen'>, <class 'angr.procedures.glibc.__ctype_b_loc.__ctype_b_loc'>, <class 'angr.procedures.libc.strcmp.strcmp'>}

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

project: Project

kb: KnowledgeBase

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

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) –

Return type
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*], *Analysis*

This class represents a control-flow graph with static analysis result.

Perform abstract interpretation analysis starting from the given function address. The output is an invariant at the beginning (or the end) of each basic block.

Steps:

- Generate a CFG first if CFG is not provided.
- Identify all merge points (denote the set of merge points as Pw) in the CFG.
- Cut those loop back edges (can be derived from Pw) so that we gain an acyclic CFG.

- **Identify all variables that are 1) from memory loading 2) from initial values, or 3) phi functions. Denote**
  the set of those variables as S_{var}.
- **Start real AI analysis and try to compute a fix point of each merge point. Perform widening/narrowing only on**
  variables in S_{var}.

__init__(*cfg=None*, *context_sensitivity_level=2*, *start=None*, *function_start=None*, *interfunction_level=0*, *initial_state=None*, *avoid_runs=None*, *remove_options=None*, *timeout=None*, *max_iterations_before_widening=8*, *max_iterations=40*, *widening_interval=3*, *final_state_callback=None*, *status_callback=None*, *record_function_final_states=False*)

Parameters

- **cfg** (Optional[*CFGEmulated*]) – 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** (Optional[int]) – The address of the function to analyze.
- **interfunction_level** (int) – The level of interfunction-ness to be
- **initial_state** (Optional[*SimState*]) – A state to use as the initial one
- **avoid_runs** (Optional[List[int]]) – A list of runs to avoid
- **remove_options** (Optional[Set[str]]) – State options to remove from the initial state. It only works when *initial_state* is None
- **timeout** (*int*) –
- **final_state_callback** (Optional[Callable[[*SimState*, *CallStack*], Any]]) – callback function when countering final state
- **status_callback** (Optional[Callable[[*VFG*], Any]]) – 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
Optional[*VFGNode*]

Parameters
addr (*int*) –

`get_all_nodes`(*addr*)

Return type
Generator[*VFGNode*, None, None]

`irsb_from_node`(*node*)

`copy`()

`project: Project`

`kb: KnowledgeBase`

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.

project: Project

kb: KnowledgeBase

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*)

project: Project

kb: KnowledgeBase

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*)

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__()

project: Project

kb: KnowledgeBase

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__()

project: Project

kb: KnowledgeBase

class angr.analyses.disassembly.DisassemblyPiece

Bases: object

addr = None

ident = nan

render(*formatting=None*)

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: Union[IRStmt, PcodeOp]

irsb: Union[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*)

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*)

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*)

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*)

Bases: *OperandPiece*

__init__(*val*, *render_with_sign*)

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*) –

project: Project

kb: KnowledgeBase

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*)

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*) –

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.

Discliamer: 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()

project: Project

kb: KnowledgeBase

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*)

project: Project

kb: KnowledgeBase

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*)

project: Project

kb: KnowledgeBase

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 Challange. 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()

project: Project

kb: KnowledgeBase

class angr.analyses.callee_cleanup_finder.**CalleeCleanupFinder**(*starts=None*, *hook_all=False*)

Bases: *Analysis*

__init__(*starts=None*, *hook_all=False*)

analyze(*addr*)

project: Project

kb: KnowledgeBase

class angr.analyses.dominance_frontier.**DominanceFrontier**(*func*, *exception_edges=False*)

Bases: *Analysis*

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*, *exception_edges=False*)

project: Project

kb: KnowledgeBase

class angr.analyses.init_finder.**SimEngineInitFinderVEX**(*project*, *replacements*, *overlay*, *pointers_only=False*)

Bases: *SimEngineLightVEXMixin*, *SimEngineLight*

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

project: Project

kb: KnowledgeBase

class angr.analyses.xrefs.**SimEngineXRefsVEX**(*xref_manager*, *project=None*, *replacements=None*)

Bases: *SimEngineLightVEXMixin*, *SimEngineLight*

The VEX engine class for XRefs analysis.

__init__(*xref_manager*, *project=None*, *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
Optional[int]

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

project: Project

kb: KnowledgeBase

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

ref_at (*Set[int] | None*) –

__init__(*func*, *ref_at=None*)

Parameters

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

- **ref_at** (*Set[int] | None*) –
- **args** (*Tuple[BaseProxiNode] | None*) –

__init__(*callee*, *ref_at=None*, *args=None*)

Parameters

- **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*) –

project: Project

kb: KnowledgeBase

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** (Optional[int]) – An address or None, Specifies where to start generation of DDG
- **end_at** (Optional[int]) – 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
- **block_addrs** –

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
Optional[DiGraph]

Parameters

- **g_node** (BaseDepNode) –
- **include_tmp_nodes** (*bool*) –
- **backwards** (*bool*) –

project: Project

kb: KnowledgeBase

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) –

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

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*)

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** (*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*) –

__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*)

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.
- **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*)

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*)

class angr.codenode.**CodeNode**(*addr*, *size*, *graph=None*, *thumb=False*)

Bases: object

Parameters

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

__init__(*addr*, *size*, *graph=None*, *thumb=False*)

Parameters

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

addr: int

size: int

thumb

successors()

Return type

List[*CodeNode*]

predecessors()

is_hook = None

class angr.codenode.**BlockNode**(*addr*, *size*, *bytestr=None*, ***kwargs*)

Bases: *CodeNode*

Parameters

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

is_hook = False

__init__(*addr*, *size*, *bytestr=None*, ***kwargs*)

Parameters
addr (*int*) –

bytestr

class angr.codenode.**SootBlockNode**(*addr*, *size*, *stmts*, ***kwargs*)

Bases: *BlockNode*

Parameters

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

__init__(*addr*, *size*, *stmts*, ***kwargs*)

stmts

class angr.codenode.**HookNode**(*addr*, *size*, *sim_procedure*, ***kwargs*)

Bases: *CodeNode*

Parameters

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

is_hook = True

__init__(*addr*, *size*, *sim_procedure*, ***kwargs*)

Parameters
sim_procedure (*type*) – the the sim_procedure class

sim_procedure

class angr.codenode.**SyscallNode**(*addr*, *size*, *sim_procedure*, ***kwargs*)

Bases: *HookNode*

Parameters

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

is_hook = False

sim_procedure

10.16 SimOS

Manage OS-level configuration.

angr.simos.**register_simos**(*name*, *cls*)

class angr.simos.simos.**SimOS**(*project*, *name=None*)

Bases: object

A class describing OS/arch-level configuration.

Parameters
project (angr.Project) –

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

Parameters
project (Project) –

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*)

syscall_abi(*state*)

Return type
str

syscall_cc(*state*)

Return type
Optional[*SimCCSyscall*]

is_syscall_addr(*addr*)

syscall_from_addr(*addr*, *allow_unsupported=True*)

syscall_from_number(*number*, *allow_unsupported=True*, *abi=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

__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**(*value*)

Bases: Enum

An enumeration.

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).

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

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*, *max_length=1000*)

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 primtive 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.

get_method_native_type(*method*)

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

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.

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.**FS**

alias of *FlirtSignature*

angr.flirt.**load_signatures**(*path*)

Recursively load all FLIRT signatures under a specific path.

Parameters
path (str) – Location of FLIRT signatures.

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 baed 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.**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.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*) – The list. Must be pre-ordered.
- **element** (*object*) – An element to insert into the list.
- **key** (*func*) – A method to get the key for each element in the list.
- **lo** (*int*) – Lower bound of the search.
- **hi** (*int*) – 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

Implements a copy-on-write version of ChainMap that supports auto-collapsing.

__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:]

clean()

class angr.utils.cowdict.**DefaultChainMapCOW**(*default_factory*, **args*, *collapse_threshold=None*)

Bases: *ChainMapCOW*

Implements a copy-on-write version of ChainMap with default values that supports auto-collapsing.

__init__(*default_factory*, **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.

clean()

class angr.utils.dynamic_dictlist.**DynamicDictList**(*max_size=None*, *content=None*)

Bases: Generic[VT]

A list-like container class that internally uses dicts to store values when the number of values is less than the threshold *LIST2DICT_THRESHOLD*. Keys must be ints.

The default thresholds are determined according to experiments described at https://github.com/angr/angr/pull/3471#issuecomment-1236515950.

__init__(*max_size=None*, *content=None*)

Parameters

- **max_size** (*int | None*) –
- **content** (*DynamicDictList | Dict[int, VT] | List[VT] | None*) –

list_content: Optional[List[TypeVar(VT)]]

max_size

dict_content: Optional[Dict[int, TypeVar(VT)]]

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.

Return type
DiGraph

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*)

Invert the given graph and generate the immediate dominator tree on the inverted graph. This is useful for computing post-dominators.

Parameters
graph (DiGraph) – The graph to invert and generate immediate dominator tree for.

Return type
Tuple[DiGraph, Optional[Dict]]

Returns
A tuple of the inverted graph and the immediate dominator tree.

angr.utils.graph.**to_acyclic_graph**(*graph*, *ordered_nodes=None*, *loop_heads=None*)

Convert a given DiGraph into an acyclic graph.

Parameters

- **graph** (DiGraph) – The graph to convert.
- **ordered_nodes** (Optional[List]) – A list of nodes sorted in a topological order.
- **loop_heads** (Optional[List]) – A list of known loop head nodes.

Return type
DiGraph

Returns
The converted acyclic graph.

angr.utils.graph.**dfs_back_edges**(*graph*, *start_node*)

Do a DFS traversal of the graph, and return with the back edges.

Note: This is just a naive recursive implementation, feel free to replace it. I couldn't find anything in networkx to do this functionality. Although the name suggest it, but *dfs_labeled_edges* is doing something different.

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*)

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: object

A container node.

Only used in dominator tree generation. We did this so we can set the index property without modifying the original object.

__init__(*obj*)

index

property obj

class angr.utils.graph.**Dominators**(*graph*, *entry_node*, *successors_func=None*, *reverse=False*)

Bases: object

Describes dominators in a graph.

__init__(*graph*, *entry_node*, *successors_func=None*, *reverse=False*)

dom: DiGraph

class angr.utils.graph.**PostDominators**(*graph*, *entry_node*, *successors_func=None*)

Bases: *Dominators*

Describe post-dominators in a graph.

__init__(*graph*, *entry_node*, *successors_func=None*)

property post_dom: DiGraph

dom: DiGraph

class angr.utils.graph.**SCCPlaceholder**(*scc_id*)

Bases: object

Describes a placeholder for strongly-connected-components in a graph.

__init__(*scc_id*)

scc_id

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 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*)

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** (Optional[List]) – A list of nodes to sort. None if you want to sort all nodes inside the graph.
- **loop_heads** (Optional[List]) – A list of nodes that should be treated loop heads.

Return type
List

Returns
A list of ordered 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[Optional[str], Optional[*SimTypeCppFunction*], Optional[str]]

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**(*demangled_name*, *specialized=True*, *qualified=True*)

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** (Optional[Callable[[str], str]]) – 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** (Optional[bool]) – 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: tuple

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*) –

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 initalizer to the same state as this initalizer, then calls each initalizer

Return type
None

angr.utils.mp.**mp_context**()

10.19 Errors

exception angr.errors.**AngrError**

Bases: Exception

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.**AngrGirlScoutError**

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.**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.**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.**UnsupportedNodeTypeError**
Bases: *AngrError*, NotImplementedError

10.20 Distributed analysis

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) –

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

angr, 157
angr.analyses, 619
angr.analyses.analysis, 619
angr.analyses.backward_slice, 627
angr.analyses.binary_optimizer, 859
angr.analyses.bindiff, 629
angr.analyses.boyscout, 632
angr.analyses.callee_cleanup_finder, 860
angr.analyses.calling_convention, 632
angr.analyses.cdg, 673
angr.analyses.cfg, 637
angr.analyses.cfg.cfb, 637
angr.analyses.cfg.cfg, 638
angr.analyses.cfg.cfg_arch_options, 656
angr.analyses.cfg.cfg_base, 645
angr.analyses.cfg.cfg_emulated, 640
angr.analyses.cfg.cfg_fast, 647
angr.analyses.cfg.cfg_fast_soot, 668
angr.analyses.cfg.cfg_job_base, 656
angr.analyses.cfg.indirect_jump_resolvers, 668
angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got, 657
angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast, 658
angr.analyses.cfg.indirect_jump_resolvers.const_resolver, 666
angr.analyses.cfg.indirect_jump_resolvers.default_resolvers, 662
angr.analyses.cfg.indirect_jump_resolvers.jumptable, 662
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast, 660
angr.analyses.cfg.indirect_jump_resolvers.resolver, 667
angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt, 661
angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat, 659
angr.analyses.cfg.segment_list, 670
angr.analyses.cfg_slice_to_sink, 814
angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink, 814
angr.analyses.cfg_slice_to_sink.graph, 815
angr.analyses.cfg_slice_to_sink.transitions, 816
angr.analyses.class_identifier, 846
angr.analyses.code_tagging, 673
angr.analyses.complete_calling_conventions, 634
angr.analyses.congruency_check, 858
angr.analyses.data_dep, 870
angr.analyses.data_dep.data_dependency_analysis, 865
angr.analyses.data_dep.dep_nodes, 867
angr.analyses.data_dep.sim_act_location, 866
angr.analyses.datagraph_meta, 673
angr.analyses.ddg, 744
angr.analyses.decompiler, 691
angr.analyses.decompiler.ail_simplifier, 691
angr.analyses.decompiler.ailgraph_walker, 692
angr.analyses.decompiler.block_simplifier, 692
angr.analyses.decompiler.callsite_maker, 693
angr.analyses.decompiler.ccall_rewriters, 693
angr.analyses.decompiler.ccall_rewriters.amd64_ccalls, 693
angr.analyses.decompiler.ccall_rewriters.rewriter_base, 693
angr.analyses.decompiler.clinic, 694
angr.analyses.decompiler.condition_processor, 695
angr.analyses.decompiler.decompilation_cache, 697
angr.analyses.decompiler.decompilation_options, 696
angr.analyses.decompiler.decompiler, 697
angr.analyses.decompiler.empty_node_remover, 698
angr.analyses.decompiler.expression_narrower, 699
angr.analyses.decompiler.graph_region, 699
angr.analyses.decompiler.jump_target_collector,

angr.analyses.decompiler.jumptable_entry_condition_rewriter, 700
angr.analyses.decompiler.optimization_passes, 700
angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier, 701
angr.analyses.decompiler.optimization_passes.const_derefs, 705
angr.analyses.decompiler.optimization_passes.div_simplifier, 701
angr.analyses.decompiler.optimization_passes.engine_base, 705
angr.analyses.decompiler.optimization_passes.expr_op_swapper, 701
angr.analyses.decompiler.optimization_passes.inline_small_functions, 709
angr.analyses.decompiler.optimization_passes.lower_switch_simplifier, 705
angr.analyses.decompiler.optimization_passes.multi_simplifier, 706
angr.analyses.decompiler.optimization_passes.multi_statement_expression_simplifier, 708
angr.analyses.decompiler.optimization_passes.optimization_pass, 708
angr.analyses.decompiler.optimization_passes.register_save_area_simplifier, 702
angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier, 710
angr.analyses.decompiler.optimization_passes.stack_canary_simplifier, 711
angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier, 704
angr.analyses.decompiler.peephole_optimizations, 711
angr.analyses.decompiler.peephole_optimizations.base, 711
angr.analyses.decompiler.redundant_label_remover, 711
angr.analyses.decompiler.region_identifier, 722
angr.analyses.decompiler.region_simplifiers, 713
angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer, 714
angr.analyses.decompiler.region_simplifiers.cascading_ifs, 714
angr.analyses.decompiler.region_simplifiers.expression_narrower, 714
angr.analyses.decompiler.region_simplifiers.goto, 714
angr.analyses.decompiler.region_simplifiers.ifelse, 717
angr.analyses.decompiler.region_simplifiers.loop, 718
angr.analyses.decompiler.region_simplifiers.node_address_finder, 718
angr.analyses.decompiler.region_simplifiers.region_simplifier, 718
angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier, 718
angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier, 722
angr.analyses.decompiler.region_walker, 722
angr.analyses.decompiler.sequence_walker, 722
angr.analyses.decompiler.structured_codegen, 723
angr.analyses.decompiler.structured_codegen.base, 723
angr.analyses.decompiler.structured_codegen.c, 724
angr.analyses.decompiler.structured_codegen.dummy, 741
angr.analyses.decompiler.structured_codegen.dwarf_import, 741
angr.analyses.decompiler.structuring, 684
angr.analyses.decompiler.structuring.dream, 684
angr.analyses.decompiler.structuring.phoenix, 690
angr.analyses.decompiler.structuring.recursive_structurer, 684
angr.analyses.decompiler.structuring.structurer_base, 680
angr.analyses.decompiler.structuring.structurer_nodes, 685
angr.analyses.decompiler.utils, 741
angr.analyses.disassembly, 846
angr.analyses.disassembly_utils, 850
angr.analyses.dominance_frontier, 861
angr.analyses.find_objects_static, 845
angr.analyses.flirt, 749
angr.analyses.forward_analysis, 621
angr.analyses.forward_analysis.forward_analysis, 621
angr.analyses.forward_analysis.job_info, 622
angr.analyses.forward_analysis.visitors, 622
angr.analyses.forward_analysis.visitors.call_graph, 622
angr.analyses.forward_analysis.visitors.function_graph, 623
angr.analyses.forward_analysis.visitors.graph, 624
angr.analyses.forward_analysis.visitors.loop,

626
angr.analyses.forward_analysis.visitors.single_node_graph, 627
angr.analyses.identifier.identify, 835
angr.analyses.init_finder, 861
angr.analyses.loop_analysis, 837
angr.analyses.loopfinder, 836
angr.analyses.propagator, 751
angr.analyses.propagator.engine_ail, 753
angr.analyses.propagator.engine_base, 752
angr.analyses.propagator.engine_vex, 753
angr.analyses.propagator.outdated_definition_walker, 754
angr.analyses.propagator.propagator, 755
angr.analyses.propagator.tmpvar_finder, 755
angr.analyses.propagator.top_checker_mixin, 757
angr.analyses.propagator.values, 751
angr.analyses.propagator.vex_vars, 752
angr.analyses.proximity_graph, 863
angr.analyses.reaching_definitions, 757
angr.analyses.reaching_definitions.call_trace, 789
angr.analyses.reaching_definitions.dep_graph, 794
angr.analyses.reaching_definitions.engine_ail, 813
angr.analyses.reaching_definitions.engine_vex, 790
angr.analyses.reaching_definitions.function_handler, 798
angr.analyses.reaching_definitions.heap_allocator, 797
angr.analyses.reaching_definitions.rd_state, 805
angr.analyses.reaching_definitions.reaching_definitions, 791
angr.analyses.reaching_definitions.subject, 813
angr.analyses.reassembler, 850
angr.analyses.soot_class_hierarchy, 636
angr.analyses.stack_pointer_tracker, 816
angr.analyses.static_hooker, 859
angr.analyses.typehoon, 835
angr.analyses.typehoon.lifter, 827
angr.analyses.typehoon.simple_solver, 828
angr.analyses.typehoon.translator, 828
angr.analyses.typehoon.typeconsts, 834
angr.analyses.typehoon.typehoon, 833
angr.analyses.typehoon.typevars, 829
angr.analyses.variable_recovery, 827
angr.analyses.variable_recovery.annotations, 818
angr.analyses.variable_recovery.engine_ail, 826
angr.analyses.variable_recovery.engine_base, 826
angr.analyses.variable_recovery.engine_vex, 826
angr.analyses.variable_recovery.irsb_scanner, 827
angr.analyses.variable_recovery.variable_recovery, 825
angr.analyses.variable_recovery.variable_recovery_base, 819
angr.analyses.variable_recovery.variable_recovery_fast, 823
angr.analyses.veritesting, 838
angr.analyses.vfg, 839
angr.analyses.vsa_ddg, 844
angr.analyses.vtable, 845
angr.analyses.xrefs, 861
angr.angrdb, 674
angr.angrdb.db, 674
angr.angrdb.models, 676
angr.angrdb.serializers, 679
angr.angrdb.serializers.cfg_model, 679
angr.angrdb.serializers.comments, 680
angr.angrdb.serializers.funcs, 680
angr.angrdb.serializers.kb, 681
angr.angrdb.serializers.labels, 681
angr.angrdb.serializers.loader, 681
angr.angrdb.serializers.structured_code, 683
angr.angrdb.serializers.variables, 682
angr.angrdb.serializers.xrefs, 682
angr.annocfg, 871
angr.blade, 870
angr.block, 220
angr.callable, 519
angr.calling_conventions, 483
angr.code_location, 612
angr.codenode, 873
angr.concretization_strategies, 335
angr.concretization_strategies.any, 381
angr.concretization_strategies.controlled_data, 381
angr.concretization_strategies.eval, 379
angr.concretization_strategies.max, 380
angr.concretization_strategies.nonzero, 381
angr.concretization_strategies.nonzero_range, 380
angr.concretization_strategies.norepeats, 379
angr.concretization_strategies.norepeats_range, 381
angr.concretization_strategies.range, 380
angr.concretization_strategies.single, 379
angr.concretization_strategies.solutions, 379

angr.concretization_strategies.unlimited_range, 381
angr.distributed, 900
angr.distributed.server, 900
angr.distributed.worker, 900
angr.engines, 427
angr.engines.concrete, 433
angr.engines.engine, 427
angr.engines.failure, 431
angr.engines.hook, 430
angr.engines.light, 750
angr.engines.light.data, 750
angr.engines.light.engine, 750
angr.engines.pcode, 434
angr.engines.pcode.behavior, 444
angr.engines.pcode.cc, 464
angr.engines.pcode.emulate, 444
angr.engines.pcode.engine, 434
angr.engines.pcode.lifter, 435
angr.engines.procedure, 430
angr.engines.soot, 431
angr.engines.soot.engine, 431
angr.engines.successors, 428
angr.engines.syscall, 431
angr.engines.unicorn, 432
angr.engines.vex, 431
angr.errors, 894
angr.exploration_techniques, 390
angr.exploration_techniques.bucketizer, 426
angr.exploration_techniques.common, 424
angr.exploration_techniques.dfs, 408
angr.exploration_techniques.director, 418
angr.exploration_techniques.driller_core, 416
angr.exploration_techniques.explorer, 408
angr.exploration_techniques.lengthlimiter, 409
angr.exploration_techniques.local_loop_seer, 422
angr.exploration_techniques.loop_seer, 421
angr.exploration_techniques.manual_mergepoint, 410
angr.exploration_techniques.memory_watcher, 426
angr.exploration_techniques.oppologist, 420
angr.exploration_techniques.slicecutor, 417
angr.exploration_techniques.spiller, 410
angr.exploration_techniques.spiller_db, 412
angr.exploration_techniques.stochastic, 423
angr.exploration_techniques.suggestions, 426
angr.exploration_techniques.symbion, 424
angr.exploration_techniques.tech_builder, 424
angr.exploration_techniques.threading, 413
angr.exploration_techniques.timeout, 407
angr.exploration_techniques.tracer, 414
angr.exploration_techniques.unique, 423
angr.exploration_techniques.veritesting, 413
angr.factory, 216
angr.flirt, 883
angr.flirt.build_sig, 883
angr.keyed_region, 613
angr.knowledge_base, 520
angr.knowledge_base.knowledge_base, 520
angr.knowledge_plugins, 521
angr.knowledge_plugins.callsite_prototypes, 523
angr.knowledge_plugins.cfg, 524
angr.knowledge_plugins.cfg.cfg_manager, 544
angr.knowledge_plugins.cfg.cfg_model, 536
angr.knowledge_plugins.cfg.cfg_node, 544
angr.knowledge_plugins.cfg.indirect_jump, 547
angr.knowledge_plugins.cfg.memory_data, 542
angr.knowledge_plugins.comments, 549
angr.knowledge_plugins.data, 549
angr.knowledge_plugins.debug_variables, 568
angr.knowledge_plugins.functions, 550
angr.knowledge_plugins.functions.function, 552
angr.knowledge_plugins.functions.function_manager, 550
angr.knowledge_plugins.functions.function_parser, 559
angr.knowledge_plugins.functions.soot_function, 560
angr.knowledge_plugins.indirect_jumps, 549
angr.knowledge_plugins.key_definitions, 570
angr.knowledge_plugins.key_definitions.atoms, 584
angr.knowledge_plugins.key_definitions.constants, 588
angr.knowledge_plugins.key_definitions.definition, 588
angr.knowledge_plugins.key_definitions.environment, 590
angr.knowledge_plugins.key_definitions.heap_address, 591
angr.knowledge_plugins.key_definitions.key_definition_mana
591
angr.knowledge_plugins.key_definitions.live_definitions, 592
angr.knowledge_plugins.key_definitions.rd_model, 601
angr.knowledge_plugins.key_definitions.tag, 603
angr.knowledge_plugins.key_definitions.undefined, 605
angr.knowledge_plugins.key_definitions.unknown_size, 605
angr.knowledge_plugins.key_definitions.uses,

605
angr.knowledge_plugins.labels, 549
angr.knowledge_plugins.patches, 521
angr.knowledge_plugins.plugin, 523
angr.knowledge_plugins.propagations, 549
angr.knowledge_plugins.structured_code, 570
angr.knowledge_plugins.structured_code.manager, 570
angr.knowledge_plugins.sync, 607
angr.knowledge_plugins.sync.sync_controller, 607
angr.knowledge_plugins.types, 549
angr.knowledge_plugins.variables, 561
angr.knowledge_plugins.variables.variable_access, 561
angr.knowledge_plugins.variables.variable_manager, 561
angr.knowledge_plugins.xrefs, 610
angr.knowledge_plugins.xrefs.xref, 610
angr.knowledge_plugins.xrefs.xref_manager, 611
angr.knowledge_plugins.xrefs.xref_types, 611
angr.misc.plugins, 222
angr.procedures, 473
angr.procedures.definitions, 475
angr.procedures.stubs.format_parser, 473
angr.project, 212
angr.protos, 617
angr.serializable, 616
angr.sim_manager, 382
angr.sim_options, 228
angr.sim_procedure, 469
angr.sim_state, 224
angr.sim_state_options, 228
angr.sim_type, 507
angr.sim_variable, 502
angr.simos, 875
angr.simos.cgc, 878
angr.simos.javavm, 880
angr.simos.linux, 877
angr.simos.simos, 875
angr.simos.userland, 878
angr.simos.windows, 879
angr.slicer, 871
angr.state_hierarchy, 389
angr.state_plugins, 231
angr.state_plugins.callstack, 263
angr.state_plugins.cgc, 271
angr.state_plugins.concrete, 292
angr.state_plugins.debug_variables, 307
angr.state_plugins.filesystem, 248
angr.state_plugins.gdb, 270
angr.state_plugins.globals, 278
angr.state_plugins.heap, 297
angr.state_plugins.heap.heap_base, 297
angr.state_plugins.heap.heap_brk, 298
angr.state_plugins.heap.heap_freelist, 300
angr.state_plugins.heap.heap_libc, 301
angr.state_plugins.heap.heap_ptmalloc, 302
angr.state_plugins.heap.utils, 306
angr.state_plugins.history, 267
angr.state_plugins.inspect, 233
angr.state_plugins.javavm_classloader, 294
angr.state_plugins.jni_references, 296
angr.state_plugins.libc, 236
angr.state_plugins.light_registers, 266
angr.state_plugins.log, 262
angr.state_plugins.loop_data, 291
angr.state_plugins.plugin, 231
angr.state_plugins.posix, 240
angr.state_plugins.preconstrainer, 282
angr.state_plugins.scratch, 280
angr.state_plugins.sim_action, 466
angr.state_plugins.sim_action_object, 468
angr.state_plugins.sim_event, 468
angr.state_plugins.solver, 254
angr.state_plugins.symbolizer, 307
angr.state_plugins.trace_additions, 273
angr.state_plugins.uc_manager, 279
angr.state_plugins.unicorn_engine, 284
angr.state_plugins.view, 309
angr.storage, 309
angr.storage.file, 314
angr.storage.memory_mixins, 336
angr.storage.memory_mixins.actions_mixin, 342
angr.storage.memory_mixins.address_concretization_mixin, 344
angr.storage.memory_mixins.bvv_conversion_mixin, 340
angr.storage.memory_mixins.clouseau_mixin, 346
angr.storage.memory_mixins.conditional_store_mixin, 346
angr.storage.memory_mixins.convenient_mappings_mixin, 348
angr.storage.memory_mixins.default_filler_mixin, 340
angr.storage.memory_mixins.dirty_addrs_mixin, 344
angr.storage.memory_mixins.hex_dumper_mixin, 341
angr.storage.memory_mixins.javavm_memory, 376
angr.storage.memory_mixins.javavm_memory.javavm_memory_mix 376
angr.storage.memory_mixins.keyvalue_memory, 375
angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory 375

angr.storage.memory_mixins.label_merger_mixin, 346
angr.storage.memory_mixins.multi_value_merger_mixin, 352
angr.storage.memory_mixins.name_resolution_mixin, 339
angr.storage.memory_mixins.paged_memory, 352
angr.storage.memory_mixins.paged_memory.page_backer_mixins, 357
angr.storage.memory_mixins.paged_memory.paged_memory_mixin, 352
angr.storage.memory_mixins.paged_memory.pages, 359
angr.storage.memory_mixins.paged_memory.pages.cooperation, 361
angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin, 360
angr.storage.memory_mixins.paged_memory.pages.ispo_mixin, 361
angr.storage.memory_mixins.paged_memory.pages.list_pages, 362
angr.storage.memory_mixins.paged_memory.pages.multi_values, 350
angr.storage.memory_mixins.paged_memory.pages.pages_mixin, 348
angr.storage.memory_mixins.paged_memory.pages.permissions_mixin, 360
angr.storage.memory_mixins.paged_memory.pages.refcount_mixin, 359
angr.storage.memory_mixins.paged_memory.pages.stack_pages, 363
angr.storage.memory_mixins.paged_memory.privileged_mixin, 358
angr.storage.memory_mixins.paged_memory.stack_allocation_mixin, 358
angr.storage.memory_mixins.regioned_memory, 365
angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor, 371
angr.storage.memory_mixins.regioned_memory.abstract_merger_mixin, 373
angr.storage.memory_mixins.regioned_memory.region_category_mixin, 370
angr.storage.memory_mixins.regioned_memory.region_data, 368
angr.storage.memory_mixins.regioned_memory.region_meta_mixin, 371
angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin, 373
angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin, 365
angr.storage.memory_mixins.regioned_memory.static_find_mixin, 370
angr.storage.memory_mixins.simple_interface_mixin, 342
angr.storage.memory_mixins.simplification_mixin, 347
angr.storage.memory_mixins.size_resolution_mixin, 343
angr.storage.memory_mixins.slotted_memory, 374
angr.storage.memory_mixins.smart_find_mixin, 339
angr.storage.memory_mixins.symbolic_merger_mixin, 342
angr.storage.memory_mixins.top_merger_mixin, 351
angr.storage.memory_mixins.underconstrained_mixin, 341
angr.storage.memory_mixins.unwrapper_mixin, 347
angr.storage.memory_object, 334
angr.storage.pcap, 335
angr.utils, 884
angr.utils.algo, 885
angr.utils.constants, 885
angr.utils.cowdict, 885
angr.utils.dynamic_dictlist, 885
angr.utils.enums_conv, 886
angr.utils.env, 886
angr.utils.formatting, 892
angr.utils.graph, 886
angr.utils.lazy_import, 890
angr.utils.library, 891
angr.utils.loader, 890
angr.utils.mp, 893
angr.utils.timing, 892
angr.utils.xmalloc, 617

INDEX

Symbols

__init__() (*angr.BP method*), 161
__init__() (*angr.Blade method*), 167
__init__() (*angr.Block method*), 170
__init__() (*angr.ExplorationTechnique method*), 178
__init__() (*angr.KnowledgeBase method*), 211
__init__() (*angr.PTChunk method*), 209
__init__() (*angr.PointerWrapper method*), 184
__init__() (*angr.Project method*), 164
__init__() (*angr.Server method*), 210
__init__() (*angr.SimCC method*), 185
__init__() (*angr.SimCC.ArgSession method*), 186
__init__() (*angr.SimFile method*), 190
__init__() (*angr.SimFileBase method*), 189
__init__() (*angr.SimFileDescriptor method*), 198
__init__() (*angr.SimFileDescriptorDuplex method*), 200
__init__() (*angr.SimFileStream method*), 194
__init__() (*angr.SimHeapBrk method*), 204
__init__() (*angr.SimHeapPTMalloc method*), 206
__init__() (*angr.SimHostFilesystem method*), 204
__init__() (*angr.SimOS method*), 168
__init__() (*angr.SimPackets method*), 192
__init__() (*angr.SimPacketsStream method*), 196
__init__() (*angr.SimProcedure method*), 159
__init__() (*angr.SimState method*), 182
__init__() (*angr.SimStatePlugin method*), 161
__init__() (*angr.SimulationManager method*), 172
__init__() (*angr.StateHierarchy method*), 180
__init__() (*angr.analyses.analysis.AnalysesHub method*), 619
__init__() (*angr.analyses.analysis.AnalysisFactory method*), 620
__init__() (*angr.analyses.analysis.AnalysisLogEntry method*), 619
__init__() (*angr.analyses.analysis.KnownAnalysesPlugin method*), 620
__init__() (*angr.analyses.backward_slice.BackwardSlice method*), 628
__init__() (*angr.analyses.binary_optimizer.BinaryOptimizer method*), 860
__init__() (*angr.analyses.binary_optimizer.ConstantPropagation method*), 859
__init__() (*angr.analyses.binary_optimizer.DeadAssignment method*), 860
__init__() (*angr.analyses.binary_optimizer.RedundantStackVariable method*), 859
__init__() (*angr.analyses.binary_optimizer.RegisterReallocation method*), 860
__init__() (*angr.analyses.bindiff.BinDiff method*), 631
__init__() (*angr.analyses.bindiff.ConstantChange method*), 629
__init__() (*angr.analyses.bindiff.Difference method*), 629
__init__() (*angr.analyses.bindiff.FunctionDiff method*), 630
__init__() (*angr.analyses.bindiff.NormalizedBlock method*), 630
__init__() (*angr.analyses.bindiff.NormalizedFunction method*), 630
__init__() (*angr.analyses.boyscout.BoyScout method*), 632
__init__() (*angr.analyses.callee_cleanup_finder.CalleeCleanupFinder method*), 860
__init__() (*angr.analyses.calling_convention.CallSiteFact method*), 633
__init__() (*angr.analyses.calling_convention.CallingConventionAnalysis method*), 633
__init__() (*angr.analyses.cdg.CDG method*), 673
__init__() (*angr.analyses.cfg.cfb.CFBlanket method*), 637
__init__() (*angr.analyses.cfg.cfb.CFBlanketView method*), 637
__init__() (*angr.analyses.cfg.cfb.MemoryRegion method*), 637
__init__() (*angr.analyses.cfg.cfb.Unknown method*), 637
__init__() (*angr.analyses.cfg.cfg.CFG method*), 638
__init__() (*angr.analyses.cfg.cfg_arch_options.CFGArchOptions method*), 656
__init__() (*angr.analyses.cfg.cfg_base.CFGBase method*), 645
__init__() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 641

__init__() (angr.analyses.cfg.cfg_emulated.CFGJob method), 640
__init__() (angr.analyses.cfg.cfg_emulated.PendingJob method), 640
__init__() (angr.analyses.cfg.cfg_fast.CFGFast method), 653
__init__() (angr.analyses.cfg.cfg_fast.CFGJob method), 652
__init__() (angr.analyses.cfg.cfg_fast.DecodingAssumption method), 648
__init__() (angr.analyses.cfg.cfg_fast.FunctionCallEdge method), 650
__init__() (angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge method), 650
__init__() (angr.analyses.cfg.cfg_fast.FunctionReturn method), 648
__init__() (angr.analyses.cfg.cfg_fast.FunctionReturnEdge method), 651
__init__() (angr.analyses.cfg.cfg_fast.FunctionTransitionEdge method), 650
__init__() (angr.analyses.cfg.cfg_fast.PendingJobs method), 649
__init__() (angr.analyses.cfg.cfg_fast_soot.CFGFastSoot method), 668
__init__() (angr.analyses.cfg.cfg_job_base.BlockID method), 656
__init__() (angr.analyses.cfg.cfg_job_base.CFGJobBase method), 657
__init__() (angr.analyses.cfg.cfg_job_base.FunctionKey method), 657
__init__() (angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got.AMD64ElfGotResolver method), 657
__init__() (angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast.ArmElfFastResolver method), 658
__init__() (angr.analyses.cfg.indirect_jump_resolvers.const_resolver.ConstantResolver method), 666
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.BSSHook method), 665
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.ConstantValueManager method), 663
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessor method), 664
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessorState method), 663
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver method), 665
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTargetBaseAddr method), 662
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.LoadHook method), 664
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.MIPSGPHook method), 665
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffsetAnnotation method), 663
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.Register method), 665
__init__() (angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.Mips method), 660
__init__() (angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.Over method), 660
__init__() (angr.analyses.cfg.indirect_jump_resolvers.resolver.IndirectJu method), 667
__init__() (angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt.X8 method), 661
__init__() (angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat.X86Pe method), 659
__init__() (angr.analyses.cfg.segment_list.Segment method), 670
__init__() (angr.analyses.cfg.segment_list.SegmentList method), 671
__init__() (angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceT method), 814
__init__() (angr.analyses.class_identifier.ClassIdentifier method), 846
__init__() (angr.analyses.code_tagging.CodeTagging method), 674
__init__() (angr.analyses.complete_calling_conventions.CompleteCallin method), 635
__init__() (angr.analyses.congruency_check.CongruencyCheck method), 858
__init__() (angr.analyses.data_dep.data_dependency_analysis.DataDep method), 866
__init__() (angr.analyses.data_dep.data_dependency_analysis.NodalAnn method), 865
__init__() (angr.analyses.data_dep.dep_nodes.BaseDepNode method), 868
__init__() (angr.analyses.data_dep.dep_nodes.ConstantDepNode method), 868
__init__() (angr.analyses.data_dep.dep_nodes.MemDepNode method), 868
__init__() (angr.analyses.data_dep.dep_nodes.RegDepNode method), 870
__init__() (angr.analyses.data_dep.dep_nodes.TmpDepNode method), 869
__init__() (angr.analyses.data_dep.dep_nodes.VarDepNode method), 869
__init__() (angr.analyses.data_dep.sim_act_location.ParsedInstruction method), 867
__init__() (angr.analyses.data_dep.sim_act_location.SimActLocation method), 867
__init__() (angr.analyses.datagraph_meta.DataGraphMeta method), 673
__init__() (angr.analyses.ddg.AST method), 744
__init__() (angr.analyses.ddg.DDG method), 747
__init__() (angr.analyses.ddg.DDGJob method), 744
__init__() (angr.analyses.ddg.DDGView method), 747
__init__() (angr.analyses.ddg.DDGViewInstruction method), 746

__init__() (angr.analyses.ddg.DDGViewItem method), 746
__init__() (angr.analyses.ddg.LiveDefinitions method), 745
__init__() (angr.analyses.ddg.ProgramVariable method), 744
__init__() (angr.analyses.decompiler.ail_simplifier.AILBlockTempCollector method), 691
__init__() (angr.analyses.decompiler.ail_simplifier.AILSimplifier method), 692
__init__() (angr.analyses.decompiler.ailgraph_walker.AILGraphWalker method), 692
__init__() (angr.analyses.decompiler.block_simplifier.BlockSimplifier method), 693
__init__() (angr.analyses.decompiler.block_simplifier.HasCallExprWalker method), 692
__init__() (angr.analyses.decompiler.callsite_maker.CallSiteMaker method), 693
__init__() (angr.analyses.decompiler.ccall_rewriters.rewriter_base.CCallRewriterBase method), 693
__init__() (angr.analyses.decompiler.clinic.Clinic method), 694
__init__() (angr.analyses.decompiler.condition_processor.ConditionProcessor method), 695
__init__() (angr.analyses.decompiler.decompilation_cache.DecompilationCache method), 697
__init__() (angr.analyses.decompiler.decompilation_options.DecompilationOption method), 697
__init__() (angr.analyses.decompiler.decompiler.Decompiler method), 698
__init__() (angr.analyses.decompiler.empty_node_remover.EmptyNodeRemover method), 699
__init__() (angr.analyses.decompiler.expression_narrower.ExpressionNarrowingWalker method), 699
__init__() (angr.analyses.decompiler.graph_region.GraphRegion method), 700
__init__() (angr.analyses.decompiler.jump_target_collector.JumpTargetCollector method), 700
__init__() (angr.analyses.decompiler.jumptable_entry_condition_rewriter.JumpTableEntryConditionRewriter method), 701
__init__() (angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePointerSaveSimplifier method), 705
__init__() (angr.analyses.decompiler.optimization_passes.constant_derefs.ConstantDereferenceSimplifier method), 701
__init__() (angr.analyses.decompiler.optimization_passes.constant_derefs.ConstantDereferenceWalker method), 701
__init__() (angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifier method), 705
__init__() (angr.analyses.decompiler.optimization_passes.engine_base.SimEngineLightAILBase method), 709
__init__() (angr.analyses.decompiler.optimization_passes.engine_base.SimplifierAILState method), 709
__init__() (angr.analyses.decompiler.optimization_passes.expr_op_swap.OpSwapper method), 710
__init__() (angr.analyses.decompiler.optimization_passes.expr_op_swap.OpSwapper method), 709
__init__() (angr.analyses.decompiler.optimization_passes.expr_op_swap.OpSwapper method), 710
__init__() (angr.analyses.decompiler.optimization_passes.expr_op_swap.OpSwapper method), 709
__init__() (angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter method), 705
__init__() (angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter method), 706
__init__() (angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter method), 706
__init__() (angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier method), 706
__init__() (angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier method), 707
__init__() (angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier method), 707
__init__() (angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier method), 709
__init__() (angr.analyses.decompiler.optimization_passes.multi_simplifier.MultiSimplifier method), 708
__init__() (angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass method), 703
__init__() (angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage method), 704
__init__() (angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage method), 704
__init__() (angr.analyses.decompiler.optimization_passes.register_save_simplifier.RegisterSaveSimplifier method), 711
__init__() (angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier method), 711
__init__() (angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier method), 705
__init__() (angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcSimplifier method), 711
__init__() (angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase method), 713
__init__() (angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationMultiStatementBase method), 712
__init__() (angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase method), 712
__init__() (angr.analyses.decompiler.redundant_label_remover.RedundantLabelRemover method), 722
__init__() (angr.analyses.decompiler.region_identifier.RegionIdentifier method), 714
__init__() (angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer.CascadingConditionTransformer method), 714
__init__() (angr.analyses.decompiler.region_simplifiers.cascading_ifs.CascadingIfsRemover method), 714
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ConstantExpressionFolding method), 715

__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ConditionalBreakLocation method), 715
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter method), 716
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionFolder method), 717
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionLocation method), 715
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionReplacer method), 716
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionUseFinder method), 716
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.MultiStatementExpressionAssignmentFinder method), 716
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.StatementLocation method), 715
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.StatementSelector method), 717
__init__() (angr.analyses.decompiler.region_simplifiers.goto_simplifier.GotoSimplifier method), 717
__init__() (angr.analyses.decompiler.region_simplifiers.if_.IfSimplifier method), 718
__init__() (angr.analyses.decompiler.region_simplifiers.ifelse_.IfElseFlattener method), 718
__init__() (angr.analyses.decompiler.region_simplifiers.loop_simplifier.LoopSimplifier method), 718
__init__() (angr.analyses.decompiler.region_simplifiers.node_address_finder.NodeAddressFinder method), 718
__init__() (angr.analyses.decompiler.region_simplifiers.region_simplifier.RegionSimplifier method), 718
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.ConditionalRegion method), 719
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.FindFirstNodeInSet method), 721
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchCaseRegion method), 719
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterFinder method), 720
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterReplacer method), 720
__init__() (angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier.SwitchExpressionSimplifier method), 722
__init__() (angr.analyses.decompiler.region_walker.RegionWalker method), 722
__init__() (angr.analyses.decompiler.sequence_walker.SequenceWalker method), 722
__init__() (angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method), 724
__init__() (angr.analyses.decompiler.structured_codegen.base.InstructionMappings method), 723
__init__() (angr.analyses.decompiler.structured_codegen.base.InstructionMappingElement method), 723
__init__() (angr.analyses.decompiler.structured_codegen.base.PositionMapping method), 723
__init__() (angr.analyses.decompiler.structured_codegen.base.PositionMapping method), 723
__init__() (angr.analyses.decompiler.structured_codegen.c.CAILBlock method), 727
__init__() (angr.analyses.decompiler.structured_codegen.c.CArrayTypeL method), 737
__init__() (angr.analyses.decompiler.structured_codegen.c.CAssignment method), 729
__init__() (angr.analyses.decompiler.structured_codegen.c.CBinaryOp method), 734
__init__() (angr.analyses.decompiler.structured_codegen.c.CBreak method), 728
__init__() (angr.analyses.decompiler.structured_codegen.c.CClosingObj method), 737
__init__() (angr.analyses.decompiler.structured_codegen.c.CConstant method), 735
__init__() (angr.analyses.decompiler.structured_codegen.c.CConstruct method), 725
__init__() (angr.analyses.decompiler.structured_codegen.c.CContinue method), 729
__init__() (angr.analyses.decompiler.structured_codegen.c.CDirtyExpre method), 737
__init__() (angr.analyses.decompiler.structured_codegen.c.CDoWhileLo method), 727
__init__() (angr.analyses.decompiler.structured_codegen.c.CExpression method), 726
__init__() (angr.analyses.decompiler.structured_codegen.c.CFakeVariab method), 732
__init__() (angr.analyses.decompiler.structured_codegen.c.CForLoop method), 727
__init__() (angr.analyses.decompiler.structured_codegen.c.CFunction method), 725
__init__() (angr.analyses.decompiler.structured_codegen.c.CFunctionCa method), 730
__init__() (angr.analyses.decompiler.structured_codegen.c.CGoto method), 730
__init__() (angr.analyses.decompiler.structured_codegen.c.CITE method), 736
__init__() (angr.analyses.decompiler.structured_codegen.c.CIfBreak method), 728
__init__() (angr.analyses.decompiler.structured_codegen.c.CIfElse method), 728
__init__() (angr.analyses.decompiler.structured_codegen.c.CIndexedVar method), 733
__init__() (angr.analyses.decompiler.structured_codegen.c.CLabel method), 731
__init__() (angr.analyses.decompiler.structured_codegen.c.CMultiStaten method), 737
__init__() (angr.analyses.decompiler.structured_codegen.c.CRegister method), 736
__init__() (angr.analyses.decompiler.structured_codegen.c.CReturn method), 730
__init__() (angr.analyses.decompiler.structured_codegen.c.CStatements method), 726

__init__() (angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method), 731
__init__() (angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method), 737
__init__() (angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWriter method), 738
__init__() (angr.analyses.decompiler.structured_codegen.c.CSwitchCase method), 729
__init__() (angr.analyses.decompiler.structured_codegen.c.CTypeCast method), 735
__init__() (angr.analyses.decompiler.structured_codegen.c.CUnaryOp method), 734
__init__() (angr.analyses.decompiler.structured_codegen.c.CUnsupportedStatement method), 731
__init__() (angr.analyses.decompiler.structured_codegen.c.CVariable method), 732
__init__() (angr.analyses.decompiler.structured_codegen.c.CVariableField method), 733
__init__() (angr.analyses.decompiler.structured_codegen.c.CWhileLoop method), 727
__init__() (angr.analyses.decompiler.structured_codegen.dummy.DummyStructuredCodeGenerator method), 741
__init__() (angr.analyses.decompiler.structured_codegen.dwarf_import.DwarfImportStructuredCodeGenerator method), 741
__init__() (angr.analyses.decompiler.structured_codegen.dwarf_import.ImportedDwarfVariable method), 741
__init__() (angr.analyses.decompiler.structuring.dream.DreamStructurer method), 685
__init__() (angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer method), 691
__init__() (angr.analyses.decompiler.structuring.recursive_structurer.RecursiveStructurer method), 684
__init__() (angr.analyses.decompiler.structuring.structurer_base.StructurerBase method), 689
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.BreakNode method), 687
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.ConditionNode method), 686
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.ContinueNode method), 686
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.ConditionalBreakNode method), 686
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.CrossJumpReplacementNode method), 688
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.CrossJumpNode method), 688
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.IncompleteSwitchCaseHeadStatement method), 689
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.IncompleteSwitchCaseNode method), 689
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.LoopNode method), 687
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.MultiNode method), 685
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.SequenceNode method), 685
__init__() (angr.analyses.decompiler.structuring.structurer_nodes.SwitchCaseNode method), 688
__init__() (angr.analyses.disassembly.BlockStart method), 847
__init__() (angr.analyses.disassembly.Comment method), 849
__init__() (angr.analyses.disassembly.Disassembly method), 850
__init__() (angr.analyses.disassembly.FuncComment method), 849
__init__() (angr.analyses.disassembly.FunctionStart method), 846
__init__() (angr.analyses.disassembly.Hook method), 847
__init__() (angr.analyses.disassembly.IROp method), 847
__init__() (angr.analyses.disassembly.Instruction method), 847
__init__() (angr.analyses.disassembly.JumpTargetLabel method), 847
__init__() (angr.analyses.disassembly.MemoryOperand method), 849
__init__() (angr.analyses.disassembly.Opcode method), 848
__init__() (angr.analyses.disassembly.Operand method), 848
__init__() (angr.analyses.disassembly.Register method), 849
__init__() (angr.analyses.disassembly.SootExpression method), 848
__init__() (angr.analyses.disassembly.SootExpressionInvoke method), 848
__init__() (angr.analyses.disassembly.SootExpressionStaticFieldRef method), 848
__init__() (angr.analyses.disassembly.SootExpressionTarget method), 848
__init__() (angr.analyses.disassembly.SootStatement method), 848
__init__() (angr.analyses.disassembly.Value method), 849
__init__() (angr.analyses.dominance_frontier.DominanceFrontier method), 861
__init__() (angr.analyses.find_objects_static.NewFunctionHandler method), 845
__init__() (angr.analyses.find_objects_static.PossibleObject method), 845
__init__() (angr.analyses.find_objects_static.StaticObjectFinder method), 846
__init__() (angr.analyses.flirt.FlirtAnalysis method), 750
__init__() (angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method), 621

__init__() (angr.analyses.forward_analysis.job_info.JobInfo method), 622
__init__() (angr.analyses.forward_analysis.visitors.call_graph.CallGraphVisitor method), 622
__init__() (angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor method), 623
__init__() (angr.analyses.forward_analysis.visitors.graph.GraphVisitor method), 624
__init__() (angr.analyses.forward_analysis.visitors.loop.LoopVisitor method), 626
__init__() (angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor method), 627
__init__() (angr.analyses.identifier.identify.FuncInfo method), 835
__init__() (angr.analyses.identifier.identify.Identifier method), 836
__init__() (angr.analyses.init_finder.InitializationFinder method), 861
__init__() (angr.analyses.init_finder.SimEngineInitFinderVEX method), 861
__init__() (angr.analyses.loop_analysis.AnnotatedVariable method), 837
__init__() (angr.analyses.loop_analysis.Condition method), 837
__init__() (angr.analyses.loop_analysis.LoopAnalysis method), 837
__init__() (angr.analyses.loop_analysis.LoopAnalysisState method), 837
__init__() (angr.analyses.loop_analysis.SootBlockProcessor method), 837
__init__() (angr.analyses.loopfinder.Loop method), 836
__init__() (angr.analyses.loopfinder.LoopFinder method), 836
__init__() (angr.analyses.propagator.engine_base.SimEnginePropagatorBase method), 752
__init__() (angr.analyses.propagator.outdated_definition_walker.OutdatedDefinitionWalker method), 755
__init__() (angr.analyses.propagator.propagator.PropagatorAnalysis method), 756
__init__() (angr.analyses.propagator.tmpvar_finder.TmpvarFinder method), 755
__init__() (angr.analyses.propagator.values.Top method), 751
__init__() (angr.analyses.propagator.vex_vars.VEXMemVar method), 752
__init__() (angr.analyses.propagator.vex_vars.VEXReg method), 752
__init__() (angr.analyses.propagator.vex_vars.VEXTmp method), 752
__init__() (angr.analyses.proximity_graph.BaseProxiNode method), 863
__init__() (angr.analyses.proximity_graph.CallProxiNode method), 864
__init__() (angr.analyses.proximity_graph.FunctionProxiNode method), 863
__init__() (angr.analyses.proximity_graph.IntegerProxiNode method), 864
__init__() (angr.analyses.proximity_graph.ProximityGraphAnalysis method), 865
__init__() (angr.analyses.proximity_graph.StringProxiNode method), 864
__init__() (angr.analyses.proximity_graph.UnknownProxiNode method), 864
__init__() (angr.analyses.proximity_graph.VariableProxiNode method), 863
__init__() (angr.analyses.reaching_definitions.Atom method), 766
__init__() (angr.analyses.reaching_definitions.ConstantSrc method), 769
__init__() (angr.analyses.reaching_definitions.Definition method), 770
__init__() (angr.analyses.reaching_definitions.FunctionCallData method), 788
__init__() (angr.analyses.reaching_definitions.FunctionHandler method), 783
__init__() (angr.analyses.reaching_definitions.GuardUse method), 769
__init__() (angr.analyses.reaching_definitions.LiveDefinitions method), 757
__init__() (angr.analyses.reaching_definitions.MemoryLocation method), 768
__init__() (angr.analyses.reaching_definitions.ReachingDefinitionsAnaly method), 771
__init__() (angr.analyses.reaching_definitions.ReachingDefinitionsMode method), 774
__init__() (angr.analyses.reaching_definitions.ReachingDefinitionsState method), 777
__init__() (angr.analyses.reaching_definitions.Register method), 768
__init__() (angr.analyses.reaching_definitions.Tmp method), 769
__init__() (angr.analyses.reaching_definitions.call_trace.CallSite method), 789
__init__() (angr.analyses.reaching_definitions.call_trace.CallTrace method), 790
__init__() (angr.analyses.reaching_definitions.dep_graph.DepGraph method), 795
__init__() (angr.analyses.reaching_definitions.dep_graph.FunctionCallR method), 794
__init__() (angr.analyses.reaching_definitions.engine_ail.SimEngineRDA method), 814
__init__() (angr.analyses.reaching_definitions.engine_vex.SimEngineRD method), 790
__init__() (angr.analyses.reaching_definitions.function_handler.Function method), 802
__init__() (angr.analyses.reaching_definitions.function_handler.Function method), 803

__init__() (angr.analyses.reaching_definitions.function_handler.FunctionHandler method), 799
__init__() (angr.analyses.reaching_definitions.function_handler.FunctionHandlerRDV method), 803
__init__() (angr.analyses.reaching_definitions.heap_allocator_handler.HeapAllocatorHandler method), 797
__init__() (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionState method), 806
__init__() (angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method), 791
__init__() (angr.analyses.reaching_definitions.subject.Subject method), 813
__init__() (angr.analyses.reassembler.BasicBlock method), 853
__init__() (angr.analyses.reassembler.Data method), 855
__init__() (angr.analyses.reassembler.DataLabel method), 851
__init__() (angr.analyses.reassembler.FunctionLabel method), 851
__init__() (angr.analyses.reassembler.Instruction method), 852
__init__() (angr.analyses.reassembler.Label method), 851
__init__() (angr.analyses.reassembler.NotypeLabel method), 851
__init__() (angr.analyses.reassembler.ObjectLabel method), 851
__init__() (angr.analyses.reassembler.Operand method), 852
__init__() (angr.analyses.reassembler.Procedure method), 853
__init__() (angr.analyses.reassembler.ProcedureChunk method), 854
__init__() (angr.analyses.reassembler.Reassembler method), 855
__init__() (angr.analyses.reassembler.Relocation method), 855
__init__() (angr.analyses.reassembler.SymbolManager method), 851
__init__() (angr.analyses.soot_class_hierarchy.NoConcreteDispatchError method), 636
__init__() (angr.analyses.soot_class_hierarchy.SootClassHierarchy method), 636
__init__() (angr.analyses.soot_class_hierarchy.SootClassHierarchy.InvalidClassInfoError method), 636
__init__() (angr.analyses.stack_pointer_tracker.Constant method), 817
__init__() (angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerState method), 817
__init__() (angr.analyses.stack_pointer_tracker.OffsetVal method), 817
__init__() (angr.analyses.stack_pointer_tracker.Register method), 817
__init__() (angr.analyses.stack_pointer_tracker.StackPointerTracker method), 818
__init__() (angr.analyses.stack_pointer_tracker.StackPointerTrackerState method), 817
__init__() (angr.analyses.static_hooker.StaticHooker method), 859
__init__() (angr.analyses.typehoon.lifter.TypeLifter method), 828
__init__() (angr.analyses.typehoon.simple_solver.RecursiveType method), 828
__init__() (angr.analyses.typehoon.simple_solver.SimpleSolver method), 828
__init__() (angr.analyses.typehoon.translator.SimTypeTempRef method), 828
__init__() (angr.analyses.typehoon.translator.TypeTranslator method), 828
__init__() (angr.analyses.typehoon.typeconsts.Array method), 835
__init__() (angr.analyses.typehoon.typeconsts.Pointer method), 835
__init__() (angr.analyses.typehoon.typeconsts.Pointer32 method), 835
__init__() (angr.analyses.typehoon.typeconsts.Pointer64 method), 835
__init__() (angr.analyses.typehoon.typeconsts.Struct method), 835
__init__() (angr.analyses.typehoon.typeconsts.TypeVariableReference method), 835
__init__() (angr.analyses.typehoon.typehoon.Typehoon method), 833
__init__() (angr.analyses.typehoon.typevars.Add method), 830
__init__() (angr.analyses.typehoon.typevars.AddN method), 832
__init__() (angr.analyses.typehoon.typevars.ConvertTo method), 832
__init__() (angr.analyses.typehoon.typevars.DerivedTypeVariable method), 831
__init__() (angr.analyses.typehoon.typevars.Equivalence method), 829
__init__() (angr.analyses.typehoon.typevars.Existence method), 829
__init__() (angr.analyses.typehoon.typevars.FuncIn method), 832
__init__() (angr.analyses.typehoon.typevars.FuncOut method), 832
__init__() (angr.analyses.typehoon.typevars.HasField method), 832
__init__() (angr.analyses.typehoon.typevars.ReinterpretAs method), 832
__init__() (angr.analyses.typehoon.typevars.Sub method), 830
__init__() (angr.analyses.typehoon.typevars.SubN method), 832

__init__() (angr.analyses.typehoon.typevars.Subtype method), 830
__init__() (angr.analyses.typehoon.typevars.TypeVariable method), 831
__init__() (angr.analyses.typehoon.typevars.TypeVariables method), 831
__init__() (angr.analyses.variable_recovery.annotations.StackLocationAnnotation method), 819
__init__() (angr.analyses.variable_recovery.annotations.VariableSourceAnnotation method), 819
__init__() (angr.analyses.variable_recovery.engine_ail.SimEngineVRAIL method), 826
__init__() (angr.analyses.variable_recovery.engine_base.RichR method), 827
__init__() (angr.analyses.variable_recovery.engine_base.SimEngineVRBase method), 827
__init__() (angr.analyses.variable_recovery.engine_vex.SimEngineVRVEX method), 826
__init__() (angr.analyses.variable_recovery.irsb_scanner.VEXIRSBScanner method), 827
__init__() (angr.analyses.variable_recovery.variable_recovery.VariableRecovery method), 826
__init__() (angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryState method), 825
__init__() (angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase method), 819
__init__() (angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method), 820
__init__() (angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFast method), 821
__init__() (angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryStateFast method), 824
__init__() (angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryStateMixin method), 823
__init__() (angr.analyses.veritesting.CallTracingFilter method), 838
__init__() (angr.analyses.veritesting.Veritesting method), 839
__init__() (angr.analyses.vfg.AnalysisTask method), 840
__init__() (angr.analyses.vfg.CallAnalysis method), 841
__init__() (angr.analyses.vfg.FunctionAnalysis method), 841
__init__() (angr.analyses.vfg.PendingJob method), 840
__init__() (angr.analyses.vfg.VFG method), 843
__init__() (angr.analyses.vfg.VFGJob method), 839
__init__() (angr.analyses.vfg.VFGNode method), 842
__init__() (angr.analyses.vsa_ddg.DefUseChain method), 844
__init__() (angr.analyses.vsa_ddg.VSA_DDG method), 844
__init__() (angr.analyses.vtable.Vtable method), 845
__init__() (angr.analyses.vtable.VtableFinder method), 845
__init__() (angr.analyses.xrefs.SimEngineXRefsVEX method), 862
__init__() (angr.analyses.xrefs.XRefsAnalysis method), 862
__init__() (angr.angrdb.db.AngrDB method), 674
__init__() (angr.angrdb.models.DbCFGModel method), 677
__init__() (angr.angrdb.models.DbComment method), 679
__init__() (angr.angrdb.models.DbFunction method), 677
__init__() (angr.angrdb.models.DbInformation method), 676
__init__() (angr.angrdb.models.DbKnowledgeBase method), 677
__init__() (angr.angrdb.models.DbLabel method), 679
__init__() (angr.angrdb.models.DbObject method), 676
__init__() (angr.angrdb.models.DbStructuredCode method), 678
__init__() (angr.angrdb.models.DbVariableCollection method), 678
__init__() (angr.angrdb.models.DbXRefs method), 679
__init__() (angr.annocfg.AnnotatedCFG method), 871
__init__() (angr.block.Block method), 221
__init__() (angr.block.DisassemblerBlock method), 220
__init__() (angr.block.SootBlock method), 222
__init__() (angr.calling_conventions.AllocHelper method), 483
__init__() (angr.calling_conventions.ArgSession method), 487
__init__() (angr.calling_conventions.MicrosoftAMD64ArgSession method), 492
__init__() (angr.calling_conventions.PointerWrapper method), 483
__init__() (angr.calling_conventions.SerializableCounter method), 484
__init__() (angr.calling_conventions.SerializableListIterator method), 484
__init__() (angr.calling_conventions.SimArrayArg method), 486
__init__() (angr.calling_conventions.SimCC method), 487
__init__() (angr.calling_conventions.SimCC.ArgSession method), 489
__init__() (angr.calling_conventions.SimCCUsercall method), 491
__init__() (angr.calling_conventions.SimComboArg

method), 486
__init__() (*angr.calling_conventions.SimFunctionArgument method*), 484
__init__() (*angr.calling_conventions.SimLyingRegArg method*), 491
__init__() (*angr.calling_conventions.SimReferenceArgument method*), 487
__init__() (*angr.calling_conventions.SimRegArg method*), 485
__init__() (*angr.calling_conventions.SimStackArg method*), 485
__init__() (*angr.calling_conventions.SimStructArg method*), 486
__init__() (*angr.calling_conventions.UsercallArgSession method*), 487
__init__() (*angr.code_location.CodeLocation method*), 612
__init__() (*angr.code_location.ExternalCodeLocation method*), 613
__init__() (*angr.codenode.BlockNode method*), 874
__init__() (*angr.codenode.CodeNode method*), 873
__init__() (*angr.codenode.HookNode method*), 874
__init__() (*angr.codenode.SootBlockNode method*), 874
__init__() (*angr.concretization_strategies.SimConcretizationStrategy method*), 335
__init__() (*angr.concretization_strategies.controlled_data.SimConcretizationStrategyControlledData method*), 381
__init__() (*angr.concretization_strategies.eval.SimConcretizationStrategyEval method*), 379
__init__() (*angr.concretization_strategies.max.SimConcretizationStrategyMax method*), 380
__init__() (*angr.concretization_strategies.nonzero_range.SimConcretizationStrategyNonzeroRange method*), 380
__init__() (*angr.concretization_strategies.norepeats.SimConcretizationStrategyNorepeats method*), 379
__init__() (*angr.concretization_strategies.norepeats_range.SimConcretizationStrategyNorepeatsRange method*), 381
__init__() (*angr.concretization_strategies.range.SimConcretizationStrategyRange method*), 380
__init__() (*angr.concretization_strategies.solutions.SimConcretizationStrategySolutions method*), 379
__init__() (*angr.concretization_strategies.unlimited_range.SimConcretizationStrategyUnlimitedRange method*), 382
__init__() (*angr.distributed.server.Server method*), 900
__init__() (*angr.distributed.worker.BadStatesDropper method*), 900
__init__() (*angr.distributed.worker.ExplorationStatusNotifier method*), 901
__init__() (*angr.distributed.worker.Worker method*), 901
__init__() (*angr.engines.concrete.SimEngineConcrete method*), 433
__init__() (*angr.engines.engine.SimEngineBase method*), 427
__init__() (*angr.engines.engine.SuccessorsMixin method*), 428
__init__() (*angr.engines.engine.TLSProperty method*), 428
__init__() (*angr.engines.light.data.ArithmeticExpression method*), 750
__init__() (*angr.engines.light.data.RegisterOffset method*), 750
__init__() (*angr.engines.light.data.SpOffset method*), 750
__init__() (*angr.engines.light.engine.SimEngineLight method*), 751
__init__() (*angr.engines.light.engine.SimEngineLightMixin method*), 751
__init__() (*angr.engines.pcode.behavior.BehaviorFactory method*), 464
__init__() (*angr.engines.pcode.behavior.OpBehavior method*), 445
__init__() (*angr.engines.pcode.behavior.OpBehaviorBoolAnd method*), 458
__init__() (*angr.engines.pcode.behavior.OpBehaviorBoolNegate method*), 457
__init__() (*angr.engines.pcode.behavior.OpBehaviorBoolOr method*), 459
__init__() (*angr.engines.pcode.behavior.OpBehaviorBoolXor method*), 458
__init__() (*angr.engines.pcode.behavior.OpBehaviorCopy method*), 446
__init__() (*angr.engines.pcode.behavior.OpBehaviorEqual method*), 446
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatAbs method*), 461
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatAdd method*), 460
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatCeil method*), 462
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatDiv method*), 460
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatEqual method*), 459
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatFloat2Float method*), 462
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatFloor method*), 462
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatInt2Float method*), 462
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatLess method*), 460
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatLessEqual method*), 460
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatMult method*), 461

__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatNan method*), 460
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatNeg method*), 461
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatNotEqual method*), 459
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatRound method*), 463
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatSqrt method*), 461
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatSub method*), 461
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatTrunc method*), 462
__init__() (*angr.engines.pcode.behavior.OpBehaviorInt2Comp method*), 452
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntAdd method*), 450
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntAnd method*), 453
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntCarry method*), 451
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntDiv method*), 456
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntLeft method*), 454
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntLess method*), 448
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual method*), 448
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntMult method*), 455
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntNegate method*), 452
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntOr method*), 454
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntRem method*), 457
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntRight method*), 454
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSborrow method*), 451
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntScarry method*), 451
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSdiv method*), 456
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSext method*), 449
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSless method*), 447
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual method*), 448
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSrem method*), 457
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSright method*), 455
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSub method*), 450
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntXor method*), 453
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntZext method*), 449
__init__() (*angr.engines.pcode.behavior.OpBehaviorNotEqual method*), 447
__init__() (*angr.engines.pcode.behavior.OpBehaviorPiece method*), 463
__init__() (*angr.engines.pcode.behavior.OpBehaviorPopcount method*), 463
__init__() (*angr.engines.pcode.behavior.OpBehaviorSubpiece method*), 463
__init__() (*angr.engines.pcode.emulate.PcodeEmulatorMixin method*), 444
__init__() (*angr.engines.pcode.engine.HeavyPcodeMixin method*), 434
__init__() (*angr.engines.pcode.lifter.ExitStatement method*), 435
__init__() (*angr.engines.pcode.lifter.IRSB method*), 436
__init__() (*angr.engines.pcode.lifter.Lifter method*), 439
__init__() (*angr.engines.pcode.lifter.PcodeBasicBlockLifter method*), 441
__init__() (*angr.engines.pcode.lifter.PcodeDisassemblerInsn method*), 435
__init__() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 443
__init__() (*angr.engines.successors.SimSuccessors method*), 429
__init__() (*angr.engines.unicorn.SimEngineUnicorn method*), 432
__init__() (*angr.errors.SimMemoryMissingError method*), 896
__init__() (*angr.errors.SimReliftException method*), 898
__init__() (*angr.errors.SimSegfaultException method*), 899
__init__() (*angr.errors.SimUninitializedAccessError method*), 898
__init__() (*angr.exploration_techniques.Bucketizer method*), 405
__init__() (*angr.exploration_techniques.CallFunctionGoal method*), 401
__init__() (*angr.exploration_techniques.DFS method*), 398
__init__() (*angr.exploration_techniques.Director method*), 400
__init__() (*angr.exploration_techniques.DrillerCore method*), 393

__init__() (angr.exploration_techniques.ExecuteAddressGoal method), 400
__init__() (angr.exploration_techniques.ExplorationTechnique method), 390
__init__() (angr.exploration_techniques.Explorer method), 396
__init__() (angr.exploration_techniques.LengthLimiter method), 398
__init__() (angr.exploration_techniques.LocalLoopSeer method), 406
__init__() (angr.exploration_techniques.LoopSeer method), 393
__init__() (angr.exploration_techniques.ManualMergepoint method), 402
__init__() (angr.exploration_techniques.MemoryWatcher method), 405
__init__() (angr.exploration_techniques.Oppologist method), 399
__init__() (angr.exploration_techniques.Slicecutor method), 392
__init__() (angr.exploration_techniques.Spiller method), 401
__init__() (angr.exploration_techniques.StochasticSearch method), 403
__init__() (angr.exploration_techniques.Suggestions method), 407
__init__() (angr.exploration_techniques.Symbion method), 404
__init__() (angr.exploration_techniques.TechniqueBuilder method), 403
__init__() (angr.exploration_techniques.Threading method), 397
__init__() (angr.exploration_techniques.Timeout method), 407
__init__() (angr.exploration_techniques.Tracer method), 395
__init__() (angr.exploration_techniques.UniqueSearch method), 403
__init__() (angr.exploration_techniques.Veritesting method), 399
__init__() (angr.exploration_techniques.bucketizer.Bucketizer method), 426
__init__() (angr.exploration_techniques.dfs.DFS method), 408
__init__() (angr.exploration_techniques.director.BaseGoal method), 418
__init__() (angr.exploration_techniques.director.CallFunctionGoal method), 419
__init__() (angr.exploration_techniques.director.Director method), 420
__init__() (angr.exploration_techniques.director.ExecuteAddressGoal method), 418
__init__() (angr.exploration_techniques.driller_core.DrillerCore method), 416
__init__() (angr.exploration_techniques.explorer.Explorer method), 409
__init__() (angr.exploration_techniques.lengthlimiter.LengthLimiter method), 409
__init__() (angr.exploration_techniques.local_loop_seer.LocalLoopSeer method), 422
__init__() (angr.exploration_techniques.loop_seer.LoopSeer method), 421
__init__() (angr.exploration_techniques.manual_mergepoint.ManualMer method), 410
__init__() (angr.exploration_techniques.memory_watcher.MemoryWatch method), 426
__init__() (angr.exploration_techniques.oppologist.Oppologist method), 420
__init__() (angr.exploration_techniques.slicecutor.Slicecutor method), 417
__init__() (angr.exploration_techniques.spiller.PickledStatesDb method), 411
__init__() (angr.exploration_techniques.spiller.PickledStatesList method), 410
__init__() (angr.exploration_techniques.spiller.Spiller method), 412
__init__() (angr.exploration_techniques.spiller_db.PickledState method), 412
__init__() (angr.exploration_techniques.stochastic.StochasticSearch method), 423
__init__() (angr.exploration_techniques.suggestions.Suggestions method), 427
__init__() (angr.exploration_techniques.symbion.Symbion method), 425
__init__() (angr.exploration_techniques.tech_builder.TechniqueBuilder method), 424
__init__() (angr.exploration_techniques.threading.Threading method), 413
__init__() (angr.exploration_techniques.timeout.Timeout method), 407
__init__() (angr.exploration_techniques.tracer.RepHook method), 414
__init__() (angr.exploration_techniques.tracer.Tracer method), 415
__init__() (angr.exploration_techniques.tracer.TracerDesyncError method), 414
__init__() (angr.exploration_techniques.unique.UniqueSearch method), 423
__init__() (angr.exploration_techniques.veritesting.Veritesting method), 413
__init__() (angr.factory.AngrObjectFactory method), 216
__init__() (angr.flirt.FlirtSignature method), 883
__init__() (angr.keyed_region.KeyedRegion method), 614
__init__() (angr.keyed_region.RegionObject method), 613
__init__() (angr.keyed_region.StoredObject method),

613
__init__() (angr.knowledge_base.knowledge_base.KnowledgeBase method), 520
__init__() (angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method), 523
__init__() (angr.knowledge_plugins.cfg.CFGENode method), 527
__init__() (angr.knowledge_plugins.cfg.CFGManager method), 536
__init__() (angr.knowledge_plugins.cfg.CFGModel method), 529
__init__() (angr.knowledge_plugins.cfg.CFGNode method), 525
__init__() (angr.knowledge_plugins.cfg.IndirectJump method), 528
__init__() (angr.knowledge_plugins.cfg.MemoryData method), 524
__init__() (angr.knowledge_plugins.cfg.cfg_manager.CFGManager method), 544
__init__() (angr.knowledge_plugins.cfg.cfg_model.CFGModel method), 536
__init__() (angr.knowledge_plugins.cfg.cfg_node.CFGENode method), 547
__init__() (angr.knowledge_plugins.cfg.cfg_node.CFGNode method), 545
__init__() (angr.knowledge_plugins.cfg.cfg_node.CFGNodeCreation method), 544
__init__() (angr.knowledge_plugins.cfg.indirect_jump.IndirectJump method), 548
__init__() (angr.knowledge_plugins.cfg.memory_data.MemoryData method), 543
__init__() (angr.knowledge_plugins.comments.Comments method), 549
__init__() (angr.knowledge_plugins.data.Data method), 549
__init__() (angr.knowledge_plugins.debug_variables.DebugVariable method), 568
__init__() (angr.knowledge_plugins.debug_variables.DebugVariableContainer method), 568
__init__() (angr.knowledge_plugins.debug_variables.DebugVariableManager method), 569
__init__() (angr.knowledge_plugins.functions.function.Function method), 552
__init__() (angr.knowledge_plugins.functions.function_manager.FunctionDict method), 550
__init__() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 551
__init__() (angr.knowledge_plugins.functions.soot_function.SootFunction method), 560
__init__() (angr.knowledge_plugins.indirect_jumps.IndirectJumps method), 549
__init__() (angr.knowledge_plugins.key_definitions.Definition method), 583
__init__() (angr.knowledge_plugins.key_definitions.KeyDefinitionManager method), 572
__init__() (angr.knowledge_plugins.key_definitions.LiveDefinitions method), 573
__init__() (angr.knowledge_plugins.key_definitions.ReachingDefinitions method), 570
__init__() (angr.knowledge_plugins.key_definitions.Uses method), 581
__init__() (angr.knowledge_plugins.key_definitions.atoms.Atom method), 584
__init__() (angr.knowledge_plugins.key_definitions.atoms.ConstantSrc method), 586
__init__() (angr.knowledge_plugins.key_definitions.atoms.GuardUse method), 586
__init__() (angr.knowledge_plugins.key_definitions.atoms.MemoryLocat method), 587
__init__() (angr.knowledge_plugins.key_definitions.atoms.Register method), 587
__init__() (angr.knowledge_plugins.key_definitions.atoms.Tmp method), 586
__init__() (angr.knowledge_plugins.key_definitions.definition.Definition method), 590
__init__() (angr.knowledge_plugins.key_definitions.definition.Definitionl method), 589
__init__() (angr.knowledge_plugins.key_definitions.environment.Environ method), 591
__init__() (angr.knowledge_plugins.key_definitions.heap_address.HeapA method), 591
__init__() (angr.knowledge_plugins.key_definitions.key_definition_mana method), 592
__init__() (angr.knowledge_plugins.key_definitions.key_definition_mana method), 592
__init__() (angr.knowledge_plugins.key_definitions.live_definitions.Defi method), 593
__init__() (angr.knowledge_plugins.key_definitions.live_definitions.LiveI method), 594
__init__() (angr.knowledge_plugins.key_definitions.rd_model.ReachingD method), 602
__init__() (angr.knowledge_plugins.key_definitions.tag.FunctionTag method), 604
__init__() (angr.knowledge_plugins.key_definitions.tag.Tag method), 604
__init__() (angr.knowledge_plugins.key_definitions.uses.Uses method), 605
__init__() (angr.knowledge_plugins.labels.Labels method), 549
__init__() (angr.knowledge_plugins.patches.Patch method), 521
__init__() (angr.knowledge_plugins.patches.PatchManager method), 521
__init__() (angr.knowledge_plugins.structured_code.manager.Structured method), 570
__init__() (angr.knowledge_plugins.sync.sync_controller.SyncController method), 608
__init__() (angr.knowledge_plugins.types.TypesStore

method), 549
__init__() (angr.knowledge_plugins.variables.variable_access.VariableAccess method), 561
__init__() (angr.knowledge_plugins.variables.variable_manager.LiveVariables method), 562
__init__() (angr.knowledge_plugins.variables.variable_manager.VariableManager method), 567
__init__() (angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method), 562
__init__() (angr.knowledge_plugins.xrefs.xref.XRef method), 610
__init__() (angr.knowledge_plugins.xrefs.xref_manager.XRefManager method), 611
__init__() (angr.misc.plugins.PluginHub method), 222
__init__() (angr.misc.plugins.PluginPreset method), 223
__init__() (angr.procedures.definitions.SimLibrary method), 476
__init__() (angr.procedures.definitions.SimSyscallLibrary method), 480
__init__() (angr.procedures.stubs.format_parser.FormatSpecifier method), 474
__init__() (angr.procedures.stubs.format_parser.FormatString method), 473
__init__() (angr.project.Project method), 213
__init__() (angr.sim_manager.ErrorRecord method), 389
__init__() (angr.sim_manager.SimulationManager method), 383
__init__() (angr.sim_procedure.SimProcedure method), 471
__init__() (angr.sim_state.SimState method), 225
__init__() (angr.sim_state_options.SimStateOptions method), 230
__init__() (angr.sim_state_options.StateOption method), 228
__init__() (angr.sim_type.NamedTypeMixin method), 508
__init__() (angr.sim_type.SimCppClass method), 517
__init__() (angr.sim_type.SimCppClassValue method), 518
__init__() (angr.sim_type.SimStruct method), 516
__init__() (angr.sim_type.SimStructValue method), 516
__init__() (angr.sim_type.SimType method), 507
__init__() (angr.sim_type.SimTypeArray method), 512
__init__() (angr.sim_type.SimTypeBottom method), 509
__init__() (angr.sim_type.SimTypeChar method), 510
__init__() (angr.sim_type.SimTypeCppFunction method), 514
__init__() (angr.sim_type.SimTypeDouble method), 515
__init__() (angr.sim_type.SimTypeFd method), 511
__init__() (angr.sim_type.SimTypeFloat method), 515
__init__() (angr.sim_type.SimTypeFunction method), 513
__init__() (angr.sim_type.SimTypeInt method), 510
__init__() (angr.sim_type.SimTypeLength method), 515
__init__() (angr.sim_type.SimTypeNum method), 509
__init__() (angr.sim_type.SimTypeNumOffset method), 518
__init__() (angr.sim_type.SimTypePointer method), 511
__init__() (angr.sim_type.SimTypeReference method), 511
__init__() (angr.sim_type.SimTypeReg method), 509
__init__() (angr.sim_type.SimTypeString method), 512
__init__() (angr.sim_type.SimTypeTop method), 509
__init__() (angr.sim_type.SimTypeWString method), 513
__init__() (angr.sim_type.SimTypeWideChar method), 511
__init__() (angr.sim_type.SimUnion method), 516
__init__() (angr.sim_type.SimUnionValue method), 517
__init__() (angr.sim_type.TypeRef method), 508
__init__() (angr.sim_variable.SimConstantVariable method), 503
__init__() (angr.sim_variable.SimMemoryVariable method), 505
__init__() (angr.sim_variable.SimRegisterVariable method), 504
__init__() (angr.sim_variable.SimStackVariable method), 506
__init__() (angr.sim_variable.SimTemporaryVariable method), 504
__init__() (angr.sim_variable.SimVariable method), 503
__init__() (angr.sim_variable.SimVariableSet method), 507
__init__() (angr.simos.cgc.SimCGC method), 878
__init__() (angr.simos.javavm.SimJavaVM method), 880
__init__() (angr.simos.linux.SimLinux method), 877
__init__() (angr.simos.simos.GlobalDescriptorTable method), 877
__init__() (angr.simos.simos.SimOS method), 875
__init__() (angr.simos.userland.SimUserland method), 878
__init__() (angr.simos.windows.SimWindows method), 879
__init__() (angr.slicer.SimLightState method), 871
__init__() (angr.slicer.SimSlicer method), 871
__init__() (angr.state_hierarchy.StateHierarchy method), 389
__init__() (angr.state_plugins.callstack.CallStack

method), 263
__init__() (angr.state_plugins.callstack.CallStackAction method), 266
__init__() (angr.state_plugins.cgc.SimStateCGC method), 272
__init__() (angr.state_plugins.concrete.Concrete method), 292
__init__() (angr.state_plugins.debug_variables.SimDebugVariable method), 307
__init__() (angr.state_plugins.filesystem.SimConcreteFilesystem method), 252
__init__() (angr.state_plugins.filesystem.SimFilesystem method), 249
__init__() (angr.state_plugins.filesystem.SimHostFilesystem method), 254
__init__() (angr.state_plugins.gdb.GDB method), 270
__init__() (angr.state_plugins.globals.SimStateGlobals method), 278
__init__() (angr.state_plugins.heap.heap_base.SimHeapBase method), 298
__init__() (angr.state_plugins.heap.heap_brk.SimHeapBrk method), 298
__init__() (angr.state_plugins.heap.heap_freelist.Chunk method), 300
__init__() (angr.state_plugins.heap.heap_ptmalloc.PTChunk method), 302
__init__() (angr.state_plugins.heap.heap_ptmalloc.PTChunkIterator method), 304
__init__() (angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method), 304
__init__() (angr.state_plugins.history.LambdaAttrIter method), 270
__init__() (angr.state_plugins.history.LambdaIterIter method), 270
__init__() (angr.state_plugins.history.SimStateHistory method), 267
__init__() (angr.state_plugins.history.TreeIter method), 270
__init__() (angr.state_plugins.inspect.BP method), 233
__init__() (angr.state_plugins.inspect.SimInspector method), 234
__init__() (angr.state_plugins.javavm_classloader.SimJavaVmClassloader method), 294
__init__() (angr.state_plugins.jni_references.SimStateJNIReferences method), 296
__init__() (angr.state_plugins.libc.SimStateLibc method), 238
__init__() (angr.state_plugins.light_registers.SimLightRegisters method), 266
__init__() (angr.state_plugins.log.SimStateLog method), 262
__init__() (angr.state_plugins.loop_data.SimStateLoopData method), 291
__init__() (angr.state_plugins.plugin.SimStatePlugin method), 231
__init__() (angr.state_plugins.posix.SimSystemPosix method), 245
__init__() (angr.state_plugins.preconstrainer.SimStatePreconstrainer method), 282
__init__() (angr.state_plugins.scratch.SimStateScratch method), 280
__init__() (angr.state_plugins.sim_action.SimAction method), 466
__init__() (angr.state_plugins.sim_action.SimActionConstraint method), 467
__init__() (angr.state_plugins.sim_action.SimActionData method), 467
__init__() (angr.state_plugins.sim_action.SimActionExit method), 467
__init__() (angr.state_plugins.sim_action.SimActionOperation method), 467
__init__() (angr.state_plugins.sim_action_object.SimActionObject method), 468
__init__() (angr.state_plugins.sim_event.SimEvent method), 468
__init__() (angr.state_plugins.solver.SimSolver method), 254
__init__() (angr.state_plugins.symbolizer.SimSymbolizer method), 307
__init__() (angr.state_plugins.trace_additions.ChallRespInfo method), 274
__init__() (angr.state_plugins.trace_additions.FormatInfoDontConstrain method), 274
__init__() (angr.state_plugins.trace_additions.FormatInfoIntToStr method), 274
__init__() (angr.state_plugins.trace_additions.FormatInfoStrToInt method), 274
__init__() (angr.state_plugins.trace_additions.ZenPlugin method), 276
__init__() (angr.state_plugins.uc_manager.SimUCManager method), 279
__init__() (angr.state_plugins.unicorn_engine.AggressiveConcretization method), 288
__init__() (angr.state_plugins.unicorn_engine.Unicorn method), 289
__init__() (angr.state_plugins.unicorn_engine.Uniwrapper method), 288
__init__() (angr.state_plugins.view.SimMemView method), 311
__init__() (angr.state_plugins.view.StructMode method), 314
__init__() (angr.storage.file.SimFile method), 317
__init__() (angr.storage.file.SimFileBase method), 316
__init__() (angr.storage.file.SimFileDescriptor method), 327
__init__() (angr.storage.file.SimFileDescriptorDuplex method), 329

__init__() (angr.storage.file.SimFileStream method), 319
__init__() (angr.storage.file.SimPackets method), 321
__init__() (angr.storage.file.SimPacketsSlots method), 332
__init__() (angr.storage.file.SimPacketsStream method), 323
__init__() (angr.storage.memory_mixins.MemoryMixin method), 336
__init__() (angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method), 344
__init__() (angr.storage.memory_mixins.convenient_mappings_mixin.ConvenientMappingsMixin method), 348
__init__() (angr.storage.memory_mixins.default_filler_mixin.DefaultFillerMixin method), 340
__init__() (angr.storage.memory_mixins.default_filler_mixin.SpecialFillerMixin method), 340
__init__() (angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method), 376
__init__() (angr.storage.memory_mixins.keyvalue_memory_mixin.KeyValueMemoryMixin method), 375
__init__() (angr.storage.memory_mixins.keyvalue_memory_mixin.TypedVariable method), 375
__init__() (angr.storage.memory_mixins.label_merger_mixin.LabelMergerMixin method), 347
__init__() (angr.storage.memory_mixins.multi_value_merger_mixin.MultiValueMergerMixin method), 352
__init__() (angr.storage.memory_mixins.paged_memory.page_backer_mixins.DictBackerMixin method), 357
__init__() (angr.storage.memory_mixins.paged_memory.page_backer_mixins.MemoryBackerMixin method), 358
__init__() (angr.storage.memory_mixins.paged_memory.page_backer_mixins.SnapshotMemoryview method), 357
__init__() (angr.storage.memory_mixins.paged_memory.paged_memory_mixin.MVListPagesMixin method), 356
__init__() (angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method), 353
__init__() (angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.HistoryTrackingMixin method), 360
__init__() (angr.storage.memory_mixins.paged_memory.pages.list_page.ListPage method), 362
__init__() (angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method), 350
__init__() (angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method), 348
__init__() (angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.PermissionsMixin method), 360
__init__() (angr.storage.memory_mixins.paged_memory.pages.refcount_mixin.RefcountMixin method), 359
__init__() (angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method), 363
__init__() (angr.storage.memory_mixins.paged_memory.stack_allocation_mixin.StackAllocationMixin method), 358
__init__() (angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor.AbstractAddressDescriptor method), 371
__init__() (angr.storage.memory_mixins.regioned_memory.region_data.RegionDescriptor method), 368
__init__() (angr.storage.memory_mixins.regioned_memory.region_data.RegionObject method), 369
__init__() (angr.storage.memory_mixins.regioned_memory.region_data.StackFrameMemory method), 369
__init__() (angr.storage.memory_mixins.regioned_memory.region_meta.RegionMeta method), 371
__init__() (angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin.RegionedAddressConcretizationMixin method), 373
__init__() (angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method), 366
__init__() (angr.storage.memory_mixins.size_resolution_mixin.SizeConcretizationMixin method), 343
__init__() (angr.storage.memory_mixins.slotted_memory.SlottedMemory method), 374
__init__() (angr.storage.memory_mixins.top_merger_mixin.TopMergerMixin method), 352
__init__() (angr.storage.memory_mixins.unconstrained_mixin.UnconstrainedMixin method), 341
__init__() (angr.storage.memory_object.SimLabeledMemoryObject method), 334
__init__() (angr.storage.memory_object.SimMemoryObject method), 334
__init__() (angr.utils.cowdict.ChainMapCOW method), 335
__init__() (angr.utils.cowdict.DefaultChainMapCOW method), 885
__init__() (angr.utils.dynamic_dictlist.DynamicDictList method), 885
__init__() (angr.utils.graph.ContainerNode method), 888
__init__() (angr.utils.graph.Dominators method), 888
__init__() (angr.utils.graph.PostDominators method), 888
__init__() (angr.utils.graph.TemporaryNode method), 888
__init__() (angr.vaults.VaultInitializer method), 893
__init__() (angr.vaults.Vault method), 617
__init__() (angr.vaults.VaultDict method), 618
__init__() (angr.vaults.VaultDir method), 618
__init__() (angr.vaults.VaultPermissionsMixin method), 618
__init__() (angr.vaults.VaultPickler method), 617
__init__() (angr.vaults.VaultSelf method), 618
__init__() (angr.vaults.VaultUnpickler method), 617

A

abort() (angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method), 622

absolutize() (*angr.storage.memory_mixins.regioned_memory.regioned_memory.RegionMap method*), 370
absorb() (*angr.sim_manager.SimulationManager method*), 386
absorb() (*angr.SimulationManager method*), 175
AbstractAddressDescriptor (*class in angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor*), 371
AbstractMemory (*class in angr.storage.memory_mixins*), 338
AbstractMergerMixin (*class in angr.storage.memory_mixins.regioned_memory.abstract_merger_mixin*), 373
access_type (*angr.knowledge_plugins.variables.variable_access.VariableAccess attribute*), 561
accessed_data_references (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 545
accessed_data_references (*angr.knowledge_plugins.cfg.CFGNode property*), 526
AccessingZeroPageError, 288
acquire_shared() (*angr.storage.memory_mixins.paged_memory.paged_mixin.PagedMemory method*), 360
acquire_unique() (*angr.storage.memory_mixins.paged_memory.paged_mixin.PagedMemory method*), 361
acquire_unique() (*angr.storage.memory_mixins.paged_memory.paged_mixin.PagedMemory method*), 360
action() (*angr.state_plugins.inspect.SimInspector method*), 234
actions (*angr.state_plugins.history.SimStateHistory property*), 269
actions (*angr.state_plugins.log.SimStateLog property*), 262
actions_of_type() (*angr.state_plugins.log.SimStateLog method*), 262
ActionsMixinHigh (*class in angr.storage.memory_mixins.actions_mixin*), 342
ActionsMixinLow (*class in angr.storage.memory_mixins.actions_mixin*), 342
activate() (*angr.misc.plugins.PluginPreset method*), 223
active (*angr.sim_manager.SimulationManager attribute*), 383
active (*angr.SimulationManager attribute*), 172
active_workers (*angr.distributed.server.Server property*), 900
active_workers (*angr.Server property*), 210
Add (*angr.engines.light.data.ArithmeticExpression attribute*), 750
Add (*class in angr.analyses.typehoon.typevars*), 830
add() (*angr.exploration_techniques.spiller.PickledStatesBase method*), 410
add() (*angr.exploration_techniques.spiller.PickledStatesDb method*), 411
add() (*angr.exploration_techniques.spiller.PickledStatesList method*), 411
add() (*angr.procedures.definitions.SimLibrary method*), 477
add() (*angr.sim_state_options.SimStateOptions method*), 230
add() (*angr.sim_variable.SimVariableSet method*), 507
add() (*angr.state_plugins.solver.SimSolver method*), 259
add_action() (*angr.state_plugins.history.SimStateHistory method*), 269
add_action() (*angr.state_plugins.log.SimStateLog method*), 262
add_alias() (*angr.procedures.definitions.SimLibrary method*), 477
add_all_from_dict() (*angr.procedures.definitions.SimLibrary method*), 477
add_block_to_whitelist() (*angr.annocfg.AnnotatedCFG method*), 872
add_breakpoint() (*angr.state_plugins.inspect.SimInspector method*), 234
add_constraints() (*angr.sim_state.SimStateMixin method*), 226
add_constraints() (*angr.SimStateMixin method*), 182
add_data_seg() (*angr.analyses.cfg.cfg_fast.DecodingAssumption method*), 648
add_def() (*angr.analyses.ddg.LiveDefinitions method*), 745
add_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 774
add_def() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDe method*), 602
add_def() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsM method*), 570
add_default_plugin() (*angr.misc.plugins.PluginPreset method*), 223
add_defs() (*angr.analyses.ddg.LiveDefinitions method*), 745
add_dependencies_for_concrete_pointers_of() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 796
add_edge() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 795
add_edge_to_buffer() (*in module angr.utils.formatting*), 892
add_event() (*angr.state_plugins.history.SimStateHistory method*), 269
add_event() (*angr.state_plugins.log.SimStateLog method*), 262
add_exit_to_whitelist()

(*angr.annocfg.AnnotatedCFG method*), 872
add_final_job() (*angr.analyses.vfg.CallAnalysis method*), 841
add_function() (*angr.analyses.cfg.cfb.CFBlanket method*), 638
add_function_edge() (*angr.analyses.cfg.cfg_fast.CFGJob method*), 652
add_goal() (*angr.exploration_techniques.Director method*), 400
add_goal() (*angr.exploration_techniques.director.Director method*), 420
add_heap_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 764
add_heap_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 811
add_heap_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 781
add_heap_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 600
add_heap_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 579
add_heap_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 764
add_heap_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 600
add_heap_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 579
add_heap_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 811
add_heap_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 782
add_history() (*angr.state_hierarchy.StateHierarchy method*), 389
add_history() (*angr.StateHierarchy method*), 180
add_job() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 649
add_job() (*angr.analyses.forward_analysis.job_info.JobInfo method*), 622
add_jumpout_site() (*angr.knowledge_plugins.functions.function.Function method*), 556
add_label() (*angr.analyses.reassembler.Reassembler method*), 857
add_loop() (*angr.annocfg.AnnotatedCFG method*), 872
add_loop_exit_stmt() (*angr.analyses.loop_analysis.LoopAnalysisState method*), 837
add_mapping() (*angr.analyses.decompiler.structured_codegen.base.InstructionMapping method*), 723
add_mapping() (*angr.analyses.decompiler.structured_codegen.base.PositionMapping method*), 723
add_memory_data() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 540
add_memory_data() (*angr.knowledge_plugins.cfg.CFGModel method*), 534
add_memory_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 764
add_memory_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 600
add_memory_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 580
add_memory_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 764
add_memory_use_by_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 812
add_memory_use_by_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 782
add_memory_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 600
add_memory_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 580
add_memory_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 812
add_memory_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 782
add_memory_variable() (*angr.sim_variable.SimVariableSet method*), 507
add_memory_variables() (*angr.sim_variable.SimVariableSet method*), 507
add_node() (*angr.analyses.decompiler.structuring.structurer_nodes.SequenceNode method*), 685
add_node() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 795
add_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 537
add_node() (*angr.knowledge_plugins.cfg.CFGModel method*), 530
add_nonreturning_function() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 649
add_number_mapping() (*angr.procedures.definitions.SimSyscallLibrary method*), 480
add_number_mapping_from_dict()

(*angr.procedures.definitions.SimSyscallLibrary method*), 481
add_obj() (*angr.analyses.cfg.cfb.CFBlanket method*), 638
add_object() (*angr.keyed_region.KeyedRegion method*), 615
add_object() (*angr.keyed_region.RegionObject method*), 614
add_patch() (*angr.knowledge_plugins.patches.PatchManager method*), 521
add_patch_obj() (*angr.knowledge_plugins.patches.PatchManager method*), 522
add_regioned_address() (*angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor.AbstractAddressDescriptor method*), 371
add_register_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 763
add_register_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 811
add_register_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 781
add_register_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 599
add_register_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
add_register_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 763
add_register_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 599
add_register_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 579
add_register_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 811
add_register_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 781
add_register_variable() (*angr.sim_variable.SimVariableSet method*), 507
add_retout_site() (*angr.knowledge_plugins.functions.function.Function method*), 556
add_returning_function() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 649
add_sinkhole() (*angr.state_plugins.cgc.SimStateCGC method*), 273
add_stack_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 763
add_stack_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 811
add_stack_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 781
add_stack_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 599
add_stack_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 579
add_stack_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 763
add_stack_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 600
add_stack_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 579
add_stack_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 811
add_stack_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 781
add_state() (*angr.misc.hierarchy.StateHierarchy method*), 389
add_state() (*angr.misc.StateHierarchy method*), 180
add_statements_to_whitelist() (*angr.annocfg.AnnotatedCFG method*), 872
add_successor() (*angr.engines.successors.SimSuccessors method*), 429
add_tmp_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 764
add_tmp_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 810
add_tmp_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 780
add_tmp_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 601
add_tmp_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 580
add_tmp_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 765
add_tmp_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 601
add_tmp_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 580
add_tmp_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 810
add_tmp_use_by_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 781
add_transitions() (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.C*

method), 815
add_type_constraint() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 822
add_type_variable() (*angr.analyses.typehoon.typevars.TypeVariables method*), 831
add_use() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 761
add_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 810
add_use() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 780
add_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 597
add_use() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 576
add_use() (*angr.knowledge_plugins.key_definitions.Uses method*), 581
add_use() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 606
add_use_by_def() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 761
add_use_by_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 810
add_use_by_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 780
add_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 597
add_use_by_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 576
add_value() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 350
add_variable() (*angr.keyed_region.KeyedRegion method*), 614
add_variable() (*angr.knowledge_plugins.debug_variables.DebugVariableManager method*), 569
add_variable() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 562
add_variable_list() (*angr.knowledge_plugins.debug_variables.DebugVariableManager method*), 570
add_xref() (*angr.analyses.xrefs.SimEngineXRefsVEX method*), 862
add_xref() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 611
add_xrefs() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 611
AddN (*class in angr.analyses.typehoon.typevars*), 832
addr (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
addr (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 697
addr (*angr.analyses.decompiler.graph_region.GraphRegion property*), 700
addr (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 725
addr (*angr.analyses.decompiler.structuring.structurer_nodes.BaseNode attribute*), 685
addr (*angr.analyses.decompiler.structuring.structurer_nodes.BreakNode attribute*), 687
addr (*angr.analyses.decompiler.structuring.structurer_nodes.CascadingCo attribute*), 687
addr (*angr.analyses.decompiler.structuring.structurer_nodes.CodeNode property*), 686
addr (*angr.analyses.decompiler.structuring.structurer_nodes.ConditionNod attribute*), 686
addr (*angr.analyses.decompiler.structuring.structurer_nodes.ContinueNode attribute*), 688
addr (*angr.analyses.decompiler.structuring.structurer_nodes.IncompleteSw attribute*), 689
addr (*angr.analyses.decompiler.structuring.structurer_nodes.IncompleteSw attribute*), 689
addr (*angr.analyses.decompiler.structuring.structurer_nodes.LoopNode property*), 687
addr (*angr.analyses.decompiler.structuring.structurer_nodes.MultiNode attribute*), 685
addr (*angr.analyses.decompiler.structuring.structurer_nodes.SequenceNod attribute*), 685
addr (*angr.analyses.decompiler.structuring.structurer_nodes.SwitchCaseNo attribute*), 688
addr (*angr.analyses.disassembly.DisassemblyPiece attribute*), 846
addr (*angr.analyses.disassembly.IROp attribute*), 847
addr (*angr.analyses.disassembly.OperandPiece attribute*), 849
addr (*angr.analyses.reaching_definitions.definition.Definition attribute*), 768
addr (*angr.angrdb.models.DbComment attribute*), 679
addr (*angr.angrdb.models.DbFunction attribute*), 677
addr (*angr.angrdb.models.DbLabel attribute*), 679
addr (*angr.Block attribute*), 170
addr (*angr.block.Block attribute*), 221
addr (*angr.block.DisassemblerBlock attribute*), 220
addr (*angr.codenode.CodeNode attribute*), 873
addr (*angr.engines.pcode.lifter.IRSB attribute*), 437
addr (*angr.engines.pcode.lifter.Lifter attribute*), 439
addr (*angr.engines.pcode.lifter.PcodeDisassemblerBlock attribute*), 435
addr (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
addr (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
addr (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
addr (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548

addr (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
addr (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 543
addr (*angr.knowledge_plugins.cfg.MemoryData attribute*), 524
addr (*angr.knowledge_plugins.functions.function.Function attribute*), 552
addr (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
addr (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation attribute*), 588
addr (*angr.sim_state.SimState property*), 225
addr (*angr.sim_variable.SimMemoryVariable attribute*), 505
addr (*angr.SimState property*), 182
addr (*angr.state_plugins.history.SimStateHistory property*), 267
addr_and_variables (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase attribute*), 819
addr_invalid() (*angr.state_plugins.cgc.SimStateCGC method*), 272
addr_to_instruction_addr() (*angr.knowledge_plugins.functions.function.Function method*), 558
address (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 800
address (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
address (*angr.block.CapstoneInsn property*), 221
address (*angr.block.DisassemblerInsn property*), 220
address (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 435
address (*angr.knowledge_plugins.cfg.memory_data.MemoryData property*), 543
address (*angr.knowledge_plugins.cfg.MemoryData property*), 525
address (*angr.state_plugins.unicorn_engine.MEM_PATCH attribute*), 284
address (*angr.state_plugins.unicorn_engine.MemoryValue attribute*), 284
address (*angr.storage.memory_mixins.regioned_memory.region_data.AddressWrapper attribute*), 369
address_multi (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 800
address_multi (*angr.analyses.reaching_definitions.function_handler.FunctionCallDataUnwrapped attribute*), 803
address_multi (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
AddressConcretizationMixin (*class in angr.storage.memory_mixins.address_concretization_mixin*), 344
AddressTransferringTypes (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 662
AddressWrapper (*class in angr.storage.memory_mixins.regioned_memory.region_data*), 368
addrs_for_hash() (*angr.storage.memory_mixins.convenient_mappings_m method*), 348
addrs_for_name() (*angr.storage.memory_mixins.convenient_mappings_m method*), 348
ADDS_EXITS (*angr.sim_procedure.SimProcedure attribute*), 471
ADDS_EXITS (*angr.SimProcedure attribute*), 159
AFTER_AIL_GRAPH_CREATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 703
AFTER_GLOBAL_SIMPLIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 703
AFTER_MAKING_CALLSITES (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 703
AFTER_SINGLE_BLOCK_SIMPLIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 703
AFTER_STRUCTURING (*angr.analyses.decompiler.optimization_passes.optim attribute*), 703
AFTER_VARIABLE_RECOVERY (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 703
AggressiveConcretizationAnnotation (*class in angr.state_plugins.unicorn_engine*), 288
AILBlockTempCollector (*class in angr.analyses.decompiler.ail_simplifier*), 691
AILGraphWalker (*class in angr.analyses.decompiler.ailgraph_walker*), 692
AILSimplifier (*class in angr.analyses.decompiler.ail_simplifier*), 691
alignment (*angr.knowledge_plugins.functions.function.Function property*), 553
alignment (*angr.sim_type.SimStruct property*), 516
alignment (*angr.sim_type.SimType property*), 508
alignment (*angr.sim_type.SimTypeArray property*), 512
alignment (*angr.sim_type.SimTypeDouble property*), 515
alignment (*angr.sim_type.SimTypeString property*), 513
alignment (*angr.sim_type.SimTypeWString property*), 513
alignment (*angr.sim_type.SimUnion property*), 517
alignment (*angr.sim_type.TypeRef property*), 508
ALL (*angr.sim_manager.SimulationManager attribute*), 383
ALL (*angr.SimulationManager attribute*), 172

all_bytes_changed_in_history() (*angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.HistoryTrackingMixin method*), 361
all_constants (*angr.engines.pcode.lifter.IRSB property*), 438
all_definitions (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState attribute*), 807
all_definitions (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 792
all_definitions (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 772
all_definitions (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 777
all_objects (*angr.state_plugins.sim_action.SimAction property*), 466
all_objects (*angr.state_plugins.sim_action.SimActionConstraint property*), 467
all_objects (*angr.state_plugins.sim_action.SimActionData property*), 468
all_objects (*angr.state_plugins.sim_action.SimActionExit property*), 467
all_objects (*angr.state_plugins.sim_action.SimActionOperation property*), 467
all_stashes (*angr.analyses.veritesting.Veritesting attribute*), 839
all_successors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 625
all_successors() (*angr.state_hierarchy.StateHierarchy method*), 389
all_successors() (*angr.StateHierarchy method*), 180
ALL_TABLES (*angr.angrdb.db.AngrDB attribute*), 674
all_uses (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 792
all_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 772
alloc() (*angr.calling_conventions.AllocHelper method*), 483
allocate() (*angr.analyses.reaching_definitions.heap_allocator.HeapAllocator method*), 797
allocate() (*angr.SimHeapBrk method*), 205
allocate() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 298
allocate_stack_pages() (*angr.storage.memory_mixins.paged_memory.stack_allocation_mixin.StackAllocationMixin method*), 358
allocated_addresses (*angr.analyses.reaching_definitions.heap_allocator.HeapAllocator property*), 798
allocated_chunks() (*angr.SimHeapPTMalloc method*), 206
allocated_chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 301
allocated_chunks() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 304
AllocHelper (*class in angr.calling_conventions*), 483
allow_arch_optimizations (*angr.engines.pcode.lifter.Lifter attribute*), 440
allow_arch_optimizations (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
ALT_NAMES (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis attribute*), 472
ALT_NAMES (*angr.SimProcedure attribute*), 159
ALWAYS (*angr.analyses.decompiler.structuring.phoenix.MultiStmtExprMode attribute*), 690
AlwaysUpdate (*angr.analyses.calling_convention.UpdateArgumentsOption attribute*), 633
AMD64CCallRewriter (*class in angr.analyses.decompiler.ccall_rewriters.amd64_ccalls*), 693
AMD64ElfGotResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got*), 657
analyses (*angr.Project property*), 164
analyses (*angr.project.Project property*), 214
AnalysesHub (*class in angr.analyses.analysis*), 619
AnalysesHubWithDefault (*class in angr.analyses.analysis*), 620
analysis (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitions attribute*), 807
analysis (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 777
Analysis (*class in angr*), 178
Analysis (*class in angr.analyses.analysis*), 620
AnalysisFactory (*class in angr.analyses.analysis*), 620
AnalysisLogEntry (*class in angr.analyses.analysis*), 619
AnalysisTask (*class in angr.analyses.vfg*), 840
analyze() (*angr.analyses.callee_cleanup_finder.CalleeCleanupFinder method*), 861
analyze() (*angr.analyses.code_tagging.CodeTagging method*), 674
analyze() (*angr.analyses.decompiler.optimization_passes.optimization_pa method*), 703
analyze() (*angr.analyses.vtable.VtableFinder method*), 845
analyze_or_mini_stack_allocation_mixin (*angr.state_plugins.trace_additions.ZenPlugin method*), 277
And (*angr.engines.light.data.ArithmeticExpression attribute*), 750
angr module, 157
angr.analyses module, 619
angr.analyses.analysis
angr.analyses.backward_slice

module, 627
angr.analyses.binary_optimizer
module, 859
angr.analyses.bindiff
module, 629
angr.analyses.boyscout
module, 632
angr.analyses.callee_cleanup_finder
module, 860
angr.analyses.calling_convention
module, 632
angr.analyses.cdg
module, 673
angr.analyses.cfg
module, 637
angr.analyses.cfg.cfb
module, 637
angr.analyses.cfg.cfg
module, 638
angr.analyses.cfg.cfg_arch_options
module, 656
angr.analyses.cfg.cfg_base
module, 645
angr.analyses.cfg.cfg_emulated
module, 640
angr.analyses.cfg.cfg_fast
module, 647
angr.analyses.cfg.cfg_fast_soot
module, 668
angr.analyses.cfg.cfg_job_base
module, 656
angr.analyses.cfg.indirect_jump_resolvers
module, 668
angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got
module, 657
angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast
module, 658
angr.analyses.cfg.indirect_jump_resolvers.constant_value
module, 666
angr.analyses.cfg.indirect_jump_resolvers.default_resolvers
module, 662
angr.analyses.cfg.indirect_jump_resolvers.jumptable
module, 662
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast
module, 660
angr.analyses.cfg.indirect_jump_resolvers.resolver
module, 667
angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt
module, 661
angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat
module, 659
angr.analyses.cfg.segment_list
module, 670
angr.analyses.cfg_slice_to_sink
module, 814
angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink
module, 814
angr.analyses.cfg_slice_to_sink.graph
module, 815
angr.analyses.cfg_slice_to_sink.transitions
module, 816
angr.analyses.class_identifier
module, 846
angr.analyses.code_tagging
module, 673
angr.analyses.complete_calling_conventions
module, 634
angr.analyses.congruency_check
module, 858
angr.analyses.data_dep
module, 870
angr.analyses.data_dep.data_dependency_analysis
module, 865
angr.analyses.data_dep.dep_nodes
module, 867
angr.analyses.data_dep.sim_act_location
module, 866
angr.analyses.datagraph_meta
module, 673
angr.analyses.ddg
module, 744
angr.analyses.decompiler
module, 691
angr.analyses.decompiler.ail_simplifier
module, 691
angr.analyses.decompiler.ailgraph_walker
module, 692
angr.analyses.decompiler.block_simplifier
module, 692
angr.analyses.decompiler.callsite_maker
module, 693
angr.analyses.decompiler.ccall_rewriters
module, 693
angr.analyses.decompiler.ccall_rewriters.amd64_ccalls
module, 693
angr.analyses.decompiler.ccall_rewriters.rewriter_base
module, 693
angr.analyses.decompiler.clinic
module, 694
angr.analyses.decompiler.condition_processor
module, 695
angr.analyses.decompiler.decompilation_cache
module, 697
angr.analyses.decompiler.decompilation_options
module, 696
angr.analyses.decompiler.decompiler
module, 697
angr.analyses.decompiler.empty_node_remover

module, 698
angr.analyses.decompiler.expression_narrower
module, 699
angr.analyses.decompiler.graph_region
module, 699
angr.analyses.decompiler.jump_target_collector
module, 700
angr.analyses.decompiler.jumptable_entry_condition_rewriter
module, 700
angr.analyses.decompiler.optimization_passes
module, 701
angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier
module, 705
angr.analyses.decompiler.optimization_passes.call_stmt_rewriter
module, 701
angr.analyses.decompiler.optimization_passes.div_simplifier
module, 705
angr.analyses.decompiler.optimization_passes.engine_base
module, 701
angr.analyses.decompiler.optimization_passes.expr_op_swapper
module, 709
angr.analyses.decompiler.optimization_passes.inline_small_functions
module, 705
angr.analyses.decompiler.optimization_passes.lower_switch_simplifier
module, 706
angr.analyses.decompiler.optimization_passes.mod_simplifier
module, 708
angr.analyses.decompiler.optimization_passes.multi_simplifier
module, 708
angr.analyses.decompiler.optimization_passes.optimization_pass
module, 702
angr.analyses.decompiler.optimization_passes.register_save_area_simplifier
module, 710
angr.analyses.decompiler.optimization_passes.return_duplicator_base
module, 711
angr.analyses.decompiler.optimization_passes.stack_canary_simplifier
module, 704
angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier
module, 711
angr.analyses.decompiler.peephole_optimizations
module, 711
angr.analyses.decompiler.peephole_optimizations.base
module, 711
angr.analyses.decompiler.redundant_label_remover
module, 722
angr.analyses.decompiler.region_identifier
module, 713
angr.analyses.decompiler.region_simplifiers
module, 714
angr.analyses.decompiler.region_simplifiers.cancellation_constructs
module, 714
angr.analyses.decompiler.region_simplifiers.expr_folding
module, 714
angr.analyses.decompiler.region_simplifiers.goto
module, 717
angr.analyses.decompiler.region_simplifiers.if_
module, 717
angr.analyses.decompiler.region_simplifiers.ifelse
module, 718
angr.analyses.decompiler.region_simplifiers.loop
module, 718
angr.analyses.decompiler.region_simplifiers.node_address_finder
module, 718
angr.analyses.decompiler.region_simplifiers.region_simplifier
module, 718
angr.analyses.decompiler.region_simplifiers.switch_cluster
module, 718
angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier
module, 722
angr.analyses.decompiler.region_walker
module, 722
angr.analyses.decompiler.sequence_walker
module, 722
angr.analyses.decompiler.structured_codegen
module, 723
angr.analyses.decompiler.structured_codegen.base
module, 723
angr.analyses.decompiler.structured_codegen.c
module, 724
angr.analyses.decompiler.structured_codegen.dummy
module, 741
angr.analyses.decompiler.structured_codegen.dwarf_import
module, 741
angr.analyses.decompiler.structuring
module, 684
angr.analyses.decompiler.structuring.dream
module, 684
angr.analyses.decompiler.structuring.phoenix
module, 690
angr.analyses.decompiler.structuring.recursive_structurer
module, 684
angr.analyses.decompiler.structuring.structurer_base
module, 689
angr.analyses.decompiler.structuring.structurer_nodes
module, 685
angr.analyses.decompiler.utils
module, 741
angr.analyses.disassembly
module, 846
angr.analyses.disassembly_utils
module, 850
angr.analyses.dominance_frontier
module, 861
angr.analyses.find_objects_static

module, 845
angr.analyses.flirt
module, 749
angr.analyses.forward_analysis
module, 621
angr.analyses.forward_analysis.forward_analysis
module, 621
angr.analyses.forward_analysis.job_info
module, 622
angr.analyses.forward_analysis.visitors
module, 622
angr.analyses.forward_analysis.visitors.call_graph
module, 622
angr.analyses.forward_analysis.visitors.function_graph
module, 623
angr.analyses.forward_analysis.visitors.graph
module, 624
angr.analyses.forward_analysis.visitors.loop
module, 626
angr.analyses.forward_analysis.visitors.single_node_graph
module, 627
angr.analyses.identifier.identify
module, 835
angr.analyses.init_finder
module, 861
angr.analyses.loop_analysis
module, 837
angr.analyses.loopfinder
module, 836
angr.analyses.propagator
module, 751
angr.analyses.propagator.engine_ail
module, 753
angr.analyses.propagator.engine_base
module, 752
angr.analyses.propagator.engine_vex
module, 753
angr.analyses.propagator.outdated_definition_walker
module, 754
angr.analyses.propagator.propagator
module, 755
angr.analyses.propagator.tmpvar_finder
module, 755
angr.analyses.propagator.top_checker_mixin
module, 757
angr.analyses.propagator.values
module, 751
angr.analyses.propagator.vex_vars
module, 752
angr.analyses.proximity_graph
module, 863
angr.analyses.reaching_definitions
module, 757
angr.analyses.reaching_definitions.call_trace
module, 789
angr.analyses.reaching_definitions.dep_graph
module, 794
angr.analyses.reaching_definitions.engine_ail
module, 813
angr.analyses.reaching_definitions.engine_vex
module, 790
angr.analyses.reaching_definitions.function_handler
module, 798
angr.analyses.reaching_definitions.heap_allocator
module, 797
angr.analyses.reaching_definitions.rd_state
module, 805
angr.analyses.reaching_definitions.reaching_definitions
module, 791
angr.analyses.reaching_definitions.subject
module, 813
angr.analyses.reassembler
module, 850
angr.analyses.soot_class_hierarchy
module, 636
angr.analyses.stack_pointer_tracker
module, 816
angr.analyses.static_hooker
module, 859
angr.analyses.typehoon
module, 835
angr.analyses.typehoon.lifter
module, 827
angr.analyses.typehoon.simple_solver
module, 828
angr.analyses.typehoon.translator
module, 828
angr.analyses.typehoon.typeconsts
module, 834
angr.analyses.typehoon.typehoon
module, 833
angr.analyses.typehoon.typevars
module, 829
angr.analyses.variable_recovery
module, 827
angr.analyses.variable_recovery.annotations
module, 818
angr.analyses.variable_recovery.engine_ail
module, 826
angr.analyses.variable_recovery.engine_base
module, 826
angr.analyses.variable_recovery.engine_vex
module, 826
angr.analyses.variable_recovery.irsb_scanner
module, 827
angr.analyses.variable_recovery.variable_recovery
module, 825
angr.analyses.variable_recovery.variable_recovery_base

module, 819
angr.analyses.variable_recovery.variable_recovery_fast
module, 823
angr.analyses.veritesting
module, 838
angr.analyses.vfg
module, 839
angr.analyses.vsa_ddg
module, 844
angr.analyses.vtable
module, 845
angr.analyses.xrefs
module, 861
angr.angrdb
module, 674
angr.angrdb.db
module, 674
angr.angrdb.models
module, 676
angr.angrdb.serializers
module, 679
angr.angrdb.serializers.cfg_model
module, 679
angr.angrdb.serializers.comments
module, 680
angr.angrdb.serializers.funcs
module, 680
angr.angrdb.serializers.kb
module, 681
angr.angrdb.serializers.labels
module, 681
angr.angrdb.serializers.loader
module, 681
angr.angrdb.serializers.structured_code
module, 683
angr.angrdb.serializers.variables
module, 682
angr.angrdb.serializers.xrefs
module, 682
angr.annocfg
module, 871
angr.blade
module, 870
angr.block
module, 220
angr.callable
module, 519
angr.calling_conventions
module, 483
angr.code_location
module, 612
angr.codenode
module, 873
angr.concretization_strategies
module, 335
angr.concretization_strategies.any
module, 381
angr.concretization_strategies.controlled_data
module, 381
angr.concretization_strategies.eval
module, 379
angr.concretization_strategies.max
module, 380
angr.concretization_strategies.nonzero
module, 381
angr.concretization_strategies.nonzero_range
module, 380
angr.concretization_strategies.norepeats
module, 379
angr.concretization_strategies.norepeats_range
module, 381
angr.concretization_strategies.range
module, 380
angr.concretization_strategies.single
module, 379
angr.concretization_strategies.solutions
module, 379
angr.concretization_strategies.unlimited_range
module, 381
angr.distributed
module, 900
angr.distributed.server
module, 900
angr.distributed.worker
module, 900
angr.engines
module, 427
angr.engines.concrete
module, 433
angr.engines.engine
module, 427
angr.engines.failure
module, 431
angr.engines.hook
module, 430
angr.engines.light
module, 750
angr.engines.light.data
module, 750
angr.engines.light.engine
module, 750
angr.engines.pcode
module, 434
angr.engines.pcode.behavior
module, 444
angr.engines.pcode.cc
module, 464
angr.engines.pcode.emulate

module, 444
angr.engines.pcode.engine
module, 434
angr.engines.pcode.lifter
module, 435
angr.engines.procedure
module, 430
angr.engines.soot
module, 431
angr.engines.soot.engine
module, 431
angr.engines.successors
module, 428
angr.engines.syscall
module, 431
angr.engines.unicorn
module, 432
angr.engines.vex
module, 431
angr.errors
module, 894
angr.exploration_techniques
module, 390
angr.exploration_techniques.bucketizer
module, 426
angr.exploration_techniques.common
module, 424
angr.exploration_techniques.dfs
module, 408
angr.exploration_techniques.director
module, 418
angr.exploration_techniques.driller_core
module, 416
angr.exploration_techniques.explorer
module, 408
angr.exploration_techniques.lengthlimiter
module, 409
angr.exploration_techniques.local_loop_seer
module, 422
angr.exploration_techniques.loop_seer
module, 421
angr.exploration_techniques.manual_mergepoint
module, 410
angr.exploration_techniques.memory_watcher
module, 426
angr.exploration_techniques.oppologist
module, 420
angr.exploration_techniques.slicecutor
module, 417
angr.exploration_techniques.spiller
module, 410
angr.exploration_techniques.spiller_db
module, 412
angr.exploration_techniques.stochastic
module, 423
angr.exploration_techniques.suggestions
module, 426
angr.exploration_techniques.symbion
module, 424
angr.exploration_techniques.tech_builder
module, 424
angr.exploration_techniques.threading
module, 413
angr.exploration_techniques.timeout
module, 407
angr.exploration_techniques.tracer
module, 414
angr.exploration_techniques.unique
module, 423
angr.exploration_techniques.veritesting
module, 413
angr.factory
module, 216
angr.flirt
module, 883
angr.flirt.build_sig
module, 883
angr.keyed_region
module, 613
angr.knowledge_base
module, 520
angr.knowledge_base.knowledge_base
module, 520
angr.knowledge_plugins
module, 521
angr.knowledge_plugins.callsite_prototypes
module, 523
angr.knowledge_plugins.cfg
module, 524
angr.knowledge_plugins.cfg.cfg_manager
module, 544
angr.knowledge_plugins.cfg.cfg_model
module, 536
angr.knowledge_plugins.cfg.cfg_node
module, 544
angr.knowledge_plugins.cfg.indirect_jump
module, 547
angr.knowledge_plugins.cfg.memory_data
module, 542
angr.knowledge_plugins.comments
module, 549
angr.knowledge_plugins.data
module, 549
angr.knowledge_plugins.debug_variables
module, 568
angr.knowledge_plugins.functions
module, 550
angr.knowledge_plugins.functions.function

module, 552
angr.knowledge_plugins.functions.function_manager
module, 550
angr.knowledge_plugins.functions.function_parser
module, 559
angr.knowledge_plugins.functions.soot_function
module, 560
angr.knowledge_plugins.indirect_jumps
module, 549
angr.knowledge_plugins.key_definitions
module, 570
angr.knowledge_plugins.key_definitions.atoms
module, 584
angr.knowledge_plugins.key_definitions.constants
module, 588
angr.knowledge_plugins.key_definitions.definition
module, 588
angr.knowledge_plugins.key_definitions.environment
module, 590
angr.knowledge_plugins.key_definitions.heap_address
module, 591
angr.knowledge_plugins.key_definitions.key_definition
module, 591
angr.knowledge_plugins.key_definitions.live_definitions
module, 592
angr.knowledge_plugins.key_definitions.rd_model
module, 601
angr.knowledge_plugins.key_definitions.tag
module, 603
angr.knowledge_plugins.key_definitions.undefined
module, 605
angr.knowledge_plugins.key_definitions.unknown_size
module, 605
angr.knowledge_plugins.key_definitions.uses
module, 605
angr.knowledge_plugins.labels
module, 549
angr.knowledge_plugins.patches
module, 521
angr.knowledge_plugins.plugin
module, 523
angr.knowledge_plugins.propagations
module, 549
angr.knowledge_plugins.structured_code
module, 570
angr.knowledge_plugins.structured_code.manager
module, 570
angr.knowledge_plugins.sync
module, 607
angr.knowledge_plugins.sync.sync_controller
module, 607
angr.knowledge_plugins.types
module, 549
angr.knowledge_plugins.variables
module, 561
angr.knowledge_plugins.variables.variable_access
module, 561
angr.knowledge_plugins.variables.variable_manager
module, 561
angr.knowledge_plugins.xrefs
module, 610
angr.knowledge_plugins.xrefs.xref
module, 610
angr.knowledge_plugins.xrefs.xref_manager
module, 611
angr.knowledge_plugins.xrefs.xref_types
module, 611
angr.misc.plugins
module, 222
angr.procedures
module, 473
angr.procedures.definitions
module, 475
angr.procedures.stubs.format_parser
module, 473
angr.project_manager
module, 212
angr.protos
module, 617
angr.serializable
module, 616
angr.sim_manager
module, 382
angr.sim_options
module, 228
angr.sim_procedure
module, 469
angr.sim_state
module, 224
angr.sim_state_options
module, 228
angr.sim_type
module, 507
angr.sim_variable
module, 502
angr.simos
module, 875
angr.simos.cgc
module, 878
angr.simos.javavm
module, 880
angr.simos.linux
module, 877
angr.simos.simos
module, 875
angr.simos.userland
module, 878
angr.simos.windows

module, 879
angr.slicer
module, 871
angr.state_hierarchy
module, 389
angr.state_plugins
module, 231
angr.state_plugins.callstack
module, 263
angr.state_plugins.cgc
module, 271
angr.state_plugins.concrete
module, 292
angr.state_plugins.debug_variables
module, 307
angr.state_plugins.filesystem
module, 248
angr.state_plugins.gdb
module, 270
angr.state_plugins.globals
module, 278
angr.state_plugins.heap
module, 297
angr.state_plugins.heap.heap_base
module, 297
angr.state_plugins.heap.heap_brk
module, 298
angr.state_plugins.heap.heap_freelist
module, 300
angr.state_plugins.heap.heap_libc
module, 301
angr.state_plugins.heap.heap_ptmalloc
module, 302
angr.state_plugins.heap.utils
module, 306
angr.state_plugins.history
module, 267
angr.state_plugins.inspect
module, 233
angr.state_plugins.javavm_classloader
module, 294
angr.state_plugins.jni_references
module, 296
angr.state_plugins.libc
module, 236
angr.state_plugins.light_registers
module, 266
angr.state_plugins.log
module, 262
angr.state_plugins.loop_data
module, 291
angr.state_plugins.plugin
module, 231
angr.state_plugins.posix
module, 240
angr.state_plugins.preconstrainer
module, 282
angr.state_plugins.scratch
module, 280
angr.state_plugins.sim_action
module, 466
angr.state_plugins.sim_action_object
module, 468
angr.state_plugins.sim_event
module, 468
angr.state_plugins.solver
module, 254
angr.state_plugins.symbolizer
module, 307
angr.state_plugins.trace_additions
module, 273
angr.state_plugins.uc_manager
module, 279
angr.state_plugins.unicorn_engine
module, 284
angr.state_plugins.view
module, 309
angr.storage
module, 309
angr.storage.file
module, 314
angr.storage.memory_mixins
module, 336
angr.storage.memory_mixins.actions_mixin
module, 342
angr.storage.memory_mixins.address_concretization_mixin
module, 344
angr.storage.memory_mixins.bvv_conversion_mixin
module, 340
angr.storage.memory_mixins.clouseau_mixin
module, 346
angr.storage.memory_mixins.conditional_store_mixin
module, 346
angr.storage.memory_mixins.convenient_mappings_mixin
module, 348
angr.storage.memory_mixins.default_filler_mixin
module, 340
angr.storage.memory_mixins.dirty_addrs_mixin
module, 344
angr.storage.memory_mixins.hex_dumper_mixin
module, 341
angr.storage.memory_mixins.javavm_memory
module, 376
angr.storage.memory_mixins.javavm_memory.javavm_memory_mix
module, 376
angr.storage.memory_mixins.keyvalue_memory
module, 375
angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory

module, 375
angr.storage.memory_mixins.label_merger_mixin
module, 346
angr.storage.memory_mixins.multi_value_merger_mixin
module, 352
angr.storage.memory_mixins.name_resolution_mixin
module, 339
angr.storage.memory_mixins.paged_memory
module, 352
angr.storage.memory_mixins.paged_memory.page_backers_mixin
module, 357
angr.storage.memory_mixins.paged_memory.paged_memory_mixin
module, 352
angr.storage.memory_mixins.paged_memory.pages
module, 359
angr.storage.memory_mixins.paged_memory.pages.cooperation
module, 361
angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin
module, 360
angr.storage.memory_mixins.paged_memory.pages.ispo_mixin
module, 361
angr.storage.memory_mixins.paged_memory.pages.list_storage
module, 362
angr.storage.memory_mixins.paged_memory.pages.multi_values
module, 350
angr.storage.memory_mixins.paged_memory.pages.pages
module, 348
angr.storage.memory_mixins.paged_memory.pages.permissions_mixin
module, 360
angr.storage.memory_mixins.paged_memory.pages.refcount_mixin
module, 359
angr.storage.memory_mixins.paged_memory.pages.underlying
module, 363
angr.storage.memory_mixins.paged_memory.privileged_mixin
module, 358
angr.storage.memory_mixins.paged_memory.stack_allocation_mixin
module, 358
angr.storage.memory_mixins.regioned_memory
module, 365
angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor
module, 371
angr.storage.memory_mixins.regioned_memory.abstract_merger_mixin
module, 373
angr.storage.memory_mixins.regioned_memory.region_categories
module, 370
angr.storage.memory_mixins.regioned_memory.region_data
module, 368
angr.storage.memory_mixins.regioned_memory.region_meta_mixin
module, 371
angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin
module, 373
angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin
module, 365
angr.storage.memory_mixins.regioned_memory.static_find
module, 370
angr.storage.memory_mixins.simple_interface_mixin
module, 342
angr.storage.memory_mixins.simplification_mixin
module, 347
angr.storage.memory_mixins.size_resolution_mixin
module, 343
angr.storage.memory_mixins.slotted_memory
module, 374
angr.storage.memory_mixins.smart_find_mixin
module, 339
angr.storage.memory_mixins.symbolic_merger_mixin
module, 342
angr.storage.memory_mixins.top_merger_mixin
module, 351
angr.storage.memory_mixins.underconstrained_mixin
module, 341
angr.storage.memory_mixins.unwrapper_mixin
module, 347
angr.storage.memory_object
module, 334
angr.storage.pcap
module, 335
angr.utils
module, 884
angr.utils.algo
module, 885
angr.utils.constants
module, 885
angr.utils.coro
module, 885
angr.utils.dynamic_dictlist
module, 885
angr.utils.enums_conv
module, 886
angr.utils.env
module, 886
angr.utils.formatting
module, 892
angr.utils.graph
module, 886
angr.utils.lazy_import
module, 890
angr.utils.library
module, 891
angr.utils.loader
module, 890
angr.utils.mp
module, 893
angr.utils.ssexy
module, 892
angr.vaults
module, 617
AngrAnalysisError, 894

AngrAnnotatedCFGError, 894
AngrAssemblyError, 895
AngrBackwardSlicingError, 894
AngrBladeError, 894
AngrBladeSimProcError, 894
AngrCallableError, 895
AngrCallableMultistateError, 895
AngrCFGError, 895
AngrCorruptDBError, 896
AngrDataGraphError, 895
AngrDB (*class in angr.angrdb.db*), 674
AngrDBError, 896
AngrDDGError, 895
AngrDelayJobNotice, 895
AngrDirectorError, 896
AngrError, 894
AngrExitError, 894
AngrExplorationTechniqueError, 895
AngrExplorerError, 896
AngrForwardAnalysisError, 895
AngrGirlScoutError, 894
AngrIncompatibleDBError, 896
AngrIncongruencyError, 895
AngrInvalidArgumentError, 894
AngrJobMergingFailureNotice, 895
AngrJobWideningFailureNotice, 895
AngrLifterError, 894
AngrLoopAnalysisError, 895
AngrNoPluginError, 899
AngrObjectFactory (*class in angr.factory*), 216
AngrPathError, 894
AngrSimOSError, 895
AngrSkipJobNotice, 895
AngrSurveyorError, 894
AngrSyscallError, 895
AngrTracerError, 896
AngrTypeError, 895
AngrUnsupportedSyscallError, 898
AngrValueError, 894
AngrVariableRecoveryError, 896
AngrVaultError, 894
AngrVFGError, 895
AngrVFGRestartAnalysisNotice, 895
annotate_mv_with_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 808
annotate_mv_with_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 778
annotate_with_def() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 759
annotate_with_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 808
annotate_with_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 778
annotate_with_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi static method*), 595
annotate_with_def() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 575
annotate_with_variables() (*angr.analyses.variable_recovery.variable_recovery_base.Variabl static method*), 821
annotated_cfg() (*angr.analyses.backward_slice.BackwardSlice method*), 628
AnnotatedCFG (*class in angr.annocfg*), 871
AnnotatedVariable (*class in angr.analyses.loop_analysis*), 837
ansi_color() (*in module angr.utils.formatting*), 892
append_data() (*angr.analyses.reassembler.Reassembler method*), 857
append_procedure() (*angr.analyses.reassembler.Reassembler method*), 857
append_state() (*angr.analyses.vfg.VFGNode method*), 842
append_statement() (*in module angr.analyses.decompiler.utils*), 741
apply() (*angr.analyses.cfg.cfg_fast.FunctionCallEdge method*), 650
apply() (*angr.analyses.cfg.cfg_fast.FunctionEdge method*), 649
apply() (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge method*), 651
apply() (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge method*), 651
apply() (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge method*), 650
apply() (*angr.calling_conventions.AllocHelper method*), 483
apply() (*angr.sim_manager.SimulationManager method*), 387
apply() (*angr.SimulationManager method*), 176
apply_at_callsite (*angr.analyses.reaching_definitions.function_handle attribute*), 798
apply_definition() (*angr.knowledge_plugins.functions.function.Functio method*), 559
apply_function_edges() (*angr.analyses.cfg.cfg_fast.CFGJob method*), 652
apply_patches_to_binary() (*angr.knowledge_plugins.patches.PatchManager method*), 522
apply_patches_to_state() (*angr.knowledge_plugins.patches.PatchManager*

method), 522
arch (*angr.analyses.cfg.cfg_arch_options.CFGArchOptions attribute*), 656
arch (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AMDTableReading attribute*), 663
arch (*angr.analyses.decompiler.ccall_rewriters.rewriter_base.CCallRewriterBase attribute*), 693
arch (*angr.analyses.reaching_definitions.engine_ail.SimEngineRDAIL attribute*), 814
arch (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
arch (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState attribute*), 807
arch (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 777
arch (*angr.analyses.reaching_definitions.Register attribute*), 768
arch (*angr.Block attribute*), 170
arch (*angr.block.Block attribute*), 221
arch (*angr.block.DisassemblerBlock attribute*), 220
ARCH (*angr.calling_conventions.SimCC attribute*), 488
ARCH (*angr.calling_conventions.SimCCAArch64 attribute*), 497
ARCH (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 497
ARCH (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 495
ARCH (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 495
ARCH (*angr.calling_conventions.SimCCARM attribute*), 496
ARCH (*angr.calling_conventions.SimCCARMHF attribute*), 496
ARCH (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 496
ARCH (*angr.calling_conventions.SimCCCdecl attribute*), 492
ARCH (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
ARCH (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 492
ARCH (*angr.calling_conventions.SimCCN64 attribute*), 498
ARCH (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 499
ARCH (*angr.calling_conventions.SimCCO32 attribute*), 498
ARCH (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 498
ARCH (*angr.calling_conventions.SimCCPowerPC attribute*), 499
ARCH (*angr.calling_conventions.SimCCPowerPC64 attribute*), 500
ARCH (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 500
ARCH (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 499
ARCH (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 497
ARCH (*angr.calling_conventions.SimCCS390X attribute*), 501
ARCH (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 502
ARCH (*angr.calling_conventions.SimCCSoot attribute*), 500
ARCH (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
ARCH (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 494
ARCH (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 494
arch (*angr.engines.pcode.lifter.IRSB attribute*), 437
arch (*angr.engines.pcode.lifter.Lifter attribute*), 439
arch (*angr.engines.pcode.lifter.PcodeDisassemblerBlock attribute*), 435
arch (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
arch (*angr.knowledge_plugins.key_definitions.atoms.Register attribute*), 587
arch (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitio attribute*), 594
arch (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
arch (*angr.procedures.stubs.format_parser.FormatParser attribute*), 475
arch (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 475
arch (*angr.Project attribute*), 164
arch (*angr.project.Project attribute*), 214
arch (*angr.sim_procedure.SimProcedure attribute*), 471
arch (*angr.sim_state.SimState property*), 226
ARCH (*angr.SimCC attribute*), 185
arch (*angr.SimState property*), 182
arch_overrideable() (*in module angr.sim_state*), 224
ARCHES (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simp attribute*), 705
ARCHES (*angr.analyses.decompiler.optimization_passes.const_derefs.Consta attribute*), 701
ARCHES (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSi attribute*), 705
ARCHES (*angr.analyses.decompiler.optimization_passes.eager_returns.Eage attribute*), 702
ARCHES (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.E attribute*), 710
ARCHES (*angr.analyses.decompiler.optimization_passes.ite_expr_converter. attribute*), 706
ARCHES (*angr.analyses.decompiler.optimization_passes.lowered_switch_sim attribute*), 707

ARCHES (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 708
ARCHES (*angr.analyses.decompiler.optimization_passes.multi_simplifier.MultiSimplifier attribute*), 708
ARCHES (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute*), 703
ARCHES (*angr.analyses.decompiler.optimization_passes.optimization_pass.StructuringOptimizationPass attribute*), 704
ARCHES (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifier attribute*), 710
ARCHES (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 711
ARCHES (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 704
ARCHES (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier attribute*), 711
arg_list (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 726
arg_locs() (*angr.calling_conventions.SimCC method*), 489
arg_locs() (*angr.SimCC method*), 186
ARG_REGS (*angr.calling_conventions.SimCC attribute*), 488
ARG_REGS (*angr.calling_conventions.SimCCAArch64 attribute*), 496
ARG_REGS (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 497
ARG_REGS (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 495
ARG_REGS (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 495
ARG_REGS (*angr.calling_conventions.SimCCARM attribute*), 495
ARG_REGS (*angr.calling_conventions.SimCCARMHF attribute*), 496
ARG_REGS (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 496
ARG_REGS (*angr.calling_conventions.SimCCCdecl attribute*), 491
ARG_REGS (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 492
ARG_REGS (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 492
ARG_REGS (*angr.calling_conventions.SimCCN64 attribute*), 498
ARG_REGS (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 498
ARG_REGS (*angr.calling_conventions.SimCCO32 attribute*), 497
ARG_REGS (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 498
ARG_REGS (*angr.calling_conventions.SimCCPowerPC attribute*), 499
ARG_REGS (*angr.calling_conventions.SimCCPowerPC64 attribute*), 500
ARG_REGS (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 500
ARG_REGS (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 499
ARG_REGS (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 497
ARG_REGS (*angr.calling_conventions.SimCCS390X attribute*), 501
ARG_REGS (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 502
ARG_REGS (*angr.calling_conventions.SimCCSoot attribute*), 503
ARG_REGS (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 493
ARG_REGS (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 493
ARG_REGS (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 494
ARG_REGS (*angr.engines.pcode.cc.SimCCM68k attribute*), 464
ARG_REGS (*angr.engines.pcode.cc.SimCCPARISC attribute*), 465
ARG_REGS (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 465
ARG_REGS (*angr.engines.pcode.cc.SimCCRISCV attribute*), 464
ARG_REGS (*angr.engines.pcode.cc.SimCCSH4 attribute*), 465
ARG_REGS (*angr.engines.pcode.cc.SimCCSPARC attribute*), 465
ARG_REGS (*angr.engines.pcode.cc.SimCCXtensa attribute*), 466
ARG_REGS (*angr.SimCC attribute*), 185
arg_session (*angr.procedures.stubs.format_parser.FormatParser attribute*), 475
arg_session (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 475
arg_session (*angr.sim_procedure.SimProcedure attribute*), 471
arg_session() (*angr.calling_conventions.SimCC method*), 489
arg_session() (*angr.SimCC method*), 186
args (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 730
args (*angr.sim_type.SimTypeCppFunction attribute*), 514
args (*angr.utils.mp.Closure attribute*), 893
args_atoms (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 801
args_atoms (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
args_defns (*angr.analyses.reaching_definitions.dep_graph.FunctionCallRefs attribute*), 794
ARGS_MISMATCH (*angr.procedures.stubs.format_parser.FormatParser*

attribute), 474
ARGS_MISMATCH (*angr.sim_procedure.SimProcedure attribute*), 472
ARGS_MISMATCH (*angr.SimProcedure attribute*), 159
args_values (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 801
args_values (*angr.analyses.reaching_definitions.FunctionCallDrop attribute*), 787
ArgSession (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
ArgSession (*angr.calling_conventions.SimCCUsercall attribute*), 491
ArgSession (*class in angr.calling_conventions*), 487
argument_types (*angr.sim_procedure.SimProcedure property*), 473
argument_types (*angr.SimProcedure property*), 161
arguments (*angr.knowledge_plugins.functions.function.Function property*), 558
ArithmeticExpression (*class in angr.engines.light.data*), 750
ARM (*angr.analyses.cfg.cfg_fast.ARMDecodingMode attribute*), 648
ARMDecodingMode (*class in angr.analyses.cfg.cfg_fast*), 648
ArmElfFastResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast*), 658
Array (*class in angr.analyses.typehoon.typeconsts*), 835
array() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 308
array() (*angr.state_plugins.view.SimMemView method*), 314
assembly() (*angr.analyses.reassembler.BasicBlock method*), 853
assembly() (*angr.analyses.reassembler.Data method*), 855
assembly() (*angr.analyses.reassembler.Instruction method*), 853
assembly() (*angr.analyses.reassembler.Operand method*), 852
assembly() (*angr.analyses.reassembler.Procedure method*), 854
assembly() (*angr.analyses.reassembler.Reassembler method*), 858
assign() (*angr.state_plugins.uc_manager.SimUCManager method*), 279
assign_labels() (*angr.analyses.reassembler.BasicBlock method*), 853
assign_labels() (*angr.analyses.reassembler.Data method*), 855
assign_labels() (*angr.analyses.reassembler.Instruction method*), 853
assign_labels() (*angr.analyses.reassembler.Procedure method*), 854
assign_unified_variable_names() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 566
assign_variable_names() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 566
Assignment (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressT attribute*), 662
AST (*angr.analyses.data_dep.dep_nodes.BaseDepNode property*), 868
AST (*class in angr.analyses.ddg*), 744
ast_graph (*angr.analyses.ddg.DDG property*), 747
ast_preserving_op() (*in module angr.state_plugins.sim_action_object*), 468
ast_stripping_decorator() (*in module angr.state_plugins.sim_action_object*), 468
ast_stripping_op() (*in module angr.state_plugins.sim_action_object*), 468
ast_weight() (*in module angr.exploration_techniques.suggestions*), 426
at_new_block() (*angr.analyses.reaching_definitions.ReachingDefinitions method*), 774
at_new_block() (*angr.knowledge_plugins.key_definitions.rd_model.Reach method*), 602
at_new_block() (*angr.knowledge_plugins.key_definitions.ReachingDefini method*), 571
at_new_stmt() (*angr.analyses.reaching_definitions.ReachingDefinitionsM method*), 774
at_new_stmt() (*angr.knowledge_plugins.key_definitions.rd_model.Reachi method*), 602
at_new_stmt() (*angr.knowledge_plugins.key_definitions.ReachingDefiniti method*), 571
atoi_dumps() (*angr.state_plugins.trace_additions.ChallRespInfo static method*), 276
atom (*angr.analyses.reaching_definitions.Definition attribute*), 770
atom (*angr.knowledge_plugins.key_definitions.Definition attribute*), 583
atom (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 590
Atom (*class in angr.analyses.reaching_definitions*), 766
Atom (*class in angr.knowledge_plugins.key_definitions.atoms*), 584
atom_hash (*angr.knowledge_plugins.variables.variable_access.VariableAc attribute*), 561
AtomKind (*class in angr.analyses.reaching_definitions*), 765
AtomKind (*class in angr.knowledge_plugins.key_definitions.atoms*), 584
available_flavors() (*angr.knowledge_plugins.structured_code.manager.StructuredCo method*), 570

B

b() (*angr.state_plugins.inspect.SimInspector method*), 234
back_edges() (*angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor method*), 624
back_edges() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 625
backend (*angr.angrdb.models.DbObject attribute*), 676
backend2name (*angr.angrdb.serializers.loader.LoaderSerializer attribute*), 682
backend_args (*angr.angrdb.models.DbObject attribute*), 676
backpatch() (*angr.analyses.typehoon.translator.TypeTranslator method*), 829
BackwardSlice (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
BackwardSlice (*class in angr.analyses.backward_slice*), 627
BadJumpkindNotification, 870
BadStatesDropper (*class in angr.distributed.worker*), 900
base (*angr.sim_type.SimType attribute*), 507
base (*angr.sim_type.SimTypeFunction attribute*), 513
base (*angr.sim_variable.SimStackVariable attribute*), 506
base (*angr.storage.memory_object.SimMemoryObject attribute*), 334
base_addr (*angr.sim_variable.SimStackVariable attribute*), 506
base_addr_available (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTargetBaseAddr property*), 662
base_address (*angr.storage.memory_mixins.regioned_memory.region_data.RegionDescriptor attribute*), 369
BaseDepNode (*class in angr.analyses.data_dep.dep_nodes*), 867
BaseGoal (*class in angr.exploration_techniques.director*), 418
BaseLabel (*class in angr.analyses.typehoon.typevars*), 832
BaseNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 685
BaseOptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 703
BasePointerSaveSimplifier (*class in angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier*), 705
BaseProxiNode (*class in angr.analyses.proximity_graph*), 863
BaseStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.base*), 724
basic_spec (*angr.procedures.stubs.format_parser.FormatParser attribute*), 474
basic_spec (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 475
BasicBlock (*class in angr.analyses.reassembler*), 853
BasicClarippyCooperation (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 362
bbl_addr (*angr.errors.SimError attribute*), 896
bbl_addr (*angr.knowledge_plugins.key_definitions.definition.DefinitionMa attribute*), 589
bbl_addrs (*angr.state_plugins.history.SimStateHistory property*), 269
bck_chunk() (*angr.PTChunk method*), 210
bck_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 301
bck_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 303
BEFORE_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 703
BehaviorFactory (*class in angr.engines.pcode.behavior*), 464
behaviors (*angr.engines.pcode.lifter.IRSB attribute*), 437
behaviors (*angr.engines.pcode.lifter.PcodeBasicBlockLifter attribute*), 441
binary (*angr.knowledge_plugins.functions.function.Function property*), 555
binary_insert() (*in module angr.utils.algo*), 885
binary_name (*angr.knowledge_plugins.functions.function.Function attribute*), 553
binary_name (*angr.knowledge_plugins.functions.soot_function.SootFunctio attribute*), 560
BinaryError, 850
BinaryOptimizer (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
BinaryOptimizer (*class in angr.analyses.binary_optimizer*), 860
BinDiff (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
BinDiff (*class in angr.analyses.bindiff*), 631
binop_operators (*angr.analyses.decompiler.decompilation_cache.Decom attribute*), 697
bitlen (*angr.analyses.stack_pointer_tracker.Register attribute*), 817
bits (*angr.analyses.propagator.values.Top property*), 752
bits (*angr.analyses.reaching_definitions.Atom property*), 766
bits (*angr.analyses.typehoon.lifter.TypeLifter attribute*), 828
bits (*angr.analyses.typehoon.typevars.HasField attribute*), 833
bits (*angr.analyses.variable_recovery.engine_base.RichR property*), 827

bits (*angr.engines.light.data.RegisterOffset property*), 750
bits (*angr.knowledge_plugins.key_definitions.atoms.Atom property*), 584
bits (*angr.sim_variable.SimMemoryVariable property*), 505
bits (*angr.sim_variable.SimRegisterVariable property*), 505
Blade (*class in angr*), 167
Blade (*class in angr.blade*), 870
blank_state() (*angr.factory.AngrObjectFactory method*), 217
blob (*angr.angrdb.models.DbCFGModel attribute*), 677
blob (*angr.angrdb.models.DbFunction attribute*), 677
blob (*angr.angrdb.models.DbVariableCollection attribute*), 678
blob (*angr.angrdb.models.DbXRefs attribute*), 679
block (*angr.analyses.decompiler.structured_codegen.c.CAILBlock attribute*), 727
Block (*angr.analyses.reaching_definitions.subject.SubjectType attribute*), 813
block (*angr.analyses.variable_recovery.engine_ail.SimEngineVRAIL attribute*), 826
block (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 546
block (*angr.knowledge_plugins.cfg.CFGNode property*), 527
Block (*class in angr*), 170
Block (*class in angr.block*), 221
block() (*angr.analyses.decompiler.clinic.Clinic method*), 694
block() (*angr.factory.AngrObjectFactory method*), 220
block() (*angr.sim_state.SimState method*), 226
block() (*angr.SimState method*), 183
block_addr (*angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionLocation attribute*), 715
block_addr (*angr.analyses.decompiler.region_simplifiers.expr_folding.StatementLocation attribute*), 715
block_addr (*angr.analyses.reaching_definitions.call_trace.CallSite attribute*), 789
block_addr (*angr.code_location.CodeLocation attribute*), 612
block_addr (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 610
block_addr (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 285
block_addr (*angr.state_plugins.unicorn_engine.StopDetails attribute*), 287
block_addrs (*angr.knowledge_plugins.functions.function.Function property*), 554
block_addrs_set (*angr.knowledge_plugins.functions.function.Function property*), 554
block_count (*angr.state_plugins.history.SimStateHistory property*), 269
block_id (*angr.analyses.cfg.cfg_emulated.CFGJob property*), 640
block_id (*angr.analyses.vfg.PendingJob attribute*), 840
block_id (*angr.analyses.vfg.VFGJob property*), 839
block_id (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
block_id (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
block_idx (*angr.analyses.decompiler.region_simplifiers.expr_folding.Expr attribute*), 715
block_idx (*angr.analyses.decompiler.region_simplifiers.expr_folding.State attribute*), 715
block_idx (*angr.analyses.decompiler.structured_codegen.c.CLabel attribute*), 731
block_idx (*angr.code_location.CodeLocation attribute*), 612
block_matches (*angr.analyses.bindiff.FunctionDiff property*), 630
BLOCK_MAX_SIZE (*angr.Block attribute*), 170
BLOCK_MAX_SIZE (*angr.block.Block attribute*), 221
block_similarity() (*angr.analyses.bindiff.FunctionDiff method*), 631
block_size (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 285
block_size (*angr.state_plugins.unicorn_engine.StopDetails attribute*), 287
block_trace_ind (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 285
BlockCache (*class in angr.analyses.decompiler.clinic*), 694
BlockDetails (*class in angr.state_plugins.unicorn_engine*), 285
BlockID (*class in angr.analyses.cfg.cfg_job_base*), 656
BlockLocator (*class in angr.analyses.decompiler.optimization_passes.ite_expr_converter*), 705
BlockNode (*class in angr.codenode*), 873
blocks (*angr.knowledge_plugins.functions.function.Function property*), 553
blocks_by_addr (*angr.analyses.decompiler.optimization_passes.optimizat property*), 704
blocks_by_addr_and_idx (*angr.analyses.decompiler.optimization_passes.optimization_pass property*), 704
blocks_probably_identical() (*angr.analyses.bindiff.FunctionDiff method*), 631
BLOCKS_THRESHOLD (*angr.analyses.binary_optimizer.BinaryOptimizer attribute*), 860
blocks_with_differing_constants (*angr.analyses.bindiff.BinDiff property*), 632
blocks_with_differing_constants (*angr.analyses.bindiff.FunctionDiff property*),

630
BlockSimplifier (*class in angr.analyses.decompiler.block_simplifier*), 692
BlockStart (*class in angr.analyses.disassembly*), 847
BlockWalker (*class in angr.analyses.decompiler.optimization_passes.constant_derefs*), 701
body (*angr.analyses.decompiler.structured_codegen.c.CDoWhileLoop attribute*), 727
body (*angr.analyses.decompiler.structured_codegen.c.CForLoop attribute*), 728
body (*angr.analyses.decompiler.structured_codegen.c.CWhileLoop attribute*), 727
booleanize() (*angr.engines.pcode.behavior.OpBehavior class method*), 446
both_iter (*angr.calling_conventions.ArgSession attribute*), 487
both_iter (*angr.calling_conventions.SimCC.ArgSession attribute*), 488
both_iter (*angr.SimCC.ArgSession attribute*), 186
Bottom (*class in angr.analyses.propagator.values*), 752
BottomType (*class in angr.analyses.stack_pointer_tracker*), 816
BottomType (*class in angr.analyses.typehoon.typeconsts*), 834
BoyScout (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 620
BoyScout (*class in angr.analyses.boyscout*), 632
BP (*class in angr*), 161
BP (*class in angr.state_plugins.inspect*), 233
BP_AFTER (*angr.state_plugins.inspect.SimInspector attribute*), 233
BP_BEFORE (*angr.state_plugins.inspect.SimInspector attribute*), 233
BP_BOTH (*angr.state_plugins.inspect.SimInspector attribute*), 233
bp_on_stack (*angr.knowledge_plugins.functions.function.Function attribute*), 553
bp_on_stack (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
branch() (*angr.analyses.ddg.LiveDefinitions method*), 745
BreakNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 687
bss_memory_read_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.BSSHook method*), 665
bss_memory_write_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.BSSHook method*), 665
BSSHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 665
Bucketizer (*class in angr.exploration_techniques*), 405
Bucketizer (*class in angr.exploration_techniques.bucketizer*), 426
build() (*angr.analyses.disassembly.Operand static method*), 848
bv_slice() (*in module angr.storage.memory_object*), 334
BVS() (*angr.state_plugins.solver.SimSolver method*), 256
byte_string (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
byte_string (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
bytes (*angr.Block property*), 170
bytes (*angr.block.Block property*), 221
bytes_at() (*angr.storage.memory_object.SimMemoryObject method*), 334
bytes_offset (*angr.engines.pcode.lifter.Lifter attribute*), 440
bytes_offset (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
bytestr (*angr.codenode.BlockNode attribute*), 874

C

c_args_as_atoms() (*angr.analyses.reaching_definitions.function_handler static method*), 805
c_args_as_atoms() (*angr.analyses.reaching_definitions.FunctionHandler static method*), 785
c_repr() (*angr.analyses.decompiler.structured_codegen.c.CConstruct method*), 725
c_repr() (*angr.analyses.typehoon.translator.SimTypeTempRef method*), 828
c_repr() (*angr.sim_type.SimStruct method*), 516
c_repr() (*angr.sim_type.SimType method*), 508
c_repr() (*angr.sim_type.SimTypeArray method*), 512
c_repr() (*angr.sim_type.SimTypeBottom method*), 509
c_repr() (*angr.sim_type.SimTypeFunction method*), 514
c_repr() (*angr.sim_type.SimTypeInt method*), 510
c_repr() (*angr.sim_type.SimTypePointer method*), 511
c_repr() (*angr.sim_type.SimTypeReference method*), 512
c_repr() (*angr.sim_type.SimUnion method*), 517
c_repr() (*angr.sim_type.TypeRef method*), 508
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CAILB method*), 727
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CAssig method*), 729
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CBina method*), 734
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CBrea method*), 729
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CCons method*), 736
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CCons method*), 725

c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.Continue method*), 729
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CDirectionExpression method*), 737
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CDoWhileLoop method*), 727
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFakeVariable method*), 732
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CForLoop method*), 728
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 726
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall method*), 730
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CGoto method*), 731
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIfBreak method*), 728
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIfElse method*), 728
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIntegralVariable method*), 733
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIfEl method*), 736
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CLabel method*), 731
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression method*), 737
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CRegister method*), 736
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CReturn method*), 730
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CStatement method*), 726
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CStructField method*), 732
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CSwitchCase method*), 729
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CTypeCast method*), 735
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp method*), 734
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CUnsupportedStatement method*), 731
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CVariableFinder method*), 733
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CVariableField method*), 733
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CWhileLoop method*), 727
c_return_as_atoms() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler static method*), 805
c_return_as_atoms() (*angr.analyses.reaching_definitions.FunctionHandler static method*), 785
cache_key (*angr.storage.memory_object.SimMemoryObject property*), 334
CALBlock (*class in angr.analyses.decompiler.structured_codegen.c*), 726
calc_size() (*angr.calling_conventions.AllocHelper class method*), 483
call() (*angr.simprocedure.SimProcedure method*), 473
call() (*angr.SimProcedure method*), 160
call() (*angr.state_plugins.callstack.CallStack method*), 266
call_c() (*angr.callable.Callable method*), 520
call_site_addr (*angr.analyses.cfg.cfg_fast.FunctionReturn attribute*), 648
call_stack (*angr.analyses.cfg.cfg_job_base.CFGJobBase property*), 657
call_stack (*angr.analyses.vfg.PendingJob attribute*), 841
call_stack_copy() (*angr.analyses.cfg.cfg_job_base.CFGJobBase method*), 657
call_state() (*angr.factory.AngrObjectFactory method*), 218
call_string (*angr.code_location.ExternalCodeLocation attribute*), 613
callable (*angr.knowledge_plugins.functions.function.Function property*), 558
Callable (*class in angr.callable*), 519
callable() (*angr.factory.AngrObjectFactory method*), 219
CallAnalysis (*class in angr.analyses.vfg*), 841
CALLEE_CLEANUP (*angr.calling_conventions.SimCC attribute*), 488
CALLEE_CLEANUP (*angr.calling_conventions.SimCCStdcall attribute*), 492
CALLEE_CLEANUP (*angr.SimCC attribute*), 185
callee_func (*angr.analyses.decompiler.structured_codegen.c.CFunctionC attribute*), 730
callee_func_addr (*angr.analyses.cfg.cfg_fast.FunctionReturn attribute*), 648
callee_func_addr (*angr.analyses.reaching_definitions.call_trace.CallSite attribute*), 789
callee_target (*angr.analyses.decompiler.structured_codegen.c.CFunctio attribute*), 730
CalleeCleanupFinder (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
CalleeCleanupFinder (*class in angr.analyses.callee_cleanup_finder*), 860
caller_func_addr (*angr.analyses.cfg.cfg_fast.FunctionReturn attribute*), 648
caller_func_addr (*angr.analyses.reaching_definitions.call_trace.CallSite attribute*), 789
CALLER_SAVED_REGS (*angr.calling_conventions.SimCC

attribute), 488
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCAMD64 attribute*), 495
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCARM attribute*), 495
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCARMHF attribute*), 496
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCCdecl attribute*), 491
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCN64 attribute*), 498
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCO32 attribute*), 497
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
CALLER_SAVED_REGS (*angr.SimCC attribute*), 185
caller_saved_regs_as_atoms() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler static method*), 805
caller_saved_regs_as_atoms() (*angr.analyses.reaching_definitions.FunctionHandler static method*), 785
caller_will_handle_single_ret (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 801
caller_will_handle_single_ret (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
CallFunctionGoal (*class in angr.exploration_techniques*), 401
CallFunctionGoal (*class in angr.exploration_techniques.director*), 419
callgraph (*angr.knowledge_base.knowledge_base.KnowledgeBase property*), 520
callgraph (*angr.KnowledgeBase property*), 211
CallGraphVisitor (*class in angr.analyses.forward_analysis.visitors.call_graph*), 622
calling_convention (*angr.knowledge_plugins.functions.function.Function attribute*), 553
calling_convention (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
CallingConvention (*angr.analyses.analysis.KnownAnalysisPlugin attribute*), 620
CallingConventionAnalysis (*class in angr.analyses.calling_convention*), 633
calloc() (*angr.SimHeapPTMalloc method*), 207
calloc() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 302
calloc() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 305
callout_sites (*angr.knowledge_plugins.functions.function.Function property*), 555
CallProxiNode (*class in angr.analyses.proximity_graph*), 864
CallSite (*angr.analyses.reaching_definitions.dep_graph.FunctionCallRela attribute*), 794
CallSite (*class in angr.analyses.reaching_definitions.call_trace*), 789
callsite_codeloc (*angr.analyses.reaching_definitions.function_handler. attribute*), 800
callsite_codeloc (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
callsite_repr() (*angr.analyses.cfg.cfg_job_base.BlockID method*), 656
callsite_repr() (*angr.analyses.cfg.cfg_job_base.FunctionKey method*), 657
CallSiteRact (*class in angr.analyses.calling_convention*), 632
CallSiteMaker (*class in angr.analyses.decompiler.callsite_maker*), 697
CallsitePrototypes (*class in angr.knowledge_plugins.callsite_prototypes*), 523
callsites (*angr.analyses.reaching_definitions.call_trace.CallTrace attribute*), 790
callsites_to() (*angr.analyses.reaching_definitions.reaching_definitions method*), 794
callsites_to() (*angr.analyses.reaching_definitions.ReachingDefinitions method*), 773
callstack (*angr.sim_state.SimState attribute*), 225
callstack (*angr.SimState attribute*), 181
CallStack (*class in angr.state_plugins.callstack*), 263
callstack_key (*angr.knowledge_plugins.cfg.cfg_node.CFGENode property*), 547
callstack_key (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 546
callstack_key (*angr.knowledge_plugins.cfg.CFGENode property*), 528
callstack_key (*angr.knowledge_plugins.cfg.CFGNode property*), 527
callstack_repr() (*angr.analyses.vfg.VFGJob method*), 839
CallStackAction (*class in angr.state_plugins.callstack*), 266
CallTrace (*angr.analyses.reaching_definitions.subject.SubjectType attribute*), 813
CallTrace (*class in angr.analyses.reaching_definitions.call_trace*), 790
CallTracingFilter (*class in angr.analyses.veritesting*), 838
can_call_same_name() (*angr.analyses.identifier.identify.Identifier method*), 836
canonical_names (*angr.sim_variable.SimVariable attribute*), 503
capstone (*angr.Block property*), 170

capstone (*angr.block.Block property*), 221
CapstoneBlock (*class in angr.block*), 220
CapstoneInsn (*class in angr.block*), 220
cardinality (*angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor.AbstractAddressDescriptor property*), 787
cardinality (*angr.storage.memory_mixins.regioned_memory.region_category_mixin.RegionCategoryMixin property*), 371
CArrayTypeLength (*class in angr.analyses.decompiler.structured_codegen.c*), 737
CascadingConditionNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 686
CascadingConditionTransformer (*class in angr.analyses.decompiler.region_simplifiers.cascading_conditions*), 714
CascadingIfsRemover (*class in angr.analyses.decompiler.region_simplifiers.cascading_ifs*), 714
Case (*class in angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier*), 706
case_addrs (*angr.analyses.decompiler.structuring.structurer_nodes.IncompleteSwitchCaseNode attribute*), 689
case_idx (*angr.analyses.decompiler.region_simplifiers.expression_folding.ExpressionCounter attribute*), 715
cases (*angr.analyses.decompiler.structured_codegen.c.CSwitchCase attribute*), 729
cases (*angr.analyses.decompiler.structuring.structurer_nodes.IncompleteSwitchCaseNode attribute*), 689
cases (*angr.analyses.decompiler.structuring.structurer_nodes.SwitchCaseNode attribute*), 688
cases_issubset() (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier static method*), 708
CAssignment (*class in angr.analyses.decompiler.structured_codegen.c*), 729
cast_primitive() (*angr.simos.javavm.SimJavaVM static method*), 881
cast_to_mem() (*angr.analyses.data_dep.dep_nodes.MemDepNode class method*), 868
CastType (*angr.state_plugins.solver.SimSolver attribute*), 259
CatchDesync (*angr.exploration_techniques.tracer.TracingMode attribute*), 414
category (*angr.sim_variable.SimVariable attribute*), 503
category (*angr.SimFile property*), 190
category (*angr.storage.file.SimFile property*), 317
category (*angr.storage.memory_mixins.MemoryMixin property*), 336
category (*angr.storage.memory_mixins.regioned_memory.region_category_mixin.RegionCategoryMixin property*), 370
CBinaryOp (*class in angr.analyses.decompiler.structured_codegen.c*), 734
CBreak (*class in angr.analyses.decompiler.structured_codegen.c*), 728
cc (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 800
cc (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
cc (*angr.analyses.reaching_definitions.subject.Subject property*), 813
cc (*angr.calling_conventions.ArgSession attribute*), 487
cc (*angr.calling_conventions.SimCC.ArgSession attribute*), 488
cc (*angr.calling_conventions.UsercallArgSession attribute*), 487
cc (*angr.procedures.stubs.format_parser.FormatParser attribute*), 475
cc (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 475
cc (*angr.sim_procedure.SimProcedure attribute*), 471
cc (*angr.SimCC.ArgSession attribute*), 185
cc() (*angr.factory.AngrObjectFactory method*), 219
CCallRewriterBase (*class in angr.analyses.decompiler.structured_codegen.c_rewriters.rewriter_base*), 693
CClosingObjectLocation (*class in angr.analyses.decompiler.structured_codegen.c*), 737
CConstant (*class in angr.analyses.decompiler.structured_codegen.c*), 725
CConstruct (*class in angr.analyses.decompiler.structured_codegen.c*), 725
CContinue (*class in angr.analyses.decompiler.structured_codegen.c*), 729
CDG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
CDG (*class in angr.analyses.cdg*), 673
CDirtyExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 737
CDoWhileLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 727
ceiling_addr() (*angr.analyses.cfg.cfb.CFBlanket method*), 638
ceiling_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 550
ceiling_func() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 551
ceiling_item() (*angr.analyses.cfg.cfb.CFBlanket method*), 638
ceiling_items() (*angr.analyses.cfg.cfb.CFBlanket method*), 638
CExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 726
CFakeVariable (*class in angr.analyses.decompiler.structured_codegen.c*),

732
CFB (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
CFBlanket (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
CFBlanket (*class in angr.analyses.cfg.cfb*), 637
CFBlanketView (*class in angr.analyses.cfg.cfb*), 637
CFG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
CFG (*class in angr.analyses.cfg.cfg*), 638
cfg_cache (*angr.analyses.veritesting.CallTracingFilter attribute*), 838
cfg_cache (*angr.analyses.veritesting.Veritesting attribute*), 838
cfg_jumpkind_from_pb() (*in module angr.utils.enums_conv*), 886
cfg_jumpkind_to_pb() (*in module angr.utils.enums_conv*), 886
CFGArchOptions (*class in angr.analyses.cfg.cfg_arch_options*), 656
CFGBase (*class in angr.analyses.cfg.cfg_base*), 645
CFGEmulated (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
CFGEmulated (*class in angr.analyses.cfg.cfg_emulated*), 641
CFGENode (*class in angr.knowledge_plugins.cfg*), 527
CFGENode (*class in angr.knowledge_plugins.cfg.cfg_node*), 546
CFGFast (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
CFGFast (*class in angr.analyses.cfg.cfg_fast*), 652
CFGFastSoot (*class in angr.analyses.cfg.cfg_fast_soot*), 668
CFGJob (*class in angr.analyses.cfg.cfg_emulated*), 640
CFGJob (*class in angr.analyses.cfg.cfg_fast*), 651
CFGJobBase (*class in angr.analyses.cfg.cfg_job_base*), 657
CFGJobType (*class in angr.analyses.cfg.cfg_fast*), 651
CFGManager (*class in angr.knowledge_plugins.cfg*), 535
CFGManager (*class in angr.knowledge_plugins.cfg.cfg_manager*), 544
CFGModel (*class in angr.knowledge_plugins.cfg*), 529
CFGModel (*class in angr.knowledge_plugins.cfg.cfg_model*), 536
CFGModelSerializer (*class in angr.angrdb.serializers.cfg_model*), 679
CFGNode (*class in angr.knowledge_plugins.cfg*), 525
CFGNode (*class in angr.knowledge_plugins.cfg.cfg_node*), 544
CFGNodeCreationFailure (*class in angr.knowledge_plugins.cfg.cfg_node*), 544
cfgs (*angr.angrdb.models.DbKnowledgeBase attribute*), 676
cfgs (*angr.knowledge_base.knowledge_base.KnowledgeBase attribute*), 520
cfgs (*angr.KnowledgeBase attribute*), 211
CFGSliceToSink (*class in angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink*), 814
CForLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 727
CFunction (*class in angr.analyses.decompiler.structured_codegen.c*), 725
CFunctionCall (*class in angr.analyses.decompiler.structured_codegen.c*), 729
CGoto (*class in angr.analyses.decompiler.structured_codegen.c*), 730
ChainMapCOW (*class in angr.utils.cowdict*), 885
ChallRespInfo (*class in angr.state_plugins.trace_additions*), 274
changed_bytes() (*angr.storage.memory_mixins.paged_memory.paged_m method*), 354
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.his method*), 361
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.lis method*), 363
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.mv method*), 350
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.ult method*), 365
changed_bytes() (*angr.storage.memory_mixins.slotted_memory.SlottedM method*), 375
changed_pages() (*angr.storage.memory_mixins.paged_memory.paged_m method*), 355
chdir() (*angr.state_plugins.filesystem.SimFilesystem method*), 250
check() (*angr.BP method*), 161
check() (*angr.exploration_techniques.CallFunctionGoal method*), 401
check() (*angr.exploration_techniques.director.BaseGoal method*), 418
check() (*angr.exploration_techniques.director.CallFunctionGoal method*), 419
check() (*angr.exploration_techniques.director.ExecuteAddressGoal method*), 418
check() (*angr.exploration_techniques.ExecuteAddressGoal method*), 400
check() (*angr.state_plugins.inspect.BP method*), 233
check_concrete_target_methods() (*angr.engines.concrete.SimEngineConcrete static method*), 433
check_offset() (*angr.calling_conventions.SimRegArg method*), 485
check_state() (*angr.exploration_techniques.CallFunctionGoal method*), 401
check_state() (*angr.exploration_techniques.director.BaseGoal method*), 418

check_state() (*angr.exploration_techniques.director.CallFunctionGoal method*), 419
check_state() (*angr.exploration_techniques.director.ExecuteAddressGoal method*), 419
check_state() (*angr.exploration_techniques.ExecuteAddressGoal method*), 400
check_tests() (*angr.analyses.identifier.identify.Identifier method*), 836
check_value_get() (*angr.calling_conventions.SimFunctionArgument method*), 484
check_value_set() (*angr.calling_conventions.SimFunctionArgument method*), 484
Chunk (*class in angr.state_plugins.heap.heap_freelist*), 300
chunk_from_mem() (*angr.SimHeapPTMalloc method*), 207
chunk_from_mem() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 301
chunk_from_mem() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 305
chunks() (*angr.SimHeapPTMalloc method*), 206
chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 301
chunks() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 304
CIfBreak (*class in angr.analyses.decompiler.structured_codegen.c*), 728
CIfElse (*class in angr.analyses.decompiler.structured_codegen.c*), 728
CIndexedVariable (*class in angr.analyses.decompiler.structured_codegen.c*), 733
CITE (*class in angr.analyses.decompiler.structured_codegen.c*), 736
CLabel (*class in angr.analyses.decompiler.structured_codegen.c*), 731
claripy_ast_from_ail_condition() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 696
claripy_ast_to_sympy_expr() (*angr.analyses.decompiler.condition_processor.ConditionProcessor static method*), 696
ClassIdentifier (*class in angr.analyses.class_identifier*), 846
clean() (*angr.utils.cowdict.ChainMapCOW method*), 885
clean() (*angr.utils.cowdict.DefaultChainMapCOW method*), 885
cleanup() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 649
cleanup() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 738
clear() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 695
clear() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 551
clear() (*angr.state_plugins.log.SimStateLog method*), 263
clear() (*angr.state_plugins.scratch.SimStateScratch method*), 282
clear() (*angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor.AbstractAddressDescriptor method*), 371
clear_cache() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 443
clear_local_references() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 296
clear_page_cache() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
clear_region_for_reflow() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 542
clear_region_for_reflow() (*angr.knowledge_plugins.cfg.CFGModel method*), 535
clear_updated_functions() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 649
ClemoryBackerMixin (*class in angr.storage.memory_mixins.paged_memory.page_backer_mixins*), 357
Clinic (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 620
clinic (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 697
Clinic (*class in angr.analyses.decompiler.clinic*), 694
CLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 727
close() (*angr.state_plugins.posix.SimSystemPosix method*), 246
close() (*angr.vaults.Vault static method*), 617
close() (*angr.vaults.VaultShelf method*), 618
closed_fds (*angr.state_plugins.posix.SimSystemPosix property*), 245
closest_common_ancestor() (*angr.state_plugins.history.SimStateHistory method*), 270
Closure (*class in angr.utils.mp*), 893
CmpOp (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier*), 718
CMultiStatementExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 736
code_constants (*angr.knowledge_plugins.functions.function.Function property*), 555
codegen (*angr.analyses.decompiler.decompilation_cache.DecompilationCache attribute*), 697
codegen (*angr.analyses.decompiler.structured_codegen.c.CConstruct*

attribute), 725
codeloc (*angr.analyses.reaching_definitions.Definition attribute*), 770
codeloc (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState attribute*), 807
codeloc (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 777
codeloc (*angr.knowledge_plugins.key_definitions.Definition attribute*), 583
codeloc (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 590
codeloc_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState attribute*), 807
codeloc_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 777
CodeLocation (*class in angr.code_location*), 612
codenode (*angr.Block property*), 170
codenode (*angr.block.Block property*), 221
codenode (*angr.block.SootBlock property*), 222
CodeNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 686
CodeNode (*class in angr.codenode*), 873
CodeReference (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 542
CodeReference (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
CodeTagging (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
CodeTagging (*class in angr.analyses.code_tagging*), 674
CodeTags (*class in angr.analyses.code_tagging*), 673
collapse() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecasts class method*), 740
collapsed (*angr.analyses.decompiler.structured_codegen.c.CExpression attribute*), 726
collapsed (*angr.analyses.decompiler.structured_codegen.c.CImplicitCast attribute*), 733
collapsed (*angr.analyses.decompiler.structured_codegen.c.CVariable attribute*), 733
collect_data_refs (*angr.engines.pcode.lifter.Lifter attribute*), 440
collect_data_refs (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
color() (*angr.analyses.disassembly.DisassemblyPiece static method*), 846
comment (*angr.angrdb.models.DbComment attribute*), 679
Comment (*class in angr.analyses.disassembly*), 849
comments (*angr.angrdb.models.DbKnowledgeBase attribute*), 676
Comments (*class in angr.knowledge_plugins.comments*), 549
CommentsSerializer (*class in angr.angrdb.serializers.comments*), 680
commit() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 608
common_type (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp attribute*), 734
compare_function_group() (*angr.analyses.congruency_check.CongruencyCheck method*), 859
compare_paths() (*angr.analyses.congruency_check.CongruencyCheck method*), 859
compare_statement_dict() (*in module angr.analyses.bindiff*), 629
compare_states() (*angr.analyses.congruency_check.CongruencyCheck method*), 859
complement() (*angr.sim_variable.SimVariableSet method*), 507
complete() (*angr.exploration_techniques.ExplorationTechnique method*), 391
complete() (*angr.exploration_techniques.Explorer method*), 397
complete() (*angr.exploration_techniques.explorer.Explorer method*), 409
complete() (*angr.exploration_techniques.Symbion method*), 405
complete() (*angr.exploration_techniques.symbion.Symbion method*), 425
complete() (*angr.exploration_techniques.Tracer method*), 395
complete() (*angr.exploration_techniques.tracer.Tracer method*), 415
complete() (*angr.ExplorationTechnique method*), 179
complete() (*angr.sim_manager.SimulationManager method*), 385
complete() (*angr.SimulationManager method*), 174
COMPLETE_SCANNING (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 651
CompleteCallingConventions (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 620
CompleteCallingConventionsAnalysis (*class in angr.analyses.complete_calling_conventions*), 634
compute() (*angr.state_plugins.trace_additions.FormatInfo method*), 273
compute() (*angr.state_plugins.trace_additions.FormatInfoDontConstrain method*), 274
compute() (*angr.state_plugins.trace_additions.FormatInfoIntToStr method*), 274
compute() (*angr.state_plugins.trace_additions.FormatInfoStrToInt method*), 274
compute_common_type() (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp static method*), 734
compute_dominance_frontier() (*in module angr.utils.graph*), 887
concat() (*angr.storage.memory_mixins.paged_memory.pages.multi_value.

method), 351
concrete (*angr.state_plugins.debug_variables.SimDebugVariable property*), 308
concrete (*angr.state_plugins.view.SimMemView property*), 314
Concrete (*class in angr.state_plugins.concrete*), 292
concrete_load() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 346
concrete_load() (*angr.storage.memory_mixins.MemoryMixin method*), 337
concrete_load() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 354
concrete_load() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValuedMemory method*), 365
concrete_path_bool() (*in module angr.state_plugins.solver*), 254
concrete_path_list() (*in module angr.state_plugins.solver*), 254
concrete_path_not_bool() (*in module angr.state_plugins.solver*), 254
concrete_path_scalar() (*in module angr.state_plugins.solver*), 254
concrete_path_tuple() (*in module angr.state_plugins.solver*), 254
concrete_states (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateMixin property*), 825
ConcreteBackerMixin (*class in angr.storage.memory_mixins.paged_memory.pages.multi_values*), 357
concretize() (*angr.concretization_strategies.SimConcretizationStrategy method*), 335
concretize() (*angr.SimFile method*), 190
concretize() (*angr.SimFileBase method*), 189
concretize() (*angr.SimFileDescriptor method*), 198
concretize() (*angr.SimFileDescriptorDuplex method*), 201
concretize() (*angr.SimPackets method*), 192
concretize() (*angr.storage.file.SimFile method*), 317
concretize() (*angr.storage.file.SimFileBase method*), 316
concretize() (*angr.storage.file.SimFileDescriptor method*), 327
concretize() (*angr.storage.file.SimFileDescriptorBase method*), 326
concretize() (*angr.storage.file.SimFileDescriptorDuplex method*), 330
concretize() (*angr.storage.file.SimPackets method*), 321
concretize() (*angr.storage.file.SimPacketsSlots method*), 332
concretize() (*in module angr.state_plugins.heap.utils*), 306
concretize_load_idx() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin.JavaVmMemoryMixin method*), 377
concretize_read_addr() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 346
concretize_store_idx() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin.JavaVmMemoryMixin method*), 377
concretize_write_addr() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 345
condition (*angr.analyses.decompiler.structured_codegen.c.CConditionalExpression attribute*), 736
condition (*angr.analyses.decompiler.structured_codegen.c.CDoWhileLoop attribute*), 727
condition (*angr.analyses.decompiler.structured_codegen.c.CForLoop attribute*), 727
condition (*angr.analyses.decompiler.structured_codegen.c.CIfBreak attribute*), 728
condition (*angr.analyses.decompiler.structured_codegen.c.CWhileLoop attribute*), 727
condition (*angr.analyses.decompiler.structuring.structurer_nodes.ConditionNode attribute*), 688
condition (*angr.analyses.decompiler.structuring.structurer_nodes.ConditionNode attribute*), 686
condition (*angr.analyses.decompiler.structuring.structurer_nodes.LoopNode attribute*), 687
Condition (*class in angr.analyses.loop_analysis*), 837
condition_and_nodes (*angr.analyses.decompiler.structured_codegen.c.CIfElse attribute*), 728
condition_and_nodes (*angr.analyses.decompiler.structuring.structurer_nodes.CascadingConditionNode attribute*), 687
condition_to_lambda() (*in module angr.exploration_techniques.common*), 424
CONDITIONAL (*angr.state_plugins.sim_action.SimActionExit attribute*), 466
ConditionalBreakLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 715
ConditionalBreakNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 688
ConditionalMixin (*class in angr.storage.memory_mixins.conditional_store_mixin*), 346
ConditionalRegion (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier*), 719
ConditionLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 715
ConditionNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*),

686
ConditionProcessor (*class in angr.analyses.decompiler.condition_processor*), 695
configuration (*angr.angrdb.models.DbStructuredCode attribute*), 678
configure_project() (*angr.SimOS method*), 168
configure_project() (*angr.simos.linux.SimLinux method*), 877
configure_project() (*angr.simos.simos.SimOS method*), 875
configure_project() (*angr.simos.userland.SimUserland method*), 878
configure_project() (*angr.simos.windows.SimWindows method*), 879
confirmed (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge attribute*), 650
CongruencyCheck (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
CongruencyCheck (*class in angr.analyses.congruency_check*), 858
connect() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 608
connected (*angr.knowledge_plugins.sync.sync_controller.SyncController property*), 608
const_formats (*angr.angrdb.models.DbStructuredCode attribute*), 678
CONST_TYPES (*angr.engines.light.data.ArithmeticExpression attribute*), 750
const_val (*angr.knowledge_plugins.key_definitions.definitions.atom.Atom attribute*), 589
Constant (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 867
CONSTANT (*angr.analyses.reaching_definitions.AtomKind attribute*), 766
CONSTANT (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 584
Constant (*class in angr.analyses.stack_pointer_tracker*), 816
constant_after() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 818
constant_after_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 818
constant_before() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 818
constant_before_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 818
constant_jump_targets (*angr.engines.pcode.lifter.IRSB property*), 439
constant_jump_targets_and_jumpkinds (*angr.engines.pcode.lifter.IRSB property*), 439
ConstantChange (*class in angr.analyses.bindiff*), 629
ConstantDepNode (*class in angr.analyses.data_dep.dep_nodes*), 868
ConstantDereferencesSimplifier (*class in angr.analyses.decompiler.optimization_passes.const_derefs*), 701
ConstantOperand (*class in angr.analyses.disassembly*), 849
ConstantPropagation (*class in angr.analyses.binary_optimizer*), 859
ConstantResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.const_resolver*), 666
constants (*angr.engines.pcode.lifter.IRSB property*), 439
ConstantSrc (*class in angr.analyses.reaching_definitions*), 769
ConstantSrc (*class in angr.knowledge_plugins.key_definitions.atoms*), 586
ConstantValueManager (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 662
constrain_all_zero() (*angr.analyses.identifier.identify.Identifier static method*), 836
constraint_hook() (*in module angr.state_plugins.trace_additions*), 274
constraints (*angr.state_plugins.solver.SimSolver property*), 257
constraints_since() (*angr.state_plugins.history.SimStateHistory method*), 270
construct() (*angr.knowledge_plugins.key_definitions.definition.Definition static method*), 589
ContainerNode (*class in angr.utils.graph*), 888
contains() (*angr.knowledge_plugins.debug_variables.DebugVariable method*), 568
contains_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 551
contains_atom() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 796
contains_memory_variable() (*angr.sim_variable.SimVariableSet method*), 507
contains_register_variable() (*angr.sim_variable.SimVariableSet method*), 507
content (*angr.analyses.reaching_definitions.subject.Subject property*), 813
content (*angr.analyses.reassembler.Data property*), 855

content (*angr.angrdb.models.DbObject attribute*), 676
content (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 543
content (*angr.knowledge_plugins.cfg.MemoryData attribute*), 525
content_gen() (*angr.storage.memory_mixins.paged_memory.page_backers.MVListPageBacker method*), 350
context (*angr.code_location.CodeLocation attribute*), 612
context (*angr.engines.pcode.lifter.PcodeBasicBlockLifter attribute*), 441
context_sensitivity_level (*angr.analyses.cfg.cfg_base.CFGBase property*), 646
context_sensitivity_level (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 644
continue_addr (*angr.analyses.decompiler.structuring.structurer_nodes.LoopNode property*), 687
ContinueNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 687
ContinueScanningNotification, 647
ConvenientMappingsMixin (*class in angr.storage.memory_mixins.convenient_mappings_mixin*), 348
convert_claripy_bool_ast() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 696
convert_claripy_bool_ast_core() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 696
convert_cppproto_to_py() (*in module angr.utils.library*), 891
convert_cproto_to_py() (*in module angr.utils.library*), 891
convert_variable_list() (*angr.knowledge_plugins.variables.variable_manager.VariableManager static method*), 568
ConvertTo (*class in angr.analyses.typehoon.typevars*), 832
CooperationBase (*class in angr.storage.memory_mixins.paged_memory.page_operation*), 361
copy() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
copy() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 642
copy() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 656
copy() (*angr.analyses.cfg.segment_list.Segment method*), 671
copy() (*angr.analyses.cfg.segment_list.SegmentList method*), 672
copy() (*angr.analyses.ddg.LiveDefinitions method*), 745
copy() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 700
copy() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifyAPIPass method*), 709
copy() (*angr.analyses.decompiler.region_simplifiers.expr_folding.StatementCounter method*), 715
copy() (*angr.analyses.decompiler.structuring.structurer_nodes.CodeNode method*), 686
copy() (*angr.analyses.decompiler.structuring.structurer_nodes.LoopNode method*), 687
copy() (*angr.analyses.decompiler.structuring.structurer_nodes.MultiNode method*), 685
copy() (*angr.analyses.decompiler.structuring.structurer_nodes.SequenceNode method*), 686
copy() (*angr.analyses.loop_analysis.LoopAnalysisState method*), 837
copy() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 790
copy() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 758
copy() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 809
copy() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 775
copy() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 779
copy() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 818
copy() (*angr.analyses.typehoon.typevars.TypeVariables method*), 831
copy() (*angr.analyses.variable_recovery.variable_recovery.VariableRecoveryState method*), 825
copy() (*angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFastState method*), 824
copy() (*angr.analyses.vfg.VFG method*), 844
copy() (*angr.concretization_strategies.norepeats.SimConcretizationStrategyNorepeats method*), 379
copy() (*angr.concretization_strategies.norepeats_range.SimConcretizationStrategyNorepeatsRange method*), 381
copy() (*angr.concretization_strategies.SimConcretizationStrategy method*), 335
copy() (*angr.engines.pcode.lifter.IRSB method*), 438
copy() (*angr.keyed_region.KeyedRegion method*), 614
copy() (*angr.keyed_region.RegionObject method*), 614
copy() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 523
copy() (*angr.knowledge_plugins.cfg.cfg_manager.CFGManager method*), 544
copy() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 536
copy() (*angr.knowledge_plugins.cfg.cfg_node.CFGENode method*), 547
copy() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode*

method), 546
copy() (angr.knowledge_plugins.cfg.CFGENode method), 528
copy() (angr.knowledge_plugins.cfg.CFGManager method), 536
copy() (angr.knowledge_plugins.cfg.CFGModel method), 530
copy() (angr.knowledge_plugins.cfg.CFGNode method), 527
copy() (angr.knowledge_plugins.cfg.memory_data.MemoryData method), 543
copy() (angr.knowledge_plugins.cfg.MemoryData method), 525
copy() (angr.knowledge_plugins.comments.Comments method), 549
copy() (angr.knowledge_plugins.data.Data method), 549
copy() (angr.knowledge_plugins.functions.function.Function method), 559
copy() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 551
copy() (angr.knowledge_plugins.indirect_jumps.IndirectJump method), 549
copy() (angr.knowledge_plugins.key_definitions.key_definition_manager.KeyDefinitionManager method), 592
copy() (angr.knowledge_plugins.key_definitions.KeyDefinitionManager method), 572
copy() (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method), 594
copy() (angr.knowledge_plugins.key_definitions.LiveDefinitions method), 574
copy() (angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method), 603
copy() (angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method), 571
copy() (angr.knowledge_plugins.key_definitions.Uses method), 582
copy() (angr.knowledge_plugins.key_definitions.uses.Uses method), 607
copy() (angr.knowledge_plugins.labels.Labels method), 550
copy() (angr.knowledge_plugins.patches.PatchManager method), 522
copy() (angr.knowledge_plugins.plugin.KnowledgeBasePlugin method), 523
copy() (angr.knowledge_plugins.structured_code.manager.StructuredCodeManager method), 570
copy() (angr.knowledge_plugins.sync.sync_controller.SyncController method), 608
copy() (angr.knowledge_plugins.types.TypesStore method), 549
copy() (angr.knowledge_plugins.variables.variable_manager.VariableManager method), 567
copy() (angr.knowledge_plugins.xrefs.xref.XRef method), 611
copy() (angr.knowledge_plugins.xrefs.xref_manager.XRefManager method), 611
copy() (angr.misc.plugins.PluginPreset method), 224
copy() (angr.procedures.definitions.SimLibrary method), 476
copy() (angr.procedures.definitions.SimSyscallLibrary method), 480
copy() (angr.sim_manager.SimulationManager method), 384
copy() (angr.sim_state.SimState method), 227
copy() (angr.sim_state_options.SimStateOptions method), 231
copy() (angr.sim_type.SimCppClass method), 517
copy() (angr.sim_type.SimCppClassValue method), 518
copy() (angr.sim_type.SimStruct method), 516
copy() (angr.sim_type.SimStructValue method), 516
copy() (angr.sim_type.SimType method), 508
copy() (angr.sim_type.SimTypeArray method), 512
copy() (angr.sim_type.SimTypeBottom method), 509
copy() (angr.sim_type.SimTypeChar method), 510
copy() (angr.sim_type.SimTypeCppFunction method), 514
copy() (angr.sim_type.SimTypeDouble method), 515
copy() (angr.sim_type.SimTypeFd method), 511
copy() (angr.sim_type.SimTypeFloat method), 515
copy() (angr.sim_type.SimTypeFunction method), 514
copy() (angr.sim_type.SimTypeInt method), 510
copy() (angr.sim_type.SimTypeLength method), 515
copy() (angr.sim_type.SimTypeNum method), 510
copy() (angr.sim_type.SimTypeNumOffset method), 518
copy() (angr.sim_type.SimTypePointer method), 511
copy() (angr.sim_type.SimTypeReference method), 512
copy() (angr.sim_type.SimTypeReg method), 509
copy() (angr.sim_type.SimTypeString method), 513
copy() (angr.sim_type.SimTypeTop method), 509
copy() (angr.sim_type.SimTypeWideChar method), 511
copy() (angr.sim_type.SimTypeWString method), 513
copy() (angr.sim_type.SimUnion method), 517
copy() (angr.sim_type.SimUnionValue method), 517
copy() (angr.sim_type.TypeRef method), 508
copy() (angr.sim_variable.SimConstantVariable method), 503
copy() (angr.sim_variable.SimMemoryVariable method), 505
copy() (angr.sim_variable.SimRegisterVariable method), 505
copy() (angr.sim_variable.SimStackVariable method), 506
copy() (angr.sim_variable.SimTemporaryVariable method), 504
copy() (angr.sim_variable.SimVariable method), 503
copy() (angr.sim_variable.SimVariableSet method), 507
copy() (angr.SimFile method), 191

copy() (*angr.SimFileBase method*), 189
copy() (*angr.SimFileDescriptor method*), 199
copy() (*angr.SimFileDescriptorDuplex method*), 201
copy() (*angr.SimFileStream method*), 195
copy() (*angr.SimHeapBrk method*), 204
copy() (*angr.SimHeapPTMalloc method*), 206
copy() (*angr.SimHostFilesystem method*), 204
copy() (*angr.SimPackets method*), 193
copy() (*angr.SimPacketsStream method*), 197
copy() (*angr.SimState method*), 183
copy() (*angr.SimStatePlugin method*), 161
copy() (*angr.SimulationManager method*), 173
copy() (*angr.state_plugins.callstack.CallStack method*), 264
copy() (*angr.state_plugins.cgc.SimStateCGC method*), 272
copy() (*angr.state_plugins.concrete.Concrete method*), 292
copy() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 252
copy() (*angr.state_plugins.filesystem.SimFilesystem method*), 249
copy() (*angr.state_plugins.filesystem.SimHostFilesystem method*), 254
copy() (*angr.state_plugins.gdb.GDB method*), 271
copy() (*angr.state_plugins.globals.SimStateGlobals method*), 279
copy() (*angr.state_plugins.heap.heap_base.SimHeapBase method*), 298
copy() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 298
copy() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 304
copy() (*angr.state_plugins.history.SimStateHistory method*), 268
copy() (*angr.state_plugins.inspect.SimInspector method*), 234
copy() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 294
copy() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 296
copy() (*angr.state_plugins.libc.SimStateLibc method*), 238
copy() (*angr.state_plugins.light_registers.SimLightRegisters method*), 267
copy() (*angr.state_plugins.log.SimStateLog method*), 262
copy() (*angr.state_plugins.loop_data.SimStateLoopData method*), 292
copy() (*angr.state_plugins.plugin.SimStatePlugin method*), 232
copy() (*angr.state_plugins.posix.PosixDevFS method*), 241
copy() (*angr.state_plugins.posix.PosixProcFS method*), 243
copy() (*angr.state_plugins.posix.SimSystemPosix method*), 245
copy() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 283
copy() (*angr.state_plugins.scratch.SimStateScratch method*), 280
copy() (*angr.state_plugins.sim_action.SimAction method*), 466
copy() (*angr.state_plugins.sim_action_object.SimActionObject method*), 468
copy() (*angr.state_plugins.solver.SimSolver method*), 256
copy() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 307
copy() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 274
copy() (*angr.state_plugins.trace_additions.FormatInfo method*), 273
copy() (*angr.state_plugins.trace_additions.FormatInfoDontConstrain method*), 274
copy() (*angr.state_plugins.trace_additions.FormatInfoIntToStr method*), 274
copy() (*angr.state_plugins.trace_additions.FormatInfoStrToInt method*), 274
copy() (*angr.state_plugins.trace_additions.ZenPlugin method*), 276
copy() (*angr.state_plugins.uc_manager.SimUCManager method*), 279
copy() (*angr.state_plugins.unicorn_engine.Unicorn method*), 289
copy() (*angr.state_plugins.view.SimMemView method*), 313
copy() (*angr.state_plugins.view.SimRegNameView method*), 309
copy() (*angr.storage.file.SimFile method*), 318
copy() (*angr.storage.file.SimFileBase method*), 316
copy() (*angr.storage.file.SimFileDescriptor method*), 328
copy() (*angr.storage.file.SimFileDescriptorDuplex method*), 330
copy() (*angr.storage.file.SimFileStream method*), 320
copy() (*angr.storage.file.SimPackets method*), 322
copy() (*angr.storage.file.SimPacketsSlots method*), 332
copy() (*angr.storage.file.SimPacketsStream method*), 324
copy() (*angr.storage.memory_mixins.address_concretization_mixin.Addre method*), 344
copy() (*angr.storage.memory_mixins.convenient_mappings_mixin.Conven method*), 348
copy() (*angr.storage.memory_mixins.default_filler_mixin.ExplicitFillerMi method*), 340
copy() (*angr.storage.memory_mixins.default_filler_mixin.SpecialFillerMix method*), 340

copy() (*angr.storage.memory_mixins.javavm_memory.JavaVmMemoryMixin method*), 377
copy() (*angr.storage.memory_mixins.keyvalue_memory.KeyValueMemoryMixin method*), 375
copy() (*angr.storage.memory_mixins.label_merger_mixin.LabelMergerMixin method*), 347
copy() (*angr.storage.memory_mixins.MemoryMixin method*), 336
copy() (*angr.storage.memory_mixins.multi_value_merger_mixin.MultiValueMergerMixin method*), 352
copy() (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.DictBackerMixin method*), 357
copy() (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.MVListPageBackerMixin method*), 358
copy() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 356
copy() (*angr.storage.memory_mixins.paged_memory.pages.cooperation.SharedPagesMixin method*), 353
copy() (*angr.storage.memory_mixins.paged_memory.pages.history_tracking.HistoryTrackingMixin method*), 361
copy() (*angr.storage.memory_mixins.paged_memory.pages.list_page.ListPage method*), 362
copy() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 348
copy() (*angr.storage.memory_mixins.paged_memory.pages.pages.PageType method*), 360
copy() (*angr.storage.memory_mixins.paged_memory.pages.refcount_mixin.RefCountMixin method*), 359
copy() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 364
copy() (*angr.storage.memory_mixins.paged_memory.stack_allocation_mixin.StackAllocationMixin method*), 358
copy() (*angr.storage.memory_mixins.regioned_memory.region_data.RegionObject method*), 369
copy() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.RegionMetaMixin method*), 371
copy() (*angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin.RegionedAddressConcretizationMixin method*), 373
copy() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 366
copy() (*angr.storage.memory_mixins.size_resolution_mixin.SizeConcretizationMixin method*), 343
copy() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method*), 374
copy() (*angr.storage.memory_mixins.top_merger_mixin.TopMergerMixin method*), 352
copy() (*angr.storage.memory_mixins.underconstrained_mixin.UnderconstrainedMixin method*), 341
copy() (*angr.storage.pcap.PCAP method*), 335
copy() (*angr.utils.cowdict.ChainMapCOW method*), 885
copy_contents() (*angr.storage.memory_mixins.MemoryMixin method*), 338
copy_contents() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 355
copy_contents() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 347
copy_graph() (*angr.analyses.clinic.Clinic method*), 694
CouldNotResolveException, 818
count (*angr.state_plugins.unicorn_engine.TRANSMIT_RECORD attribute*), 284
count() (*angr.exploration_techniques.spiller.PickledStatesDb method*), 411
count() (*angr.state_plugins.history.TreeIter method*), 770
count() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MVListPageBackerMixin method*), 351
cprotos2py() (*in module angr.utils.library*), 892
crash_window() (*angr.exploration_techniques.Tracer class method*), 396
crash_window() (*angr.exploration_techniques.tracer.Tracer class method*), 416
create_extract_vtables() (*angr.analyses.vtable.VtableFinder method*), 845
create_jump_target_var() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 696
create_new_reference() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 796
creation_failed (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 547
creation_failed (*angr.knowledge_plugins.cfg.CFGENode property*), 528
creation_failure_info (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 547
creation_failure_info (*angr.knowledge_plugins.cfg.CFGENode property*), 528
CRegister (*class in angr.analyses.decompiler.structured_codegen.c*), 736
CReturn (*class in angr.analyses.decompiler.structured_codegen.c*), 730
cs_operand (*angr.analyses.disassembly.Operand property*), 848
CStatement (*class in angr.analyses.decompiler.structured_codegen.c*), 726
CStatements (*class in angr.analyses.decompiler.structured_codegen.c*), 726
CStructField (*class in angr.analyses.decompiler.structured_codegen.c*), 731
CStructFieldNameDef (*class in angr.analyses.decompiler.structured_codegen.c*), 737

CStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.c*), 738
CStructuredCodeWalker (*class in angr.analyses.decompiler.structured_codegen.c*), 739
cstyle_ifs (*angr.analyses.decompiler.structured_codegen.CFBlock attribute*), 728
cstyle_ifs (*angr.analyses.decompiler.structured_codegen.CFailure attribute*), 728
CSwitchCase (*class in angr.analyses.decompiler.structured_codegen.c*), 729
CTypeCast (*class in angr.analyses.decompiler.structured_codegen.c*), 735
CUnaryOp (*class in angr.analyses.decompiler.structured_codegen.c*), 733
CUnsupportedStatement (*class in angr.analyses.decompiler.structured_codegen.c*), 731
current_function_address (*angr.state_plugins.callstack.CallStack property*), 265
current_function_address() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 790
current_return_target (*angr.state_plugins.callstack.CallStack property*), 265
current_stack_pointer (*angr.analyses.cfg.cfg_job_base.CFGJobBase property*), 657
current_stack_pointer (*angr.state_plugins.callstack.CallStack property*), 265
CVariable (*class in angr.analyses.decompiler.structured_codegen.c*), 732
CVariableField (*class in angr.analyses.decompiler.structured_codegen.c*), 733
CWhileLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 727
cyclic (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 700
cyclic_ancestor (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 700
cyclomatic_complexity (*angr.knowledge_plugins.functions.function.Function property*), 553

D

data (*angr.analyses.variable_recovery.engine_base.RichR attribute*), 827
data (*angr.engines.pcode.lifter.Lifter attribute*), 439
data (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
data (*angr.state_plugins.unicorn_engine.TRANSMIT_RECORD attribute*), 284
Data (*class in angr.analyses.reassembler*), 855
Data (*class in angr.knowledge_plugins.data*), 549
data_graph (*angr.analyses.ddg.DDG property*), 747
data_ptr() (*angr.PTChunk method*), 209
data_ptr() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 300
data_ptr() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 303
data_refs (*angr.engines.pcode.lifter.IRSB attribute*), 437
data_sub_graph() (*angr.analyses.ddg.DDG method*), 748
DataDependencyGraphAnalysis (*class in angr.analyses.data_dep.data_dependency_analysis*), 865
DataGraphError, 673
DataGraphMeta (*class in angr.analyses.datagraph_meta*), 673
DataLabel (*class in angr.analyses.reassembler*), 851
DataNormalizationMixin (*class in angr.storage.memory_mixins.bvv_conversion_mixin*), 340
DATAREF_HINTS (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 651
db_compatible() (*angr.angrdb.db.AngrDB method*), 675
DbCFGModel (*class in angr.angrdb.models*), 677
DbComment (*class in angr.angrdb.models*), 679
DbFunction (*class in angr.angrdb.models*), 677
dbg_comments() (*angr.analyses.reassembler.Instruction method*), 853
dbg_draw() (*angr.knowledge_plugins.functions.function.Function method*), 558
dbg_draw() (*angr.knowledge_plugins.functions.function_manager.Function method*), 552
dbg_get_repr() (*angr.analyses.decompiler.graph_region.GraphRegion static method*), 700
dbg_print() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 700
dbg_print() (*angr.knowledge_plugins.functions.function.Function method*), 558
dbg_print() (*angr.storage.memory_mixins.regioned_memory.region_met method*), 373
dbg_print_irsb() (*angr.annocfg.AnnotatedCFG method*), 873
dbg_print_stack() (*angr.sim_state.SimState method*), 228
dbg_print_stack() (*angr.SimState method*), 184
dbg_repr() (*angr.analyses.backward_slice.BackwardSlice method*), 628

dbg_repr() (*angr.analyses.cfg.cfb.CFBlanket method*), 638
dbg_repr() (*angr.analyses.ddg.DDG method*), 748
dbg_repr() (*angr.analyses.decompiler.clinic.Clinic method*), 694
dbg_repr() (*angr.analyses.decompiler.structuring.structurer_nodes.BaseNode method*), 685
dbg_repr() (*angr.analyses.decompiler.structuring.structurer_nodes.BreakNode method*), 687
dbg_repr() (*angr.analyses.decompiler.structuring.structurer_nodes.ConditionNode method*), 686
dbg_repr() (*angr.analyses.decompiler.structuring.structurer_nodes.ConditionalBreakNode method*), 688
dbg_repr() (*angr.analyses.decompiler.structuring.structurer_nodes.ContinueNode method*), 686
dbg_repr() (*angr.analyses.decompiler.structuring.structurer_nodes.CrossJumpNode method*), 688
dbg_repr() (*angr.analyses.decompiler.structuring.structurer_nodes.GotoNode method*), 687
dbg_repr() (*angr.analyses.decompiler.structuring.structurer_nodes.MultiNode method*), 685
dbg_repr() (*angr.analyses.decompiler.structuring.structurer_nodes.SequenceNode method*), 686
dbg_repr() (*angr.annocfg.AnnotatedCFG method*), 873
dbg_repr() (*angr.Blade method*), 168
dbg_repr() (*angr.blade.Blade method*), 871
dbg_repr() (*angr.keyed_region.KeyedRegion method*), 614
dbg_repr() (*angr.state_plugins.callstack.CallStack method*), 266
dbg_repr_run() (*angr.analyses.backward_slice.BackwardSlice method*), 628
DbInformation (*class in angr.angrdb.models*), 676
DbKnowledgeBase (*class in angr.angrdb.models*), 676
DbLabel (*class in angr.angrdb.models*), 679
DbObject (*class in angr.angrdb.models*), 676
DbStructuredCode (*class in angr.angrdb.models*), 678
DbVariableCollection (*class in angr.angrdb.models*), 677
DbXRefs (*class in angr.angrdb.models*), 678
DDG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
DDG (*class in angr.analyses.ddg*), 747
DDGJob (*class in angr.analyses.ddg*), 744
DDGView (*class in angr.analyses.ddg*), 746
DDGViewInstruction (*class in angr.analyses.ddg*), 746
DDGViewItem (*class in angr.analyses.ddg*), 746
deactivate() (*angr.misc.plugins.PluginPreset method*), 223
dead_ref() (*angr.state_hierarchy.StateHierarchy method*), 389
dead_ref() (*angr.StateHierarchy method*), 180
DeadAssignment (*class in angr.analyses.binary_optimizer*), 860
deadended (*angr.sim_manager.SimulationManager attribute*), 383
deadended (*angr.SimulationManager attribute*), 172
deadends (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 644
debug() (*angr.sim_manager.ErrorRecord method*), 389
DebugVariable (*class in angr.knowledge_plugins.debug_variables*), 568
DebugVariableContainer (*class in angr.knowledge_plugins.debug_variables*), 568
DebugVariableManager (*class in angr.knowledge_plugins.debug_variables*), 569
decanonicalize() (*angr.distributed.server.Server method*), 900
decanonicalize() (*angr.Server method*), 210
decode_instruction() (*in module angr.analyses.disassembly_utils*), 850
DecodingAssumption (*class in angr.analyses.cfg.cfg_fast*), 648
DecompilationCache (*class in angr.analyses.decompiler.decompilation_cache*), 697
DecompilationOption (*class in angr.analyses.decompiler.decompilation_options*), 696
decompile_functions() (*in module angr.analyses.decompiler.utils*), 744
Decompiler (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 620
Decompiler (*class in angr.analyses.decompiler.decompiler*), 697
decorate() (*angr.analyses.reaching_definitions.function_handler.Function static method*), 803
default (*angr.analyses.decompiler.structured_codegen.c.CSwitchCase attribute*), 729
default (*angr.sim_state_options.StateOption attribute*), 228
DEFAULT (*angr.state_plugins.sim_action.SimActionExit attribute*), 467
default_cc() (*in module angr*), 184
default_cc() (*in module angr.calling_conventions*), 502
default_exit_target (*angr.engines.pcode.lifter.IRSB attribute*), 437
default_indirect_jump_resolvers() (*in module angr.analyses.cfg.indirect_jump_resolvers.default_resolvers*), 662
default_node (*angr.analyses.decompiler.structuring.structurer_nodes.Sw attribute*), 688
default_simtype_from_size() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCod*

method), 739
DefaultChainMapCOW (*class in angr.utils.cowdict*), 885
DefaultFillerMixin (*class in angr.storage.memory_mixins.default_filler_mixin*), 340
DefaultListPagesMemory (*class in angr.storage.memory_mixins*), 338
DefaultMemory (*class in angr.storage.memory_mixins*), 338
defer_cleanup() (*angr.state_hierarchy.StateHierarchy method*), 389
defer_cleanup() (*angr.StateHierarchy method*), 180
definition (*angr.knowledge_plugins.key_definitions.live_definitions.Definition attribute*), 593
Definition (*class in angr.analyses.reaching_definitions*), 769
Definition (*class in angr.knowledge_plugins.key_definitions*), 583
Definition (*class in angr.knowledge_plugins.key_definitions.definition*), 590
DefinitionAnnotation (*class in angr.knowledge_plugins.key_definitions.live_definitions*), 593
DefinitionMatchPredicate (*class in angr.knowledge_plugins.key_definitions.definition*), 588
definitions (*angr.analyses.ddg.DDGViewInstruction property*), 746
defs (*angr.knowledge_base.knowledge_base.KnowledgeBase attribute*), 520
defs (*angr.KnowledgeBase attribute*), 211
DefUseChain (*class in angr.analyses.vsa_ddg*), 844
delete() (*angr.SimMount method*), 203
delete() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 252
delete() (*angr.state_plugins.filesystem.SimFilesystem method*), 250
delete() (*angr.state_plugins.filesystem.SimMount method*), 251
delete() (*angr.state_plugins.posix.PosixDevFS method*), 240
delete() (*angr.state_plugins.posix.PosixProcFS method*), 242
delete_reference() (*angr.state_plugins.jni_references.SimJNIReferences method*), 296
delete_uc() (*angr.state_plugins.unicorn_engine.Unicorn static method*), 290
demangled_name (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 726
demangled_name (*angr.knowledge_plugins.functions.function.Function property*), 559
demote() (*angr.state_plugins.history.SimStateHistory method*), 269
dep_graph (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 809
dep_graph (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 793
dep_graph (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 772
dep_graph (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
dependents (*angr.analyses.ddg.DDGViewItem property*), 746
depends() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 801
depends() (*angr.analyses.reaching_definitions.FunctionCallData method*), 788
depends_on (*angr.analyses.ddg.DDGViewItem property*), 746
DepGraph (*class in angr.analyses.reaching_definitions.dep_graph*), 795
DepNodeTypes (*class in angr.analyses.data_dep.dep_nodes*), 867
depth (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 547
depth (*angr.knowledge_plugins.cfg.CFGENode attribute*), 528
deref (*angr.state_plugins.debug_variables.SimDebugVariable property*), 308
deref (*angr.state_plugins.view.SimMemView property*), 314
deref() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 765
deref() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 813
deref() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 783
deref() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 601
deref() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 580
DerefSize (*class in angr.knowledge_plugins.key_definitions*), 581
DerefSize (*class in angr.knowledge_plugins.key_definitions.live_definitions*), 592
DerivedTypeVariable (*class in angr.analyses.typehoon.typevars*), 831
describe_variables() (*angr.state_plugins.solver.SimSolver method*), 255
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePointerSaveSimplifier attribute*), 705
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.const_derefs.ConstantDereferencesSimplification attribute*), 701
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifier attribute*), 705
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.eager_returns.EagerReturnsSimplifier attribute*), 702

DESCRIPTION (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.ExprOpSwapper attribute*), 710
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 706
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier attribute*), 707
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.inter_simplifier.MultiSimplifier attribute*), 708
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.multi_simplifier.MultiSimplifier attribute*), 708
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.retrocoder.Retrocoder attribute*), 703
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifier attribute*), 711
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 711
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 705
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_pcfi_simplifier.X86GccGetPcSimplifier attribute*), 711
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase attribute*), 713
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.base_peephole_optimizations.MultiStmtBase attribute*), 712
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.base_peephole_optimizations.PeepholeOptimizationStmtBase attribute*), 712
description (*angr.sim_state_options.StateOption attribute*), 228
descriptions (*angr.state_plugins.history.SimStateHistory property*), 269
dest (*angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute*), 798
destroy() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
desymbolize() (*angr.analyses.reassembler.Data method*), 855
determine() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 828
DFS (*class in angr.exploration_techniques*), 398
DFS (*class in angr.exploration_techniques.dfs*), 408
dfs_back_edges() (*in module angr.utils.graph*), 887
dict_content (*angr.utils.dynamic_dictlist.DynamicDictList attribute*), 886
dict_strkey_to_intkey() (*angr.angrdb.serializers.structured_code.StructuredCodeManagerSerializer static method*), 683
DictBackerMixin (*class in angr.storage.memory_mixins.paged_memory.paged_backer_mixin*), 357
Difference (*class in angr.analyses.bindiff*), 629
difference() (*angr.sim_state_options.SimStateOptions method*), 230
differing_blocks (*angr.analyses.bindiff.BinDiff property*), 632
differing_blocks (*angr.analyses.bindiff.FunctionDiff property*), 630
differing_constants() (*in module angr.analyses.bindiff*), 629
differing_functions (*angr.analyses.bindiff.BinDiff property*), 632
differing_functions_with_consts() (*angr.analyses.bindiff.BinDiff method*), 632
direct_next (*angr.engines.pcode.lifter.IRSB property*), 438
Directoz (*class in angr.exploration_techniques*), 399
Director (*class in angr.exploration_techniques.director*), 419
dirty (*angr.analyses.decompiler.structured_codegen.c.CDirtyExpression attribute*), 737
DirtyAddrsMixin (*class in angr.storage.memory_mixins.dirty_addrs_mixin*), 344
disable_profiling() (*in module angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 690
disable_timing() (*in module angr.state_plugins.solver.MultiSimtBase*), 254
DisassemblerBlock (*class in angr.block*), 220
DisassemblerInsn (*class in angr.block*), 220
Disassembly (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
disassembly (*angr.Block property*), 170
disassembly (*angr.block.Block property*), 221
disassembly (*angr.engines.pcode.lifter.IRSB property*), 439
Disassembly (*class in angr.analyses.disassembly*), 849
DisassemblyPiece (*class in angr.analyses.disassembly*), 846
discard() (*angr.knowledge_plugins.structured_code.manager.StructuredC method*), 570
discard() (*angr.sim_state_options.SimStateOptions method*), 230
discard() (*angr.sim_variable.SimVariableSet method*), 507
discard_input() (*angr.state_plugins.cgc.SimStateCGC method*), 272
discard_memory_variable() (*angr.sim_variable.SimVariableSet method*), 507
discard_output() (*angr.state_plugins.cgc.SimStateCGC method*), 272
discard_plugin_preset() (*angr.misc.plugins.PluginHub method*), 223
discard_register_variable() (*angr.sim_variable.SimVariableSet method*), 507
display_name (*angr.analyses.data_dep.dep_nodes.VarDepNode property*), 869

dissect_instruction() (*angr.analyses.disassembly.Instruction method*), 848
dissect_instruction_by_default() (*angr.analyses.disassembly.Instruction method*), 848
dissect_instruction_for_arm() (*angr.analyses.disassembly.Instruction method*), 848
DivSimplifier (*class in angr.analyses.decompiler.optimization_passes.div_simplifier*), 705
DivSimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.div_simplifier*), 705
do_full_xrefs() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 655
do_preprocess() (*in module angr.sim_type*), 518
do_trace() (*angr.analyses.identifier.identify.Identifier method*), 836
dom (*angr.utils.graph.Dominators attribute*), 888
dom (*angr.utils.graph.PostDominators attribute*), 888
dominance_frontiers (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase property*), 822
DominanceFrontier (*class in angr.analyses.dominance_frontier*), 861
dominates() (*in module angr.utils.graph*), 887
Dominators (*class in angr.utils.graph*), 888
done (*angr.analyses.vfg.AnalysisTask property*), 840
done (*angr.analyses.vfg.CallAnalysis property*), 841
done (*angr.analyses.vfg.FunctionAnalysis property*), 841
DoNotUpdate (*angr.analyses.calling_convention.UpdateArgumentsOption attribute*), 633
Double (*class in angr.analyses.typehoon.typeconsts*), 835
downsize() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 643
downsize() (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method*), 622
downsize() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 813
downsize() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 783
downsize() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 823
downsize() (*angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFastState method*), 824
downsize() (*angr.knowledge_plugins.cfg.cfg_node.CFGENode method*), 547
downsize() (*angr.knowledge_plugins.cfg.CFGENode method*), 528
downsize() (*angr.sim_state.SimState method*), 226
downsize() (*angr.SimState method*), 183
downsize() (*angr.state_plugins.inspect.SimInspector method*), 234
downsize() (*angr.state_plugins.sim_action.SimAction method*), 466
downsize() (*angr.state_plugins.sim_action.SimActionData method*), 468
downsize() (*angr.state_plugins.solver.SimSolver method*), 257
downsize_region() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 823
DreamStructurer (*class in angr.analyses.decompiler.structuring.dream*), 684
DrillerCore (*class in angr.exploration_techniques*), 393
DrillerCore (*class in angr.exploration_techniques.driller_core*), 416
DROP (*angr.sim_manager.SimulationManager attribute*), 383
DROP (*angr.SimulationManager attribute*), 172
drop() (*angr.sim_manager.SimulationManager method*), 387
drop() (*angr.SimulationManager method*), 176
dst (*angr.analyses.cfg.cfg_fast.IndirectJumpResolver attribute*), 435
dst (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 610
dst_addr (*angr.analyses.cfg.cfg_fast.FunctionCallEdge attribute*), 650
dst_addr (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge attribute*), 650
dst_addr (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute*), 650
dst_func_addr (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge attribute*), 651
dst_func_addr (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute*), 650
dst_type (*angr.analyses.decompiler.structured_codegen.c.CTypeCast attribute*), 735
dummy (*angr.analyses.reaching_definitions.definition.Definition attribute*), 770
dummy (*angr.knowledge_plugins.key_definitions.Definition attribute*), 583
dummy (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 590
DummyStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.dummy*), 741
dump() (*angr.angrdb.db.AngrDB method*), 675
dump() (*angr.angrdb.serializers.cfg_model.CFGModelSerializer static method*), 680
dump() (*angr.angrdb.serializers.comments.CommentsSerializer static method*), 680
dump() (*angr.angrdb.serializers.funcs.FunctionManagerSerializer*

static method), 680
dump() (*angr.angrdb.serializers.kb.KnowledgeBaseSerializer static method*), 681
dump() (*angr.angrdb.serializers.labels.LabelsSerializer static method*), 681
dump() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 682
dump() (*angr.angrdb.serializers.structured_code.StructuredCodeManagerSerializer static method*), 683
dump() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 682
dump() (*angr.angrdb.serializers.xrefs.XRefsSerializer static method*), 682
dump() (*angr.calling_conventions.AllocHelper method*), 483
dump_file_by_path() (*angr.state_plugins.posix.SimSystemPosix method*), 247
dump_graph() (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer static method*), 691
dump_internal() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 683
dumps() (*angr.state_plugins.posix.SimSystemPosix method*), 247
dumps() (*angr.vaults.Vault method*), 617
DUPLICATION_CHECK (*angr.analyses.decompiler.structured_codegen.base.StructuredCodeGenerationCache attribute*), 723
DURING_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 703
dwarf_cfa (*angr.state_plugins.debug_variables.SimDebugVariablePlugin property*), 309
dwarf_cfa_approx (*angr.state_plugins.debug_variables.SimDebugVariablePlugin property*), 309
DYNAMIC_RET (*angr.sim_procedure.SimProcedure attribute*), 471
DYNAMIC_RET (*angr.SimProcedure attribute*), 159
dynamic_returns() (*angr.sim_procedure.SimProcedure method*), 472
dynamic_returns() (*angr.SimProcedure method*), 159
DynamicDictList (*class in angr.utils.dynamic_dictlist*), 885

E

E2BIG (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EACCES (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EAGAIN (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EagerReturnsSimplifier (*class in angr.analyses.decompiler.optimization_passes.eager_returns*), 701
EBADF (*angr.state_plugins.cgc.SimStateCGC attribute*), 271
EBADF (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EBUSY (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
ECHILD (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
edges_to_repair (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 536
edges_to_repair (*angr.knowledge_plugins.cfg.CFGModel attribute*), 530
EDOM (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
EEXIST (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EFAULT (*angr.state_plugins.cgc.SimStateCGC attribute*), 271
EFBIG (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EFBIG (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
effects (*angr.analyses.reaching_definitions.function_handler.FunctionCall attribute*), 801
effects (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
EINTR (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EINVAL (*angr.state_plugins.cgc.SimStateCGC attribute*), 272
EINVAL (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EIO (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EISDIR (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
ELFHeader (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 542
ELFHeader (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
eliminatable (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegO property*), 663
eliminatable (*angr.analyses.data_dep.data_dependency_analysis.NodalA property*), 865
eliminatable (*angr.analyses.variable_recovery.annotations.StackLocatio property*), 819
eliminatable (*angr.analyses.variable_recovery.annotations.VariableSour property*), 819
eliminatable (*angr.analyses.variable_recovery.variable_recovery_base.V property*), 820
eliminatable (*angr.knowledge_plugins.key_definitions.live_definitions.De property*), 593
eliminatable (*angr.storage.memory_mixins.address_concretization_mixi property*), 344

else_node (*angr.analyses.decompiler.structured_codegen.c.CIfElse attribute*), 728
else_node (*angr.analyses.decompiler.structuring.structurer_nodes.CascadingConditionNode attribute*), 687
EMFILE (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
EMLINK (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
Empty (*angr.analyses.proximity_graph.ProximityNodeTypes attribute*), 863
empty_block() (*angr.engines.pcode.lifter.IRSB static method*), 437
EmptyBlockNotice, 685
EmptyNodeRemover (*class in angr.analyses.decompiler.empty_node_remover*), 698
enable_profiling() (*in module angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 660
enable_timing() (*in module angr.state_plugins.solver*), 254
end (*angr.analyses.cfg.segment_list.Segment attribute*), 670
end (*angr.keyed_region.RegionObject property*), 613
end_info_hook() (*in module angr.state_plugins.trace_additions*), 274
endness (*angr.analyses.reaching_definitions.MemoryLocation attribute*), 769
endness (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation attribute*), 588
endness (*angr.storage.memory_object.SimMemoryObject attribute*), 334
endpoints (*angr.knowledge_plugins.functions.function.Function property*), 555
endpoints_with_type (*angr.knowledge_plugins.functions.function.Function property*), 555
ENFILE (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
ENODEV (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
ENOENT (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
ENOEXEC (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
ENOMEM (*angr.state_plugins.cgc.SimStateCGC attribute*), 272
ENOMEM (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
ENOSPC (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
ENOSYS (*angr.state_plugins.cgc.SimStateCGC attribute*), 272
ENOTBLK (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
ENOTDIR (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
ENOTTY (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
entry_state() (*angr.factory.AngrObjectFactory method*), 217
entrypoints (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSlice property*), 815
environment (*angr.analyses.reaching_definitions.rd_state.ReachingDefini property*), 809
environment (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
Environment (*class in angr.knowledge_plugins.key_definitions.environment*), 590
ENXIO (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
eof() (*angr.SimFileDescriptor method*), 198
eof() (*angr.SimFileDescriptorDuplex method*), 201
eof() (*angr.storage.file.SimFileDescriptor method*), 327
eof() (*angr.storage.file.SimFileDescriptorBase method*), 326
eof() (*angr.storage.file.SimFileDescriptorDuplex method*), 330
EPERM (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
EPIPE (*angr.state_plugins.cgc.SimStateCGC attribute*), 272
EPIPE (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
EQ (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.C attribute*), 719
Equal (*angr.analyses.loop_analysis.Condition attribute*), 837
Equivalence (*class in angr.analyses.typehoon.typevars*), 829
ERANGE (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
erase() (*angr.storage.memory_mixins.MemoryMixin method*), 337
erase() (*angr.storage.memory_mixins.paged_memory.paged_memory_mix method*), 353
erase() (*angr.storage.memory_mixins.paged_memory.pages.list_page.List method*), 362
erase() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page method*), 349
EROFS (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
errno (*angr.state_plugins.libc.SimStateLibc property*), 240
error_converter() (*in module angr.state_plugins.solver*), 254
ERROR_REG (*angr.calling_conventions.SimCCN64LinuxSyscall*

attribute), 499
ERROR_REG (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 498
ERROR_REG (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 500
ERROR_REG (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 499
ERROR_REG (*angr.calling_conventions.SimCCSyscall attribute*), 493
errored (*angr.sim_manager.SimulationManager property*), 383
errored (*angr.SimulationManager property*), 172
ErrorRecord (*class in angr.sim_manager*), 389
errors (*angr.analyses.analysis.Analysis attribute*), 621
errors (*angr.Analysis attribute*), 178
ESPIPE (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
ESRCH (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
ETXTBSY (*angr.state_plugins.posix.SimSystemPosix attribute*), 245
eval() (*angr.state_plugins.solver.SimSolver method*), 260
eval_atleast() (*angr.state_plugins.solver.SimSolver method*), 261
eval_atmost() (*angr.state_plugins.solver.SimSolver method*), 260
eval_exact() (*angr.state_plugins.solver.SimSolver method*), 261
eval_one() (*angr.state_plugins.solver.SimSolver method*), 260
eval_to_ast() (*angr.state_plugins.solver.SimSolver method*), 258
eval_upto() (*angr.state_plugins.solver.SimSolver method*), 259
evaluate_binary() (*angr.engines.pcode.behavior.OpBehavior method*), 445
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorBoolAnd method*), 458
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorBoolOr method*), 459
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorBoolXor method*), 458
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorFloatEqual method*), 446
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntAdd method*), 450
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntCarry method*), 453
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntDiv method*), 451
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntEqual method*), 456
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntLess method*), 454
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntLess method*), 448
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual method*), 448
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntMult method*), 455
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntOr method*), 454
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntRem method*), 457
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntRight method*), 454
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSborrow method*), 451
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntScarry method*), 451
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSdiv method*), 456
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSless method*), 447
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual method*), 448
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSrem method*), 457
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSright method*), 455
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSub method*), 450
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntXor method*), 453
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorNotEqual method*), 447
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorSubpiece method*), 463
evaluate_unary() (*angr.engines.pcode.behavior.OpBehavior method*), 445
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorBoolNegate method*), 457
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorCopy method*), 446
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorInt2Comp method*), 452
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorIntNegate method*), 452
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorIntSext method*), 449
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorIntZext method*), 449
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorPopcount method*), 464
events (*angr.state_plugins.history.SimStateHistory property*), 269
events_of_type() (*angr.state_plugins.log.SimStateLog*

method), 262
EXC_COUNTER (*angr.analyses.decompiler.condition_processor.ConditionProcessor attribute*), 696
EXDEV (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
execute() (*angr.Project method*), 166
execute() (*angr.project.Project method*), 216
execute() (*angr.sim_procedure.SimProcedure method*), 471
execute() (*angr.SimProcedure method*), 159
ExecuteAddressGoal (*class in angr.exploration_techniques*), 400
ExecuteAddressGoal (*class in angr.exploration_techniques.director*), 418
executed_instruction_count (*angr.errors.SimError attribute*), 896
Existence (*class in angr.analyses.typehoon.typevars*), 829
exists_in_replacements() (*in module angr.analyses.cfg.indirect_jump_resolvers.const_resolver*), 666
exit() (*angr.sim_procedure.SimProcedure method*), 473
exit() (*angr.SimProcedure method*), 160
exit_hook() (*in module angr.state_plugins.trace_additions*), 274
exit_observe() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 793
exit_observe() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 773
exit_observed (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState attribute*), 807
exit_observed (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 777
exit_statements (*angr.engines.pcode.lifter.IRSB property*), 438
ExitStatement (*class in angr.engines.pcode.lifter*), 435
ExplicitFillerMixin (*class in angr.storage.memory_mixins.default_filler_mixin*), 340
ExplorationStatusNotifier (*class in angr.distributed.worker*), 900
ExplorationTechnique (*class in angr*), 178
ExplorationTechnique (*class in angr.exploration_techniques*), 390
explore() (*angr.sim_manager.SimulationManager method*), 384
explore() (*angr.SimulationManager method*), 173
Explorer (*class in angr.exploration_techniques*), 396
Explorer (*class in angr.exploration_techniques.explorer*), 408
expr (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.CmpExpr attribute*), 707
expr (*angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression attribute*), 737
expr (*angr.analyses.decompiler.structured_codegen.c.CTypeCast attribute*), 735
expr_classes (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase attribute*), 713
expr_comments (*angr.angrdb.models.DbStructuredCode attribute*), 678
expr_idx (*angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter attribute*), 715
ExpressionCounter (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 716
ExpressionFolder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 716
ExpressionLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 715
ExpressionNarrowingWalker (*class in angr.analyses.decompiler.expression_narrower*), 699
ExpressionReplacer (*class in angr.analyses.decompiler.optimization_passes.expr_op_swapper*), 709
ExpressionReplacer (*class in angr.analyses.decompiler.optimization_passes.ite_expr_converter*), 706
ExpressionReplacer (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 716
expressions (*angr.engines.pcode.lifter.IRSB property*), 438
ExpressionUseFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 716
ExprOpSwapper (*class in angr.analyses.decompiler.optimization_passes.expr_op_swapper*), 710
extend() (*angr.engines.pcode.lifter.IRSB method*), 438
extend_actions() (*angr.state_plugins.history.SimStateHistory method*), 269
extend_actions() (*angr.state_plugins.log.SimStateLog method*), 262
extern (*angr.knowledge_plugins.key_definitions.definition.DefinitionMatch attribute*), 589
ExternalCodeLocation (*class in angr.code_location*), 613
extract() (*angr.sim_type.SimCppClass method*), 517
extract() (*angr.sim_type.SimStruct method*), 516
extract() (*angr.sim_type.SimTypeArray method*), 512
extract() (*angr.sim_type.SimTypeBool method*), 511
extract() (*angr.sim_type.SimTypeChar method*), 510
extract() (*angr.sim_type.SimTypeFloat method*), 515
extract() (*angr.sim_type.SimTypeInt method*), 510
extract() (*angr.sim_type.SimTypeNum method*), 510

extract() (*angr.sim_type.SimTypeNumOffset method*), 518
extract() (*angr.sim_type.SimTypeReg method*), 509
extract() (*angr.sim_type.SimTypeString method*), 513
extract() (*angr.sim_type.SimTypeWideChar method*), 511
extract() (*angr.sim_type.SimTypeWString method*), 513
extract() (*angr.sim_type.SimUnion method*), 517
extract() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 351
extract_claripy() (*angr.sim_type.SimType method*), 508
extract_components() (*angr.procedures.stubs.format_parser.FormatParser method*), 475
extract_defs() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 759
extract_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 808
extract_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 779
extract_defs() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 595
extract_defs() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 575
extract_defs_from_annotations() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 759
extract_defs_from_annotations() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 596
extract_defs_from_annotations() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 575
extract_defs_from_mv() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 759
extract_defs_from_mv() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 596
extract_defs_from_mv() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 575
extract_jump_targets() (*in module angr.analyses.decompiler.utils*), 741
extract_offset_to_sp() (*angr.analyses.propagator.engine_ail.SimEnginePropagatorAIL method*), 753
extract_offset_to_sp() (*angr.engines.light.engine.SimEngineLightMixin static method*), 751
extract_stack_offset_from_addr() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 822
extract_terms() (*in module angr.analyses.decompiler.structured_codegen.c*), 724
extract_value_if_concrete() (*angr.analyses.xrefs.SimEngineXRefsVEX static method*), 862
extract_variables() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 821

F

f (*angr.utils.mp.Closure attribute*), 893
failure() (*angr.engines.successors.SimSuccessors class method*), 429
false_node (*angr.analyses.decompiler.structuring.structurer_nodes.CondBr attribute*), 686
fast_memory_load() (*angr.analyses.reassembler.Reassembler method*), 856
FastMemory (*class in angr.storage.memory_mixins*), 338
fd (*angr.state_plugins.unicorn_engine.TRANSMIT_RECORD attribute*), 284
FD_SETSIZE (*angr.state_plugins.cgc.SimStateCGC attribute*), 272
FetchingZeroPageError, 288
field (*angr.analyses.decompiler.structured_codegen.c.CStructField attribute*), 732
FieldReferenceCleanup (*class in angr.analyses.decompiler.structured_codegen.c*), 740
fields (*angr.sim_type.SimUnion attribute*), 516
file_exists (*angr.SimFileDescriptor property*), 199
file_exists (*angr.storage.file.SimFileDescriptor property*), 328
file_exists (*angr.storage.file.SimFileDescriptorBase property*), 327
fill_content() (*angr.knowledge_plugins.cfg.memory_data.MemoryData method*), 543
fill_content() (*angr.knowledge_plugins.cfg.MemoryData method*), 525
fill_reg_map() (*in module angr.analyses.reassembler*), 851
filter() (*angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got.AMD64 method*), 657
filter() (*angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast.ArmElfFast method*), 658
filter() (*angr.analyses.cfg.indirect_jump_resolvers.const_resolver.ConstantResolver method*), 666
filter() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver method*), 666
filter() (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.MipsElfFast method*), 660
filter() (*angr.analyses.cfg.indirect_jump_resolvers.resolver.IndirectJumpResolver method*), 667

filter() (*angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt_unresolved_indirect_jump_resolver.X86ElfPicPltUnresolvedIndirectJumpResolver method*), 661
filter() (*angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat_indirect_jump_resolver.X86PeIatIndirectJumpResolver method*), 659
filter() (*angr.analyses.veritesting.CallTracingFilter method*), 838
filter() (*angr.exploration_techniques.ExplorationTechnique method*), 390
filter() (*angr.exploration_techniques.Explorer method*), 397
filter() (*angr.exploration_techniques.explorer.Explorer method*), 409
filter() (*angr.exploration_techniques.local_loop_seer.LocalLoopSeer method*), 422
filter() (*angr.exploration_techniques.LocalLoopSeer method*), 406
filter() (*angr.exploration_techniques.loop_seer.LoopSeer method*), 421
filter() (*angr.exploration_techniques.LoopSeer method*), 394
filter() (*angr.exploration_techniques.Slicecutor method*), 392
filter() (*angr.exploration_techniques.slicecutor.Slicecutor method*), 417
filter() (*angr.exploration_techniques.Tracer method*), 395
filter() (*angr.exploration_techniques.tracer.Tracer method*), 415
filter() (*angr.ExplorationTechnique method*), 179
filter() (*angr.sim_manager.SimulationManager method*), 386
filter() (*angr.SimulationManager method*), 175
filter_actions() (*angr.state_plugins.history.SimStateHistory method*), 268
filter_cond_regions() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier*), 720
filter_constraints() (*angr.state_plugins.trace_additions.ZenPlugin method*), 277
filter_variables() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifierAILState method*), 709
final_states (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 547
final_states (*angr.knowledge_plugins.cfg.CFGENode attribute*), 528
find() (*angr.storage.memory_mixins.MemoryMixin method*), 336
find() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 367
find() (*angr.storage.memory_mixins.regioned_memory.static_find.StaticFindMixin method*), 371
find() (*angr.storage.memory_mixins.smart_find_mixin.SmartFindMixin method*), 339
find() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 347
find_all_predecessors() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 796
find_all_successors() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 796
find_cc() (*angr.calling_conventions.SimCC static method*), 490
find_cc() (*angr.SimCC static method*), 187
find_consumers() (*angr.analyses.ddg.DDG method*), 748
find_data_references_and_update_memory_data() (*angr.analyses.decompiler.decompiler.Decompiler method*), 698
find_declaration() (*angr.knowledge_plugins.functions.function.Functio method*), 558
find_definition() (*angr.analyses.decompiler.peephole_optimizations.b static method*), 713
find_definitions() (*angr.analyses.ddg.DDG method*), 748
find_definitions() (*angr.analyses.reaching_definitions.dep_graph.Dep method*), 796
find_defs_at() (*angr.analyses.reaching_definitions.ReachingDefinitions method*), 775
find_defs_at() (*angr.knowledge_plugins.key_definitions.rd_model.Reach method*), 602
find_defs_at() (*angr.knowledge_plugins.key_definitions.ReachingDefini method*), 571
find_function_for_reflow_into_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 542
find_function_for_reflow_into_addr() (*angr.knowledge_plugins.cfg.CFGModel method*), 535
find_killers() (*angr.analyses.ddg.DDG method*), 749
find_merge_points() (*angr.utils.graph.GraphUtils static method*), 888
find_path() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 797
find_paths() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 797
find_sources() (*angr.analyses.ddg.DDG method*), 749
find_stack_vars_x86() (*angr.analyses.identifier.identify.Identifier method*), 836
find_variable_by_atom() (*angr.knowledge_plugins.variables.variable_manager.VariableM method*), 564
find_variables_by_atom() (*angr.knowledge_plugins.variables.variable_manager.VariableM method*), 563
find_variables_by_atom()

(*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 564
find_variables_by_insn() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 563
find_variables_by_register() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 564
find_variables_by_stack_offset() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 564
find_variables_by_stmt() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 563
find_widening_points() (*angr.utils.graph.GraphUtils static method*), 889
FindFirstNodeInSet (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier*), 721
finish() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
fire() (*angr.BP method*), 161
fire() (*angr.state_plugins.inspect.BP method*), 233
first_nonlabel_node() (*in module angr.analyses.decompiler.utils*), 743
first_nonlabel_statement() (*in module angr.analyses.decompiler.utils*), 742
fix_prototype_returnty() (*angr.sim_procedure.SimProcedure method*), 472
fix_prototype_returnty() (*angr.SimProcedure method*), 160
flags (*angr.procedures.stubs.format_parser.FormatParser attribute*), 474
Flags (*class in angr.storage.file*), 314
flavor (*angr.angrdb.models.DbStructuredCode attribute*), 678
FlirtAnalysis (*class in angr.analyses.flirt*), 749
FlirtSignature (*class in angr.flirt*), 883
Float (*class in angr.analyses.typehoon.typeconsts*), 834
float_len_mod (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 475
float_spec (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 475
float_type() (*in module angr.analyses.typehoon.typeconsts*), 835
FloatBase (*class in angr.analyses.typehoon.typeconsts*), 834
FloatingPoint (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 542
FloatingPoint (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
floor_addr() (*angr.analyses.cfg.cfb.CFBlanket method*), 637
floor_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 550
floor_func() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 551
floor_item() (*angr.analyses.cfg.cfb.CFBlanket method*), 637
floor_items() (*angr.analyses.cfg.cfb.CFBlanket method*), 638
flush_page() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 355
fmt (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 735
fmt_char (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 736
fmt_hex (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 735
fmt_int() (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 736
fmt_neg (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 735
force_unroll_loops() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 643
FormatInfo (*class in angr.state_plugins.trace_additions*), 273
FormatInfoDontConstrain (*class in angr.state_plugins.trace_additions*), 274
FormatInfoIntToStr (*class in angr.state_plugins.trace_additions*), 274
FormatInfoStrToInt (*class in angr.state_plugins.trace_additions*), 274
FormatParser (*class in angr.procedures.stubs.format_parser*), 474
FormatSpecifier (*class in angr.procedures.stubs.format_parser*), 474
FormatString (*class in angr.procedures.stubs.format_parser*), 473
ForwardAnalysis (*class in angr.analyses.forward_analysis.forward_analysis*), 621
found (*angr.sim_manager.SimulationManager attribute*), 383
found (*angr.SimulationManager attribute*), 172
FP_ARG_REGS (*angr.calling_conventions.SimCC attribute*), 488
FP_ARG_REGS (*angr.calling_conventions.SimCCAArch64 attribute*), 496
FP_ARG_REGS (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 497
FP_ARG_REGS (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 495
FP_ARG_REGS (*angr.calling_conventions.SimCCARM attribute*), 495

FP_ARG_REGS (*angr.calling_conventions.SimCCARMHF attribute*), 496
FP_ARG_REGS (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 496
FP_ARG_REGS (*angr.calling_conventions.SimCCCdecl attribute*), 491
FP_ARG_REGS (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
FP_ARG_REGS (*angr.calling_conventions.SimCCN64 attribute*), 498
FP_ARG_REGS (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 499
FP_ARG_REGS (*angr.calling_conventions.SimCCO32 attribute*), 497
FP_ARG_REGS (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 498
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPC attribute*), 499
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPC64 attribute*), 500
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 500
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 499
FP_ARG_REGS (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 497
FP_ARG_REGS (*angr.calling_conventions.SimCCS390X attribute*), 501
FP_ARG_REGS (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 502
FP_ARG_REGS (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
FP_ARG_REGS (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 493
FP_ARG_REGS (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 494
FP_ARG_REGS (*angr.engines.pcode.cc.SimCCM68k attribute*), 464
FP_ARG_REGS (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 465
FP_ARG_REGS (*angr.engines.pcode.cc.SimCCXtensa attribute*), 466
FP_ARG_REGS (*angr.SimCC attribute*), 185
fp_args (*angr.calling_conventions.SimCC property*), 488
fp_args (*angr.SimCC property*), 185
fp_iter (*angr.calling_conventions.ArgSession attribute*), 487
fp_iter (*angr.calling_conventions.SimCC.ArgSession attribute*), 488
fp_iter (*angr.SimCC.ArgSession attribute*), 186
FP_RETURN_VAL (*angr.calling_conventions.SimCC attribute*), 488
FP_RETURN_VAL (*angr.calling_conventions.SimCCARMHF attribute*), 496
FP_RETURN_VAL (*angr.calling_conventions.SimCCCdecl attribute*), 492
FP_RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
FP_RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
FP_RETURN_VAL (*angr.SimCC attribute*), 185
free() (*angr.analyses.reaching_definitions.heap_allocator.HeapAllocator method*), 798
free() (*angr.SimHeapPTMalloc method*), 207
free() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 301
free() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 305
free_chunks() (*angr.SimHeapPTMalloc method*), 206
free_chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 301
free_chunks() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 304
freeze() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 818
free_block() (*angr.factory.AngrObjectFactory method*), 220
free_constraints (*angr.state_plugins.log.SimStateLog property*), 262
from_ail_expr() (*angr.analyses.reaching_definitions.Atom static method*), 766
from_ail_expr() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 584
from_argument() (*angr.analyses.reaching_definitions.Atom static method*), 766
from_argument() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 584
from_cle_variable() (*angr.state_plugins.debug_variables.SimDebugVariable static method*), 308
from_digraph() (*angr.annocfg.AnnotatedCFG method*), 872
from_name() (*angr.knowledge_plugins.debug_variables.DebugVariableM method*), 569
from_name_and_pc() (*angr.knowledge_plugins.debug_variables.DebugV method*), 569
from_opstr() (*angr.analyses.loop_analysis.Condition class method*), 837
from_pc() (*angr.knowledge_plugins.debug_variables.DebugVariable method*), 568
from_pc() (*angr.knowledge_plugins.debug_variables.DebugVariableCont method*), 568
from_signature (*angr.knowledge_plugins.functions.function.Function attribute*), 553
from_signature (*angr.knowledge_plugins.functions.soot_function.SootFu attribute*), 561
from_state() (*angr.analyses.variable_recovery.annotations.VariableSour*

static method), 819
FrozenStackPointerTrackerState (*class in angr.analyses.stack_pointer_tracker*), 817
FS (*in module angr.flirt*), 883
fstat() (*angr.state_plugins.posix.SimSystemPosix method*), 246
fstat_with_result() (*angr.state_plugins.posix.SimSystemPosix method*), 246
full_graph (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 700
full_init_state() (*angr.factory.AngrObjectFactory method*), 217
full_simplify() (*angr.state_hierarchy.StateHierarchy method*), 389
full_simplify() (*angr.StateHierarchy method*), 180
func (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessorState attribute*), 663
func_addr (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
func_addr (*angr.analyses.cfg.cfg_job_base.BlockID property*), 657
func_addr (*angr.analyses.cfg.cfg_job_base.CFGJobBase property*), 657
func_addr (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase attribute*), 713
func_addr (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationMultiStmtBase attribute*), 713
func_addr (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase attribute*), 712
func_addr (*angr.analyses.variable_recovery.engine_base.SimEngineVRBase property*), 827
func_addr (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase property*), 822
func_addr (*angr.angrdb.models.DbStructuredCode attribute*), 678
func_addr (*angr.angrdb.models.DbVariableCollection attribute*), 678
func_addr (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
func_addr (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
func_edge_type_from_pb() (*in module angr.utils.enums_conv*), 886
func_edge_type_to_pb() (*in module angr.utils.enums_conv*), 886
func_graph (*angr.analyses.reaching_definitions.subject.Subject property*), 813
func_lookup() (*angr.analyses.disassembly.Disassembly method*), 850
FuncComment (*class in angr.analyses.disassembly*), 849
FuncIn (*class in angr.analyses.typehoon.typevars*), 832
FuncInfo (*class in angr.analyses.identifier.identify*), 835
FuncOut (*class in angr.analyses.typehoon.typevars*), 832
funcs (*angr.angrdb.models.DbKnowledgeBase attribute*), 676
Function (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 863
function (*angr.analyses.reaching_definitions.function_handler.FunctionCallDataUnwrapped attribute*), 800
function (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
Function (*angr.analyses.reaching_definitions.subject.SubjectType attribute*), 813
Function (*class in angr.knowledge_plugins.functions.function*), 552
function() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 551
function_address (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
function_address (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
function_address (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedAddress attribute*), 369
function_calls (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis attribute*), 792
function_codeloc (*angr.analyses.reaching_definitions.function_handler.FunctionCallDataUnwrapped attribute*), 800
function_codeloc (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
function_dependency_graph() (*angr.analyses.ddg.DDG method*), 748
function_final_states (*angr.analyses.vfg.VFG property*), 843
function_initial_states (*angr.analyses.vfg.VFG property*), 843
function_label (*angr.analyses.disassembler.FunctionLabel property*), 851
function_needs_variable_recovery() (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis static method*), 636
FUNCTION_PROLOGUE (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 651
function_prototype() (*angr.factory.AngrObjectFactory method*), 220
FunctionAnalysis (*class in angr.analyses.vfg*), 840
FunctionCall (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 863
FunctionCallData (*class in angr.analyses.reaching_definitions*), 785
FunctionCallData (*class in angr.analyses.reaching_definitions.function_handler*), 799
FunctionCallDataUnwrapped (*class in angr.analyses.reaching_definitions.function_handler*), 802
FunctionCallEdge (*class in angr.analyses.cfg.cfg_fast*),

650
FunctionCallRelationships (*class in angr.analyses.reaching_definitions.dep_graph*), 794
FunctionDict (*class in angr.knowledge_plugins.functions.function_manager*), 550
FunctionDiff (*class in angr.analyses.bindiff*), 630
FunctionEdge (*class in angr.analyses.cfg.cfg_fast*), 649
FunctionEffect (*class in angr.analyses.reaching_definitions.function_handler*), 798
FunctionFakeRetEdge (*class in angr.analyses.cfg.cfg_fast*), 650
FunctionGraphVisitor (*class in angr.analyses.forward_analysis.visitors.function_graph*), 623
FunctionHandler (*class in angr.analyses.reaching_definitions*), 783
FunctionHandler (*class in angr.analyses.reaching_definitions.function_handler*), 803
FunctionKey (*class in angr.analyses.cfg.cfg_job_base*), 657
FunctionLabel (*class in angr.analyses.reassembler*), 851
FunctionManager (*class in angr.knowledge_plugins.functions.function_manager*), 550
FunctionManagerSerializer (*class in angr.angrdb.serializers.funcs*), 680
FunctionParser (*class in angr.knowledge_plugins.functions.function_parser*), 559
FunctionProxiNode (*class in angr.analyses.proximity_graph*), 863
FunctionReturn (*class in angr.analyses.cfg.cfg_fast*), 648
FunctionReturnEdge (*class in angr.analyses.cfg.cfg_fast*), 651
functions (*angr.analyses.cfg.cfg_base.CFGBase property*), 646
functions (*angr.knowledge_base.knowledge_base.KnowledgeBase attribute*), 520
functions (*angr.KnowledgeBase attribute*), 211
functions_called() (*angr.knowledge_plugins.functions.function.Function method*), 559
functions_probably_identical() (*angr.analyses.bindiff.BinDiff method*), 631
FunctionStart (*class in angr.analyses.disassembly*), 846
FunctionTag (*class in angr.knowledge_plugins.key_definitions.tag*), 604
FunctionTransitionEdge (*class in angr.analyses.cfg.cfg_fast*), 650
functy (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 726
fwd_chunk() (*angr.PTChunk method*), 209
fwd_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 300
fwd_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 303

G

g_label_ctr (*angr.analyses.reassembler.Label attribute*), 851
GDB (*class in angr.state_plugins.gdb*), 270
generate_code_cover() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 656
generate_gdt() (*angr.SimOS method*), 169
generate_gdt() (*angr.simos.simos.SimOS method*), 876
generate_index() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
generate_symbolic_cmd_line_arg() (*angr.simos.javavm.SimJavaVM static method*), 881
generic_compare() (*angr.engines.pcode.behavior.OpBehavior static method*), 446
generic_info_hook() (*in module angr.state_plugins.trace_additions*), 274
get() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 818
get() (*angr.knowledge_plugins.functions.function_manager.FunctionDict method*), 550
get() (*angr.knowledge_plugins.key_definitions.environment.Environment method*), 591
get() (*angr.knowledge_plugins.labels.Labels method*), 549
get() (*angr.procedures.definitions.SimCppLibrary method*), 479
get() (*angr.procedures.definitions.SimLibrary method*), 477
get() (*angr.procedures.definitions.SimSyscallLibrary method*), 481
get() (*angr.SimMount method*), 203
get() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 252
get() (*angr.state_plugins.filesystem.SimFilesystem method*), 250
get() (*angr.state_plugins.filesystem.SimMount method*), 251
get() (*angr.state_plugins.globals.SimStateGlobals method*), 279
get() (*angr.state_plugins.posix.PosixDevFS method*), 240

get() (*angr.state_plugins.posix.PosixProcFS method*), 242
get() (*angr.state_plugins.view.SimRegNameView method*), 310
get() (*angr.utils.mp.Initializer class method*), 893
get_abstract_locations() (*angr.storage.memory_mixins.regioned_memory.region_memory_mixin.RegionMemoryRegionMetaMixin method*), 371
get_addr() (*angr.annocfg.AnnotatedCFG method*), 872
get_addr_of_native_method() (*angr.simos.javavm.SimJavaVM method*), 882
get_all_definitions() (*in module angr.analyses.reaching_definitions*), 789
get_all_nodes() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
get_all_nodes() (*angr.analyses.vfg.VFG method*), 844
get_all_nodes() (*angr.analyses.vsa_ddg.VSA_DDG method*), 845
get_all_nodes() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 538
get_all_nodes() (*angr.knowledge_plugins.cfg.CFGModel method*), 531
get_all_nodes_intersecting_region() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 538
get_all_nodes_intersecting_region() (*angr.knowledge_plugins.cfg.CFGModel method*), 531
get_all_patches() (*angr.knowledge_plugins.patches.PatchManager method*), 522
get_all_predecessors() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
get_all_predecessors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 540
get_all_predecessors() (*angr.knowledge_plugins.cfg.CFGModel method*), 533
get_all_successors() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
get_all_successors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 540
get_all_successors() (*angr.knowledge_plugins.cfg.CFGModel method*), 534
get_all_variables() (*angr.keyed_region.KeyedRegion method*), 616
get_alloc_depth() (*angr.state_plugins.uc_manager.SimUCManager method*), 280
get_any_node() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
get_any_node() (*angr.analyses.vfg.VFG method*), 843
get_any_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 537
get_any_node() (*angr.knowledge_plugins.cfg.CFGModel method*), 531
get_arg_info() (*angr.calling_conventions.SimCC method*), 491
get_arg_info() (*angr.SimCC method*), 188
get_args() (*angr.calling_conventions.SimCC method*), 489
get_args() (*angr.SimCC method*), 187
get_ast_subexprs() (*in module angr.analyses.decompiler.utils*), 742
get_base_addr() (*angr.keyed_region.KeyedRegion method*), 615
get_basic_info() (*in module angr.flirt.build_sig*), 883
get_behavior_for_opcode() (*angr.engines.pcode.behavior.BehaviorFactory method*), 464
get_block() (*angr.knowledge_plugins.functions.function.Function method*), 554
get_block_size() (*angr.knowledge_plugins.functions.function.Function method*), 554
get_branching_nodes() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
get_branching_nodes() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 540
get_branching_nodes() (*angr.knowledge_plugins.cfg.CFGModel method*), 534
get_by_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 551
get_byte() (*angr.state_plugins.trace_additions.ChallRespInfo static method*), 276
get_call_args() (*angr.analyses.identifier.identify.Identifier method*), 836
get_call_return() (*angr.knowledge_plugins.functions.function.Function method*), 557
get_call_sites() (*angr.knowledge_plugins.functions.function.Function method*), 556
get_call_stack_suffix() (*angr.analyses.cfg.cfg_job_base.CFGJobBase method*), 657
get_call_target() (*angr.knowledge_plugins.functions.function.Function method*), 557
get_cc() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 523
get_class() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 294
get_class_hierarchy()

(*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 294
get_cmd_line_args() (*angr.simos.javavm.SimJavaVM static method*), 882
get_concrete_fd() (*angr.state_plugins.posix.SimSystemPosix method*), 246
get_concrete_state() (*angr.analyses.variable_recovery.variable_recovery.VariableRecovery method*), 825
get_concrete_value() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 763
get_concrete_value() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 812
get_concrete_value() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 782
get_concrete_value() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 599
get_concrete_value() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
get_concrete_value_from_atom() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_concrete_value_from_atom() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 599
get_concrete_value_from_atom() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
get_concrete_value_from_definition() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_concrete_value_from_definition() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_concrete_value_from_definition() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
get_cpp_function_name() (*in module angr.utils.library*), 892
get_data_dep() (*angr.analyses.data_dep.data_dependency_analysis.DataDependencyGraphAnalysis method*), 866
get_data_references() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 545
get_data_references() (*angr.knowledge_plugins.cfg.CFGNode method*), 526
get_data_size() (*angr.PTChunk method*), 209
get_data_size() (*angr.state_plugins.heap.heap_freelist.ChunkFreeList method*), 300
get_data_size() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 302
get_dbinfo() (*angr.angrdb.db.AngrDB method*), 675
get_default_optimization_passes() (*in module angr.analyses.decompiler.optimization_passes*), 701
get_default_value_by_type() (*angr.simos.javavm.SimJavaVM static method*), 881
get_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 761
get_definitions() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 812
get_definitions() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 782
get_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 597
get_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 577
get_definitions_from_atoms() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_definitions_from_atoms() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_definitions_from_atoms() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
get_defs() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 775
get_defs() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 602
get_defs() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 571
get_dependants() (*angr.analyses.cdg.CDG method*), 673
get_display_name() (*angr.analyses.decompiler.structured_codegen.base.PositionMapping method*), 723
get_exit_livedefinitions() (*in module angr.analyses.reaching_definitions.function_handler*), 798
get_exit_stmt_idx() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
get_exit_stmt_idx() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 540
get_exit_stmt_idx() (*angr.knowledge_plugins.cfg.CFGModel method*), 534
get_expr_depth() (*angr.state_plugins.trace_additions.ZenPlugin method*), 276
get_fd() (*angr.state_plugins.posix.SimSystemPosix*

method), 246
get_flag_bytes() (*angr.state_plugins.trace_additions.ZenPlugin method*), 277
get_flag_rand_args() (*angr.state_plugins.trace_additions.ZenPlugin static method*), 276
get_footprint() (*angr.calling_conventions.SimArrayArg method*), 486
get_footprint() (*angr.calling_conventions.SimComboArg method*), 486
get_footprint() (*angr.calling_conventions.SimFunctionArgument method*), 484
get_footprint() (*angr.calling_conventions.SimReferenceArgument method*), 487
get_footprint() (*angr.calling_conventions.SimRegArg method*), 485
get_footprint() (*angr.calling_conventions.SimStackArg method*), 485
get_footprint() (*angr.calling_conventions.SimStructArg method*), 486
get_func_addr_from_addr() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 609
get_func_info() (*angr.analyses.identifier.identify.Identifier method*), 836
get_function_diff() (*angr.analyses.bindiff.BinDiff method*), 632
get_function_manager() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 567
get_function_name() (*in module angr.utils.library*), 891
get_function_subgraph() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 644
get_global_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 565
get_guardians() (*angr.analyses.cdg.CDG method*), 673
get_heap_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_heap_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_heap_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 577
get_heap_offset() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 765
get_heap_offset() (*angr.analyses.reaching_definitions.rd_state.RDState static method*), 808
get_heap_offset() (*angr.analyses.reaching_definitions.ReachingDefinitionsState static method*), 778
get_heap_offset() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 601
get_heap_offset() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 580
get_implementers() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 636
get_info() (*angr.angrdb.db.AngrDB static method*), 675
get_intersecting_functions() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 541
get_intersecting_functions() (*angr.knowledge_plugins.cfg.CFGModel method*), 535
get_irsb_at() (*angr.analyses.datagraph_meta.DataGraphMeta method*), 673
get_knowledge() (*angr.knowledge_base.knowledge_base.KnowledgeBase method*), 521
get_knowledge() (*angr.KnowledgeBase method*), 211
get_last_statement() (*angr.analyses.decompiler.condition_processor.ConditionProcessor class method*), 696
get_last_statement_index() (*angr.annocfg.AnnotatedCFG method*), 872
get_last_statements() (*angr.analyses.decompiler.condition_processor.ConditionProcessor class method*), 696
get_loop_back_edges() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
get_loops() (*angr.annocfg.AnnotatedCFG method*), 872
get_max_sinkhole() (*angr.state_plugins.cgc.SimStateCGC method*), 273
get_memory_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_memory_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_memory_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 577
get_method_definitive_type() (*angr.simos.javavm.SimJavaVM method*), 882
get_model() (*angr.knowledge_plugins.key_definitions.key_definition_manager.KeyDefinitionManager method*), 592
get_model() (*angr.knowledge_plugins.key_definitions.KeyDefinitionManager method*), 572
get_most_accurate_definitions() (*angr.knowledge_plugins.cfg.cfg_manager.CFGManager method*), 544

get_most_accurate() (*angr.knowledge_plugins.cfg.CFGManager method*), 536
get_mountpoint() (*angr.state_plugins.filesystem.SimFilesystem method*), 250
get_native_cc() (*angr.simos.javavm.SimJavaVM method*), 882
get_native_type() (*angr.simos.javavm.SimJavaVM method*), 882
get_nearest_pos() (*angr.analyses.decompiler.structured_codegen.base.StructuredCodeGenerator method*), 723
get_new_uuid() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin static method*), 376
get_node() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
get_node() (*angr.analyses.decompiler.structured_codegen.base.StructuredCodeGenerator method*), 723
get_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 537
get_node() (*angr.knowledge_plugins.cfg.CFGModel method*), 530
get_node() (*angr.knowledge_plugins.functions.function.Function method*), 554
get_normalized_block() (*angr.analyses.bindiff.FunctionDiff static method*), 631
get_objects_by_offset() (*angr.keyed_region.KeyedRegion method*), 616
get_observation_by_exit() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 775
get_observation_by_exit() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 603
get_observation_by_exit() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 572
get_observation_by_insn() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 775
get_observation_by_insn() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 603
get_observation_by_insn() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 571
get_observation_by_node() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 775
get_observation_by_node() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 603
get_observation_by_node() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 572
get_observation_by_stmt() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 775
get_observation_by_stmt() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 603
get_observation_by_stmt() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 572
get_one_value() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_one_value() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 812
get_one_value() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 782
get_one_value() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 599
get_one_value() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
get_one_value_from_atom() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_one_value_from_atom() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_one_value_from_atom() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
get_one_value_from_definition() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_one_value_from_definition() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_one_value_from_definition() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
get_optimization_passes() (*in module angr.analyses.decompiler.optimization_passes*), 701
get_patch() (*angr.knowledge_plugins.patches.PatchManager method*), 522
get_phi_subvariables() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 565
get_phi_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 565
get_plugin() (*angr.knowledge_base.knowledge_base.KnowledgeBase method*), 531
get_plugin() (*angr.KnowledgeBase method*), 211
get_plugin() (*angr.misc.plugins.PluginHub method*),

223
get_plugin() (*angr.sim_state.SimState method*), 226
get_plugin() (*angr.SimState method*), 182
get_possible_len() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 276
get_post_dominators() (*angr.analyses.cdg.CDG method*), 673
get_predecessors() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
get_predecessors() (*angr.analyses.ddg.DDG method*), 748
get_predecessors() (*angr.analyses.vsa_ddg.VSA_DDG method*), 844
get_predecessors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 538
get_predecessors() (*angr.knowledge_plugins.cfg.CFGModel method*), 532
get_predecessors_and_jumpkind() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 540
get_predecessors_and_jumpkind() (*angr.knowledge_plugins.cfg.CFGModel method*), 533
get_predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 539
get_predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.CFGModel method*), 533
get_prototype() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 523
get_prototype() (*angr.procedures.definitions.SimCppLibrary method*), 479
get_prototype() (*angr.procedures.definitions.SimLibrary method*), 478
get_prototype() (*angr.procedures.definitions.SimSyscallLibrary method*), 482
get_prototype_type() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 523
get_reaching_definitions() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 793
get_reaching_definitions() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 772
get_reaching_definitions_by_insn() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 793
get_reaching_definitions_by_insn() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 772
get_reaching_definitions_by_node() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 793
get_reaching_definitions_by_node() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 772
get_real_len() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 276
get_recent_bbl_addrs() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
get_recent_n() (*angr.exploration_techniques.spiller.PickledStatesDb method*), 411
get_ref() (*angr.state_hierarchy.StateHierarchy method*), 389
get_ref() (*angr.StateHierarchy method*), 180
get_reg_name() (*angr.analyses.identifier.identify.Identifier static method*), 836
get_register_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 761
get_register_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 597
get_register_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 577
get_regs() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
get_run() (*angr.annocfg.AnnotatedCFG method*), 872
get_same_length_constraints() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 276
get_segment_register_name() (*angr.simos.linux.SimLinux method*), 878
get_segment_register_name() (*angr.simos.windows.SimWindows method*), 880
get_size() (*angr.PTChunk method*), 209
get_size() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 300
get_size() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 302
get_sp() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 760
get_sp() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 809
get_sp() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 779
get_sp() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 596
get_sp() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 575
get_sp_offset() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 760
get_sp_offset() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 596

method), 596
get_sp_offset() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 575
get_stack_address() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 760
get_stack_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 809
get_stack_address() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 779
get_stack_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 596
get_stack_address() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 575
get_stack_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_stack_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_stack_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 577
get_stack_offset() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 759
get_stack_offset() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 808
get_stack_offset() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 778
get_stack_offset() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 822
get_stack_offset() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 595
get_stack_offset() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 574
get_stack_values() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 761
get_stack_values() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_stack_values() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 577
get_stdin_indices() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 276
get_stdout_indices() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 276
get_stop_details() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
get_stop_msg() (*angr.state_plugins.unicorn_engine.STOP static method*), 287
get_structurer_option() (*in module angr.analyses.decompiler.decompilation_options*), 697
get_stub() (*angr.procedures.definitions.SimCppLibrary method*), 479
get_stub() (*angr.procedures.definitions.SimLibrary method*), 477
get_stub() (*angr.procedures.definitions.SimSyscallLibrary method*), 482
get_sub_classes() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 637
get_sub_classes_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 637
get_sub_interfaces() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 636
get_sub_interfaces_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 636
get_subgraph() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 644
get_successors() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
get_successors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 538
get_successors() (*angr.knowledge_plugins.cfg.CFGModel method*), 532
get_successors_and_jumpkind() (*angr.analyses.cfg.cfg_base.CFGBase method*), 686
get_successors_and_jumpkind() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 539
get_successors_and_jumpkinds() (*angr.knowledge_plugins.cfg.CFGModel method*), 533
get_successors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 539
get_successors_and_jumpkinds() (*angr.knowledge_plugins.cfg.CFGModel method*), 532
get_super_classes() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 636
get_super_classes_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 636
get_superclass() (*angr.state_plugins.javavm_classloader.SimJavaVmCl method*), 294
get_symbolic_addrs() (*angr.storage.memory_mixins.convenient_mappings_mixin.Conve

method), 348
get_targets() (*angr.annocfg.AnnotatedCFG method*), 872
get_tmp_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 761
get_tmp_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 597
get_tmp_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 577
get_topological_order() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 644
get_type() (*angr.state_plugins.trace_additions.FormatInfo method*), 274
get_type() (*angr.state_plugins.trace_additions.FormatInfoDontCommit method*), 274
get_type() (*angr.state_plugins.trace_additions.FormatInfoIntToStr method*), 274
get_type() (*angr.state_plugins.trace_additions.FormatInfoStrToInt method*), 274
get_type_variable() (*angr.analyses.typehoon.typevars.TypeVariables method*), 831
get_unconstrained_simprocedure() (*angr.engines.soot.engine.SootMixin method*), 432
get_unified_local_vars() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 726
get_unified_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 564
get_unique_label() (*angr.knowledge_plugins.labels.Labels method*), 550
get_unique_strings() (*in module angr.flirt.build_sig*), 884
get_unique_symbol_name() (*angr.analyses.reassembler.SymbolManager method*), 852
get_uses() (*angr.knowledge_plugins.key_definitions.Uses method*), 581
get_uses() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 606
get_uses_by_insaddr() (*angr.knowledge_plugins.key_definitions.Uses method*), 582
get_uses_by_insaddr() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 607
get_uses_by_location() (*angr.knowledge_plugins.key_definitions.Uses method*), 582
get_uses_by_location() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 606
get_uses_with_expr() (*angr.knowledge_plugins.key_definitions.Uses method*), 581
get_uses_with_expr() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 606
get_value() (*angr.calling_conventions.SimArrayArg method*), 486
get_value() (*angr.calling_conventions.SimComboArg method*), 486
get_value() (*angr.calling_conventions.SimFunctionArgument method*), 484
get_value() (*angr.calling_conventions.SimLyingRegArg method*), 491
get_value() (*angr.calling_conventions.SimReferenceArgument method*), 487
get_value() (*angr.calling_conventions.SimRegArg method*), 485
get_value() (*angr.calling_conventions.SimStackArg method*), 485
get_value() (*angr.calling_conventions.SimStructArg method*), 486
get_value_from_atom() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_value_from_atom() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_value_from_atom() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
get_value_from_definition() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_value_from_definition() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 598
get_value_from_definition() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578
get_values() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 762
get_values() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitions method*), 812
get_values() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 782
get_values() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 599
get_values() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 578

get_variable() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifierAILState method*), 709
get_variable() (*angr.state_plugins.debug_variables.SimDebugVariablePlugin method*), 309
get_variable_accesses() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 567
get_variable_accesses() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 564
get_variable_definitions() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase method*), 820
get_variable_definitions() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 822
get_variable_type() (*angr.knowledge_plugins.variables.variable_manager.VariableManager.Interval method*), 566
get_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 564
get_variables() (*angr.state_plugins.solver.SimSolver method*), 255
get_variables_by_offset() (*angr.keyed_region.KeyedRegion method*), 615
get_variables_without_writes() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 565
get_whitelisted_statements() (*angr.annocfg.AnnotatedCFG method*), 872
get_xrefs_by_dst() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 611
get_xrefs_by_dst_region() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 611
get_xrefs_by_ins_addr() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 611
get_xrefs_by_ins_addr_region() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 611
getpiece() (*angr.analyses.disassembly.DisassemblyPiece method*), 846
getstate() (*angr.calling_conventions.ArgSession method*), 487
getstate() (*angr.calling_conventions.SerializableCounter method*), 484
getstate() (*angr.calling_conventions.SerializableIterator method*), 484
getstate() (*angr.calling_conventions.SerializableListIterator method*), 484
getstate() (*angr.calling_conventions.SimCC.ArgSession method*), 489
getstate() (*angr.calling_conventions.UsercallArgSession method*), 487
getstate() (*angr.SimCC.ArgSession method*), 186
give_up_on_memory_tracking() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 818
global_addr (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 589
GlobalDescriptorTable (*class in angr.simos.simos*), 876
GotoSimplifier (*class in angr.analyses.decompiler.region_simplifiers.goto*), 717
GOTPLTEntry (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 542
GOTPLTEntry (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
gp (*angr.analyses.cfg.Interval.CFGJob attribute*), 652
gp_register_read_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.MIPSGPHo method*), 665
gp_register_write_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.MIPSGPHo method*), 665
graph (*angr.analyses.cdg.CDG property*), 673
graph (*angr.analyses.cfg.cfg_base.CFGBase property*), 647
graph (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 644
graph (*angr.analyses.cfg.cfg_fast.CFGFast property*), 655
graph (*angr.analyses.data_dep.data_dependency_analysis.DataDependenc property*), 866
graph (*angr.analyses.ddg.DDG property*), 747
graph (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 700
graph (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis property*), 621
graph (*angr.analyses.reaching_definitions.dep_graph.DepGraph property*), 795
graph (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 536
graph (*angr.knowledge_plugins.cfg.CFGModel attribute*), 530
graph (*angr.knowledge_plugins.functions.function.Function property*), 557
graph_ex() (*angr.knowledge_plugins.functions.function.Function method*), 557
graph_with_successors (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 700
GraphChangedNotification, 690
GraphRegion (*class in angr.analyses.decompiler.graph_region*),

699
GraphUtils (*class in angr.utils.graph*), 888
GraphVisitor (*class in angr.analyses.forward_analysis.visitors.graph*), 624
GT (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.CaseCriterion attribute*), 719
GUARD (*angr.analyses.reaching_definitions.AtomKind attribute*), 766
guard (*angr.errors.SimError attribute*), 896
GUARD (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 584
GuardUse (*class in angr.analyses.reaching_definitions*), 769
GuardUse (*class in angr.knowledge_plugins.key_definitions.atoms*), 586
guess_prototype() (*angr.calling_conventions.SimCC static method*), 489
guess_prototype() (*angr.calling_conventions.SimCCSoot static method*), 501
guess_prototype() (*angr.SimCC static method*), 186
guess_value_type() (*in module angr.analyses.decompiler.structured_codegen.c*), 724
guessed_cc (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 801
guessed_cc (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
guessed_prototype (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 801
guessed_prototype (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787

H

handle() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker class method*), 739
handle_CAssignment() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker class method*), 739
handle_CAssignment() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecastsImplicit class method*), 740
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker class method*), 740
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecastsImplicit class method*), 740
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.c.PointerArithmeticFixer class method*), 741
handle_CDoWhileLoop() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker class method*), 739
handle_CForLoop() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker class method*), 739
handle_CFunction() (*angr.analyses.decompiler.structured_codegen.c.CS class method*), 739
handle_CFunctionCall() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCode class method*), 739
handle_CFunctionCall() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecastsI class method*), 740
handle_CGoto() (*angr.analyses.decompiler.structured_codegen.c.CStruct class method*), 739
handle_CIfBreak() (*angr.analyses.decompiler.structured_codegen.c.CSt class method*), 739
handle_CIfElse() (*angr.analyses.decompiler.structured_codegen.c.CStru class method*), 739
handle_CIndexedVariable() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCode class method*), 740
handle_CITE() (*angr.analyses.decompiler.structured_codegen.c.CStructu class method*), 740
handle_CReturn() (*angr.analyses.decompiler.structured_codegen.c.CStru class method*), 739
handle_CReturn() (*angr.analyses.decompiler.structured_codegen.c.Make class method*), 740
handle_CStatements() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCode class method*), 739
handle_CSwitchCase() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCode class method*), 739
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.CS class method*), 740
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.Fie class method*), 740
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.Ma class method*), 740
handle_CUnaryOp() (*angr.analyses.decompiler.structured_codegen.c.CSt class method*), 740
handle_CVariableField() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCode class method*), 740
handle_CWhileLoop() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCode class method*), 739
handle_default() (*angr.analyses.decompiler.structured_codegen.c.CStru class method*), 739
handle_exception() (*angr.SimOS method*), 169
handle_exception() (*angr.simos.simos.SimOS method*), 876
handle_exception() (*angr.simos.windows.SimWindows method*), 880
handle_external_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHa method*), 804
handle_external_function()

(*angr.analyses.reaching_definitions.FunctionHandler method*), 784
handle_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 803
handle_function() (*angr.analyses.reaching_definitions.FunctionHandler method*), 784
handle_generic_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 804
handle_generic_function() (*angr.analyses.reaching_definitions.FunctionHandler method*), 784
handle_indirect_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 804
handle_indirect_function() (*angr.analyses.reaching_definitions.FunctionHandler method*), 784
handle_local_function() (*angr.analyses.find_objects_static.NewFunctionHandler method*), 845
handle_local_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 804
handle_local_function() (*angr.analyses.reaching_definitions.FunctionHandler method*), 784
handle_pcode_block() (*angr.engines.pcode.emulate.PcodeEmulatorMixin method*), 444
hardcopy (*angr.state_plugins.history.TreeIter property*), 270
HAS_BITSHIFTS (*angr.analyses.code_tagging.CodeTags attribute*), 674
has_bitshifts() (*angr.analyses.code_tagging.CodeTagging method*), 674
has_blocks (*angr.analyses.cfg.segment_list.SegmentList property*), 673
has_clobbered() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 801
has_clobbered() (*angr.analyses.reaching_definitions.FunctionHandler method*), 787
has_default_value (*angr.sim_state_options.StateOption property*), 228
has_function_manager() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 567
has_implementation() (*angr.procedures.definitions.SimCppLibrary method*), 479
has_implementation() (*angr.procedures.definitions.SimLibrary method*), 478
has_implementation() (*angr.procedures.definitions.SimSyscallLibrary method*), 482
has_join() (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method*), 622
has_load (*angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter attribute*), 716
has_memory_dep (*angr.state_plugins.unicorn_engine.VEXStmtDetails attribute*), 285
has_metadata() (*angr.procedures.definitions.SimCppLibrary method*), 479
has_metadata() (*angr.procedures.definitions.SimLibrary method*), 478
has_metadata() (*angr.procedures.definitions.SimSyscallLibrary method*), 482
has_model() (*angr.knowledge_plugins.key_definitions.key_definition_manager.KeyDefinitionManager method*), 592
has_model() (*angr.knowledge_plugins.key_definitions.KeyDefinitionManager method*), 572
has_nonlabel_statements() (*in module angr.analyses.decompiler.utils*), 742
has_plugin() (*angr.knowledge_base.knowledge_base.KnowledgeBase method*), 521
has_plugin() (*angr.KnowledgeBase method*), 211
has_plugin() (*angr.misc.plugins.PluginHub method*), 223
has_plugin() (*angr.sim_state.SimState method*), 226
has_plugin() (*angr.SimState method*), 182
has_plugin_preset (*angr.misc.plugins.PluginHub property*), 223
has_prototype() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 523
has_prototype() (*angr.procedures.definitions.SimCppLibrary method*), 480
has_prototype() (*angr.procedures.definitions.SimLibrary method*), 478
has_prototype() (*angr.procedures.definitions.SimSyscallLibrary method*), 482
has_remote (*angr.knowledge_plugins.sync.sync_controller.SyncController property*), 608
has_return (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
has_return (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
has_return (*angr.knowledge_plugins.functions.function.Function property*), 558
HAS_VSA (*angr.analyses.code_tagging.CodeTags attribute*), 674
has_sql() (*angr.analyses.code_tagging.CodeTagging method*), 674
has_statements (*angr.engines.pcode.lifter.IRSB property*), 438
has_store() (*angr.analyses.decompiler.region_simplifiers.expr_folding.StackCanonicalizer method*), 717
has_super_class() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy

method), 636
has_symbolic_exit (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 285
has_tmpexpr() (*angr.analyses.propagator.engine_ail.SimEnginePropagatorAIL static method*), 754
has_type_variable_for() (*angr.analyses.typehoon.typevars.TypeVariables method*), 831
has_unresolved_calls (*angr.knowledge_plugins.functions.function.Function property*), 554
has_unresolved_jumps (*angr.knowledge_plugins.functions.function.Function property*), 554
HAS_XOR (*angr.analyses.code_tagging.CodeTags attribute*), 674
has_xor() (*angr.analyses.code_tagging.CodeTagging method*), 674
HasCallExprWalker (*class in angr.analyses.decompiler.block_simplifier*), 692
HasCallNotification, 691
HasField (*class in angr.analyses.typehoon.typevars*), 832
HasNext (*angr.analyses.loop_analysis.VariableTypes attribute*), 837
head (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 700
head (*angr.analyses.decompiler.structuring.structurer_nodes.LoopNode attribute*), 689
heap (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
heap (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 809
heap (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
heap (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594
heap (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
heap_address() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 765
heap_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 807
heap_address() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 777
heap_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 601
heap_address() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 581
heap_allocator (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState attribute*), 807
heap_allocator (*angr.analyses.reaching_definitions.ReachingDefinitionsState attribute*), 777
heap_definitions (*angr.analyses.reaching_definitions.LiveDefinitions property*), 758
heap_definitions (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions property*), 594
heap_definitions (*angr.knowledge_plugins.key_definitions.LiveDefinitions property*), 574
heap_offset (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 589
heap_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
heap_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 809
heap_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
heap_uses (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594
heap_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
HeapAddress (*class in angr.knowledge_plugins.key_definitions.heap_address*), 591
HeapAllocator (*class in angr.analyses.reaching_definitions.heap_allocator*), 797
HeavyPcodeMixin (*class in angr.engines.pcode.engine*), 434
height() (*angr.analyses.disassembly.Comment method*), 849
height() (*angr.analyses.disassembly.DisassemblyPiece method*), 846
height() (*angr.analyses.disassembly.FunctionStart method*), 847
hexdump() (*angr.storage.memory_mixins.hex_dumper_mixin.HexDumperMixin method*), 341
HexDumperMixin (*class in angr.storage.memory_mixins.hex_dumper_mixin*), 341
highlight() (*angr.analyses.disassembly.DisassemblyPiece method*), 846
history (*angr.sim_state.SimState attribute*), 225
history (*angr.SimState attribute*), 181
history_contains() (*angr.state_hierarchy.StateHierarchy method*), 390
history_contains() (*angr.StateHierarchy method*), 180
history_predecessors() (*angr.state_hierarchy.StateHierarchy method*), 390
history_predecessors() (*angr.StateHierarchy method*), 180
history_successors() (*angr.state_hierarchy.StateHierarchy method*), 389
history_successors() (*angr.StateHierarchy method*),

180
HistoryIter (*class in angr.state_plugins.history*), 270
HistoryTrackingMixin (*class in angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin*), 360
Hook (*class in angr.analyses.disassembly*), 847
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.ExitHandler static method*), 664
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegisterInitializer method*), 665
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.StoreHandler static method*), 664
hook() (*angr.analyses.find_objects_static.NewFunctionHandler method*), 845
hook() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 803
hook() (*angr.analyses.reaching_definitions.FunctionHandler method*), 783
hook() (*angr.Project method*), 164
hook() (*angr.project.Project method*), 214
hook() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
hook_add() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 288
hook_after() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.ExitHandler method*), 664
hook_before() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.ExitHandler method*), 664
hook_del() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 288
hook_reset() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 288
hook_symbol() (*angr.Project method*), 165
hook_symbol() (*angr.project.Project method*), 215
hooked_by() (*angr.Project method*), 165
hooked_by() (*angr.project.Project method*), 214
HookNode (*class in angr.codenode*), 874
HooksMixin (*class in angr.engines.hook*), 430

I

id (*angr.angrdb.models.DbCFGModel attribute*), 677
id (*angr.angrdb.models.DbComment attribute*), 679
id (*angr.angrdb.models.DbFunction attribute*), 677
id (*angr.angrdb.models.DbInformation attribute*), 676
id (*angr.angrdb.models.DbKnowledgeBase attribute*), 676
id (*angr.angrdb.models.DbLabel attribute*), 679
id (*angr.angrdb.models.DbObject attribute*), 676
id (*angr.angrdb.models.DbStructuredCode attribute*), 678
id (*angr.angrdb.models.DbVariableCollection attribute*), 678
id (*angr.angrdb.models.DbXRefs attribute*), 678
id (*angr.exploration_techniques.spiller_db.PickledState attribute*), 412
ident (*angr.analyses.disassembly.DisassemblyPiece attribute*), 846
ident (*angr.analyses.disassembly.OperandPiece attribute*), 849
ident (*angr.angrdb.models.DbCFGModel attribute*), 677
ident (*angr.angrdb.models.DbVariableCollection attribute*), 678
ident (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 536
ident (*angr.knowledge_plugins.cfg.CFGModel attribute*), 529
ident (*angr.sim_variable.SimVariable attribute*), 503
identical_blocks (*angr.analyses.bindiff.BinDiff property*), 632
identical_blocks (*angr.analyses.bindiff.FunctionDiff property*), 630
identical_functions (*angr.analyses.bindiff.BinDiff property*), 631
Identifier (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
Identifier (*class in angr.analyses.identifier.identify*), 836
identify_func() (*angr.analyses.identifier.identify.Identifier method*), 836
idom (*angr.analyses.decompiler.structuring.structurer_nodes.CodeNode property*), 686
idx (*angr.analyses.decompiler.structuring.structurer_nodes.MultiNode attribute*), 685
idx (*angr.analyses.typehoon.typevars.TypeVariable attribute*), 831
IfElseFlattener (*class in angr.analyses.decompiler.region_simplifiers.ifelse*), 718
iffalse (*angr.analyses.decompiler.structured_codegen.c.CITE attribute*), 736
IfSimplifier (*class in angr.analyses.decompiler.region_simplifiers.if_*), 717
iftrue (*angr.analyses.decompiler.structured_codegen.c.CITE attribute*), 736
IFUNC_HINTS (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 651
immediate_dominators() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 643
immediate_postdominators() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 643
import_binsync() (*in module angr.knowledge_plugins.sync.sync_controller*), 607
ImportedLine (*class in*

angr.analyses.decompiler.structured_codegen.dwarf_import), 741
ImportSourceCode (*class in angr.analyses.decompiler.structured_codegen.dwarf_import*), 741
inc_active_workers() (*angr.distributed.server.Server method*), 900
inc_active_workers() (*angr.Server method*), 210
includes() (*angr.keyed_region.RegionObject method*), 614
includes() (*angr.storage.memory_object.SimMemoryObject method*), 334
includes_function() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 790
IncompleteSwitchCaseHeadStatement (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 689
IncompleteSwitchCaseNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 688
inconsistent (*angr.analyses.stack_pointer_tracker.StackPointerTracker property*), 818
inconsistent_for() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 818
indent_str() (*angr.analyses.decompiler.structured_codegen.c.CConstruct static method*), 725
index (*angr.utils.graph.ContainerNode attribute*), 888
indirect_jumps (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 656
indirect_jumps (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot attribute*), 670
IndirectJump (*class in angr.knowledge_plugins.cfg*), 528
IndirectJump (*class in angr.knowledge_plugins.cfg.indirect_jump*), 547
IndirectJumpResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.resolver*), 667
IndirectJumps (*class in angr.knowledge_plugins.indirect_jumps*), 549
IndirectJumpType (*class in angr.knowledge_plugins.cfg*), 529
IndirectJumpType (*class in angr.knowledge_plugins.cfg.indirect_jump*), 547
info (*angr.code_location.CodeLocation attribute*), 612
info (*angr.knowledge_plugins.functions.function.Function attribute*), 553
info (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
init_checker() (*in module angr.knowledge_plugins.sync.sync_controller*), 607
init_class() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 294
init_hierarchy() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 636
init_state() (*angr.SimHeapPTMalloc method*), 208
init_state() (*angr.SimStatePlugin method*), 163
init_state() (*angr.state_plugins.heap.heap_base.SimHeapBase method*), 298
init_state() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 306
init_state() (*angr.state_plugins.history.SimStateHistory method*), 267
init_state() (*angr.state_plugins.plugin.SimStatePlugin method*), 233
init_state() (*angr.state_plugins.posix.SimSystemPosix method*), 245
init_state() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 307
init_static_field() (*angr.simos.javavm.SimJavaVM static method*), 881
INITIAL_SP_32BIT (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 757
INITIAL_SP_32BIT (*angr.knowledge_plugins.key_definitions.live_definitions attribute*), 594
INITIAL_SP_32BIT (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
INITIAL_SP_64BIT (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 757
INITIAL_SP_64BIT (*angr.knowledge_plugins.key_definitions.live_definitions attribute*), 594
INITIAL_SP_64BIT (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
InitializationFinder (*class in angr.analyses.init_finder*), 861
initialize() (*angr.storage.pcap.PCAP method*), 335
initialize() (*angr.utils.mp.Initializer method*), 893
initialize_dominance_frontiers() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase method*), 820
initialize_gdt_x86() (*angr.simos.linux.SimLinux method*), 878
initialize_gdt_x86() (*angr.simos.windows.SimWindows method*), 880
initialize_segment_register_x64() (*angr.simos.linux.SimLinux method*), 877
initialize_segment_register_x64() (*angr.simos.windows.SimWindows method*), 880
initialize_variable_names() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 567

initialized_classes (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader property*), 294
initializer (*angr.analyses.decompiler.structured_codegen.c.CForLoop attribute*), 727
initializer (*angr.analyses.decompiler.structuring.structurer_nodes.LoopNode attribute*), 687
Initializer (*class in angr.utils.mp*), 893
InitialValueTag (*class in angr.knowledge_plugins.key_definitions.tag*), 605
inline_call() (*angr.sim_procedure.SimProcedure method*), 472
inline_call() (*angr.SimProcedure method*), 160
inner_step() (*angr.exploration_techniques.Threading method*), 397
inner_step() (*angr.exploration_techniques.threading.Threading method*), 413
input_state (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 547
input_state (*angr.knowledge_plugins.cfg.CFGENode attribute*), 527
input_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 565
ins_addr (*angr.analyses.cfg.cfg_fast.FunctionEdge attribute*), 650
ins_addr (*angr.analyses.decompiler.structured_codegen.base.InstructionMapping attribute*), 723
ins_addr (*angr.analyses.decompiler.structured_codegen.c.CStatement attribute*), 731
ins_addr (*angr.code_location.CodeLocation attribute*), 612
ins_addr (*angr.errors.SimError attribute*), 896
ins_addr (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
ins_addr (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
ins_addr (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 589
ins_addr (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 610
ins_addrs (*angr.state_plugins.history.SimStateHistory property*), 270
insert() (*angr.SimMount method*), 203
insert() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 252
insert() (*angr.state_plugins.filesystem.SimFilesystem method*), 250
insert() (*angr.state_plugins.filesystem.SimMount method*), 251
insert() (*angr.state_plugins.posix.PosixDevFS method*), 240
insert() (*angr.state_plugins.posix.PosixProcFS method*), 242
insert_asm() (*angr.analyses.reassembler.Reassembler method*), 857
insert_node() (*angr.analyses.decompiler.structuring.structurer_nodes.SequenceNode method*), 685
insert_node() (*in module angr.analyses.decompiler.utils*), 742
inserted_asm_after_label (*angr.analyses.reassembler.Reassembler property*), 856
inserted_asm_before_label (*angr.analyses.reassembler.Reassembler property*), 856
insn (*angr.block.CapstoneInsn attribute*), 220
insn_addr_to_memory_data (*angr.analyses.cfg.cfg_fast.CFGFast property*), 655
insn_addr_to_memory_data (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 536
insn_addr_to_memory_data (*angr.knowledge_plugins.cfg.CFGModel attribute*), 530
insn_observe() (*angr.knowledge_plugins.key_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 793
insn_observe() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 773
insn_opc_idx (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 610
insn_op_type (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 611
insns (*angr.block.DisassemblerBlock attribute*), 220
insns (*angr.engines.pcode.lifter.PcodeDisassemblerBlock attribute*), 435
inspect (*angr.sim_state.SimState attribute*), 225
inspect (*angr.SimState attribute*), 181
InspectMixinHigh (*class in angr.storage.memory_mixins.clouseau_mixin*), 345
InstMatchPredicate
Instruction (*class in angr.analyses.disassembly*), 847
Instruction (*class in angr.analyses.reassembler*), 852
instruction_addresses (*angr.engines.pcode.lifter.IRSB property*), 438
instruction_addresses() (*angr.analyses.reassembler.BasicBlock method*), 853
instruction_addresses() (*angr.analyses.reassembler.Procedure method*), 854
instruction_addrs (*angr.Block property*), 170
instruction_addrs (*angr.block.Block property*), 221
instruction_addrs (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
instruction_addrs (*angr.knowledge_plugins.cfg.CFGNode*

attribute), 526
instruction_size() (*angr.knowledge_plugins.functions.function.Function method*), 558
InstructionError, 850
InstructionMapping (*class in angr.analyses.decompiler.structured_codegen.base*), 723
InstructionMappingElement (*class in angr.analyses.decompiler.structured_codegen.base*), 723
instructions (*angr.analyses.reassembler.Reassembler property*), 855
instructions (*angr.Block property*), 170
instructions (*angr.block.Block property*), 221
instructions (*angr.engines.pcode.lifter.IRSB property*), 438
Int (*class in angr.analyses.typehoon.typeconsts*), 834
Int1 (*class in angr.analyses.typehoon.typeconsts*), 834
Int128 (*class in angr.analyses.typehoon.typeconsts*), 834
Int16 (*class in angr.analyses.typehoon.typeconsts*), 834
int2base() (*in module angr.state_plugins.trace_additions*), 274
Int32 (*class in angr.analyses.typehoon.typeconsts*), 834
Int64 (*class in angr.analyses.typehoon.typeconsts*), 834
Int8 (*class in angr.analyses.typehoon.typeconsts*), 834
int_args (*angr.calling_conventions.SimCC property*), 488
int_args (*angr.SimCC property*), 185
int_iter (*angr.calling_conventions.ArgSession attribute*), 487
int_iter (*angr.calling_conventions.SimCC.ArgSession attribute*), 488
int_iter (*angr.SimCC.ArgSession attribute*), 186
int_len_mod (*angr.procedures.stubs.format_parser.FormatSpecifier attribute*), 474
int_sign (*angr.procedures.stubs.format_parser.FormatParser attribute*), 474
int_type() (*in module angr.analyses.typehoon.typeconsts*), 835
Integer (*angr.analyses.proximity_graph.ProximityNodeTypes attribute*), 863
Integer (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 542
Integer (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
IntegerProximityNode (*class in angr.analyses.proximity_graph*), 864
internal_objects (*angr.keyed_region.RegionObject property*), 614
interpret() (*angr.procedures.stubs.format_parser.FormatString method*), 473
intersection() (*angr.state_plugins.sim_action_object.SimActionObject method*), 468
invalidate_direct_next() (*angr.engines.pcode.lifter.IRSB method*), 438
inverted_idoms() (*in module angr.utils.graph*), 886
ip (*angr.sim_state.SimState property*), 225
ip (*angr.SimState property*), 182
IROp (*class in angr.analyses.disassembly*), 847
irsb (*angr.analyses.disassembly.IROp attribute*), 847
irsb (*angr.engines.pcode.lifter.Lifter attribute*), 440
irsb (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
irsb (*angr.engines.UberEngine attribute*), 427
irsb (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
irsb (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
IRSB (*class in angr.engines.pcode.lifter*), 435
irsb_from_node() (*angr.analyses.vfg.VFG method*), 844
is_a_jump_target() (*angr.analyses.decompiler.structuring.structurer_ba static method*), 690
is_alignment (*angr.knowledge_plugins.functions.function.Function attribute*), 553
is_alignment (*angr.knowledge_plugins.functions.soot_function.SootFunct attribute*), 561
is_alignment_mask() (*in module angr.utils.constants*), 885
is_arm (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 536
is_arm (*angr.knowledge_plugins.cfg.CFGModel attribute*), 529
is_base (*angr.engines.light.data.SpOffset attribute*), 750
is_bool_expr() (*angr.analyses.decompiler.peephole_optimizations.base.I static method*), 713
is_bounded() (*angr.state_plugins.uc_manager.SimUCManager method*), 280
is_bytes (*angr.storage.memory_object.SimMemoryObject attribute*), 334
is_class_initialized() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 294
is_concrete() (*angr.analyses.init_finder.SimEngineInitFinderVEX static method*), 861
is_cross_referenced() (*angr.analyses.vtable.VtableFinder method*), 845
is_default_name (*angr.knowledge_plugins.functions.function.Function attribute*), 553
is_default_name (*angr.knowledge_plugins.functions.soot_function.SootF attribute*), 560
is_empty (*angr.engines.successors.SimSuccessors property*), 429
is_empty (*angr.keyed_region.RegionObject property*), 613
is_empty (*angr.storage.memory_mixins.regioned_memory.region_data.Re

property), 369
is_empty() (angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink method), 815
is_empty_node() (in module angr.analyses.decompiler.utils), 742
is_empty_or_label_only_node() (in module angr.analyses.decompiler.utils), 742
is_exception (angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute), 650
is_expr (angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute), 730
is_false() (angr.state_plugins.solver.SimSolver method), 259
is_fp_arg() (angr.calling_conventions.SimCC method), 488
is_fp_arg() (angr.SimCC method), 185
is_fp_value() (angr.calling_conventions.SimCC static method), 489
is_fp_value() (angr.SimCC static method), 186
is_free() (angr.PTChunk method), 209
is_free() (angr.state_plugins.heap.heap_freelist.Chunk method), 300
is_free() (angr.state_plugins.heap.heap_ptmalloc.PTChunk method), 303
IS_FUNCTION (angr.sim_procedure.SimProcedure attribute), 472
IS_FUNCTION (angr.SimProcedure attribute), 159
is_function() (angr.analyses.vtable.VtableFinder method), 845
is_function_argument (angr.sim_variable.SimVariable property), 503
is_global_variable_address() (angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase method), 822
is_heap_address() (angr.analyses.reaching_definitions.LiveDefinitions static method), 765
is_heap_address() (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState static method), 807
is_heap_address() (angr.analyses.reaching_definitions.ReachingDefinitionsState static method), 778
is_heap_address() (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method), 601
is_heap_address() (angr.knowledge_plugins.key_definitions.LiveDefinitions static method), 580
is_hex() (in module angr.analyses.reassembler), 851
is_hook (angr.codenode.BlockNode attribute), 873
is_hook (angr.codenode.CodeNode attribute), 873
is_hook (angr.codenode.HookNode attribute), 874
is_hook (angr.codenode.SyscallNode attribute), 874
is_hooked() (angr.Project method), 165
is_hooked() (angr.project.Project method), 214
is_immediate (angr.analyses.reassembler.Operand property), 852
is_in_readonly_section() (in module angr.utils.loader), 890
is_in_readonly_segment() (in module angr.utils.loader), 890
is_java (angr.sim_procedure.SimProcedure property), 473
is_java (angr.SimProcedure property), 160
is_jumptable (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver attribute), 663
is_machine_word_size_type() (in module angr.analyses.decompiler.structured_codegen.c), 724
is_not_in_cfg() (angr.analyses.veritesting.Veritesting method), 839
is_occupied() (angr.analyses.cfg.segment_list.SegmentList method), 671
is_on_stack (angr.analyses.reaching_definitions.MemoryLocation property), 769
is_on_stack (angr.knowledge_plugins.key_definitions.atoms.MemoryLocation property), 588
is_on_stack (angr.storage.memory_mixins.regioned_memory.region_data attribute), 369
is_overbound() (angr.analyses.veritesting.Veritesting method), 839
is_pc() (in module angr.utils.loader), 890
is_phi_variable() (angr.knowledge_plugins.variables.variable_manager method), 565
is_plt (angr.analyses.reassembler.Procedure property), 854
is_plt (angr.knowledge_plugins.functions.function.Function attribute), 553
is_plt (angr.knowledge_plugins.functions.soot_function.SootFunction attribute), 560
is_prev_free() (angr.PTChunk method), 209
is_prev_free() (angr.state_plugins.heap.heap_ptmalloc.PTChunk method), 303
is_prototype_guessed (angr.knowledge_plugins.functions.function.Function attribute), 553
is_prototype_guessed (angr.knowledge_plugins.functions.soot_function.SootFunction attribute), 561
is_pyinstaller() (in module angr.utils.env), 886
is_simple_jump_node() (in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier), 720
is_simprocedure (angr.knowledge_plugins.cfg.cfg_node.CFGNode property), 546
is_simprocedure (angr.knowledge_plugins.cfg.CFGNode property), 526
is_simprocedure (angr.knowledge_plugins.functions.function.Function attribute), 553
is_simprocedure (angr.knowledge_plugins.functions.soot_function.SootFunction attribute), 560

is_special (*angr.engines.pcode.behavior.OpBehavior attribute*), 445
is_special (*angr.engines.pcode.behavior.OpBehaviorBoolAnd attribute*), 459
is_special (*angr.engines.pcode.behavior.OpBehaviorBoolNegate attribute*), 458
is_special (*angr.engines.pcode.behavior.OpBehaviorBoolOr attribute*), 459
is_special (*angr.engines.pcode.behavior.OpBehaviorBoolXor attribute*), 458
is_special (*angr.engines.pcode.behavior.OpBehaviorCopy attribute*), 446
is_special (*angr.engines.pcode.behavior.OpBehaviorEqual attribute*), 447
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatAbs attribute*), 461
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatAdd attribute*), 460
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatCeil attribute*), 462
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatDiv attribute*), 460
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatEqual attribute*), 459
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatFloat2Float attribute*), 462
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatFloor attribute*), 463
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatInt2Float attribute*), 462
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatLess attribute*), 460
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatLessEqual attribute*), 460
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatMult attribute*), 461
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatNan attribute*), 460
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatNeg attribute*), 461
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatNotEqual attribute*), 459
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatRound attribute*), 463
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatSqrt attribute*), 462
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatSub attribute*), 461
is_special (*angr.engines.pcode.behavior.OpBehaviorFloatTrunc attribute*), 462
is_special (*angr.engines.pcode.behavior.OpBehaviorInt2Comp attribute*), 452
is_special (*angr.engines.pcode.behavior.OpBehaviorIntAdd attribute*), 450
is_special (*angr.engines.pcode.behavior.OpBehaviorIntAnd attribute*), 453
is_special (*angr.engines.pcode.behavior.OpBehaviorIntCarry attribute*), 451
is_special (*angr.engines.pcode.behavior.OpBehaviorIntDiv attribute*), 456
is_special (*angr.engines.pcode.behavior.OpBehaviorIntLeft attribute*), 454
is_special (*angr.engines.pcode.behavior.OpBehaviorIntLess attribute*), 448
is_special (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual attribute*), 449
is_special (*angr.engines.pcode.behavior.OpBehaviorIntMult attribute*), 456
is_special (*angr.engines.pcode.behavior.OpBehaviorIntNegate attribute*), 453
is_special (*angr.engines.pcode.behavior.OpBehaviorIntOr attribute*), 454
is_special (*angr.engines.pcode.behavior.OpBehaviorIntRem attribute*), 457
is_special (*angr.engines.pcode.behavior.OpBehaviorIntRight attribute*), 455
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSborrow attribute*), 452
is_special (*angr.engines.pcode.behavior.OpBehaviorIntScarry attribute*), 451
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSdiv attribute*), 456
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSext attribute*), 450
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSless attribute*), 447
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual attribute*), 448
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSrem attribute*), 457
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSright attribute*), 455
is_special (*angr.engines.pcode.behavior.OpBehaviorIntSub attribute*), 450
is_special (*angr.engines.pcode.behavior.OpBehaviorIntXor attribute*), 453
is_special (*angr.engines.pcode.behavior.OpBehaviorIntZext attribute*), 449
is_special (*angr.engines.pcode.behavior.OpBehaviorNotEqual attribute*), 447
is_special (*angr.engines.pcode.behavior.OpBehaviorPiece attribute*), 463
is_special (*angr.engines.pcode.behavior.OpBehaviorPopcount attribute*), 464
is_special (*angr.engines.pcode.behavior.OpBehaviorSubpiece attribute*), 463
is_stack (*angr.storage.memory_mixins.regioned_memory.region_meta_m property*), 371

is_stack_address() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 759
is_stack_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 808
is_stack_address() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 778
is_stack_address() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 822
is_stack_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 595
is_stack_address() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 574
is_statement_terminating() (*in module angr.analyses.decompiler.utils*), 743
is_stored() (*angr.vaults.Vault method*), 617
is_stored() (*angr.vaults.VaultDict method*), 618
is_subclass() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 636
is_subclass_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 636
is_symbol_hooked() (*angr.Project method*), 166
is_symbol_hooked() (*angr.project.Project method*), 215
is_symbolic (*angr.state_plugins.sim_action.SimAction property*), 466
is_symbolic (*angr.state_plugins.sim_action.SimActionConstraint property*), 467
is_symbolic (*angr.state_plugins.sim_action.SimActionData property*), 468
is_symbolic (*angr.state_plugins.sim_action.SimActionExit property*), 467
is_symbolic (*angr.state_plugins.sim_action.SimActionOperation property*), 467
is_syscall (*angr.analyses.cfg.cfg_emulated.CFGJob property*), 640
is_syscall (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
is_syscall (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
is_syscall (*angr.knowledge_plugins.functions.function.Function attribute*), 553
is_syscall (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
is_syscall_addr() (*angr.SimOS method*), 169
is_syscall_addr() (*angr.simos.simos.SimOS method*), 876
is_syscall_addr() (*angr.simos.userland.SimUserland method*), 879
is_taint_impacting_stack_pointers() (*angr.analyses.backward_slice.BackwardSlice method*), 629
is_taint_related_to_ip() (*angr.analyses.backward_slice.BackwardSlice method*), 628
is_thumb_addr() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
is_top() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 758
is_top() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 807
is_top() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 777
is_top() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 821
is_top() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 595
is_top() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 574
is_tracking_memory (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTracker attribute*), 817
is_tracking_memory (*angr.analyses.stack_pointer_tracker.StackPointerTracker attribute*), 817
is_true() (*angr.state_plugins.solver.SimSolver method*), 258
is_unary (*angr.engines.pcode.behavior.OpBehavior attribute*), 445
is_unary (*angr.engines.pcode.behavior.OpBehaviorBoolAnd attribute*), 459
is_unary (*angr.engines.pcode.behavior.OpBehaviorBoolNegate attribute*), 458
is_unary (*angr.engines.pcode.behavior.OpBehaviorBoolOr attribute*), 459
is_unary (*angr.engines.pcode.behavior.OpBehaviorBoolXor attribute*), 458
is_unary (*angr.engines.pcode.behavior.OpBehaviorCopy attribute*), 446
is_unary (*angr.engines.pcode.behavior.OpBehaviorEqual attribute*), 447
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatAbs attribute*), 461
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatAdd attribute*), 460
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatCeil attribute*), 462
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatDiv attribute*), 460
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatEqual attribute*), 459
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatFloat2Float attribute*), 462
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatFloor attribute*), 463
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatInt2Float attribute*), 462
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatLess attribute*), 460
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatLessEqual*

attribute), 460
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatMult attribute*), 461
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatNan attribute*), 460
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatNeg attribute*), 461
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatNotEqual attribute*), 459
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatRound attribute*), 463
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatSqrt attribute*), 461
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatSub attribute*), 461
is_unary (*angr.engines.pcode.behavior.OpBehaviorFloatTrunc attribute*), 462
is_unary (*angr.engines.pcode.behavior.OpBehaviorInt2Comp attribute*), 452
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntAdd attribute*), 450
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntAnd attribute*), 453
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntCarry attribute*), 451
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntDiv attribute*), 456
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntLeft attribute*), 454
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntLess attribute*), 448
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual attribute*), 449
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntMult attribute*), 456
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntNegate attribute*), 453
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntOr attribute*), 454
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntRem attribute*), 457
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntRight attribute*), 455
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSborrow attribute*), 452
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntScarry attribute*), 451
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSdiv attribute*), 456
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSext attribute*), 450
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSless attribute*), 447
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual attribute*), 448
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSrem attribute*), 457
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSright attribute*), 455
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntSub attribute*), 450
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntXor attribute*), 453
is_unary (*angr.engines.pcode.behavior.OpBehaviorIntZext attribute*), 449
is_unary (*angr.engines.pcode.behavior.OpBehaviorNotEqual attribute*), 447
is_unary (*angr.engines.pcode.behavior.OpBehaviorPiece attribute*), 463
is_unary (*angr.engines.pcode.behavior.OpBehaviorPopcount attribute*), 464
is_unary (*angr.engines.pcode.behavior.OpBehaviorSubpiece attribute*), 463
is_using_outdated_def() (*angr.analyses.propagator.engine_ail.SimEnginePropagatorAIL method*), 754
is_va_start_amd64() (*angr.analyses.calling_convention.CallingConventionAnalysis method*), 634
is_value_set (*angr.state_plugins.unicorn_engine.MemoryValue attribute*), 284
is_value_symbolic (*angr.state_plugins.unicorn_engine.MemoryValue attribute*), 284
is_variable_used_at() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 563
is_visible_class() (*angr.analyses.soot_class_hierarchy.SootClassHier method*), 636
is_visible_method() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 636
IsArray (*class in angr.analyses.typehoon.typevars*), 833
ISPOMixin (*class in angr.storage.memory_mixins.paged_memory.pages.isp* 361
ite_exprs (*angr.analyses.decompiler.decompilation_cache.Decompilation attribute*), 697
ite_exprs (*angr.angrdb.models.DbStructuredCode attribute*), 678
ITEExprConverter (*class in angr.analyses.decompiler.optimization_passes.ite_expr_converter* 706
items() (*angr.analyses.ddg.LiveDefinitions method*), 746
items() (*angr.analyses.decompiler.structured_codegen.base.InstructionMa method*), 723
items() (*angr.analyses.decompiler.structured_codegen.base.PositionMapp method*), 723
iteritems() (*angr.knowledge_plugins.labels.Labels*

method), 549
items() (*angr.knowledge_plugins.patches.PatchManager method*), 522
items() (*angr.state_plugins.globals.SimStateGlobals method*), 279
items() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 351
iter_own() (*angr.knowledge_plugins.types.TypesStore method*), 549
iterator (*angr.analyses.decompiler.structured_codegen.c.CExpression attribute*), 728
iterator (*angr.analyses.decompiler.structuring.structurer_nodes.LoopNode attribute*), 687
Iterator (*angr.analyses.loop_analysis.VariableTypes attribute*), 837
itervariables() (*angr.analyses.ddg.LiveDefinitions method*), 746

J

javavm_memory (*angr.sim_state.SimState property*), 226
javavm_memory (*angr.SimState property*), 182
javavm_registers (*angr.sim_state.SimState property*), 226
javavm_registers (*angr.SimState property*), 182
JavaVmMemory (*class in angr.storage.memory_mixins*), 339
JavaVmMemoryMixin (*class in angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin*), 376
jni_references (*angr.sim_state.SimState attribute*), 225
jni_references (*angr.SimState attribute*), 181
job (*angr.analyses.forward_analysis.job_info.JobInfo property*), 622
job_type (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
JobInfo (*class in angr.analyses.forward_analysis.job_info*), 622
jobs (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis property*), 622
jump() (*angr.sim_procedure.SimProcedure method*), 473
jump() (*angr.SimProcedure method*), 160
jump_guards (*angr.state_plugins.history.SimStateHistory property*), 269
jump_sources (*angr.state_plugins.history.SimStateHistory property*), 269
jump_tables (*angr.analyses.cfg.cfg_fast.CFGFast property*), 655
jump_tables (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 536
jump_tables (*angr.knowledge_plugins.cfg.CFGModel attribute*), 530
jump_targets (*angr.state_plugins.history.SimStateHistory property*), 269
jumpkind (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
jumpkind (*angr.engines.pcode.lifter.ExitStatement attribute*), 435
jumpkind (*angr.engines.pcode.lifter.IRSB attribute*), 437
jumpkind (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
jumpkind (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
jumpkinds (*angr.state_plugins.history.SimStateHistory property*), 269
jumpout_site (*angr.knowledge_plugins.functions.function.Function property*), 555
jumptable (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
jumptable (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
jumptable_addr (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
jumptable_addr (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
Jumptable_AddressComputed (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 547
Jumptable_AddressComputed (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 529
Jumptable_AddressLoadedFromMemory (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 547
Jumptable_AddressLoadedFromMemory (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 529
jumptable_entries (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
jumptable_entries (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
jumptable_entry_size (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
jumptable_entry_size (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
jumptable_size (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
jumptable_size (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
JumpTableEntryConditionRewriter (*class in angr.analyses.decompiler.jumptable_entry_condition_rewriter*), 700
JumpTableProcessor (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 664
JumpTableProcessorState (*class in*

angr.analyses.cfg.indirect_jump_resolvers.jumptable), 663
JumpTableResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 665
JumpTargetBaseAddr (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 662
JumpTargetCollector (*class in angr.analyses.decompiler.jump_target_collector*), 700

K

K (*angr.knowledge_base.knowledge_base.KnowledgeBase attribute*), 521
K (*angr.KnowledgeBase attribute*), 211
kb (*angr.analyses.analysis.Analysis attribute*), 621
kb (*angr.analyses.backward_slice.BackwardSlice attribute*), 629
kb (*angr.analyses.binary_optimizer.BinaryOptimizer attribute*), 860
kb (*angr.analyses.bindiff.BinDiff attribute*), 632
kb (*angr.analyses.boyscout.BoyScout attribute*), 632
kb (*angr.analyses.callee_cleanup_finder.CalleeCleanupFinder attribute*), 861
kb (*angr.analyses.calling_convention.CallingConventionAnalysis attribute*), 634
kb (*angr.analyses.cdg.CDG attribute*), 673
kb (*angr.analyses.cfg.cfb.CFBlanket attribute*), 638
kb (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 656
kb (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot attribute*), 670
kb (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.ConstantValueManager attribute*), 663
kb (*angr.analyses.class_identifier.ClassIdentifier attribute*), 846
kb (*angr.analyses.code_tagging.CodeTagging attribute*), 674
kb (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis attribute*), 635
kb (*angr.analyses.congruency_check.CongruencyCheck attribute*), 859
kb (*angr.analyses.data_dep.data_dependency_analysis.DataDependencyGraphAnalysis attribute*), 866
kb (*angr.analyses.ddg.DDG attribute*), 749
kb (*angr.analyses.decompiler.ail_simplifier.AILSimplifier attribute*), 692
kb (*angr.analyses.decompiler.block_simplifier.BlockSimplifier attribute*), 693
kb (*angr.analyses.decompiler.callsite_maker.CallSiteMaker attribute*), 693
kb (*angr.analyses.decompiler.clinic.Clinic attribute*), 695
kb (*angr.analyses.decompiler.decompiler.Decompiler attribute*), 698
kb (*angr.analyses.decompiler.optimization_passes.optimization_pass.BaseOptimizationPass property*), 703
kb (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimi attribute*), 713
kb (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimi attribute*), 712
kb (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimi attribute*), 712
kb (*angr.analyses.decompiler.region_identifier.RegionIdentifier attribute*), 714
kb (*angr.analyses.decompiler.region_simplifiers.region_simplifier.RegionSi attribute*), 718
kb (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGener attribute*), 739
kb (*angr.analyses.decompiler.structured_codegen.dwarf_import.ImportSou attribute*), 741
kb (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer attribute*), 691
kb (*angr.analyses.decompiler.structuring.recursive_structurer.RecursiveStru attribute*), 684
kb (*angr.analyses.disassembly.Disassembly attribute*), 850
kb (*angr.analyses.dominance_frontier.DominanceFrontier attribute*), 861
kb (*angr.analyses.find_objects_static.StaticObjectFinder attribute*), 846
kb (*angr.analyses.flirt.FlirtAnalysis attribute*), 750
kb (*angr.analyses.identifier.identify.Identifier attribute*), 836
kb (*angr.analyses.init_finder.InitializationFinder attribute*), 861
kb (*angr.analyses.loop_analysis.LoopAnalysis attribute*), 829
kb (*angr.analyses.loopfinder.LoopFinder attribute*), 837
kb (*angr.analyses.propagator.propagator.PropagatorAnalysis attribute*), 757
kb (*angr.analyses.proximity_graph.ProximityGraphAnalysis attribute*), 865
kb (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefini attribute*), 794
kb (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis attribute*), 773
kb (*angr.analyses.reassembler.Reassembler attribute*), 858
kb (*angr.analyses.soot_class_hierarchy.SootClassHierarchy attribute*), 637
kb (*angr.analyses.stack_pointer_tracker.StackPointerTracker attribute*), 818
kb (*angr.analyses.static_hooker.StaticHooker attribute*), 859
kb (*angr.analyses.typehoon.typehoon.Typehoon attribute*), 833
kb (*angr.analyses.variable_recovery.variable_recovery.VariableRecovery attribute*), 826

kb (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase attribute*), 820
kb (*angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFast attribute*), 825
kb (*angr.analyses.veritesting.Veritesting attribute*), 839
kb (*angr.analyses.vfg.VFG attribute*), 844
kb (*angr.analyses.vsa_ddg.VSA_DDG attribute*), 845
kb (*angr.analyses.vtable.VtableFinder attribute*), 845
kb (*angr.analyses.xrefs.XRefsAnalysis attribute*), 863
kb (*angr.Analysis attribute*), 178
kb (*angr.angrdb.models.DbCFGModel attribute*), 677
kb (*angr.angrdb.models.DbComment attribute*), 679
kb (*angr.angrdb.models.DbFunction attribute*), 677
kb (*angr.angrdb.models.DbLabel attribute*), 679
kb (*angr.angrdb.models.DbStructuredCode attribute*), 678
kb (*angr.angrdb.models.DbVariableCollection attribute*), 678
kb (*angr.angrdb.models.DbXRefs attribute*), 679
kb_id (*angr.angrdb.models.DbCFGModel attribute*), 677
kb_id (*angr.angrdb.models.DbComment attribute*), 679
kb_id (*angr.angrdb.models.DbFunction attribute*), 677
kb_id (*angr.angrdb.models.DbLabel attribute*), 679
kb_id (*angr.angrdb.models.DbStructuredCode attribute*), 678
kb_id (*angr.angrdb.models.DbVariableCollection attribute*), 678
kb_id (*angr.angrdb.models.DbXRefs attribute*), 679
keep_path() (*angr.annocfg.AnnotatedCFG method*), 873
key (*angr.angrdb.models.DbInformation attribute*), 676
KeyDefinitionManager (*class in angr.knowledge_plugins.key_definitions*), 572
KeyDefinitionManager (*class in angr.knowledge_plugins.key_definitions.key_definition_manager*), 592
KeyedRegion (*class in angr.keyed_region*), 614
keys() (*angr.knowledge_plugins.patches.PatchManager method*), 522
keys() (*angr.state_plugins.globals.SimStateGlobals method*), 279
keys() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 351
keys() (*angr.vaults.Vault method*), 617
keys() (*angr.vaults.VaultDict method*), 618
keys() (*angr.vaults.VaultDir method*), 618
keys() (*angr.vaults.VaultDirShelf method*), 618
KeyValueMemory (*class in angr.storage.memory_mixins*), 339
KeyValueMemoryMixin (*class in angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory_mixin*), 375
kill_and_add_definition() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 760
kill_and_add_definition() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 810
kill_and_add_definition() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 780
kill_and_add_definition() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 597
kill_and_add_definition() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 576
kill_def() (*angr.analyses.ddg.LiveDefinitions method*), 745
kill_def() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 774
kill_def() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 602
kill_def() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsState method*), 570
kill_definitions() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 760
kill_definitions() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 810
kill_definitions() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 780
kill_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 596
kill_definitions() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 576
kind (*angr.knowledge_plugins.key_definitions.definition.DefinitionMatchPredicate attribute*), 589
KnowledgeBase (*class in angr*), 211
KnowledgeBase (*class in angr.knowledge_base.knowledge_base*), 520
KnowledgeBasePlugin (*class in angr.knowledge_plugins.plugin*), 523
KnowledgeBaseSerializer (*class in angr.angrdb.serializers.kb*), 681
KnownAnalysesPlugin (*class in angr.knowledge_plugins.analysis*), 619
kwargs (*angr.utils.mp.Closure attribute*), 893

L

label (*angr.analyses.typehoon.typevars.DerivedTypeVariable attribute*), 831
label (*angr.storage.memory_object.SimLabeledMemoryObject attribute*), 334
Label (*class in angr.analyses.disassembly*), 847
Label (*class in angr.analyses.reassembler*), 851
label_got() (*angr.analyses.reassembler.SymbolManager method*), 852

LabeledMemory (*class in angr.storage.memory_mixins*), 339
LabeledPagesMixin (*class in angr.storage.memory_mixins.paged_memory.pages.list_page*), 355
LabelMergerMixin (*class in angr.storage.memory_mixins.label_merger_mixin*), 346
labels (*angr.angrdb.models.DbKnowledgeBase attribute*), 676
Labels (*class in angr.knowledge_plugins.labels*), 549
LabelsSerializer (*class in angr.angrdb.serializers.labels*), 681
LambdaAttrIter (*class in angr.state_plugins.history*), 270
LambdaIterIter (*class in angr.state_plugins.history*), 270
LARGE_SWITCH (*angr.analyses.code_tagging.CodeTags attribute*), 674
last_addr (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
last_addr (*angr.storage.memory_object.SimMemoryObject property*), 334
last_nonlabel_statement() (*in module angr.analyses.decompiler.utils*), 743
lazy_import() (*in module angr.utils.lazy_import*), 890
length (*angr.analyses.decompiler.structured_codegen.base.PositionMappingElement attribute*), 723
length (*angr.state_plugins.unicorn_engine.MEM_PATCH attribute*), 284
length (*angr.storage.memory_object.SimMemoryObject attribute*), 334
length_spec (*angr.procedures.stubs.format_parser.FormatSpecifier attribute*), 474
LengthLimiter (*class in angr.exploration_techniques*), 398
LengthLimiter (*class in angr.exploration_techniques.lengthlimiter*), 409
lhs (*angr.analyses.decompiler.structured_codegen.c.CAssignment attribute*), 729
lhs (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp attribute*), 734
lift() (*angr.analyses.typehoon.lifter.TypeLifter method*), 828
lift() (*angr.engines.pcode.lifter.Lifter method*), 440
lift() (*angr.engines.pcode.lifter.PcodeBasicBlockLifter method*), 441
lift() (*angr.engines.pcode.lifter.PcodeLifter method*), 442
lift() (*in module angr.engines.pcode.lifter*), 440
lift_pcode() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 443
lift_soot() (*angr.engines.soot.engine.SootMixin method*), 432
lift_vex() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 443
lifter (*class in angr.engines.pcode.lifter*), 439
lineage (*angr.state_plugins.history.SimStateHistory property*), 269
lineage() (*angr.state_hierarchy.StateHierarchy method*), 389
lineage() (*angr.StateHierarchy method*), 180
linux_syscall_update_error_reg() (*angr.calling_conventions.SimCCSyscall method*), 493
list_content (*angr.utils.dynamic_dictlist.DynamicDictList attribute*), 886
list_default_plugins() (*angr.misc.plugins.PluginPreset method*), 223
ListPage (*class in angr.storage.memory_mixins.paged_memory.pages.list_page*), 362
ListPagesMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 355
ListPagesWithLabelsMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 356
live_definitions (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionState attribute*), 807
live_definitions (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis attribute*), 777
LiveDefinitions (*class in angr.analyses.ddg*), 744
LiveDefinitions (*class in angr.analyses.reaching_definitions*), 757
LiveDefinitions (*class in angr.knowledge_plugins.key_definitions*), 573
LiveDefinitions (*class in angr.knowledge_plugins.key_definitions.live_definitions*), 593
LiveVariables (*class in angr.knowledge_plugins.variables.variable_manager*), 562
load (*class in angr.analyses.typehoon.typevars*), 832
load() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 818
load() (*angr.angrdb.db.AngrDB method*), 675
load() (*angr.angrdb.serializers.cfg_model.CFGModelSerializer static method*), 680
load() (*angr.angrdb.serializers.comments.CommentsSerializer static method*), 680
load() (*angr.angrdb.serializers.funcs.FunctionManagerSerializer static method*), 681
load() (*angr.angrdb.serializers.kb.KnowledgeBaseSerializer static method*), 681
load() (*angr.angrdb.serializers.labels.LabelsSerializer*

static method), 681
load() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 682
load() (*angr.angrdb.serializers.structured_code.StructuredCodeSerializer static method*), 683
load() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 683
load() (*angr.angrdb.serializers.xrefs.XRefsSerializer static method*), 682
load() (*angr.state_plugins.light_registers.SimLightRegisters method*), 267
load() (*angr.storage.memory_mixins.actions_mixin.ActionsMixin method*), 342
load() (*angr.storage.memory_mixins.actions_mixin.ActionsMixin method*), 342
load() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 346
load() (*angr.storage.memory_mixins.bvv_conversion_mixin.BVVConversionMixin method*), 341
load() (*angr.storage.memory_mixins.clouseau_mixin.InspectMixin method*), 346
load() (*angr.storage.memory_mixins.conditional_store_mixin.ConditionalStoreMixin method*), 346
load() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin.JavaVmMemoryMixin method*), 376
load() (*angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory_mixin.KeyValueMemoryMixin method*), 375
load() (*angr.storage.memory_mixins.MemoryMixin method*), 336
load() (*angr.storage.memory_mixins.name_resolution_mixin.NameResolutionMixin method*), 339
load() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 353
load() (*angr.storage.memory_mixins.paged_memory.pages.base.ListPage method*), 362
load() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MVListPage method*), 349
load() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 364
load() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.MemoryRegionMetaMixin method*), 372
load() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 366
load() (*angr.storage.memory_mixins.simple_interface_mixin.SimpleInterfaceMixin method*), 342
load() (*angr.storage.memory_mixins.size_resolution_mixin.SizeConcretizationMixin method*), 344
load() (*angr.storage.memory_mixins.size_resolution_mixin.SizeNormalizationMixin method*), 343
load() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method*), 375
load() (*angr.storage.memory_mixins.underconstrained_mixin.UnderconstrainedMixin method*), 342
load() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 347
load() (*angr.vaults.Vault method*), 617
load() (*angr.vaults.VaultDirShelf method*), 618
load_all_definitions() (*in module angr.procedures.definitions*), 483
load_array_element() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin.JavaVmMemoryMixin method*), 376
load_array_elements() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin.JavaVmMemoryMixin method*), 376
load_high_dwarf() (*angr.knowledge_plugins.debug_variables.DebugVariables method*), 570
load_from_dwarf() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 568
load_instance() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 683
load_shellcode() (*in module angr*), 167
load_shellcode() (*in module angr.project*), 212
load_signatures() (*in module angr.flirt*), 883
load_win32api_definitions() (*in module angr.procedures.definitions*), 483
load_with_labels() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 355
LoaderSerializer (*class in angrdb.serializers.loader*), 681
LoadHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 664
loads() (*angr.vaults.Vault method*), 617
LocalInPython (*angr.analyses.decompiler.ailment.typevars.FuncIn attribute*), 832
LocalOutPython (*angr.analyses.decompiler.ailment.typevars.FuncOut attribute*), 832
loc_repr() (*angr.sim_variable.SimConstantVariable method*), 503
loc_repr() (*angr.sim_variable.SimMemoryVariable method*), 505
loc_repr() (*angr.sim_variable.SimRegisterVariable method*), 505
loc_repr() (*angr.sim_variable.SimStackVariable method*), 506
loc_repr() (*angr.sim_variable.SimTemporaryVariable method*), 504
loc_repr() (*angr.sim_variable.SimVariable method*), 503
local_concretization_mixin (*angr.knowledge_plugins.functions.function.Function property*), 455
local_types (*angr.analyses.decompiler.decompilation_cache.DecompilationCache property*), 697
local_vars (*angr.sim_procedure.SimProcedure attribute*), 471
local_vars (*angr.SimProcedure attribute*), 159
LOCALE_ARRAY (*angr.state_plugins.libc.SimStateLibc at-*

tribute), 236
LocalLoopSeer (*class in angr.exploration_techniques*), 406
LocalLoopSeer (*class in angr.exploration_techniques.local_loop_seer*), 422
LocalVariableTag (*class in angr.knowledge_plugins.key_definitions.tag*), 604
location (*angr.knowledge_plugins.variables.variable_access.VariableAccess attribute*), 561
LocationBase (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 714
long_reason (*angr.knowledge_plugins.cfg.cfg_node.CFGNodeCreationFailure attribute*), 544
looks_like_sql() (*in module angr.utils*), 884
lookup() (*angr.knowledge_plugins.labels.Labels method*), 550
lookup() (*angr.SimMount method*), 203
lookup() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 252
lookup() (*angr.state_plugins.filesystem.SimMount method*), 251
lookup() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 296
lookup() (*angr.state_plugins.posix.PosixDevFS method*), 240
lookup() (*angr.state_plugins.posix.PosixProcFS method*), 242
lookup_defs() (*angr.analyses.ddg.LiveDefinitions method*), 746
lookup_original() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 276
Loop (*class in angr.analyses.loopfinder*), 836
LoopAnalysis (*class in angr.analyses.loop_analysis*), 837
LoopAnalysisState (*class in angr.analyses.loop_analysis*), 837
LoopFinder (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
LoopFinder (*class in angr.analyses.loopfinder*), 836
looping_times (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 547
looping_times (*angr.knowledge_plugins.cfg.CFGENode attribute*), 528
LoopNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 687
LoopSeer (*class in angr.exploration_techniques*), 393
LoopSeer (*class in angr.exploration_techniques.loop_seer*), 421
LoopSimplifier (*class in angr.analyses.decompiler.region_simplifiers.loop*), 718
LoopVisitor (*class in angr.analyses.forward_analysis.visitors.loop*), 626
LoweredSwitchSimplifier (*class in angr.analyses.decompiler.optimization_passes.lowered_switch_si* 707
LShift (*angr.engines.light.data.ArithmeticExpression attribute*), 750
LT (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.C attribute*), 719

M

main() (*in module angr.flirt.build_sig*), 884
main_executable_region_limbos_contain() (*angr.analyses.reassembler.Reassembler method*), 856
main_executable_regions (*angr.analyses.reassembler.Reassembler property*), 856
main_executable_regions_contain() (*angr.analyses.reassembler.Reassembler method*), 856
main_nonexecutable_region_limbos_contain() (*angr.analyses.reassembler.Reassembler method*), 856
main_nonexecutable_regions (*angr.analyses.reassembler.Reassembler property*), 856
main_nonexecutable_regions_contain() (*angr.analyses.reassembler.Reassembler method*), 856
main_object (*angr.angrdb.models.DbObject attribute*), 876
make() (*angr.sim_type.SimTypePointer method*), 511
make() (*angr.sim_type.SimTypeReference method*), 512
make_breakpoint() (*angr.state_plugins.inspect.SimInspector method*), 234
make_bv_sizes_equal() (*in module angr.engines.pcode.behavior*), 444
make_child() (*angr.state_plugins.history.SimStateHistory method*), 270
make_concrete_int() (*angr.sim_state.SimState method*), 227
make_concrete_int() (*angr.SimState method*), 184
make_continuation() (*angr.sim_procedure.SimProcedure method*), 441
make_continuation() (*angr.SimProcedure method*), 159
make_copy() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
make_function_codeloc() (*angr.analyses.reaching_definitions.function_handler.FunctionHa method*), 803

make_function_codeloc() (*angr.analyses.reaching_definitions.FunctionHandler method*), 783
make_functions() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
make_functions() (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot method*), 670
make_ident() (*angr.SimFileBase static method*), 189
make_ident() (*angr.storage.file.SimFileBase static method*), 316
make_initial_state() (*angr.analyses.identifier.identify.Identifier static method*), 836
make_liveness_snapshot() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 774
make_liveness_snapshot() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 602
make_liveness_snapshot() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 571
make_methods() (*in module angr.state_plugins.sim_action_object*), 468
make_phi_node() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 563
make_ro_state() (*in module angr.knowledge_plugins.sync.sync_controller*), 607
make_state() (*in module angr.knowledge_plugins.sync.sync_controller*), 607
make_symbolic_state() (*angr.analyses.identifier.identify.Identifier static method*), 836
MakeTypecastsImplicit (*class in angr.analyses.decompiler.structured_codegen.c*), 740
malloc() (*angr.SimHeapPTMalloc method*), 207
malloc() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 301
malloc() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 305
ManualMergepoint (*class in angr.exploration_techniques*), 402
ManualMergepoint (*class in angr.exploration_techniques.manual_mergepoint*), 410
map() (*angr.storage.memory_mixins.regioned_memory.region_data.RegionMap method*), 369
map_callsites() (*angr.analyses.identifier.identify.Identifier method*), 836
map_region() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 346
map_region() (*angr.storage.memory_mixins.MemoryMixin method*), 337
map_region() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 354
mapping (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.ConstantValueManager attribute*), 663
mark_const() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 812
mark_const() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 783
mark_function_alignments() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
mark_guard() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 812
mark_guard() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 783
mark_merge() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 410
mark_nofilter() (*angr.exploration_techniques.ManualMergepoint method*), 402
mark_nonreturning_calls_endpoints() (*angr.knowledge_plugins.functions.function.Function method*), 556
mark_novalue() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 410
mark_okfilter() (*angr.exploration_techniques.ManualMergepoint method*), 402
match_stmt_classes() (*in module angr.analyses.decompiler.utils*), 743
matches() (*angr.analyses.reaching_definitions.Definition method*), 770
matches() (*angr.knowledge_plugins.key_definitions.Definition method*), 583
matches() (*angr.knowledge_plugins.key_definitions.definition.Definition method*), 590
matches() (*angr.knowledge_plugins.key_definitions.definition.DefinitionMixin method*), 589
max() (*angr.state_plugins.solver.SimSolver method*), 258
max_allocation (*angr.state_plugins.cgc.SimStateCGC attribute*), 272
max_bytes (*angr.engines.pcode.lifter.Lifter attribute*), 440
max_bytes (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
MAX_DATA_REFS (*angr.engines.pcode.lifter.IRSB attribute*), 436
MAX_EXITS (*angr.engines.pcode.lifter.IRSB attribute*), 436
max_inst (*angr.engines.pcode.lifter.Lifter attribute*), 440
max_inst (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
max_int() (*angr.state_plugins.solver.SimSolver method*), 261

MAX_ONE_CALL (*angr.analyses.decompiler.structuring.phoenix.MultiStmtExprMode attribute*), 690
max_size (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 543
max_size (*angr.knowledge_plugins.cfg.MemoryData attribute*), 525
max_size (*angr.utils.dynamic_dictlist.DynamicDictList attribute*), 886
maximum_syscall_number() (*angr.procedures.definitions.SimSyscallLibrary method*), 480
mem (*angr.sim_state.SimState attribute*), 225
mem (*angr.SimState attribute*), 181
mem (*angr.state_plugins.debug_variables.SimDebugVariable property*), 308
MEM (*angr.state_plugins.sim_action.SimAction attribute*), 466
mem() (*angr.analyses.reaching_definitions.Atom static method*), 767
mem() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 585
mem_concrete() (*angr.sim_state.SimState method*), 227
mem_concrete() (*angr.SimState method*), 183
mem_map() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 288
mem_map_ptr() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 288
MEM_PATCH (*class in angr.state_plugins.unicorn_engine*), 284
mem_reset() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 288
mem_unmap() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 288
mem_untyped (*angr.state_plugins.debug_variables.SimDebugVariable property*), 308
member() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 308
member() (*angr.state_plugins.view.SimMemView method*), 314
members (*angr.sim_type.SimCppClass property*), 517
MemDepNode (*class in angr.analyses.data_dep.dep_nodes*), 868
memo() (*angr.SimStatePlugin static method*), 161
memo() (*angr.state_plugins.plugin.SimStatePlugin static method*), 232
Memory (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 867
MEMORY (*angr.analyses.reaching_definitions.AtomKind attribute*), 765
memory (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
memory (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 809
memory (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
memory (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerSta attribute*), 817
memory (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState attribute*), 817
MEMORY (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 584
memory (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefini attribute*), 594
memory (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
MEMORY (*angr.knowledge_plugins.variables.variable_manager.VariableType attribute*), 562
memory (*angr.sim_state.SimState attribute*), 225
memory (*angr.SimState attribute*), 181
memory() (*angr.analyses.reaching_definitions.Atom static method*), 767
memory() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 585
memory_args (*angr.calling_conventions.SimCC property*), 488
memory_args (*angr.SimCC property*), 185
memory_data (*angr.analyses.cfg.cfg_fast.CFGFast property*), 655
memory_data (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 536
memory_data (*angr.knowledge_plugins.cfg.CFGModel attribute*), 530
memory_data (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 610
memory_definitions (*angr.analyses.reaching_definitions.LiveDefinitions property*), 758
memory_definitions (*angr.knowledge_plugins.key_definitions.live_defini property*), 594
memory_definitions (*angr.knowledge_plugins.key_definitions.LiveDefini property*), 574
memory_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
memory_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinit property*), 809
memory_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
memory_uses (*angr.knowledge_plugins.key_definitions.live_definitions.Live attribute*), 594
memory_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
memory_values (*angr.state_plugins.unicorn_engine.VEXStmtDetails attribute*), 285
memory_values_count (*angr.state_plugins.unicorn_engine.VEXStmtDetails attribute*), 285
MemoryData (*class in angr.knowledge_plugins.cfg*), 524
MemoryData (*class in angr.knowledge_plugins.cfg.memory_data*), 542

MemoryDataSort (*class in angr.knowledge_plugins.cfg*), 524
MemoryDataSort (*class in angr.knowledge_plugins.cfg.memory_data*), 542
MemoryLocation (*class in angr.analyses.reaching_definitions*), 768
MemoryLocation (*class in angr.knowledge_plugins.key_definitions.atoms*), 587
MemoryMappingError, 288
MemoryMixin (*class in angr.storage.memory_mixins*), 336
MemoryObjectMixin (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 361
MemoryObjectSetMixin (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 361
MemoryOperand (*class in angr.analyses.disassembly*), 849
MemoryRegion (*class in angr.analyses.cfg.cfb*), 637
MemoryRegionMetaMixin (*class in angr.storage.memory_mixins.regioned_memory.region_meta_mixin*), 371
MemoryValue (*class in angr.state_plugins.unicorn_engine*), 284
MemoryWatcher (*class in angr.exploration_techniques*), 405
MemoryWatcher (*class in angr.exploration_techniques.memory_watcher*), 426
merge() (*angr.analyses.decompiler.optimization_passes.engine_base.SimEngineLight method*), 709
merge() (*angr.analyses.loop_analysis.LoopAnalysisState method*), 837
merge() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 760
merge() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 809
merge() (*angr.analyses.reaching_definitions.ReachingDefinitionsModel method*), 775
merge() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 779
merge() (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerState method*), 817
merge() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 818
merge() (*angr.analyses.variable_recovery.variable_recovery.VariableRecoveryState method*), 825
merge() (*angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFastState method*), 824
merge() (*angr.concretization_strategies.norepeats.SimConcretizationStrategyNorepeats method*), 379
merge() (*angr.concretization_strategies.norepeats_range.SimConcretizationStrategyNorepeatsRange method*), 381
merge() (*angr.concretization_strategies.SimConcretizationStrategy method*), 335
merge() (*angr.keyed_region.KeyedRegion method*), 614
merge() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 546
merge() (*angr.knowledge_plugins.cfg.CFGNode method*), 527
merge() (*angr.knowledge_plugins.key_definitions.environment.Environment method*), 591
merge() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 596
merge() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 576
merge() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 603
merge() (*angr.knowledge_plugins.key_definitions.ReachingDefinitionsModel method*), 571
merge() (*angr.knowledge_plugins.key_definitions.Uses method*), 583
merge() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 607
merge() (*angr.sim_manager.SimulationManager method*), 388
merge() (*angr.sim_state.SimState method*), 227
merge() (*angr.SimFile method*), 191
merge() (*angr.SimFileDescriptor method*), 199
merge() (*angr.SimFileDescriptorDuplex method*), 202
merge() (*angr.SimFileStream method*), 195
merge() (*angr.SimHeapBrk method*), 205
merge() (*angr.SimHeapPTMalloc method*), 207
merge() (*angr.SimLibFullState method*), 193
merge() (*angr.SimPacketsStream method*), 197
merge() (*angr.SimState method*), 183
merge() (*angr.SimStatePlugin method*), 162
merge() (*angr.SimulationManager method*), 177
merge() (*angr.state_plugins.callstack.CallStack method*), 264
merge() (*angr.state_plugins.cgc.SimStateCGC method*), 272
merge() (*angr.state_plugins.concrete.Concrete method*), 293
merge() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 252
merge() (*angr.state_plugins.filesystem.SimFilesystem method*), 249
merge() (*angr.state_plugins.globals.SimStateGlobals method*), 378
merge() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 299
merge() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 305
merge() (*angr.state_plugins.history.SimStateHistory*

method), 267
merge() (*angr.state_plugins.inspect.SimInspector method*), 235
merge() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 295
merge() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 296
merge() (*angr.state_plugins.libc.SimStateLibc method*), 239
merge() (*angr.state_plugins.log.SimStateLog method*), 262
merge() (*angr.state_plugins.loop_data.SimStateLoopData method*), 291
merge() (*angr.state_plugins.plugin.SimStatePlugin method*), 232
merge() (*angr.state_plugins.posix.PosixDevFS method*), 240
merge() (*angr.state_plugins.posix.PosixProcFS method*), 242
merge() (*angr.state_plugins.posix.SimSystemPosix method*), 246
merge() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 282
merge() (*angr.state_plugins.scratch.SimStateScratch method*), 281
merge() (*angr.state_plugins.solver.SimSolver method*), 256
merge() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 275
merge() (*angr.state_plugins.trace_additions.ZenPlugin method*), 276
merge() (*angr.state_plugins.unicorn_engine.Unicorn method*), 289
merge() (*angr.state_plugins.view.SimMemView method*), 313
merge() (*angr.state_plugins.view.SimRegNameView method*), 309
merge() (*angr.storage.file.SimFile method*), 318
merge() (*angr.storage.file.SimFileDescriptor method*), 328
merge() (*angr.storage.file.SimFileDescriptorDuplex method*), 331
merge() (*angr.storage.file.SimFileStream method*), 320
merge() (*angr.storage.file.SimPackets method*), 322
merge() (*angr.storage.file.SimPacketsSlots method*), 333
merge() (*angr.storage.file.SimPacketsStream method*), 324
merge() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 345
merge() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin.JavaVmMemoryMixin method*), 377
merge() (*angr.storage.memory_mixins.MemoryMixin method*), 336
merge() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 353
merge() (*angr.storage.memory_mixins.paged_memory.pages.list_page.ListPage method*), 362
merge() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 351
merge() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 349
merge() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 364
merge() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.RegionMetaMixin method*), 372
merge() (*angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin.RegionedAddressConcretizationMixin method*), 373
merge() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 366
merge() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method*), 374
merge_jobs() (*angr.analyses.vfg.CallAnalysis method*), 842
merge_points() (*angr.annocfg.AnnotatedCFG method*), 873
merge_to_top() (*angr.keyed_region.KeyedRegion method*), 614
merge_transitions() (*in module angr.analyses.cfg_slice_to_sink.transitions*), 816
merged_jobs (*angr.analyses.forward_analysis.job_info.JobInfo property*), 622
MicrosoftAMD64ArgSession (*class in angr.calling_conventions*), 492
min() (*angr.state_plugins.solver.SimSolver method*), 258
min_int() (*angr.state_plugins.solver.SimSolver method*), 261
minimum_syscall_number() (*angr.procedures.definitions.SimSyscallLibrary method*), 480
MipsElfFastResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 660
MIPSGPHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 665
MixedPermissonsError, 288
mnemonic (*angr.analyses.disassembly.Instruction property*), 847
mnemonic (*angr.block.CapstoneInsn property*), 221
mnemonic (*angr.block.DisassemblerInsn property*), 220
mnemonic (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 435
model (*angr.analyses.cfg.cfg_base.CFGBase property*), 646
model (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionAnalysis attribute*), 792
ModSimplifier (*class in angr.analyses.decompiler.optimization_passes.mod_simplifier*),

708
ModSimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.mod_simplifier*), 708
module
angr, 157
angr.analyses, 619
angr.analyses.analysis, 619
angr.analyses.backward_slice, 627
angr.analyses.binary_optimizer, 859
angr.analyses.bindiff, 629
angr.analyses.boyscout, 632
angr.analyses.callee_cleanup_finder, 860
angr.analyses.calling_convention, 632
angr.analyses.cdg, 673
angr.analyses.cfg, 637
angr.analyses.cfg.cfb, 637
angr.analyses.cfg.cfg, 638
angr.analyses.cfg.cfg_arch_options, 656
angr.analyses.cfg.cfg_base, 645
angr.analyses.cfg.cfg_emulated, 640
angr.analyses.cfg.cfg_fast, 647
angr.analyses.cfg.cfg_fast_soot, 668
angr.analyses.cfg.cfg_job_base, 656
angr.analyses.cfg.indirect_jump_resolvers, 668
angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got, 657
angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast, 658
angr.analyses.cfg.indirect_jump_resolvers.const_resolver, 666
angr.analyses.cfg.indirect_jump_resolvers.default_resolvers, 662
angr.analyses.cfg.indirect_jump_resolvers.jumptable, 662
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast, 660
angr.analyses.cfg.indirect_jump_resolvers.resolver, 667
angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt, 661
angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat, 659
angr.analyses.cfg.segment_list, 670
angr.analyses.cfg_slice_to_sink, 814
angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink, 814
angr.analyses.cfg_slice_to_sink.graph, 815
angr.analyses.cfg_slice_to_sink.transitions, 816
angr.analyses.class_identifier, 846
angr.analyses.code_tagging, 673
angr.analyses.complete_calling_conventions, 634
angr.analyses.congruency_check, 858
angr.analyses.data_dep, 870
angr.analyses.data_dep.data_dependency_analysis, 865
angr.analyses.data_dep.dep_nodes, 867
angr.analyses.data_dep.sim_act_location, 866
angr.analyses.datagraph_meta, 673
angr.analyses.ddg, 744
angr.analyses.decompiler, 691
angr.analyses.decompiler.ail_simplifier, 691
angr.analyses.decompiler.ailgraph_walker, 692
angr.analyses.decompiler.block_simplifier, 692
angr.analyses.decompiler.callsite_maker, 693
angr.analyses.decompiler.ccall_rewriters, 693
angr.analyses.decompiler.ccall_rewriters.amd64_ccalls, 693
angr.analyses.decompiler.ccall_rewriters.rewriter_base, 693
angr.analyses.decompiler.clinic, 694
angr.analyses.decompiler.condition_processor, 695
angr.analyses.decompiler.decompilation_cache, 697
angr.analyses.decompiler.decompilation_options, 696
angr.analyses.decompiler.decompiler, 697
angr.analyses.decompiler.empty_node_remover, 698
angr.analyses.decompiler.expression_narrower, 699
angr.analyses.decompiler.graph_region, 699
angr.analyses.decompiler.jump_target_collector, 700
angr.analyses.decompiler.jumptable_entry_condition_rew 700
angr.analyses.decompiler.optimization_passes, 701
angr.analyses.decompiler.optimization_passes.base_ptr_ 705
angr.analyses.decompiler.optimization_passes.const_der 701
angr.analyses.decompiler.optimization_passes.div_simpl 705
angr.analyses.decompiler.optimization_passes.eager_ret 701

angr.analyses.decompiler.optimization_passes.engine_base, 709
angr.analyses.decompiler.optimization_passes.expr_op_swapper, 709
angr.analyses.decompiler.optimization_passes.inline_small_callee, 705
angr.analyses.decompiler.optimization_passes.lower_switch_simplifier, 706
angr.analyses.decompiler.optimization_passes.multi_simplifier, 708
angr.analyses.decompiler.optimization_passes.multi_statement, 708
angr.analyses.decompiler.optimization_passes.optimization_pass, 702
angr.analyses.decompiler.optimization_passes.register_save_area_simplifier, 710
angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier, 711
angr.analyses.decompiler.optimization_passes.stack_canary_simplifier, 704
angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier, 711
angr.analyses.decompiler.peephole_optimizations, 711
angr.analyses.decompiler.peephole_optimizations.base, 711
angr.analyses.decompiler.redundant_label_remover, 722
angr.analyses.decompiler.region_identifier, 713
angr.analyses.decompiler.region_simplifiers, 714
angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer, 714
angr.analyses.decompiler.region_simplifiers.cascading_ifs, 714
angr.analyses.decompiler.region_simplifiers.expression_folding, 714
angr.analyses.decompiler.region_simplifiers.goto, 717
angr.analyses.decompiler.region_simplifiers.if_, 717
angr.analyses.decompiler.region_simplifiers.ifelse, 718
angr.analyses.decompiler.region_simplifiers.loop, 718
angr.analyses.decompiler.region_simplifiers.node_duplication_finder, 718
angr.analyses.decompiler.region_simplifiers.region_append, 718
angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier, 718
angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier, 722
angr.analyses.decompiler.region_walker, 722
angr.analyses.decompiler.sequence_walker, 722
angr.analyses.decompiler.structured_codegen, 723
angr.analyses.decompiler.structured_codegen.base, 723
angr.analyses.decompiler.structured_codegen.c, 724
angr.analyses.decompiler.structured_codegen.dummy, 741
angr.analyses.decompiler.structured_codegen.dwarf_impo 741
angr.analyses.decompiler.structuring, 684
angr.analyses.decompiler.structuring.dream, 684
angr.analyses.decompiler.structuring.phoenix, 690
angr.analyses.decompiler.structuring.recursive_structu 684
angr.analyses.decompiler.structuring.structurer_base, 689
angr.analyses.decompiler.structuring.structurer_nodes, 685
angr.analyses.decompiler.utils, 741
angr.analyses.disassembly, 846
angr.analyses.disassembly_utils, 850
angr.analyses.dominance_frontier, 861
angr.analyses.find_objects_static, 845
angr.analyses.flirt, 749
angr.analyses.forward_analysis, 621
angr.analyses.forward_analysis.forward_analysis, 621
angr.analyses.forward_analysis.job_info, 622
angr.analyses.forward_analysis.visitors, 622
angr.analyses.forward_analysis.visitors.call_graph, 622
angr.analyses.forward_analysis.visitors.function_graph 623
angr.analyses.forward_analysis.visitors.graph, 624
angr.analyses.forward_analysis.visitors.loop, 626
angr.analyses.forward_analysis.visitors.single_node_gr 627
angr.analyses.identifier.identify, 835
angr.analyses.init_finder, 861
angr.analyses.loopfinder, 836
angr.analyses.propagator, 751
angr.analyses.propagator.engine_ail, 753

angr.analyses.propagator.engine_base, 752
angr.analyses.propagator.engine_vex, 753
angr.analyses.propagator.outdated_definition_walker, 754
angr.analyses.propagator.propagator, 755
angr.analyses.propagator.tmpvar_finder, 755
angr.analyses.propagator.top_checker_mixin, 757
angr.analyses.propagator.values, 751
angr.analyses.propagator.vex_vars, 752
angr.analyses.proximity_graph, 863
angr.analyses.reaching_definitions, 757
angr.analyses.reaching_definitions.call_trace, 789
angr.analyses.reaching_definitions.dep_graph, 794
angr.analyses.reaching_definitions.engine_ail, 813
angr.analyses.reaching_definitions.engine_vex, 790
angr.analyses.reaching_definitions.function_handler, 798
angr.analyses.reaching_definitions.heap_allocator, 797
angr.analyses.reaching_definitions.rd_state, 805
angr.analyses.reaching_definitions.reaching_definitions, 791
angr.analyses.reaching_definitions.subject, 813
angr.analyses.reassembler, 850
angr.analyses.soot_class_hierarchy, 636
angr.analyses.stack_pointer_tracker, 816
angr.analyses.static_hooker, 859
angr.analyses.typehoon, 835
angr.analyses.typehoon.lifter, 827
angr.analyses.typehoon.simple_solver, 828
angr.analyses.typehoon.translator, 828
angr.analyses.typehoon.typeconsts, 834
angr.analyses.typehoon.typehoon, 833
angr.analyses.typehoon.typevars, 829
angr.analyses.variable_recovery, 827
angr.analyses.variable_recovery.annotations, 818
angr.analyses.variable_recovery.engine_ail, 826
angr.analyses.variable_recovery.engine_base, 826
angr.analyses.variable_recovery.engine_vex, 826
angr.analyses.variable_recovery.irsb_scanner, 827
angr.analyses.variable_recovery.variable_recovery, 825
angr.analyses.variable_recovery.variable_recovery_base, 819
angr.analyses.variable_recovery.variable_recovery_fast, 823
angr.analyses.veritesting, 838
angr.analyses.vfg, 839
angr.analyses.vsa_ddg, 844
angr.analyses.vtable, 845
angr.analyses.xrefs, 861
angr.angrdb, 674
angr.angrdb.db, 674
angr.angrdb.models, 676
angr.angrdb.serializers, 679
angr.angrdb.serializers.cfg_model, 679
angr.angrdb.serializers.comments, 680
angr.angrdb.serializers.funcs, 680
angr.angrdb.serializers.kb, 681
angr.angrdb.serializers.labels, 681
angr.angrdb.serializers.loader, 681
angr.angrdb.serializers.structured_code, 683
angr.angrdb.serializers.variables, 682
angr.angrdb.serializers.xrefs, 682
angr.annocfg, 871
angr.blade, 870
angr.block, 220
angr.callable, 519
angr.calling_conventions, 483
angr.code_location, 612
angr.codenode, 873
angr.concretization_strategies, 335
angr.concretization_strategies.any, 381
angr.concretization_strategies.controlled_data, 381
angr.concretization_strategies.eval, 379
angr.concretization_strategies.max, 380
angr.concretization_strategies.nonzero, 381
angr.concretization_strategies.nonzero_range, 380
angr.concretization_strategies.norepeats, 379
angr.concretization_strategies.norepeats_range, 381
angr.concretization_strategies.range, 380
angr.concretization_strategies.single, 379
angr.concretization_strategies.solutions, 379
angr.concretization_strategies.unlimited_range, 381
angr.distributed, 900
angr.distributed.server, 900

angr.distributed.worker, 900
angr.engines, 427
angr.engines.concrete, 433
angr.engines.engine, 427
angr.engines.failure, 431
angr.engines.hook, 430
angr.engines.light, 750
angr.engines.light.data, 750
angr.engines.light.engine, 750
angr.engines.pcode, 434
angr.engines.pcode.behavior, 444
angr.engines.pcode.cc, 464
angr.engines.pcode.emulate, 444
angr.engines.pcode.engine, 434
angr.engines.pcode.lifter, 435
angr.engines.procedure, 430
angr.engines.soot, 431
angr.engines.soot.engine, 431
angr.engines.successors, 428
angr.engines.syscall, 431
angr.engines.unicorn, 432
angr.engines.vex, 431
angr.errors, 894
angr.exploration_techniques, 390
angr.exploration_techniques.bucketizer, 426
angr.exploration_techniques.common, 424
angr.exploration_techniques.dfs, 408
angr.exploration_techniques.director, 418
angr.exploration_techniques.driller_core, 416
angr.exploration_techniques.explorer, 408
angr.exploration_techniques.lengthlimiter, 409
angr.exploration_techniques.local_loop_seer, 422
angr.exploration_techniques.loop_seer, 421
angr.exploration_techniques.manual_mergepoint, 410
angr.exploration_techniques.memory_watcher, 426
angr.exploration_techniques.oppologist, 420
angr.exploration_techniques.slicecutor, 417
angr.exploration_techniques.spiller, 410
angr.exploration_techniques.spiller_db, 412
angr.exploration_techniques.stochastic, 423
angr.exploration_techniques.suggestions, 426
angr.exploration_techniques.symbion, 424
angr.exploration_techniques.tech_builder, 424
angr.exploration_techniques.threading, 413
angr.exploration_techniques.timeout, 407
angr.exploration_techniques.tracer, 414
angr.exploration_techniques.unique, 423
angr.exploration_techniques.veritesting, 413
angr.factory, 216
angr.flirt, 883
angr.flirt.build_sig, 883
angr.keyed_region, 613
angr.knowledge_base, 520
angr.knowledge_base.knowledge_base, 520
angr.knowledge_plugins, 521
angr.knowledge_plugins.callsite_prototypes, 523
angr.knowledge_plugins.cfg, 524
angr.knowledge_plugins.cfg.cfg_manager, 544
angr.knowledge_plugins.cfg.cfg_model, 536
angr.knowledge_plugins.cfg.cfg_node, 544
angr.knowledge_plugins.cfg.indirect_jump, 547
angr.knowledge_plugins.cfg.memory_data, 542
angr.knowledge_plugins.comments, 549
angr.knowledge_plugins.data, 549
angr.knowledge_plugins.debug_variables, 568
angr.knowledge_plugins.functions, 550
angr.knowledge_plugins.functions.function, 552
angr.knowledge_plugins.functions.function_manager, 550
angr.knowledge_plugins.functions.function_parser, 559
angr.knowledge_plugins.functions.soot_function, 560
angr.knowledge_plugins.indirect_jumps, 549
angr.knowledge_plugins.key_definitions, 570
angr.knowledge_plugins.key_definitions.atoms, 584
angr.knowledge_plugins.key_definitions.constants, 588
angr.knowledge_plugins.key_definitions.definition, 588
angr.knowledge_plugins.key_definitions.environment, 590
angr.knowledge_plugins.key_definitions.heap_address, 591

angr.knowledge_plugins.key_definitions.key_definition, 591
angr.knowledge_plugins.key_definitions.live_definitions, 592
angr.knowledge_plugins.key_definitions.rd_model, 601
angr.knowledge_plugins.key_definitions.tag, 603
angr.knowledge_plugins.key_definitions.undefined, 605
angr.knowledge_plugins.key_definitions.unknown_size, 605
angr.knowledge_plugins.key_definitions.uses, 605
angr.knowledge_plugins.labels, 549
angr.knowledge_plugins.patches, 521
angr.knowledge_plugins.plugin, 523
angr.knowledge_plugins.propagations, 549
angr.knowledge_plugins.structured_code, 570
angr.knowledge_plugins.structured_code.manager, 570
angr.knowledge_plugins.sync, 607
angr.knowledge_plugins.sync.sync_controller, 607
angr.knowledge_plugins.types, 549
angr.knowledge_plugins.variables, 561
angr.knowledge_plugins.variables.variable_access, 561
angr.knowledge_plugins.variables.variable_manager, 561
angr.knowledge_plugins.xrefs, 610
angr.knowledge_plugins.xrefs.xref, 610
angr.knowledge_plugins.xrefs.xref_manager, 611
angr.knowledge_plugins.xrefs.xref_types, 611
angr.misc.plugins, 222
angr.procedures, 473
angr.procedures.definitions, 475
angr.procedures.stubs.format_parser, 473
angr.project, 212
angr.protos, 617
angr.serializable, 616
angr.sim_manager, 382
angr.sim_options, 228
angr.sim_procedure, 469
angr.sim_state, 224
angr.sim_state_options, 228
angr.sim_type, 507
angr.sim_variable, 502
angr.simos, 875
angr.simos.cgc, 878
angr.simos.javavm, 880
angr.simos.linux, 877
angr.simos.simos, 875
angr.simos.userland, 878
angr.simos.windows, 879
angr.slicer, 871
angr.state_hierarchy, 389
angr.state_plugins, 231
angr.state_plugins.callstack, 263
angr.state_plugins.cgc, 271
angr.state_plugins.concrete, 292
angr.state_plugins.debug_variables, 307
angr.state_plugins.filesystem, 248
angr.state_plugins.gdb, 270
angr.state_plugins.globals, 278
angr.state_plugins.heap, 297
angr.state_plugins.heap.heap_base, 297
angr.state_plugins.heap.heap_brk, 298
angr.state_plugins.heap.heap_freelist, 300
angr.state_plugins.heap.heap_libc, 301
angr.state_plugins.heap.heap_ptmalloc, 302
angr.state_plugins.heap.utils, 306
angr.state_plugins.history, 267
angr.state_plugins.inspect, 233
angr.state_plugins.javavm_classloader, 294
angr.state_plugins.jni_references, 296
angr.state_plugins.libc, 236
angr.state_plugins.light_registers, 266
angr.state_plugins.log, 262
angr.state_plugins.loop_data, 291
angr.state_plugins.plugin, 231
angr.state_plugins.posix, 240
angr.state_plugins.preconstrainer, 282
angr.state_plugins.scratch, 280
angr.state_plugins.sim_action, 466
angr.state_plugins.sim_action_object, 468
angr.state_plugins.sim_event, 468
angr.state_plugins.solver, 254
angr.state_plugins.symbolizer, 307
angr.state_plugins.trace_additions, 273
angr.state_plugins.uc_manager, 279
angr.state_plugins.unicorn_engine, 284
angr.state_plugins.view, 309
angr.storage, 309
angr.storage.file, 314
angr.storage.memory_mixins, 336
angr.storage.memory_mixins.actions_mixin, 342
angr.storage.memory_mixins.address_concretization_mixi
344
angr.storage.memory_mixins.bvv_conversion_mixin, 340

angr.storage.memory_mixins.clouseau_mixin, 346
angr.storage.memory_mixins.conditional_store_mixin, 346
angr.storage.memory_mixins.convenient_mappings_mixin, 348
angr.storage.memory_mixins.default_filler_mixin, 340
angr.storage.memory_mixins.dirty_addrs_mixin, 344
angr.storage.memory_mixins.hex_dumper_mixin, 341
angr.storage.memory_mixins.javavm_memory, 376
angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin, 376
angr.storage.memory_mixins.keyvalue_memory, 375
angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory_mixin, 375
angr.storage.memory_mixins.label_merger_mixin, 346
angr.storage.memory_mixins.multi_value_merger_mixin, 352
angr.storage.memory_mixins.name_resolution_mixin, 339
angr.storage.memory_mixins.paged_memory, 352
angr.storage.memory_mixins.paged_memory.page_backer_mixins, 357
angr.storage.memory_mixins.paged_memory.paged_memory_mixin, 352
angr.storage.memory_mixins.paged_memory.pages, 359
angr.storage.memory_mixins.paged_memory.pages.cooperation, 361
angr.storage.memory_mixins.paged_memory.pages.history_tree, 360
angr.storage.memory_mixins.paged_memory.pages.mv_mixin, 361
angr.storage.memory_mixins.paged_memory.pages.pages, 362
angr.storage.memory_mixins.paged_memory.pages.ultra_page, 350
angr.storage.memory_mixins.paged_memory.pages.multi_page, 348
angr.storage.memory_mixins.paged_memory.pages.vectored_page_mixin, 360
angr.storage.memory_mixins.paged_memory.pages.vectored_pages, 359
angr.storage.memory_mixins.paged_memory.pages.vectored_utils, 363
angr.storage.memory_mixins.paged_memory.privileged_mixin, 358
angr.storage.memory_mixins.paged_memory.stack_allocation_mixin, 358
angr.storage.memory_mixins.regioned_memory, 365
angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor, 371
angr.storage.memory_mixins.regioned_memory.abstract_memory_mixin, 373
angr.storage.memory_mixins.regioned_memory.region_category, 370
angr.storage.memory_mixins.regioned_memory.region_data, 368
angr.storage.memory_mixins.regioned_memory.region_meta, 371
angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin, 373
angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin, 365
angr.storage.memory_mixins.regioned_memory.static_find, 370
angr.storage.memory_mixins.simple_interface_mixin, 342
angr.storage.memory_mixins.simplification_mixin, 347
angr.storage.memory_mixins.size_resolution_mixin, 343
angr.storage.memory_mixins.slotted_memory, 374
angr.storage.memory_mixins.smart_find_mixin, 339
angr.storage.memory_mixins.symbolic_merger_mixin, 342
angr.storage.memory_mixins.top_merger_mixin, 351
angr.storage.memory_mixins.underconstrained_mixin, 341
angr.storage.memory_mixins.unwrapper_mixin, 347
angr.storage.memory_object, 334
angr.storage.pcap, 335
angr.utils, 884
angr.utils.algo, 885
angr.utils.constants, 885
angr.utils.cowdict, 885
angr.utils.dynamic_dictlist, 885
angr.utils.enums_conv, 886
angr.utils.env, 886
angr.utils.formatting, 892
angr.utils.graph, 886
angr.utils.lazy_import, 890
angr.utils.library, 891
angr.utils.loader, 890
angr.utils.mp, 893
angr.utils.timing, 892

angr.vaults, 617
most_mergeable() (*angr.state_hierarchy.StateHierarchy method*), 390
most_mergeable() (*angr.StateHierarchy method*), 180
mount() (*angr.state_plugins.filesystem.SimFilesystem method*), 250
move() (*angr.sim_manager.SimulationManager method*), 386
move() (*angr.SimulationManager method*), 175
move_codelocs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 809
move_codelocs() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 780
mp_context() (*in module angr.utils.mp*), 894
Mul (*angr.engines.light.data.ArithmeticExpression attribute*), 750
mulpyplex() (*angr.sim_manager.SimulationManager method*), 383
mulpyplex() (*angr.SimulationManager method*), 172
MultiNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 685
MultipleBlocksException, 702
MultiSimplifier (*class in angr.analyses.decompiler.optimization_passes.multi_simplifier*), 708
MultiSimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.multi_simplifier*), 708
MultiStatementExpressionAssignmentFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 716
MultiStmtExprMode (*class in angr.analyses.decompiler.structuring.phoenix*), 690
MultiValuedMemory (*class in angr.storage.memory_mixins*), 339
MultiValueMergerMixin (*class in angr.storage.memory_mixins.multi_value_merger_mixin*), 352
MultiValues (*class in angr.storage.memory_mixins.paged_memory.pages.multi_values*), 350
MultiwriteAnnotation (*class in angr.storage.memory_mixins.address_concretization_mixin*), 344
MVListPage (*class in angr.storage.memory_mixins.paged_memory.pages.mv_list_page*), 348
MVListPagesMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 356
MVListPagesWithLabelsMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 356

N

n (*angr.analyses.typehoon.typevars.AddN attribute*), 832
n (*angr.analyses.typehoon.typevars.SubN attribute*), 832
NAME (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplif attribute*), 705
NAME (*angr.analyses.decompiler.optimization_passes.const_derefs.Constant attribute*), 701
NAME (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimp attribute*), 705
NAME (*angr.analyses.decompiler.optimization_passes.eager_returns.EagerR attribute*), 702
NAME (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.Exp attribute*), 710
NAME (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITE attribute*), 706
NAME (*angr.analyses.decompiler.optimization_passes.lowered_switch_simpl attribute*), 707
NAME (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSi attribute*), 708
NAME (*angr.analyses.decompiler.optimization_passes.multi_simplifier.MultiS attribute*), 708
NAME (*angr.analyses.decompiler.optimization_passes.optimization_pass.Bas attribute*), 703
NAME (*angr.analyses.decompiler.optimization_passes.register_save_area_si attribute*), 711
NAME (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplif attribute*), 711
NAME (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifi attribute*), 704
NAME (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simpli attribute*), 711
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOpti attribute*), 713
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOpti attribute*), 712
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOpti attribute*), 712
name (*angr.analyses.decompiler.structured_codegen.c.CFakeVariable attribute*), 732
name (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 726
name (*angr.analyses.decompiler.structured_codegen.c.CLabel attribute*), 731
name (*angr.analyses.decompiler.structured_codegen.c.CStructFieldNameDe attribute*), 738
NAME (*angr.analyses.decompiler.structuring.dream.DreamStructurer attribute*), 684
NAME (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer attribute*), 691
NAME (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase attribute*), 689
name (*angr.analyses.reaching_definitions.function_handler.FunctionCallDa attribute*), 800
name (*angr.analyses.reaching_definitions.FunctionCallData*

attribute), 787
name (*angr.analyses.reaching_definitions.Register property*), 768
name (*angr.analyses.reassembler.Procedure property*), 854
name (*angr.angrdb.models.DbKnowledgeBase attribute*), 676
name (*angr.angrdb.models.DbLabel attribute*), 679
name (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 545
name (*angr.knowledge_plugins.cfg.CFGNode property*), 526
name (*angr.knowledge_plugins.functions.function.Function property*), 553
name (*angr.knowledge_plugins.key_definitions.atoms.Register property*), 587
name (*angr.procedures.definitions.SimLibrary property*), 476
name (*angr.sim_state_options.StateOption attribute*), 228
name (*angr.sim_type.NamedTypeMixin property*), 508
name (*angr.sim_type.TypeRef property*), 508
name (*angr.sim_variable.SimVariable attribute*), 503
name_stop() (*angr.state_plugins.unicorn_engine.STOP static method*), 287
named_errors (*angr.analyses.analysis.Analysis attribute*), 621
named_errors (*angr.Analysis attribute*), 178
NamedTypeMixin (*class in angr.sim_type*), 508
NameResolutionMixin (*class in angr.storage.memory_mixins.name_resolution_mixin*), 339
native_arch (*angr.simos.javavm.SimJavaVM property*), 882
NE (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.Cond_Op attribute*), 719
NEVER (*angr.analyses.decompiler.structuring.phoenix.MultiStmtExprMode attribute*), 690
new() (*angr.analyses.cfg.cfg_job_base.BlockID static method*), 657
new() (*angr.analyses.cfg.cfg_job_base.FunctionKey static method*), 657
new() (*angr.analyses.typehoon.typeconsts.Pointer method*), 835
new_block_addr() (*angr.analyses.decompiler.clinic.Clinic method*), 695
new_block_addr() (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass method*), 704
new_from_shared() (*angr.storage.memory_mixins.paged_memory.pages.ultra.UltraPage class method*), 364
new_label() (*angr.analyses.reassembler.Label static method*), 851
new_label() (*angr.analyses.reassembler.SymbolManager method*), 852
new_model() (*angr.knowledge_plugins.cfg.cfg_manager.CFGManager method*), 544
new_model() (*angr.knowledge_plugins.cfg.CFGManager method*), 536
NewFunctionHandler (*class in angr.analyses.find_objects_static*), 845
Next (*angr.analyses.loop_analysis.VariableTypes attribute*), 837
next (*angr.engines.pcode.lifter.IRSB attribute*), 437
next (*angr.state_plugins.unicorn_engine.MEM_PATCH attribute*), 284
next_addr (*angr.analyses.decompiler.optimization_passes.lowered_switch attribute*), 707
next_arg() (*angr.calling_conventions.SimCC method*), 489
next_arg() (*angr.calling_conventions.SimCCARM method*), 496
next_arg() (*angr.calling_conventions.SimCCARMHF method*), 496
next_arg() (*angr.calling_conventions.SimCCCdecl method*), 492
next_arg() (*angr.calling_conventions.SimCCMicrosoftAMD64 method*), 493
next_arg() (*angr.calling_conventions.SimCCO32 method*), 498
next_arg() (*angr.calling_conventions.SimCCSystemVAMD64 method*), 494
next_arg() (*angr.calling_conventions.SimCCUsercall method*), 491
next_arg() (*angr.SimCC method*), 186
next_chunk() (*angr.PTChunk method*), 209
next_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 300
next_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 303
next_free_pos() (*angr.analyses.cfg.segment_list.SegmentList method*), 671
next_node() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 625
next_node() (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor method*), 627
next_pos_with_sort_not_in() (*angr.analyses.cfg.segment_list.SegmentList method*), 671
next_variable_ident() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 562
no_ret (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
no_ret (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
NO_RET (*angr.sim_procedure.SimProcedure attribute*), 471
NO_RET (*angr.SimProcedure attribute*), 159
NoConcreteDispatch, 636

NodalAnnotation (*class in angr.analyses.data_dep.data_dependency_analysis*), 865
node (*angr.analyses.decompiler.region_simplifiers.switch_chosen_simplifier.ConditionalRegion attribute*), 719
node (*angr.analyses.decompiler.region_simplifiers.switch_chosen_simplifier.SwitchCaseRegion attribute*), 720
node (*angr.analyses.decompiler.structuring.structurer_nodes.CodeNode attribute*), 686
node (*angr.analyses.decompiler.structuring.structurer_nodes.CodeLiveNode attribute*), 686
node (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor attribute*), 627
node_addr (*angr.analyses.decompiler.region_simplifiers.expr_folding_conditionalRegion attribute*), 715
node_addr (*angr.analyses.decompiler.region_simplifiers.expr_folding_conditionalRegion attribute*), 715
node_observe() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 793
node_observe() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 772
node_position() (*angr.analyses.decompiler.structuring.structurer_nodes.SequenceNode method*), 685
node_returned (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor attribute*), 627
node_type (*angr.analyses.decompiler.optimization_passes.ConstantLoadSimplifier.Case attribute*), 707
NodeAddressFinder (*class in angr.analyses.decompiler.region_simplifiers.node_address_finder*), 718
NodeFoundNotification, 705
nodes (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 815
nodes (*angr.analyses.decompiler.structuring.structurer_nodes.MultiNode attribute*), 685
nodes (*angr.analyses.decompiler.structuring.structurer_nodes.SequenceNode attribute*), 685
nodes (*angr.knowledge_plugins.functions.function.Function property*), 554
nodes() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
nodes() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 625
nodes() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 795
nodes() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 538
nodes() (*angr.knowledge_plugins.cfg.CFGModel method*), 532
nodes_iter() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
nodes_iter() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 625
NONE (*angr.simos.windows.SecurityCookieInit attribute*), 879
NORMAL (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 651
normalize() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
normalize() (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot method*), 669
normalize() (*angr.knowledge_plugins.functions.function.Function method*), 558
normalize() (*angr.knowledge_plugins.functions.soot_function.SootFunction method*), 560
normalize() (*angr.knowledge_plugins.key_definitions.definition.Definition method*), 589
normalize_conditionalRegion_name() (*in module angr.sim_type*), 519
normalized (*angr.analyses.cfg.cfg_base.CFGBase property*), 646
normalized (*angr.knowledge_plugins.cfg.cfg_model.CFGModel attribute*), 536
normalized (*angr.knowledge_plugins.cfg.CFGModel attribute*), 530
normalized (*angr.knowledge_plugins.functions.function.Function attribute*), 552
normalized (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
NormalizedBlock (*class in angr.analyses.bindiff*), 630
NormalizedFunction (*class in angr.analyses.bindiff*), 630
NotAssumptableNotification, 662
NotEqual (*angr.analyses.loop_analysis.Condition attribute*), 837
NotSemantic (*class in angr.storage.memory_mixins.paged_memory.page_backer_mixins*), 357
NotypeLabel (*class in angr.analyses.reassembler*), 851
NULL_TERMINATE (*angr.knowledge_plugins.key_definitions.DerefSize attribute*), 581
NULL_TERMINATE (*angr.knowledge_plugins.key_definitions.live_definitions.*
attribute), 592
num_arguments (*angr.knowledge_plugins.functions.function.Function property*), 555

O

O (*in module angr.analyses.decompiler.decompilation_options*), 697
O_ACCMODE (*angr.storage.file.Flags attribute*), 315
O_APPEND (*angr.storage.file.Flags attribute*), 315
O_ASYNC (*angr.storage.file.Flags attribute*), 315
O_CLOEXEC (*angr.storage.file.Flags attribute*), 315
O_CREAT (*angr.storage.file.Flags attribute*), 315
O_DIRECT (*angr.storage.file.Flags attribute*), 315
O_DIRECTORY (*angr.storage.file.Flags attribute*), 315
O_DSYNC (*angr.storage.file.Flags attribute*), 315
O_EXCL (*angr.storage.file.Flags attribute*), 315

O_LARGEFILE (*angr.storage.file.Flags attribute*), 315
O_NDELAY (*angr.storage.file.Flags attribute*), 315
O_NOATIME (*angr.storage.file.Flags attribute*), 315
O_NOCTTY (*angr.storage.file.Flags attribute*), 315
O_NOFOLLOW (*angr.storage.file.Flags attribute*), 315
O_NONBLOCK (*angr.storage.file.Flags attribute*), 315
O_PATH (*angr.storage.file.Flags attribute*), 315
O_RDONLY (*angr.storage.file.Flags attribute*), 314
O_RDWR (*angr.storage.file.Flags attribute*), 314
O_SYNC (*angr.storage.file.Flags attribute*), 315
O_TMPFILE (*angr.storage.file.Flags attribute*), 315
O_TRUNC (*angr.storage.file.Flags attribute*), 315
O_WRONLY (*angr.storage.file.Flags attribute*), 314
obj (*angr.analyses.decompiler.structured_codegen.base.PositionMappingElement attribute*), 723
obj (*angr.analyses.disassembly.IROp attribute*), 847
obj (*angr.keyed_region.StoredObject attribute*), 613
obj (*angr.utils.graph.ContainerNode property*), 888
obj_bit_size() (*in module angr.storage.memory_object*), 334
obj_id (*angr.keyed_region.StoredObject property*), 613
object (*angr.storage.memory_object.SimMemoryObject attribute*), 334
ObjectLabel (*class in angr.analyses.reassembler*), 851
ObservationPointType (*class in angr.analyses.reaching_definitions*), 765
ObservationPointType (*class in angr.knowledge_plugins.key_definitions.constants*), 588
observed_results (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 792
observed_results (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 772
occupied_by() (*angr.analyses.cfg.segment_list.SegmentList method*), 672
occupied_by_sort() (*angr.analyses.cfg.segment_list.SegmentList method*), 672
occupied_size (*angr.analyses.cfg.segment_list.SegmentList property*), 673
occupy() (*angr.analyses.cfg.segment_list.SegmentList method*), 672
offset (*angr.analyses.decompiler.structured_codegen.c.CStructField attribute*), 732
offset (*angr.analyses.propagator.vex_vars.VEXReg attribute*), 752
offset (*angr.analyses.reaching_definitions.Definition property*), 770
offset (*angr.analyses.reassembler.Label property*), 851
offset (*angr.analyses.stack_pointer_tracker.OffsetVal property*), 817
offset (*angr.analyses.stack_pointer_tracker.Register attribute*), 817
offset (*angr.analyses.typehoon.typevars.HasField attribute*), 833
offset (*angr.engines.light.data.RegisterOffset attribute*), 750
offset (*angr.knowledge_plugins.functions.function.Function property*), 555
offset (*angr.knowledge_plugins.key_definitions.Definition property*), 583
offset (*angr.knowledge_plugins.key_definitions.definition.Definition property*), 590
offset (*angr.knowledge_plugins.variables.variable_access.VariableAccess attribute*), 561
Offset (*angr.knowledge_plugins.xrefs.xref_types.XRefType attribute*), 611
offset (*angr.sim_variable.SimStackVariable attribute*), 596
offset (*angr.state_plugins.unicorn_engine.RegisterValue attribute*), 285
offset_after() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 818
offset_after_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 818
offset_before() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 818
offset_before_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 818
offsets (*angr.sim_type.SimStruct property*), 516
OffsetVal (*class in angr.analyses.stack_pointer_tracker*), 817
offsIP (*angr.engines.ReachingDefinitionsAnalysis property*), 438
on_worker_exit() (*angr.distributed.server.Server method*), 900
on_worker_exit() (*angr.Server method*), 211
one_active (*angr.sim_manager.SimulationManager attribute*), 383
one_active (*angr.SimulationManager attribute*), 172
one_deadended (*angr.sim_manager.SimulationManager attribute*), 383
one_deadended (*angr.SimulationManager attribute*), 172
one_found (*angr.sim_manager.SimulationManager attribute*), 383
one_found (*angr.SimulationManager attribute*), 172
one_pruned (*angr.sim_manager.SimulationManager attribute*), 383
one_pruned (*angr.SimulationManager attribute*), 172
one_result (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 792
one_result (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 772
one_stashed (*angr.sim_manager.SimulationManager attribute*), 383
one_stashed (*angr.SimulationManager attribute*), 172
one_type() (*angr.sim_state_options.StateOption*

method), 228
one_unconstrained (*angr.sim_manager.SimulationManager attribute*), 383
one_unconstrained (*angr.SimulationManager attribute*), 172
one_unsat (*angr.sim_manager.SimulationManager attribute*), 383
one_unsat (*angr.SimulationManager attribute*), 172
one_value() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 351
op (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterCondition attribute*), 719
op (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp attribute*), 734
op (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp attribute*), 734
op (*angr.engines.light.data.ArithmeticExpression attribute*), 750
OP_AFTER (*angr.analyses.reaching_definitions.ObservationPointType attribute*), 765
OP_AFTER (*angr.knowledge_plugins.key_definitions.constants.ObservationPointType attribute*), 588
OP_BEFORE (*angr.analyses.reaching_definitions.ObservationPointType attribute*), 765
OP_BEFORE (*angr.knowledge_plugins.key_definitions.constants.ObservationPointType attribute*), 588
op_precedence (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp property*), 734
op_str (*angr.block.CapstoneInsn property*), 221
op_str (*angr.block.DisassemblerInsn property*), 220
op_str (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 435
OpBehavior (*class in angr.engines.pcode.behavior*), 445
OpBehaviorBoolAnd (*class in angr.engines.pcode.behavior*), 458
OpBehaviorBoolNegate (*class in angr.engines.pcode.behavior*), 457
OpBehaviorBoolOr (*class in angr.engines.pcode.behavior*), 459
OpBehaviorBoolXor (*class in angr.engines.pcode.behavior*), 458
OpBehaviorCopy (*class in angr.engines.pcode.behavior*), 446
OpBehaviorEqual (*class in angr.engines.pcode.behavior*), 446
OpBehaviorFloatAbs (*class in angr.engines.pcode.behavior*), 461
OpBehaviorFloatAdd (*class in angr.engines.pcode.behavior*), 460
OpBehaviorFloatCeil (*class in angr.engines.pcode.behavior*), 462
OpBehaviorFloatDiv (*class in angr.engines.pcode.behavior*), 460
OpBehaviorFloatEqual (*class in angr.engines.pcode.behavior*), 459
OpBehaviorFloatFloat2Float (*class in angr.engines.pcode.behavior*), 462
OpBehaviorFloatFloor (*class in angr.engines.pcode.behavior*), 462
OpBehaviorFloatInt2Float (*class in angr.engines.pcode.behavior*), 462
OpBehaviorFloatLess (*class in angr.engines.pcode.behavior*), 459
OpBehaviorFloatLessEqual (*class in angr.engines.pcode.behavior*), 460
OpBehaviorFloatMult (*class in angr.engines.pcode.behavior*), 461
OpBehaviorFloatNan (*class in angr.engines.pcode.behavior*), 460
OpBehaviorFloatNeg (*class in angr.engines.pcode.behavior*), 461
OpBehaviorFloatNotEqual (*class in angr.engines.pcode.behavior*), 459
OpBehaviorFloatRound (*class in angr.engines.pcode.behavior*), 463
OpBehaviorFloatSqrt (*class in angr.engines.pcode.behavior*), 461
OpBehaviorFloatSub (*class in angr.engines.pcode.behavior*), 461
OpBehaviorFloatTrunc (*class in angr.engines.pcode.behavior*), 462
OpBehaviorInt2Comp (*class in angr.engines.pcode.behavior*), 452
OpBehaviorIntAdd (*class in angr.engines.pcode.behavior*), 450
OpBehaviorIntAnd (*class in angr.engines.pcode.behavior*), 453
OpBehaviorIntCarry (*class in angr.engines.pcode.behavior*), 450
OpBehaviorIntDiv (*class in angr.engines.pcode.behavior*), 456
OpBehaviorIntLeft (*class in angr.engines.pcode.behavior*), 454
OpBehaviorIntLess (*class in angr.engines.pcode.behavior*), 448
OpBehaviorIntLessEqual (*class in angr.engines.pcode.behavior*), 448
OpBehaviorIntMult (*class in angr.engines.pcode.behavior*), 455
OpBehaviorIntNegate (*class in angr.engines.pcode.behavior*), 452
OpBehaviorIntOr (*class in angr.engines.pcode.behavior*), 453
OpBehaviorIntRem (*class in angr.engines.pcode.behavior*), 456
OpBehaviorIntRight (*class in angr.engines.pcode.behavior*), 454
OpBehaviorIntSborrow (*class in*

angr.engines.pcode.behavior), 451
OpBehaviorIntScarry (*class in angr.engines.pcode.behavior*), 451
OpBehaviorIntSdiv (*class in angr.engines.pcode.behavior*), 456
OpBehaviorIntSext (*class in angr.engines.pcode.behavior*), 449
OpBehaviorIntSless (*class in angr.engines.pcode.behavior*), 447
OpBehaviorIntSlessEqual (*class in angr.engines.pcode.behavior*), 448
OpBehaviorIntSrem (*class in angr.engines.pcode.behavior*), 457
OpBehaviorIntSright (*class in angr.engines.pcode.behavior*), 455
OpBehaviorIntSub (*class in angr.engines.pcode.behavior*), 450
OpBehaviorIntXor (*class in angr.engines.pcode.behavior*), 453
OpBehaviorIntZext (*class in angr.engines.pcode.behavior*), 449
OpBehaviorNotEqual (*class in angr.engines.pcode.behavior*), 447
OpBehaviorPiece (*class in angr.engines.pcode.behavior*), 463
OpBehaviorPopcount (*class in angr.engines.pcode.behavior*), 463
OpBehaviorSubpiece (*class in angr.engines.pcode.behavior*), 463
opcode (*angr.engines.pcode.behavior.OpBehavior attribute*), 445
opcode (*angr.engines.pcode.behavior.OpBehaviorBoolAnd attribute*), 459
opcode (*angr.engines.pcode.behavior.OpBehaviorBoolNegate attribute*), 458
opcode (*angr.engines.pcode.behavior.OpBehaviorBoolOr attribute*), 459
opcode (*angr.engines.pcode.behavior.OpBehaviorBoolXor attribute*), 458
opcode (*angr.engines.pcode.behavior.OpBehaviorCopy attribute*), 446
opcode (*angr.engines.pcode.behavior.OpBehaviorEqual attribute*), 447
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatAbs attribute*), 461
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatAdd attribute*), 460
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatCeil attribute*), 462
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatDiv attribute*), 460
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatEqual attribute*), 459
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatFloor attribute*), 462
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatInt2Float attribute*), 462
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatLess attribute*), 460
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatLessEqual attribute*), 460
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatMult attribute*), 461
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatNan attribute*), 460
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatNeg attribute*), 461
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatNotEqual attribute*), 459
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatRound attribute*), 463
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatSqrt attribute*), 461
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatSub attribute*), 461
opcode (*angr.engines.pcode.behavior.OpBehaviorFloatTrunc attribute*), 462
opcode (*angr.engines.pcode.behavior.OpBehaviorInt2Comp attribute*), 452
opcode (*angr.engines.pcode.behavior.OpBehaviorIntAdd attribute*), 450
opcode (*angr.engines.pcode.behavior.OpBehaviorIntAnd attribute*), 453
opcode (*angr.engines.pcode.behavior.OpBehaviorIntCarry attribute*), 451
opcode (*angr.engines.pcode.behavior.OpBehaviorIntDiv attribute*), 456
opcode (*angr.engines.pcode.behavior.OpBehaviorIntLeft attribute*), 454
opcode (*angr.engines.pcode.behavior.OpBehaviorIntLess attribute*), 448
opcode (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual attribute*), 449
opcode (*angr.engines.pcode.behavior.OpBehaviorIntMult attribute*), 456
opcode (*angr.engines.pcode.behavior.OpBehaviorIntNegate attribute*), 452
opcode (*angr.engines.pcode.behavior.OpBehaviorIntOr attribute*), 454
opcode (*angr.engines.pcode.behavior.OpBehaviorIntRem attribute*), 457
opcode (*angr.engines.pcode.behavior.OpBehaviorIntRight attribute*), 455
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSborrow attribute*), 452
opcode (*angr.engines.pcode.behavior.OpBehaviorIntScarry*

attribute), 451
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSdiv attribute*), 456
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSext attribute*), 449
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSless attribute*), 447
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual attribute*), 448
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSrem attribute*), 457
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSright attribute*), 455
opcode (*angr.engines.pcode.behavior.OpBehaviorIntSub attribute*), 450
opcode (*angr.engines.pcode.behavior.OpBehaviorIntXor attribute*), 453
opcode (*angr.engines.pcode.behavior.OpBehaviorIntZext attribute*), 449
opcode (*angr.engines.pcode.behavior.OpBehaviorNotEqual attribute*), 447
opcode (*angr.engines.pcode.behavior.OpBehaviorPiece attribute*), 463
opcode (*angr.engines.pcode.behavior.OpBehaviorPopcount attribute*), 464
opcode (*angr.engines.pcode.behavior.OpBehaviorSubpiece attribute*), 463
Opcode (*class in angr.analyses.disassembly*), 848
OpDescriptor (*class in angr.analyses.decompiler.optimization_passes.expr_op_swap_pass*), 709
open() (*angr.state_plugins.posix.SimSystemPosix method*), 245
open_db() (*angr.angrdb.db.AngrDB static method*), 674
open_socket() (*angr.state_plugins.posix.SimSystemPosix method*), 246
opening_symbol (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator attribute*), 737
operand (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp attribute*), 734
Operand (*class in angr.analyses.disassembly*), 848
Operand (*class in angr.analyses.reassembler*), 852
operand_str (*angr.analyses.reassembler.DataLabel property*), 851
operand_str (*angr.analyses.reassembler.FunctionLabel property*), 851
operand_str (*angr.analyses.reassembler.Label property*), 851
operand_str (*angr.analyses.reassembler.NotypeLabel property*), 851
operand_str (*angr.analyses.reassembler.ObjectLabel property*), 851
OperandPiece (*class in angr.analyses.disassembly*), 849
operands (*angr.engines.light.data.ArithmeticExpression attribute*), 750
OPERATE (*angr.state_plugins.sim_action.SimActionData attribute*), 467
operations (*angr.engines.pcode.lifter.IRSB property*), 438
operations (*angr.knowledge_plugins.functions.function.Function property*), 554
Oppologist (*class in angr.exploration_techniques*), 399
Oppologist (*class in angr.exploration_techniques.oppologist*), 420
opt_level (*angr.engines.pcode.lifter.Lifter attribute*), 440
opt_level (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
OptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 703
OptimizationPassStage (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 702
optimize() (*angr.analyses.binary_optimizer.BinaryOptimizer method*), 860
optimize() (*angr.analyses.decompiler.peephole_optimizations.base.Peeph method*), 713
optimize() (*angr.analyses.decompiler.peephole_optimizations.base.Peeph method*), 713
optimize() (*angr.analyses.decompiler.peephole_optimizations.base.Peeph method*), 712
OPTIONS (*angr.analyses.cfg.cfg_arch_options.CFGArchOptions attribute*), 656
OPTIONS (*angr.sim_state_options.SimStateOptions attribute*), 228
options (*angr.slicer.SimLightState attribute*), 871
options_to_params() (*angr.analyses.decompiler.decompiler.Decompiler static method*), 698
Or (*angr.engines.light.data.ArithmeticExpression attribute*), 750
Ora (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransferr attribute*), 662
original_node (*angr.analyses.decompiler.optimization_passes.lowered_s attribute*), 707
other_input_defns (*angr.analyses.reaching_definitions.dep_graph.Funct attribute*), 794
other_output_defns (*angr.analyses.reaching_definitions.dep_graph.Fun attribute*), 794
other_types (*angr.procedures.stubs.format_parser.FormatParser attribute*), 474
other_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
other_uses (*angr.knowledge_plugins.key_definitions.live_definitions.Live attribute*), 594
other_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 574

others (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
others (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594
others (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
OutdatedDefinitionWalker (*class in angr.analyses.propagator.outdated_definition_walker*), 754
OutdatedError, 638
OuterWalker (*class in angr.analyses.decompiler.optimization_passes.expression_prop*), 709
output() (*angr.analyses.cfg.cfg_base.CFGBase method*), 646
output() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 656
OVERFLOW_FP_RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCC attribute*), 488
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCARM attribute*), 495
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCCdecl attribute*), 491
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCO32 attribute*), 498
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
OVERFLOW_RETURN_VAL (*angr.SimCC attribute*), 185
overlap() (*angr.knowledge_plugins.patches.PatchManager static method*), 522
overwrite_tmp_value() (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.OverwriteTmpValueCallback method*), 660
OverwriteTmpValueCallback (*class in angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 660

P

packed (*angr.sim_type.SimStruct property*), 516
pad_chr (*angr.procedures.stubs.format_parser.FormatSpecifier attribute*), 474
PAGE_TYPE (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.ListPagesMixin attribute*), 356
PAGE_TYPE (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.TypedPagesMixin attribute*), 353
PAGE_TYPE (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.UltraPagesMixin attribute*), 357
PageBase (*class in angr.storage.memory_mixins.paged_memory.pages*), 359
PagedMemoryMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 352
ParameterTag (*class in angr.knowledge_plugins.key_definitions.tag*), 604
parent (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterSimplifier attribute*), 719
parent (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterSimplifierCopy attribute*), 720
parentop (*angr.analyses.disassembly.OperandPiece attribute*), 849
parents (*angr.state_plugins.history.SimStateHistory property*), 269
parents() (*angr.storage.memory_mixins.paged_memory.pages.history_trackable.HistoryTrackable method*), 361
parse() (*angr.serializable.Serializable class method*), 617
parse_block() (*angr.analyses.disassembly.Disassembly method*), 850
parse_cpp_file() (*in module angr.sim_type*), 519
parse_defns() (*in module angr.sim_type*), 518
parse_file() (*in module angr.sim_type*), 518
parse_from_cmessage() (*angr.Block class method*), 171
parse_from_cmessage() (*angr.block.Block class method*), 222
parse_from_cmessage() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel class method*), 536
parse_from_cmessage() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode class method*), 546
parse_from_cmessage() (*angr.knowledge_plugins.cfg.CFGModel class method*), 530
parse_from_cmessage() (*angr.knowledge_plugins.cfg.CFGNode class method*), 527
parse_from_cmessage() (*angr.knowledge_plugins.cfg.memory_data.MemoryData class method*), 544
parse_from_cmessage() (*angr.knowledge_plugins.cfg.MemoryData class method*), 525
parse_from_cmessage()

(*angr.knowledge_plugins.functions.function.Function class method*), 555
parse_from_cmessage() (*angr.knowledge_plugins.variables.variable_access.VariableAccess class method*), 561
parse_from_cmessage() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal class method*), 562
parse_from_cmessage() (*angr.knowledge_plugins.xrefs.xref.XRef class method*), 610
parse_from_cmessage() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager class method*), 611
parse_from_cmessage() (*angr.serializable.Serializable class method*), 616
parse_from_cmessage() (*angr.sim_variable.SimMemoryVariable class method*), 506
parse_from_cmessage() (*angr.sim_variable.SimRegisterVariable class method*), 505
parse_from_cmessage() (*angr.sim_variable.SimStackVariable class method*), 506
parse_from_cmessage() (*angr.sim_variable.SimTemporaryVariable class method*), 504
parse_from_cmsg() (*angr.knowledge_plugins.functions.function.Function static method*), 559
parse_signature() (*in module angr.sim_type*), 518
parse_stack_pointer() (*in module angr.analyses.variable_recovery.variable_recovery_base*), 819
parse_type() (*in module angr.sim_type*), 519
parse_type_with_name() (*in module angr.sim_type*), 519
parse_types() (*in module angr.sim_type*), 518
parse_variable_addr() (*angr.analyses.decompiler.clinic.Clinic method*), 695
parsedcprotos2py() (*in module angr.utils.library*), 891
ParsedInstruction (*class in angr.analyses.data_dep.sim_act_location*), 867
Patch (*class in angr.knowledge_plugins.patches*), 521
patch_addrs() (*angr.knowledge_plugins.patches.PatchManager method*), 522
patched_entry_state (*angr.knowledge_plugins.patches.PatchManager property*), 522
PatchManager (*class in angr.knowledge_plugins.patches*), 521
path (*angr.angrdb.models.DbObject attribute*), 676
path_between() (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGS method*), 815
PathUnreachableError, 894
PCAP (*class in angr.storage.pcap*), 335
PcodeBasicBlockLifter (*class in angr.engines.pcode.lifter*), 441
PcodeDisassemblerBlock (*class in angr.engines.pcode.lifter*), 435
PcodeDisassemblerInsn (*class in angr.engines.pcode.lifter*), 435
PcodeEmulatorMixin (*class in angr.engines.pcode.emulate*), 444
PcodeLifter (*class in angr.engines.pcode.lifter*), 441
PcodeLifterEngineMixin (*class in angr.engines.pcode.lifter*), 442
peek_input() (*angr.state_plugins.cgc.SimStateCGC method*), 272
peek_output() (*angr.state_plugins.cgc.SimStateCGC method*), 272
peephole_optimize_expr() (*in module angr.analyses.decompiler.utils*), 743
peephole_optimize_exprs() (*in module angr.analyses.decompiler.utils*), 743
peephole_optimize_multistmts() (*in module angr.analyses.decompiler.utils*), 744
peephole_optimize_stmts() (*in module angr.analyses.decompiler.utils*), 743
PeepholeOptimizationExprBase (*class in angr.analyses.decompiler.peephole_optimizations.base*), 713
PeepholeOptimizationMultiStmtBase (*class in angr.analyses.decompiler.peephole_optimizations.base*), 712
PeepholeOptimizationStmtBase (*class in angr.analyses.decompiler.peephole_optimizations.base*), 711
PendingJob (*class in angr.analyses.cfg.cfg_emulated*), 640
PendingJob (*class in angr.analyses.vfg*), 839
PendingJobs (*class in angr.analyses.cfg.cfg_fast*), 648
perform_call() (*angr.callable.Callable method*), 520
perm_exec (*angr.storage.memory_mixins.paged_memory.pages.permission property*), 360
perm_read (*angr.storage.memory_mixins.paged_memory.pages.permission property*), 360
perm_write (*angr.storage.memory_mixins.paged_memory.pages.permissio property*), 360
permissions() (*angr.storage.memory_mixins.address_concretization_mix method*), 346
permissions() (*angr.storage.memory_mixins.MemoryMixin method*), 337
permissions() (*angr.storage.memory_mixins.paged_memory.paged_mem

method), 354
PermissionsMixin (*class in angr.storage.memory_mixins.paged_memory.pages.permissions_mixin*), 360
Permissive (*angr.exploration_techniques.tracer.TracingMode attribute*), 414
persistent_id() (*angr.vaults.VaultPickler method*), 617
persistent_load() (*angr.vaults.VaultUnpickler method*), 617
PhoenixStructurer (*class in angr.analyses.decompiler.structuring.phoenix*), 690
PickledState (*class in angr.exploration_techniques.spiller_db*), 412
PickledStatesBase (*class in angr.exploration_techniques.spiller*), 410
PickledStatesDb (*class in angr.exploration_techniques.spiller*), 411
PickledStatesList (*class in angr.exploration_techniques.spiller*), 410
PLATFORMS (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePointerSaveSimplifier attribute*), 705
PLATFORMS (*angr.analyses.decompiler.optimization_passes.const_derefs.ConstantDereferencesSimplifier attribute*), 701
PLATFORMS (*angr.analyses.decompiler.optimization_passes.flag_setting_cmp_simplifier.FlagSettingCmpSimplifier attribute*), 705
PLATFORMS (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifier attribute*), 702
PLATFORMS (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier attribute*), 710
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 706
PLATFORMS (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier attribute*), 707
PLATFORMS (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 708
PLATFORMS (*angr.analyses.decompiler.optimization_passes.multi_simplifier.MultiSimplifier attribute*), 708
PLATFORMS (*angr.analyses.decompiler.optimization_passes.optimization_pass.BaseOptimizationPass attribute*), 703
PLATFORMS (*angr.analyses.decompiler.optimization_passes.optimization_pass.SequenceOptimizationPass attribute*), 704
PLATFORMS (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifier attribute*), 711
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 711
PLATFORMS (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 704
PLATFORMS (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier attribute*), 711
plugin_preset (*angr.misc.plugins.PluginHub property*), 222
PluginHub (*class in angr.misc.plugins*), 222
PluginPreset (*class in angr.misc.plugins*), 223
plugins (*angr.misc.mixins.PluginHostMixin property*), 225
plugins (*angr.SimState property*), 182
PluginVendor (*class in angr.misc.plugins*), 224
Pointer (*class in angr.analyses.typehoon.typeconsts*), 835
Pointer32 (*class in angr.analyses.typehoon.typeconsts*), 835
Pointer64 (*class in angr.analyses.typehoon.typeconsts*), 835
pointer_addr (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 543
pointer_addr (*angr.knowledge_plugins.cfg.MemoryData attribute*), 525
pointer_to_atom() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 813
pointer_to_atom() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 783
pointer_to_atoms() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 813
pointer_to_atoms() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 783
PointerArithmeticFixer (*class in angr.analyses.decompiler.structured_codegen.c*), 740
PointerArray (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 542
PointerArray (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
PointerWrapper (*class in angr.calling_conventions*), 184
PointerWrapper (*class in angr.calling_conventions*), 483
pop() (*angr.state_plugins.callstack.CallStack method*), 265
pop() (*angr.state_plugins.globals.SimStateGlobals method*), 278
pop_from_backup() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 278
pop_job() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 349
pop_n() (*angr.exploration_techniques.spiller.PickledStatesBase method*), 410
pop_n() (*angr.exploration_techniques.spiller.PickledStatesDb method*), 411
pop_n() (*angr.exploration_techniques.spiller.PickledStatesList method*), 411
pop_priv() (*angr.state_plugins.scratch.SimStateScratch method*), 280
pop_stack_frame() (*angr.storage.memory_mixins.javavm_memory.JavaVmMemoryMixin method*), 316
populate() (*angr.sim_manager.SimulationManager method*), 386
populate() (*angr.SimulationManager method*), 175

pos (*angr.SimFileBase attribute*), 188
pos (*angr.storage.file.SimFileBase attribute*), 316
PositionMapping (*class in angr.analyses.decompiler.structured_codegen.base*), 723
PositionMappingElement (*class in angr.analyses.decompiler.structured_codegen.base*), 723
posix (*angr.sim_state.SimState attribute*), 225
posix (*angr.SimState attribute*), 181
PosixDevFS (*class in angr.state_plugins.posix*), 240
PosixProcFS (*class in angr.state_plugins.posix*), 242
posmap_pos (*angr.analyses.decompiler.structured_codegen.base.PositionMappingElement attribute*), 723
PossibleObject (*class in angr.analyses.find_objects_static*), 845
post_dom (*angr.utils.graph.PostDominators property*), 888
PostDominators (*class in angr.utils.graph*), 888
pp() (*angr.analyses.datagraph_meta.DataGraphMeta method*), 673
pp() (*angr.analyses.ddg.DDG method*), 748
pp() (*angr.Block method*), 170
pp() (*angr.block.Block method*), 221
pp() (*angr.block.DisassemblerBlock method*), 220
pp() (*angr.engines.pcode.lifter.IRSB method*), 438
pp() (*angr.knowledge_plugins.functions.function.Function method*), 559
pp_constraints() (*angr.analyses.typehoon.typehoon.TypeChecker method*), 833
pp_solution() (*angr.analyses.typehoon.typehoon.Typehoon method*), 833
pp_str() (*angr.analyses.typehoon.typeconsts.TypeConstant method*), 834
pp_str() (*angr.analyses.typehoon.typevars.Add method*), 830
pp_str() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 831
pp_str() (*angr.analyses.typehoon.typevars.Equivalence method*), 829
pp_str() (*angr.analyses.typehoon.typevars.Existence method*), 829
pp_str() (*angr.analyses.typehoon.typevars.Sub method*), 830
pp_str() (*angr.analyses.typehoon.typevars.Subtype method*), 830
pp_str() (*angr.analyses.typehoon.typevars.TypeConstraint method*), 829
pp_str() (*angr.analyses.typehoon.typevars.TypeVariable method*), 831
preconstrain() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 283
preconstrain_file() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 283
preconstrain_flag_page() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 283
predecessors (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 545
predecessors (*angr.knowledge_plugins.cfg.CFGNode property*), 526
predecessors() (*angr.analyses.forward_analysis.visitors.call_graph.CallGraphVisitor method*), 623
predecessors() (*angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor method*), 623
predecessors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 624
predecessors() (*angr.analyses.forward_analysis.visitors.loop.LoopVisitor method*), 626
predecessors() (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor method*), 627
predecessors() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 795
predecessors() (*angr.codenode.CodeNode method*), 873
predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 545
predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.CFGNode method*), 526
prep() (*angr.analyses.analysis.AnalysisFactory method*), 620
prep_tracer() (*angr.state_plugins.trace_additions.ChallRespInfo static method*), 276
prep_tracer() (*angr.state_plugins.trace_additions.ZenPlugin static method*), 278
prepare_call_state() (*angr.SimOS method*), 169
prepare_call_state() (*angr.simos.simos.SimOS method*), 875
prepare_callsite() (*angr.sim_state.SimState method*), 227
prepare_callsite() (*angr.SimState method*), 184
prepare_function_symbol() (*angr.SimOS method*), 169
prepare_function_symbol() (*angr.simos.linux.SimLinux method*), 877
prepare_function_symbol() (*angr.simos.simos.SimOS method*), 875
prepare_native_return_state() (*angr.engines.soot.engine.SootMixin static method*), 432
prepare_native_return_state() (*in module angr.simos.javavm*), 882
prepare_return_state() (*angr.engines.soot.engine.SootMixin static method*), 432

prepared_registers (*angr.knowledge_plugins.functions.function.Function attribute*), 553
prepared_registers (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
prepared_stack_variables (*angr.knowledge_plugins.functions.function.Function attribute*), 553
prepared_stack_variables (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
prev_chunk() (*angr.PTChunk method*), 209
prev_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 300
prev_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 303
prev_size() (*angr.PTChunk method*), 209
prev_size() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 303
print_all_chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 301
print_heap_state() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 301
PRINTABLES (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 653
prioritize_functions() (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis method*), 635
priority (*angr.exploration_techniques.spiller_db.PickledState attribute*), 412
priv (*angr.state_plugins.scratch.SimStateScratch property*), 280
PrivilegedPagingMixin (*class in angr.storage.memory_mixins.paged_memory.privileged_mixin*), 358
probably_identical (*angr.analyses.bindiff.FunctionDiff property*), 630
Procedure (*class in angr.analyses.reassembler*), 853
ProcedureChunk (*class in angr.analyses.reassembler*), 854
ProcedureEngine (*class in angr.engines.procedure*), 430
ProcedureMixin (*class in angr.engines.procedure*), 430
process() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifyAllEngine method*), 709
process() (*angr.analyses.loop_analysis.SootBlockProcessor method*), 837
process() (*angr.analyses.propagator.engine_base.SimEnginePropagatorBase method*), 753
process() (*angr.analyses.reaching_definitions.engine_ail.SimEngineRDAIL method*), 814
process() (*angr.analyses.reaching_definitions.engine_vex.SimEngineRDVEX method*), 790
process() (*angr.analyses.variable_recovery.engine_base.SimEngineVRBase method*), 827
process() (*angr.engines.engine.SimEngine method*), 427
process() (*angr.engines.engine.SuccessorsMixin method*), 428
process() (*angr.engines.light.engine.SimEngineLight method*), 751
process_exc_file() (*in module angr.flirt.build_sig*), 884
process_procedure() (*angr.engines.procedure.ProcedureMixin method*), 430
process_successors() (*angr.engines.concrete.SimEngineConcrete method*), 433
process_successors() (*angr.engines.engine.SuccessorsMixin method*), 428
process_successors() (*angr.engines.failure.SimEngineFailure method*), 431
process_successors() (*angr.engines.hook.HooksMixin method*), 430
process_successors() (*angr.engines.pcode.engine.HeavyPcodeMixin method*), 434
process_successors() (*angr.engines.procedure.ProcedureEngine method*), 430
process_successors() (*angr.engines.soot.engine.SootMixin method*), 432
process_successors() (*angr.engines.syscall.SimEngineSyscall method*), 431
process_successors() (*angr.engines.unicorn.SimEngineUnicorn method*), 432
ProgramVariable (*class in angr.analyses.ddg*), 744
project (*angr.analyses.analysis.Analysis attribute*), 621
project (*angr.analyses.backward_slice.BackwardSlice attribute*), 629
project (*angr.analyses.binary_optimizer.BinaryOptimizer attribute*), 860
project (*angr.analyses.bindiff.BinDiff attribute*), 632
project (*angr.analyses.boyscout.BoyScout attribute*), 632
project (*angr.analyses.callee_cleanup_finder.CalleeCleanupFinder attribute*), 861
project (*angr.analyses.calling_convention.CallingConventionAnalysis attribute*), 634
project (*angr.analyses.cdg.CDG attribute*), 673
project (*angr.analyses.cfg.cfb.CFBlanket attribute*), 638

project (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 656
project (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot attribute*), 670
project (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver attribute*), 663
project (*angr.analyses.class_identifier.ClassIdentifier attribute*), 846
project (*angr.analyses.code_tagging.CodeTagging attribute*), 674
project (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis attribute*), 635
project (*angr.analyses.congruency_check.CongruencyCheck attribute*), 859
project (*angr.analyses.data_dep.data_dependency_analysis.DataDependencyAnalysis attribute*), 866
project (*angr.analyses.ddg.DDG attribute*), 749
project (*angr.analyses.decompiler.ail_simplifier.AILSimplifier attribute*), 692
project (*angr.analyses.decompiler.block_simplifier.BlockSimplifier attribute*), 693
project (*angr.analyses.decompiler.callsite_maker.CallSiteMaker attribute*), 693
project (*angr.analyses.decompiler.clinic.Clinic attribute*), 695
project (*angr.analyses.decompiler.decompiler.Decompiler attribute*), 698
project (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass property*), 703
project (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase attribute*), 713
project (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationMultiStmtBase attribute*), 712
project (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase attribute*), 712
project (*angr.analyses.decompiler.region_identifier.RegionIdentifier attribute*), 714
project (*angr.analyses.decompiler.region_simplifiers.region_simplifier.RegionSimplifier attribute*), 718
project (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator attribute*), 739
project (*angr.analyses.decompiler.structured_codegen.dwarf_import.DwarfImporter attribute*), 741
project (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer attribute*), 691
project (*angr.analyses.decompiler.structuring.recursive_structurer.RecursiveStructurer attribute*), 684
project (*angr.analyses.disassembly.Disassembly attribute*), 850
project (*angr.analyses.disassembly.Value property*), 849
project (*angr.analyses.dominance_frontier.DominanceFrontier attribute*), 861
project (*angr.analyses.find_objects_static.StaticObjectFinder attribute*), 846
project (*angr.analyses.flirt.FlirtAnalysis attribute*), 750
project (*angr.analyses.identifier.identify.Identifier attribute*), 836
project (*angr.analyses.init_finder.InitializationFinder attribute*), 861
project (*angr.analyses.loop_analysis.LoopAnalysis attribute*), 838
project (*angr.analyses.loopfinder.LoopFinder attribute*), 837
project (*angr.analyses.propagator.propagator.PropagatorAnalysis attribute*), 757
project (*angr.analyses.proximity_graph.ProximityGraphAnalysis attribute*), 865
project (*angr.analyses.reaching_definitions.live_definitions.LiveDefinitions attribute*), 758
project (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis attribute*), 794
project (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis attribute*), 773
project (*angr.analyses.reassembler.Reassembler attribute*), 858
project (*angr.analyses.soot_class_hierarchy.SootClassHierarchy attribute*), 637
project (*angr.analyses.stack_pointer_tracker.StackPointerTracker attribute*), 818
project (*angr.analyses.static_hooker.StaticHooker attribute*), 850
project (*angr.analyses.typehoon.typehoon.Typehoon attribute*), 833
project (*angr.analyses.variable_recovery.variable_recovery.VariableRecovery attribute*), 826
project (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase attribute*), 820
project (*angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFast attribute*), 825
project (*angr.analyses.veritesting.Veritesting attribute*), 839
project (*angr.analyses.vfg.VFG attribute*), 844
project (*angr.analyses.vsa_ddg.VSA_DDG attribute*), 845
project (*angr.analyses.vtable.VtableFinder attribute*), 845
project (*angr.analyses.xrefs.XRefsAnalysis attribute*), 862
project (*angr.block.CapstoneInsn attribute*), 178
project (*angr.knowledge_plugins.cfg.cfg_model.CFGModel property*), 536
project (*angr.knowledge_plugins.cfg.CFGModel property*), 530
project (*angr.knowledge_plugins.functions.function.Function property*), 553
project (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594

project (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
project (*angr.procedures.stubs.format_parser.FormatParser attribute*), 475
project (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 475
project (*angr.sim_procedure.SimProcedure attribute*), 471
Project (*class in angr*), 163
Project (*class in angr.project*), 212
prop (*angr.analyses.decompiler.clinic.BlockCache attribute*), 694
prop_key (*angr.analyses.propagator.propagator.PropagatorAnalysis property*), 757
propagations (*angr.knowledge_base.knowledge_base.KnowledgeBase attribute*), 520
propagations (*angr.KnowledgeBase attribute*), 211
Propagator (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 620
PropagatorAnalysis (*class in angr.analyses.propagator.propagator*), 755
prototype (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall property*), 730
prototype (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 801
prototype (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
prototype (*angr.knowledge_plugins.functions.function.Function attribute*), 553
prototype (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
prototype (*angr.procedures.stubs.format_parser.FormatParser attribute*), 475
prototype (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 475
prototype (*angr.sim_procedure.SimProcedure attribute*), 471
ProximityGraphAnalysis (*class in angr.analyses.proximity_graph*), 865
ProxiNodeTypes (*class in angr.analyses.proximity_graph*), 863
prune() (*angr.sim_manager.SimulationManager method*), 386
prune() (*angr.SimulationManager method*), 175
pruned (*angr.sim_manager.SimulationManager attribute*), 383
pruned (*angr.SimulationManager attribute*), 172
pseudocode (*angr.knowledge_plugins.functions.function.Function property*), 556
PTChunk (*class in angr*), 208
PTChunk (*class in angr.state_plugins.heap.heap_ptmalloc*), 302
PTChunkIterator (*class in angr.state_plugins.heap.heap_ptmalloc*), 304
pull() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 608
pull_comment() (*angr.knowledge_plugins.sync.sync_controller.SyncCont method*), 609
pull_comments() (*angr.knowledge_plugins.sync.sync_controller.SyncCon method*), 609
pull_function() (*angr.knowledge_plugins.sync.sync_controller.SyncCon method*), 609
pull_patches() (*angr.knowledge_plugins.sync.sync_controller.SyncCont method*), 609
pull_stack_variables() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 609
push() (*angr.state_plugins.callstack.CallStack method*), 265
push_comment() (*angr.knowledge_plugins.sync.sync_controller.SyncCont method*), 608
push_comments() (*angr.knowledge_plugins.sync.sync_controller.SyncCon method*), 608
push_function() (*angr.knowledge_plugins.sync.sync_controller.SyncCon method*), 608
push_priv() (*angr.state_plugins.scratch.SimStateScratch method*), 280
push_stack_frame() (*angr.storage.memory_mixins.javavm_memory.java method*), 376
push_stack_variable() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 608
push_stack_variables() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 608
put() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 818
PutHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 664

Q

qualifies_for_implicit_cast() (*in module angr.analyses.decompiler.structured_codegen.c*), 724
qualifies_for_simple_cast() (*in module angr.analyses.decompiler.structured_codegen.c*), 724
quasi_topological_sort_nodes() (*angr.utils.graph.GraphUtils static method*), 889

R

ran_cca (*angr.knowledge_plugins.functions.function.Function attribute*), 553
ran_cca (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 561

RANDOM (*angr.simos.windows.SecurityCookieInit attribute*), 879
randomize_procedures() (*angr.analyses.reassembler.Reassembler method*), 858
rd (*angr.analyses.decompiler.clinic.BlockCache attribute*), 694
rda_observe_callback() (*angr.knowledge_plugins.key_definitions.key_definition_manager.RDAObserverControl method*), 592
RDAObserverControl (*class in angr.knowledge_plugins.key_definitions.key_definition_manager*), 591
reachable() (*angr.state_plugins.history.SimStateHistory method*), 269
reached_fixedpoint() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 626
reaching_condition (*angr.analyses.decompiler.structuring.structurer_nodes.MultiNode attribute*), 686
reaching_condition (*angr.analyses.decompiler.structuring.structurer_nodes.SequenceNode attribute*), 686
ReachingDefinitions (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 620
ReachingDefinitionsAnalysis (*class in angr.analyses.reaching_definitions*), 770
ReachingDefinitionsAnalysis (*class in angr.analyses.reaching_definitions.reaching_definitions*), 791
ReachingDefinitionsModel (*class in angr.analyses.reaching_definitions*), 774
ReachingDefinitionsModel (*class in angr.knowledge_plugins.key_definitions*), 570
ReachingDefinitionsModel (*class in angr.knowledge_plugins.key_definitions.rd_model*), 601
ReachingDefinitionsState (*class in angr.analyses.reaching_definitions*), 776
ReachingDefinitionsState (*class in angr.analyses.reaching_definitions.rd_state*), 805
READ (*angr.knowledge_plugins.variables.variable_access.VariableAccessSort attribute*), 561
Read (*angr.knowledge_plugins.xrefs.xref_types.XRefType attribute*), 611
READ (*angr.state_plugins.sim_action.SimActionData attribute*), 467
read() (*angr.SimFile method*), 190
read() (*angr.SimFileBase method*), 189
read() (*angr.SimFileStream method*), 194
read() (*angr.SimPackets method*), 192
read() (*angr.SimPacketsStream method*), 196
read() (*angr.storage.file.SimFile method*), 317
read() (*angr.storage.file.SimFileBase method*), 316
read() (*angr.storage.file.SimFileDescriptorBase method*), 325
read() (*angr.storage.file.SimFileStream method*), 319
read() (*angr.storage.file.SimPackets method*), 321
read() (*angr.storage.file.SimPacketsSlots method*), 332
read() (*angr.storage.file.SimPacketsStream method*), 323
read_data() (*angr.SimFileDescriptor method*), 198
read_data() (*angr.SimFileDescriptorDuplex method*), 200
read_data() (*angr.storage.file.SimFileDescriptor method*), 327
read_data() (*angr.storage.file.SimFileDescriptorBase method*), 325
read_data() (*angr.storage.file.SimFileDescriptorDuplex method*), 329
read_from() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 563
read_mem() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
read_pos (*angr.SimFileDescriptor property*), 199
read_pos (*angr.SimFileDescriptorDuplex property*), 201
read_pos (*angr.storage.file.SimFileDescriptor property*), 328
read_pos (*angr.storage.file.SimFileDescriptorBase property*), 326
read_pos (*angr.storage.file.SimFileDescriptorDuplex property*), 330
read_storage (*angr.SimFileDescriptor property*), 199
read_storage (*angr.SimFileDescriptorDuplex property*), 201
read_storage (*angr.storage.file.SimFileDescriptor property*), 328
read_storage (*angr.storage.file.SimFileDescriptorBase property*), 326
read_storage (*angr.storage.file.SimFileDescriptorDuplex property*), 330
real_args (*angr.calling_conventions.UsercallArgSession attribute*), 487
real_length() (*angr.utils.dynamic_dictlist.DynamicDictList method*), 886
realloc() (*angr.SimHeapPTMalloc method*), 207
realloc() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 302
realloc() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 305
reapply_options() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 724
reapply_options() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 738
Reassembler (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619

Reassembler (*class in angr.analyses.reassembler*), 855
ReassemblerFailureNotice, 850
rebuild_callgraph() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 552
recent_actions (*angr.state_plugins.history.SimStateHistory property*), 269
recent_constraints (*angr.state_plugins.history.SimStateHistory property*), 269
reconstrain() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 284
record_state() (*angr.errors.SimError method*), 896
record_variable() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 563
recover_edge_condition() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 695
recover_edge_conditions() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 695
recover_reaching_conditions() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 695
recurse_analysis() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 804
recurse_analysis() (*angr.analyses.reaching_definitions.FunctionHandler method*), 785
recursive_copy() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 700
RecursiveStructurer (*class in angr.analyses.decompiler.structuring.recursive_structurer*), 684
RecursiveType (*class in angr.analyses.typehoon.simple_solver*), 828
recv() (*angr.storage.pcap.PCAP method*), 335
redefine_locals (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 801
redefine_locals (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
RedundantLabelRemover (*class in angr.analyses.decompiler.redundant_label_remover*), 722
RedundantStackVariable (*class in angr.analyses.binary_optimizer*), 859
RefcountMixin (*class in angr.storage.memory_mixins.paged_memory.pages.refcount_mixin*), 359
REFERENCE (*angr.knowledge_plugins.variables.variable_access.VariableAccessSort attribute*), 561
reference_at() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 563
reference_size (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 543
reference_size (*angr.knowledge_plugins.cfg.MemoryData attribute*), 525
reference_values (*angr.analyses.decompiler.structured_codegen.c.CCon attribute*), 735
refine() (*angr.calling_conventions.SimFunctionArgument method*), 484
refine() (*angr.calling_conventions.SimLyingRegArg method*), 491
refine() (*angr.calling_conventions.SimRegArg method*), 485
refine() (*angr.calling_conventions.SimStackArg method*), 485
refine_locs_with_struct_type() (*in module angr.analyses.variable_recovery.variable_recovery_base*), 483
reflow_variable_types() (*angr.analyses.decompiler.decompiler.Decompiler method*), 698
reg (*angr.analyses.decompiler.structured_codegen.c.CRegister attribute*), 736
reg (*angr.analyses.stack_pointer_tracker.OffsetVal property*), 817
reg (*angr.engines.light.data.RegisterOffset attribute*), 750
reg (*angr.sim_variable.SimRegisterVariable attribute*), 504
REG (*angr.state_plugins.sim_action.SimAction attribute*), 466
reg() (*angr.analyses.reaching_definitions.Atom static method*), 766
reg() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 585
reg_concrete() (*angr.sim_state.SimState method*), 227
reg_concrete() (*angr.SimState method*), 183
reg_deps (*angr.state_plugins.sim_action.SimAction property*), 466
reg_deps (*angr.state_plugins.sim_action.SimActionData property*), 468
reg_name (*angr.knowledge_plugins.key_definitions.definition.DefinitionMa attribute*), 589
reg_offset (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffs attribute*), 663
reg_offset (*angr.analyses.reaching_definitions.Register attribute*), 768
reg_offset (*angr.knowledge_plugins.key_definitions.atoms.Register attribute*), 587
reg_read_callback() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.ConstantVa method*), 663
reg_vex_offset (*angr.analyses.data_dep.dep_nodes.RegDepNode property*), 870
RegDepNode (*class in angr.analyses.data_dep.dep_nodes*), 869
regenerate_text() (*angr.analyses.decompiler.structured_codegen.base.I method*), 724
regenerate_text() (*angr.analyses.decompiler.structured_codegen.c.CSt

method), 738
regenerate_text() (*angr.analyses.decompiler.structured_codegen.dummy.DummyStructuredCodeGenerator method*), 741
region (*angr.sim_variable.SimVariable attribute*), 503
region (*angr.storage.memory_mixins.regioned_memory.region_data.AddressWrapper attribute*), 369
region_base_addr (*angr.storage.memory_mixins.regioned_memory.region_data.AddressWrapper attribute*), 369
region_id (*angr.storage.memory_mixins.regioned_memory.region_data.AddressWrapper attribute*), 369
region_ids (*angr.storage.memory_mixins.regioned_memory.RegionedMemoryMixin property*), 369
RegionCategoryMixin (*class in angr.storage.memory_mixins.regioned_memory.region_category_mixin*), 370
RegionDescriptor (*class in angr.storage.memory_mixins.regioned_memory.region_data*), 369
RegionedAddressConcretizationMixin (*class in angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin*), 373
RegionedMemory (*class in angr.storage.memory_mixins*), 339
RegionedMemoryMixin (*class in angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin*), 365
RegionIdentifier (*class in angr.analyses.decompiler.region_identifier*), 713
RegionMap (*class in angr.storage.memory_mixins.regioned_memory.region_data*), 369
RegionObject (*class in angr.keyed_region*), 613
regions (*angr.analyses.cfg.cfb.CFBlanket property*), 637
RegionSimplifier (*class in angr.analyses.decompiler.region_simplifiers.region_simplifier*), 718
RegionWalker (*class in angr.analyses.decompiler.region_walker*), 722
Register (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 867
register (*angr.analyses.disassembly.RegisterOperand property*), 849
REGISTER (*angr.analyses.reaching_definitions.AtomKind attribute*), 765
REGISTER (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 584
REGISTER (*angr.knowledge_plugins.variables.variable_manager.VariableType attribute*), 562
Register (*class in angr.analyses.disassembly*), 849
Register (*class in angr.analyses.reaching_definitions*), 768
Register (*class in angr.analyses.stack_pointer_tracker*), 817
register (*angr.knowledge_plugins.key_definitions.atoms.Atom property*), 587
register() (*angr.analyses.reaching_definitions.Atom method*), 767
register() (*angr.knowledge_plugins.key_definitions.atoms.Atom method*), 585
register() (*angr.utils.mp.Initializer method*), 893
register_analysis() (*in module angr*), 178
register_analysis() (*in module angr.analyses*), 619
register_bool_option() (*angr.sim_state_options.SimStateOptions class method*), 231
register_callbacks() (*angr.analyses.variable_recovery.variable_recovery.VariableRecovery method*), 825
register_data_reference() (*angr.analyses.reassembler.Reassembler method*), 857
register_default() (*angr.knowledge_plugins.plugin.KnowledgeBasePlugin static method*), 523
register_default() (*angr.misc.plugins.PluginHub class method*), 222
register_default() (*angr.SimStatePlugin class method*), 162
register_default() (*angr.state_plugins.plugin.SimStatePlugin class method*), 233
register_default_cc() (*in module angr.calling_conventions*), 502
register_definitions (*angr.analyses.reaching_definitions.LiveDefinitions property*), 758
register_definitions (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions property*), 594
register_definitions (*angr.knowledge_plugins.key_definitions.LiveDefinitions property*), 574
register_function_analysis() (*angr.analyses.vfg.CallAnalysis method*), 841
register_instruction_reference() (*angr.analyses.reassembler.Reassembler method*), 857
register_kernel_types() (*in module angr.utils.library*), 891
register_optimization_pass() (*in module angr.analyses.decompiler.optimization_passes*), 701
register_option() (*angr.sim_state_options.SimStateOptions class method*), 231
register_pcode_arch_default_cc() (*in module angr.engines.pcode.cc*), 466
register_plugin() (*angr.knowledge_base.knowledge_base.KnowledgeBa

method), 521
register_plugin() (*angr.KnowledgeBase method*), 211
register_plugin() (*angr.misc.plugins.PluginHub method*), 223
register_plugin() (*angr.misc.plugins.PluginVendor method*), 224
register_plugin() (*angr.sim_state.SimState method*), 226
register_plugin() (*angr.SimState method*), 182
register_preset() (*angr.misc.plugins.PluginHub class method*), 222
register_region (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal attribute*), 562
register_simos() (*in module angr.simos*), 875
register_syscall_cc() (*in module angr.calling_conventions*), 502
register_types() (*in module angr.sim_type*), 518
register_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
register_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 809
register_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
register_uses (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594
register_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
register_values (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 285
register_values_count (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 285
register_variable() (*angr.state_plugins.solver.SimSolver method*), 255
RegisterInitializerHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 664
RegisterOffset (*class in angr.engines.light.data*), 750
RegisterOperand (*class in angr.analyses.disassembly*), 849
RegisterReallocation (*class in angr.analyses.binary_optimizer*), 859
registers (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
registers (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 809
registers (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
registers (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594
registers (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
registers (*angr.sim_state.SimState attribute*), 225
registers (*angr.SimState attribute*), 181
registers_read_afterwards (*angr.knowledge_plugins.functions.function.Function attribute*), 553
registers_read_afterwards (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
RegisterSaveAreaSimplifier (*class in angr.analyses.decompiler.optimization_passes.register_save_area*), 710
RegisterValue (*class in angr.state_plugins.unicorn_engine*), 285
RegOffsetAnnotation (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 663
regs (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerState attribute*), 817
regs (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState attribute*), 817
regs (*angr.sim_state.SimState attribute*), 225
regs (*angr.SimState attribute*), 181
regs (*angr.simos.userland.SimLightState attribute*), 871
regs_to_initialize (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffsetAnnotation attribute*), 663
rehook_symbol() (*angr.Project method*), 166
rehook_symbol() (*angr.project.Project method*), 216
ReinterpretAs (*class in angr.analyses.typehoon.typevars*), 832
related_function_addr (*angr.storage.memory_mixins.regioned_memory.region_meta_mi property*), 371
related_function_address (*angr.storage.memory_mixins.regioned_memory.region_data.Reg attribute*), 369
relativize() (*angr.storage.memory_mixins.regioned_memory.region_da method*), 370
release() (*angr.analyses.cfg.segment_list.SegmentList method*), 672
release() (*angr.SimHeapBrk method*), 205
release() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 299
release_plugin() (*angr.knowledge_base.knowledge_base.KnowledgeBas method*), 521
release_plugin() (*angr.KnowledgeBase method*), 211
release_plugin() (*angr.misc.plugins.PluginHub method*), 223
release_plugin() (*angr.misc.plugins.PluginVendor method*), 224
release_shared() (*angr.storage.memory_mixins.paged_memory.pages.re method*), 360
reload_analyses() (*angr.analyses.analysis.AnalysesHub method*), 619
reload_format() (*angr.analyses.disassembly.Instruction*

method), 847
reload_solver() (*angr.state_plugins.solver.SimSolver method*), 254
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 724
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 739
relocatable (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffsetAnnotation property*), 663
relocatable (*angr.analyses.data_dep.data_dependency_analysis.NodeAnnotation property*), 865
relocatable (*angr.analyses.variable_recovery.annotations.StackLocationAnnotation property*), 819
relocatable (*angr.analyses.variable_recovery.annotations.VariableSetAnnotation property*), 819
relocatable (*angr.analyses.variable_recovery.variable_recovery_base.VariableAnnotation property*), 820
relocatable (*angr.knowledge_plugins.key_definitions.live_definitions.MultiDefinitionAnnotation property*), 593
relocateable (*angr.storage.memory_mixins.address_concretization_mixin.MultiwriteAnnotation property*), 344
Relocation (*class in angr.analyses.reassembler*), 855
relocations (*angr.analyses.reassembler.Reassembler property*), 855
remove() (*angr.sim_state_options.SimStateOptions method*), 230
remove_breakpoint() (*angr.state_plugins.inspect.SimInspector method*), 234
remove_cgc_attachments() (*angr.analyses.reassembler.Reassembler method*), 858
remove_claripy_bool_asts() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 695
remove_cycles() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 643
remove_edge() (*angr.analyses.cfg.cfg_base.CFGBase method*), 647
remove_empty_nodes() (*angr.analyses.decompiler.clinic.Clinic static method*), 695
remove_fakerets() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 644
remove_instruction() (*angr.analyses.reassembler.Reassembler method*), 857
remove_labels() (*in module angr.analyses.decompiler.utils*), 743
remove_last_statement() (*in module angr.analyses.decompiler.utils*), 741
remove_node() (*angr.analyses.decompiler.structuring.structurer_nodes.SequenceNode method*), 685
remove_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 537
remove_node() (*angr.knowledge_plugins.cfg.CFGModel method*), 530
remove_node_and_graph_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 541
remove_node_and_graph_node() (*angr.knowledge_plugins.cfg.CFGModel method*), 535
remove_patch() (*angr.knowledge_plugins.patches.PatchManager method*), 522
remove_preconstraints() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 282
remove_technique() (*angr.sim_manager.SimulationManager method*), 178
remove_technique() (*angr.SimulationManager method*), 173
remove_types() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 586
remove_unnecessary_stuff() (*angr.analyses.reassembler.Reassembler method*), 858
remove_unnecessary_stuff_glibc() (*angr.analyses.reassembler.Reassembler method*), 858
remove_use() (*angr.knowledge_plugins.key_definitions.Uses method*), 582
remove_use() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 606
remove_uses() (*angr.knowledge_plugins.key_definitions.Uses method*), 582
remove_uses() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 606
remove_variable() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifierAILEngine method*), 709
RemoveNodeNotice, 692
rename() (*angr.knowledge_plugins.types.TypesStore method*), 549
renamed (*angr.sim_variable.SimVariable attribute*), 503
render() (*angr.analyses.disassembly.Disassembly method*), 850
render() (*angr.analyses.disassembly.DisassemblyPiece method*), 846
render_text() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 739
RENDER_TYPE (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator attribute*), 738
RepHook (*class in angr.exploration_techniques.tracer*), 414
replace() (*angr.analyses.typehoon.typevars.AddN method*), 830

replace() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 831
replace() (*angr.analyses.typehoon.typevars.Existence method*), 829
replace() (*angr.analyses.typehoon.typevars.Sub method*), 830
replace() (*angr.analyses.typehoon.typevars.Subtype method*), 830
replace() (*angr.keyed_region.KeyedRegion method*), 614
replace() (*angr.procedures.stubs.format_parser.FormatString method*), 473
replace_all() (*angr.storage.memory_mixins.convenient_mappings_mixin.ConvenientMappingsMixin method*), 348
replace_all() (*angr.storage.memory_mixins.MemoryMixin method*), 338
replace_all() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 367
replace_all_with_offsets() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 365
replace_last_statement() (*in module angr.analyses.decompiler.utils*), 741
replace_node_in_node() (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase static method*), 690
replace_nodes() (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase static method*), 690
replace_region() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 700
replace_region_with_region() (*angr.analyses.decompiler.graph_region.GraphRegion method*), 700
replacements (*angr.analyses.propagator.propagator.PropagatorAnalysis property*), 757
report() (*angr.exploration_techniques.Suggestions static method*), 407
report() (*angr.exploration_techniques.suggestions.Suggestions static method*), 427
repr_addr() (*in module angr.codenode*), 873
request_knowledge() (*angr.knowledge_base.knowledge_base.KnowledgeBase method*), 521
request_knowledge() (*angr.KnowledgeBase method*), 212
request_plugin() (*angr.misc.plugins.PluginPreset method*), 223
REQUIRE_CFG_STATES (*angr.exploration_techniques.CallFunctionGoal attribute*), 401
REQUIRE_CFG_STATES (*angr.exploration_techniques.director.BaseGoal attribute*), 418
REQUIRE_CFG_STATES (*angr.exploration_techniques.director.CallFunctionGoal attribute*), 419
REQUIRE_DATA_C (*angr.engines.pcode.lifter.Lifter attribute*), 439
REQUIRE_DATA_PY (*angr.engines.pcode.lifter.Lifter attribute*), 439
reraise() (*angr.sim_manager.ErrorRecord method*), 389
reset() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 625
reset() (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor method*), 627
reset() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 288
reset_initial_regs() (*angr.Block static method*), 170
reset_initial_regs() (*angr.block.Block static method*), 221
reset_prototype() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 801
reset_prototype() (*angr.analyses.reaching_definitions.FunctionCallDataCollector method*), 788
reset_uses() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 758
reset_uses() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 595
reset_uses() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 574
resolvable (*angr.state_plugins.debug_variables.SimDebugVariable property*), 308
resolvable (*angr.state_plugins.view.SimMemView property*), 314
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got.AMD64ElfGotResolver method*), 658
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast.ArmElfFastResolver method*), 659
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.const_resolver.ConstantResolver method*), 667
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver method*), 666
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.MipsElfFastResolver method*), 661
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.resolver.IndirectJumpResolver method*), 667
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt.X86ElfPicPltResolver method*), 661
resolve() (*angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat.X86PeIatResolver method*), 659
resolve_abstract_dispatch() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 637
resolve_concrete_dispatch() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 637
resolve_invoke() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 637
resolve_register() (*angr.state_plugins.light_registers.SimLightRegisterPlugin*

method), 267
resolve_special_dispatch() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 637
resolved (*angr.state_plugins.debug_variables.SimDebugVariable property*), 308
resolved (*angr.state_plugins.view.SimMemView property*), 314
resolved_indirect_jumps (*angr.knowledge_base.knowledge_base.KnowledgeBase property*), 520
resolved_indirect_jumps (*angr.KnowledgeBase property*), 211
resolved_targets (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
resolved_targets (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
resource_event() (*in module angr.state_plugins.sim_event*), 468
restore_graph() (*angr.analyses.decompiler.optimization_passes.cross_jump_reverter.CrossJumpReverter static method*), 707
result (*angr.analyses.decompiler.ccall_rewriters.rewriter_base.CCallRewriterBase attribute*), 693
resume() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 642
resume_with_new_graph() (*angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor method*), 623
resymbolize() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 307
ret() (*angr.sim_procedure.SimProcedure method*), 472
ret() (*angr.SimProcedure method*), 160
ret() (*angr.state_plugins.callstack.CallStack method*), 266
ret_addr (*angr.analyses.cfg.cfg_fast.FunctionCallEdge attribute*), 650
ret_atoms (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 801
ret_atoms (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
ret_defns (*angr.analyses.reaching_definitions.dep_graph.FunctionCallRelation attribute*), 794
ret_errno() (*angr.state_plugins.libc.SimStateLibc method*), 240
ret_expr (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 730
ret_from_addr (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge attribute*), 651
ret_sites (*angr.knowledge_plugins.functions.function.Function property*), 555
ret_target (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
ret_to_addr (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge attribute*), 651
ret_values (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 801
ret_values (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
ret_values_deps (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 801
ret_values_deps (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
retaddr_on_stack (*angr.knowledge_plugins.functions.function.Function attribute*), 553
retaddr_on_stack (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
retaddr_popped (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 801
retaddr_popped (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
RetAddrSaveSimplifier (*class in angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier*), 711
returning (*angr.knowledge_plugins.functions.function.Function property*), 555
RETURN_ADDR (*angr.calling_conventions.SimCC attribute*), 488
return_addr (*angr.calling_conventions.SimCC property*), 489
RETURN_ADDR (*angr.calling_conventions.SimCCAArch64 attribute*), 496
RETURN_ADDR (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 497
RETURN_ADDR (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 495
RETURN_ADDR (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 495
RETURN_ADDR (*angr.calling_conventions.SimCCARM attribute*), 495
RETURN_ADDR (*angr.calling_conventions.SimCCARMHF attribute*), 496
RETURN_ADDR (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 496
RETURN_ADDR (*angr.calling_conventions.SimCCCdecl attribute*), 492
RETURN_ADDR (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
RETURN_ADDR (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 492
RETURN_ADDR (*angr.calling_conventions.SimCCN64 attribute*), 498
RETURN_ADDR (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 499
RETURN_ADDR (*angr.calling_conventions.SimCCO32 attribute*), 497
RETURN_ADDR (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 498
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPC

attribute), 499
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPC64 attribute*), 500
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 500
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 499
RETURN_ADDR (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 497
RETURN_ADDR (*angr.calling_conventions.SimCCS390X attribute*), 501
RETURN_ADDR (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 502
RETURN_ADDR (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
RETURN_ADDR (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 493
RETURN_ADDR (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 494
RETURN_ADDR (*angr.engines.pcode.cc.SimCCM68k attribute*), 464
RETURN_ADDR (*angr.engines.pcode.cc.SimCCPARISC attribute*), 465
RETURN_ADDR (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 465
RETURN_ADDR (*angr.engines.pcode.cc.SimCCRISCV attribute*), 464
RETURN_ADDR (*angr.engines.pcode.cc.SimCCSH4 attribute*), 465
RETURN_ADDR (*angr.engines.pcode.cc.SimCCSPARC attribute*), 465
RETURN_ADDR (*angr.engines.pcode.cc.SimCCXtensa attribute*), 466
RETURN_ADDR (*angr.SimCC attribute*), 185
return_addr (*angr.SimCC property*), 186
return_in_implicit_outparam() (*angr.calling_conventions.SimCC method*), 489
return_in_implicit_outparam() (*angr.calling_conventions.SimCCCdecl method*), 492
return_in_implicit_outparam() (*angr.calling_conventions.SimCCMicrosoftAMD64 method*), 493
return_in_implicit_outparam() (*angr.calling_conventions.SimCCSystemVAMD64 method*), 495
return_in_implicit_outparam() (*angr.SimCC method*), 186
return_target (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 547
return_target (*angr.knowledge_plugins.cfg.CFGENode attribute*), 528
return_to (*angr.analyses.cfg.cfg_fast.FunctionReturn attribute*), 648
return_type (*angr.sim_procedure.SimProcedure property*), 473
return_type (*angr.SimProcedure property*), 161
RETURN_VAL (*angr.calling_conventions.SimCC attribute*), 488
RETURN_VAL (*angr.calling_conventions.SimCCAArch64 attribute*), 497
RETURN_VAL (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 497
RETURN_VAL (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 495
RETURN_VAL (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 495
RETURN_VAL (*angr.calling_conventions.SimCCARM attribute*), 495
RETURN_VAL (*angr.calling_conventions.SimCCARMHF attribute*), 496
RETURN_VAL (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 496
RETURN_VAL (*angr.calling_conventions.SimCCCdecl attribute*), 491
RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 492
RETURN_VAL (*angr.calling_conventions.SimCCN64 attribute*), 498
RETURN_VAL (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 499
RETURN_VAL (*angr.calling_conventions.SimCCO32 attribute*), 497
RETURN_VAL (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 498
RETURN_VAL (*angr.calling_conventions.SimCCPowerPC attribute*), 499
RETURN_VAL (*angr.calling_conventions.SimCCPowerPC64 attribute*), 500
RETURN_VAL (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 500
RETURN_VAL (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 499
RETURN_VAL (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 497
RETURN_VAL (*angr.calling_conventions.SimCCS390X attribute*), 501
RETURN_VAL (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 502
RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
RETURN_VAL (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 493
RETURN_VAL (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 494

RETURN_VAL (*angr.engines.pcode.cc.SimCCM68k attribute*), 464
RETURN_VAL (*angr.engines.pcode.cc.SimCCPARISC attribute*), 465
RETURN_VAL (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 465
RETURN_VAL (*angr.engines.pcode.cc.SimCCRISCV attribute*), 465
RETURN_VAL (*angr.engines.pcode.cc.SimCCSH4 attribute*), 465
RETURN_VAL (*angr.engines.pcode.cc.SimCCSPARC attribute*), 465
RETURN_VAL (*angr.engines.pcode.cc.SimCCXtensa attribute*), 466
RETURN_VAL (*angr.SimCC attribute*), 185
return_val() (*angr.calling_conventions.SimCC method*), 489
return_val() (*angr.calling_conventions.SimCCCdecl method*), 492
return_val() (*angr.calling_conventions.SimCCSystemVAMD64 method*), 494
return_val() (*angr.calling_conventions.SimCCUsercall method*), 491
return_val() (*angr.SimCC method*), 186
returning (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 730
returning (*angr.knowledge_plugins.functions.function.Function property*), 553
returning_source (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
returnty (*angr.sim_type.SimTypeCppFunction attribute*), 514
ReturnValueTag (*class in angr.knowledge_plugins.key_definitions.tag*), 605
retval (*angr.analyses.decompiler.structured_codegen.c.CReturn attribute*), 730
reverse_post_order_sort_nodes() (*angr.utils.graph.GraphUtils static method*), 889
revisit_node() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 626
revisit_successors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 625
rhs (*angr.analyses.decompiler.structured_codegen.c.CAssignment attribute*), 729
rhs (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp attribute*), 734
RichR (*class in angr.analyses.variable_recovery.engine_base*), 826
RShift (*angr.engines.light.data.ArithmeticExpression attribute*), 750
run() (*angr.analyses.congruency_check.CongruencyCheck method*), 859
run() (*angr.analyses.identifier.identify.Identifier method*), 836
run() (*angr.distributed.server.Server method*), 900
run() (*angr.distributed.worker.Worker method*), 901
run() (*angr.exploration_techniques.tracer.RepHook method*), 414
run() (*angr.Server method*), 211
run() (*angr.sim_manager.SimulationManager method*), 384
run() (*angr.sim_procedure.SimProcedure method*), 472
run() (*angr.SimProcedure method*), 159
run() (*angr.SimulationManager method*), 173
run_pelf() (*in module angr.flirt.build_sig*), 884
run_sigmake() (*in module angr.flirt.build_sig*), 884

S

s2u() (*in module angr.analyses.decompiler.optimization_passes.register_sa* 710
s2u() (*in module angr.analyses.decompiler.optimization_passes.stack_cana* 704
satisfiable() (*angr.sim_state.SimState method*), 226
satisfiable() (*angr.SimState method*), 183
satisfiable() (*angr.state_plugins.solver.SimSolver method*), 259
save_info() (*angr.angrdb.db.AngrDB static method*), 674
SCANF_DELIMITERS (*angr.procedures.stubs.format_parser.FormatString attribute*), 473
ScanfFormatParser (*class in angr.procedures.stubs.format_parser*), 475
scc_id (*angr.utils.graph.SCCPlaceholder attribute*), 888
SCCPlaceholder (*class in angr.utils.graph*), 888
scratch (*angr.sim_state.SimState attribute*), 225
scratch (*angr.SimState attribute*), 181
sdiv() (*angr.state_plugins.sim_action_object.SimActionObject method*), 468
se (*angr.sim_state.SimState property*), 225
se (*angr.SimState property*), 182
search() (*angr.analyses.cfg.segment_list.SegmentList method*), 671
section_alignment() (*angr.analyses.reassembler.Reassembler method*), 856
SecurityCookieInit (*class in angr.simos.windows*), 879
seek() (*angr.SimFileDescriptor method*), 198
seek() (*angr.SimFileDescriptorDuplex method*), 201
seek() (*angr.storage.file.SimFileDescriptor method*), 327
seek() (*angr.storage.file.SimFileDescriptorBase method*), 326
seek() (*angr.storage.file.SimFileDescriptorDuplex method*), 330

seekable (*angr.SimFileBase attribute*), 188
seekable (*angr.storage.file.SimFileBase attribute*), 316
seekable (*angr.storage.file.SimPacketsSlots attribute*), 332
SegfaultError, 288
Segment (*class in angr.analyses.cfg.segment_list*), 670
SegmentBoundary (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 542
SegmentBoundary (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
SegmentList (*class in angr.analyses.cfg.segment_list*), 671
selector() (*angr.exploration_techniques.ExplorationTechnique method*), 391
selector() (*angr.ExplorationTechnique method*), 179
selector() (*angr.sim_manager.SimulationManager method*), 386
selector() (*angr.SimulationManager method*), 175
seq (*angr.analyses.disassembly.IROp attribute*), 847
sequence_matcher_similarity() (*angr.exploration_techniques.unique.UniqueSearch static method*), 424
sequence_matcher_similarity() (*angr.exploration_techniques.UniqueSearch static method*), 404
sequence_node (*angr.analyses.decompiler.structuring.structurer_nodes.LoopNode attribute*), 687
SequenceNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 685
SequenceOptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 704
SequenceWalker (*class in angr.analyses.decompiler.sequence_walker*), 722
Serializable (*class in angr.serializable*), 616
SerializableCounter (*class in angr.calling_conventions*), 484
SerializableIterator (*class in angr.calling_conventions*), 483
SerializableListIterator (*class in angr.calling_conventions*), 484
serialize() (*angr.knowledge_plugins.functions.function_server.FunctionServer static method*), 559
serialize() (*angr.serializable.Serializable method*), 616
serialize_to_cmessage() (*angr.Block method*), 170
serialize_to_cmessage() (*angr.block.Block method*), 221
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 536
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 546
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.CFGModel method*), 530
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.CFGNode method*), 527
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.memory_data.MemoryData method*), 543
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.MemoryData method*), 525
serialize_to_cmessage() (*angr.knowledge_plugins.functions.function.Function method*), 555
serialize_to_cmessage() (*angr.knowledge_plugins.variables.variable_access.VariableAcce method*), 561
serialize_to_cmessage() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 562
serialize_to_cmessage() (*angr.knowledge_plugins.xrefs.xref.XRef method*), 610
serialize_to_cmessage() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 611
serialize_to_cmessage() (*angr.serializable.Serializable method*), 616
serialize_to_cmessage() (*angr.sim_variable.SimMemoryVariable method*), 505
serialize_to_cmessage() (*angr.sim_variable.SimRegisterVariable method*), 505
serialize_to_cmessage() (*angr.sim_variable.SimStackVariable method*), 506
serialize_to_cmessage() (*angr.sim_variable.SimTemporaryVariable method*), 504
Server (*class in angr.distributed.server*), 900
session_scope() (*angr.angrdb.db.AngrDB static method*), 674
set() (*angr.knowledge_plugins.key_definitions.environment.Environment method*), 591
set_abi_cc() (*angr.procedures.definitions.SimSyscallLibrary method*), 481
set_args() (*angr.sim_procedure.SimProcedure method*), 472
set_args() (*angr.SimProcedure method*), 160

set_base_state() (*angr.callable.Callable method*), 520
set_bck_chunk() (*angr.PTChunk method*), 210
set_bck_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 301
set_bck_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 304
set_brk() (*angr.state_plugins.posix.SimSystemPosix method*), 245
set_c_prototype() (*angr.procedures.definitions.SimLibrary method*), 477
set_data() (*angr.state_plugins.gdb.GDB method*), 271
set_default_cc() (*angr.procedures.definitions.SimLibrary method*), 476
set_entry_register_values() (*angr.simos.linux.SimLinux method*), 877
set_fd_data() (*angr.exploration_techniques.Tracer method*), 395
set_fd_data() (*angr.exploration_techniques.tracer.Tracer method*), 415
set_fwd_chunk() (*angr.PTChunk method*), 210
set_fwd_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 300
set_fwd_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 303
set_heap() (*angr.state_plugins.gdb.GDB method*), 271
set_initial_regs() (*angr.Block method*), 170
set_initial_regs() (*angr.block.Block method*), 221
set_last_block_details() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
set_last_statement() (*angr.annocfg.AnnotatedCFG method*), 872
set_library_names() (*angr.procedures.definitions.SimLibrary method*), 476
set_live_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 563
set_manager() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 562
set_mode() (*angr.sim_state.SimState method*), 228
set_mode() (*angr.SimState method*), 184
set_non_returning() (*angr.procedures.definitions.SimLibrary method*), 476
set_object() (*angr.keyed_region.KeyedRegion method*), 615
set_object() (*angr.keyed_region.RegionObject method*), 614
set_prev_freeness() (*angr.PTChunk method*), 209
set_prev_freeness() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 303
set_prototype() (*angr.knowledge_plugins.callsite_prototypes.CallsitePr method*), 523
set_prototype() (*angr.procedures.definitions.SimLibrary method*), 476
set_prototype() (*angr.procedures.definitions.SimSyscallLibrary method*), 481
set_prototypes() (*angr.procedures.definitions.SimLibrary method*), 476
set_prototypes() (*angr.procedures.definitions.SimSyscallLibrary method*), 481
set_regs() (*angr.state_plugins.gdb.GDB method*), 271
set_regs() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
set_return_val() (*angr.calling_conventions.SimCC method*), 489
set_return_val() (*angr.calling_conventions.SimCCSyscall method*), 493
set_return_val() (*angr.SimCC method*), 187
set_simgr() (*angr.analyses.congruency_check.CongruencyCheck method*), 859
set_size() (*angr.PTChunk method*), 209
set_size() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 300
set_size() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 302
set_stack() (*angr.state_plugins.gdb.GDB method*), 271
set_stack_address_mapping() (*angr.storage.memory_mixins.regioned_memory.regioned_memo method*), 367
set_stack_size() (*angr.storage.memory_mixins.regioned_memory.regio method*), 368
set_state() (*angr.SimFile method*), 190
set_state() (*angr.SimFileDescriptor method*), 199
set_state() (*angr.SimFileDescriptorDuplex method*), 201
set_state() (*angr.SimFileStream method*), 194
set_state() (*angr.SimPacket method*), 192
set_state() (*angr.SimPackets method*), 192
set_state() (*angr.SimStatePlugin method*), 161
set_state() (*angr.state_plugins.callstack.CallStack method*), 264
set_state() (*angr.state_plugins.concrete.Concrete method*), 293
set_state() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 252
set_state() (*angr.state_plugins.filesystem.SimFilesystem method*), 249
set_state() (*angr.state_plugins.globals.SimStateGlobals method*), 278
set_state() (*angr.state_plugins.inspect.SimInspector method*), 236
set_state() (*angr.state_plugins.light_registers.SimLightRegisters method*), 267
set_state() (*angr.state_plugins.plugin.SimStatePlugin

method), 232
set_state() (*angr.state_plugins.posix.SimSystemPosix method*), 245
set_state() (*angr.state_plugins.uc_manager.SimUCManager method*), 280
set_state() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
set_state() (*angr.state_plugins.view.SimMemView method*), 311
set_state() (*angr.storage.file.SimFile method*), 317
set_state() (*angr.storage.file.SimFileDescriptor method*), 328
set_state() (*angr.storage.file.SimFileDescriptorDuplex method*), 330
set_state() (*angr.storage.file.SimFileStream method*), 319
set_state() (*angr.storage.file.SimPackets method*), 321
set_state() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 344
set_state() (*angr.storage.memory_mixins.javavm_memory.JavaVmMemoryMixin method*), 377
set_state() (*angr.storage.memory_mixins.paged_memory.pages.ispo.ISPOMixin method*), 361
set_state() (*angr.storage.memory_mixins.regioned_memory.abstract_region_mapper.AbstractRegionMapper method*), 373
set_state() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 367
set_state() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method*), 374
set_state_options() (*angr.analyses.congruency_check.CongruencyCheck method*), 859
set_states() (*angr.analyses.congruency_check.CongruencyCheck method*), 859
set_stops() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
set_strongref_state() (*angr.SimStatePlugin method*), 161
set_strongref_state() (*angr.state_plugins.history.SimStateHistory method*), 267
set_strongref_state() (*angr.state_plugins.plugin.SimStatePlugin method*), 232
set_symbolization_for_all_pages() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 307
set_symbolized_target_range() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 307
set_tracking() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
set_tyenv() (*angr.state_plugins.scratch.SimStateScratch method*), 280
set_type() (*angr.analyses.decompiler.structured_codegen.c.CExpression method*), 726
set_unified_variable() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 566
set_value() (*angr.calling_conventions.SimArrayArg method*), 486
set_value() (*angr.calling_conventions.SimComboArg method*), 486
set_value() (*angr.calling_conventions.SimFunctionArgument method*), 484
set_value() (*angr.calling_conventions.SimLyingRegArg method*), 491
set_value() (*angr.calling_conventions.SimReferenceArgument method*), 487
set_value() (*angr.calling_conventions.SimRegArg method*), 485
set_value() (*angr.calling_conventions.SimStackArg method*), 485
set_value() (*angr.calling_conventions.SimStructArg method*), 486
set_variable() (*angr.storage.memory_mixins.paged_memory.pages.keyed_region.KeyedRegion method*), 615
set_variable_addr() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 563
set_variable_type() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 366
setstate() (*angr.calling_conventions.ArgSession method*), 487
setstate() (*angr.calling_conventions.SerializableCounter method*), 484
setstate() (*angr.calling_conventions.SerializableIterator method*), 484
setstate() (*angr.calling_conventions.SerializableListIterator method*), 484
setstate() (*angr.calling_conventions.SimCC.ArgSession method*), 489
setstate() (*angr.calling_conventions.UsercallArgSession method*), 487
setstate() (*angr.SimCC.ArgSession method*), 186
setup() (*angr.exploration_techniques.DFS method*), 398
setup() (*angr.exploration_techniques.dfs.DFS method*), 408
setup() (*angr.exploration_techniques.driller_core.DrillerCore method*), 416
setup() (*angr.exploration_techniques.DrillerCore method*), 393
setup() (*angr.exploration_techniques.ExplorationTechnique method*), 390
setup() (*angr.exploration_techniques.Explorer method*), 396
setup() (*angr.exploration_techniques.explorer.Explorer

method), 409
setup() (*angr.exploration_techniques.local_loop_seer.LocalLoopSeer method*), 422
setup() (*angr.exploration_techniques.LocalLoopSeer method*), 406
setup() (*angr.exploration_techniques.loop_seer.LoopSeer method*), 421
setup() (*angr.exploration_techniques.LoopSeer method*), 394
setup() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 410
setup() (*angr.exploration_techniques.ManualMergepoint method*), 402
setup() (*angr.exploration_techniques.memory_watcher.MemoryWatcher method*), 426
setup() (*angr.exploration_techniques.MemoryWatcher method*), 405
setup() (*angr.exploration_techniques.Slicecutor method*), 392
setup() (*angr.exploration_techniques.slicecutor.Slicecutor method*), 417
setup() (*angr.exploration_techniques.Symbion method*), 404
setup() (*angr.exploration_techniques.symbion.Symbion method*), 425
setup() (*angr.exploration_techniques.Timeout method*), 407
setup() (*angr.exploration_techniques.timeout.Timeout method*), 407
setup() (*angr.exploration_techniques.Tracer method*), 395
setup() (*angr.exploration_techniques.tracer.Tracer method*), 415
setup() (*angr.exploration_techniques.unique.UniqueSearch method*), 423
setup() (*angr.exploration_techniques.UniqueSearch method*), 403
setup() (*angr.ExplorationTechnique method*), 178
setup() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
setup_arguments() (*angr.engines.soot.engine.SootMixin static method*), 432
setup_callsite() (*angr.calling_conventions.SimCC method*), 489
setup_callsite() (*angr.calling_conventions.SimCCSoot method*), 500
setup_callsite() (*angr.engines.soot.engine.SootMixin class method*), 432
setup_callsite() (*angr.SimCC method*), 187
setup_flags() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
setup_gdt() (*angr.SimOS method*), 169
setup_gdt() (*angr.simos.simos.SimOS method*), 876
setup_gdt() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
sh_terminal() (*in module angr.utils.formatting*), 892
shallow_reverse() (*in module angr.utils.graph*), 886
ShiftLeft (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressT attribute*), 662
ShiftRight (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.Address attribute*), 662
short_reason (*angr.knowledge_plugins.cfg.cfg_node.CFGNodeCreationF attribute*), 544
short_repr (*angr.analyses.ddg.ProgramVariable property*), 744
short_repr (*angr.code_location.CodeLocation property*), 613
should_abort (*angr.analyses.forward_analysis.forward_analysis.Forwara property*), 621
should_add_successors (*angr.sim_procedure.SimProcedure property*), 472
should_add_successors (*angr.SimProcedure property*), 160
should_execute_statement() (*angr.annocfg.AnnotatedCFG method*), 872
should_force_replace() (*angr.analyses.propagator.engine_ail.SimEnginePropagatorAIL method*), 754
should_take_exit() (*angr.annocfg.AnnotatedCFG method*), 872
show_demangled_name (*angr.analyses.decompiler.structured_codegen.c.CFunction attribute*), 726
show_demangled_name (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 730
shrink() (*angr.analyses.reassembler.Data method*), 855
SideEffectTag (*class in angr.knowledge_plugins.key_definitions.tag*), 604
SIG_BLOCK (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
SIG_SETMASK (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
SIG_UNBLOCK (*angr.state_plugins.posix.SimSystemPosix attribute*), 244
sigmask() (*angr.state_plugins.posix.SimSystemPosix method*), 246
signed (*angr.procedures.stubs.format_parser.FormatSpecifier attribute*), 474
signed (*angr.sim_type.SimTypeFloat attribute*), 515
SignedExtension (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.A attribute*), 662
sigprocmask() (*angr.state_plugins.posix.SimSystemPosix method*), 246
silence_logger() (*in module*

angr.state_plugins.heap.heap_ptmalloc), 302
sim_procedure (*angr.code_location.CodeLocation attribute*), 612
sim_procedure (*angr.codenode.HookNode attribute*), 874
sim_procedure (*angr.codenode.SyscallNode attribute*), 874
SimAbstractMemoryError, 896
SimAction (*class in angr.state_plugins.sim_action*), 466
SimActionConstraint (*class in angr.state_plugins.sim_action*), 467
SimActionData (*class in angr.state_plugins.sim_action*), 467
SimActionError, 899
SimActionExit (*class in angr.state_plugins.sim_action*), 466
SimActionObject (*class in angr.state_plugins.sim_action_object*), 468
SimActionOperation (*class in angr.state_plugins.sim_action*), 467
SimActLocation (*class in angr.analyses.data_dep.sim_act_location*), 866
SimArrayArg (*class in angr.calling_conventions*), 486
SimCC (*class in angr*), 184
SimCC (*class in angr.calling_conventions*), 487
SimCC.ArgSession (*class in angr*), 185
SimCC.ArgSession (*class in angr.calling_conventions*), 488
SimCCAArch64 (*class in angr.calling_conventions*), 496
SimCCAArch64LinuxSyscall (*class in angr.calling_conventions*), 497
SimCCallError, 898
SimCCAMD64LinuxSyscall (*class in angr.calling_conventions*), 495
SimCCAMD64WindowsSyscall (*class in angr.calling_conventions*), 495
SimCCARM (*class in angr.calling_conventions*), 495
SimCCARMHF (*class in angr.calling_conventions*), 496
SimCCARMLinuxSyscall (*class in angr.calling_conventions*), 496
SimCCCdecl (*class in angr.calling_conventions*), 491
SimCCError, 899
SimCCM68k (*class in angr.engines.pcode.cc*), 464
SimCCMicrosoftAMD64 (*class in angr.calling_conventions*), 492
SimCCMicrosoftCdecl (*class in angr.calling_conventions*), 492
SimCCMicrosoftFastcall (*class in angr.calling_conventions*), 492
SimCCN64 (*class in angr.calling_conventions*), 498
SimCCN64LinuxSyscall (*class in angr.calling_conventions*), 498
SimCCO32 (*class in angr.calling_conventions*), 497
SimCCO32LinuxSyscall (*class in angr.calling_conventions*), 498
SimCCO64 (*in module angr.calling_conventions*), 498
SimCCPARISC (*class in angr.engines.pcode.cc*), 465
SimCCPowerPC (*class in angr.calling_conventions*), 499
SimCCPowerPC (*class in angr.engines.pcode.cc*), 465
SimCCPowerPC64 (*class in angr.calling_conventions*), 499
SimCCPowerPC64LinuxSyscall (*class in angr.calling_conventions*), 500
SimCCPowerPCLinuxSyscall (*class in angr.calling_conventions*), 499
SimCCRISCV (*class in angr.engines.pcode.cc*), 464
SimCCRISCV64LinuxSyscall (*class in angr.calling_conventions*), 497
SimCCS390X (*class in angr.calling_conventions*), 501
SimCCS390XLinuxSyscall (*class in angr.calling_conventions*), 501
SimCCSH4 (*class in angr.engines.pcode.cc*), 465
SimCCSoot (*class in angr.calling_conventions*), 500
SimCCSPARC (*class in angr.engines.pcode.cc*), 465
SimCCStdcall (*class in angr.calling_conventions*), 492
SimCCSyscall (*class in angr.calling_conventions*), 493
SimCCSystemVAMD64 (*class in angr.calling_conventions*), 494
SimCCUnknown (*class in angr.calling_conventions*), 501
SimCCUsercall (*class in angr.calling_conventions*), 491
SimCCX86LinuxSyscall (*class in angr.calling_conventions*), 493
SimCCX86WindowsSyscall (*class in angr.calling_conventions*), 494
SimCCXtensa (*class in angr.engines.pcode.cc*), 466
SimCGC (*class in angr.simos.cgc*), 878
SimComboArg (*class in angr.calling_conventions*), 485
SimConcreteBreakpointError, 899
SimConcreteFilesystem (*class in angr.state_plugins.filesystem*), 251
SimConcreteMemoryError, 899
SimConcreteRegisterError, 899
SimConcretizationStrategy (*class in angr.concretization_strategies*), 335
SimConcretizationStrategyAny (*class in angr.concretization_strategies.any*), 381
SimConcretizationStrategyControlledData (*class in angr.concretization_strategies.controlled_data*), 381
SimConcretizationStrategyEval (*class in angr.concretization_strategies.eval*), 379
SimConcretizationStrategyMax (*class in angr.concretization_strategies.max*), 380
SimConcretizationStrategyNonzero (*class in angr.concretization_strategies.nonzero*), 381
SimConcretizationStrategyNonzeroRange (*class in*

angr.concretization_strategies.nonzero_range), 380
SimConcretizationStrategyNorepeats (*class in angr.concretization_strategies.norepeats*), 379
SimConcretizationStrategyNorepeatsRange (*class in angr.concretization_strategies.norepeats_range*), 381
SimConcretizationStrategyRange (*class in angr.concretization_strategies.range*), 380
SimConcretizationStrategySingle (*class in angr.concretization_strategies.single*), 379
SimConcretizationStrategySolutions (*class in angr.concretization_strategies.solutions*), 379
SimConcretizationStrategyUnlimitedRange (*class in angr.concretization_strategies.unlimited_range*), 381
SimConstantVariable (*class in angr.sim_variable*), 503
SimCppClass (*class in angr.sim_type*), 517
SimCppClassValue (*class in angr.sim_type*), 517
SimCppLibrary (*class in angr.procedures.definitions*), 478
SimDebugVariable (*class in angr.state_plugins.debug_variables*), 307
SimDebugVariablePlugin (*class in angr.state_plugins.debug_variables*), 308
SimEmptyCallStackError, 899
SimEngine (*class in angr.engines.engine*), 427
SimEngineBase (*class in angr.engines.engine*), 427
SimEngineConcrete (*class in angr.engines.concrete*), 433
SimEngineError, 898
SimEngineFailure (*class in angr.engines.failure*), 431
SimEngineInitFinderVEX (*class in angr.analyses.init_finder*), 861
SimEngineLight (*class in angr.engines.light.engine*), 751
SimEngineLightAIL (*in module angr.engines.light.engine*), 751
SimEngineLightAILMixin (*class in angr.engines.light.engine*), 751
SimEngineLightMixin (*class in angr.engines.light.engine*), 750
SimEngineLightVEX (*in module angr.engines.light.engine*), 751
SimEngineLightVEXMixin (*class in angr.engines.light.engine*), 751
SimEnginePropagatorAIL (*class in angr.analyses.propagator.engine_ail*), 753
SimEnginePropagatorBase (*class in angr.analyses.propagator.engine_base*), 752
SimEnginePropagatorVEX (*class in angr.analyses.propagator.engine_vex*), 753
SimEngineRDAIL (*class in angr.analyses.reaching_definitions.engine_ail*), 813
SimEngineRDVEX (*class in angr.analyses.reaching_definitions.engine_vex*), 790
SimEngineSyscall (*class in angr.engines.syscall*), 431
SimEngineUnicorn (*class in angr.engines.unicorn*), 432
SimEngineVRAIL (*class in angr.analyses.variable_recovery.engine_ail*), 826
SimEngineVRBase (*class in angr.analyses.variable_recovery.engine_base*), 827
SimEngineVRVEX (*class in angr.analyses.variable_recovery.engine_vex*), 826
SimEngineXRefsVEX (*class in angr.analyses.xrefs*), 861
SimError, 896
SimEvent (*class in angr.state_plugins.sim_event*), 468
SimEventError, 897
SimException, 899
SimExpressionError, 897
SimFastMemoryError, 897
SimFastPathError, 898
SimFile (*class in angr*), 189
SimFile (*class in angr.storage.file*), 317
SimFileBase (*class in angr*), 188
SimFileBase (*class in angr.storage.file*), 315
SimFileDescriptor (*class in angr*), 198
SimFileDescriptor (*class in angr.storage.file*), 327
SimFileDescriptorBase (*class in angr.storage.file*), 325
SimFileDescriptorDuplex (*class in angr*), 200
SimFileDescriptorDuplex (*class in angr.storage.file*), 329
SimFileError, 897
SimFileStream (*class in angr*), 194
SimFileStream (*class in angr.storage.file*), 319
SimFilesystem (*class in angr.state_plugins.filesystem*), 249
SimFilesystemError, 897
SimFunctionArgument (*class in angr.calling_conventions*), 484
simgr() (*angr.factory.AngrObjectFactory method*), 219
SimHeapBase (*class in angr.state_plugins.heap.heap_base*), 297
SimHeapBrk (*class in angr*), 204
SimHeapBrk (*class in angr.state_plugins.heap.heap_brk*), 298
SimHeapError, 897
SimHeapFreelist (*class in angr.state_plugins.heap.heap_freelist*), 301
SimHeapLibc (*class in angr.state_plugins.heap.heap_libc*), 301

SimHeapPTMalloc (*class in angr*), 206
SimHeapPTMalloc (*class in angr.state_plugins.heap.heap_ptmalloc*), 304
SimHostFilesystem (*class in angr*), 203
SimHostFilesystem (*class in angr.state_plugins.filesystem*), 253
similarity() (*angr.exploration_techniques.unique.UniqueSearch static method*), 424
similarity() (*angr.exploration_techniques.UniqueSearch static method*), 404
SimInspector (*class in angr.state_plugins.inspect*), 233
SimIRSBError, 898
SimIRSBNoDecodeError, 898
SimJavaVM (*class in angr.simos.javavm*), 880
SimJavaVmClassloader (*class in angr.state_plugins.javavm_classloader*), 294
SimLabeledMemoryObject (*class in angr.storage.memory_object*), 334
SimLibrary (*class in angr.procedures.definitions*), 475
SimLightRegisters (*class in angr.state_plugins.light_registers*), 266
SimLightState (*class in angr.slicer*), 871
SimLinux (*class in angr.simos.linux*), 877
SimLyingRegArg (*class in angr.calling_conventions*), 491
SimMemoryAddressError, 897
SimMemoryError, 896
SimMemoryLimitError, 897
SimMemoryMissingError, 896
SimMemoryObject (*class in angr.storage.memory_object*), 334
SimMemoryVariable (*class in angr.sim_variable*), 505
SimMemView (*class in angr.state_plugins.view*), 310
SimMergeError, 896
SimMissingTempError, 898
SimMount (*class in angr*), 203
SimMount (*class in angr.state_plugins.filesystem*), 251
SimOperationError, 897
SimOS (*class in angr*), 168
SimOS (*class in angr.simos.simos*), 875
SIMOS_CGC (*angr.state_plugins.unicorn_engine.SimOSEnum attribute*), 287
SIMOS_LINUX (*angr.state_plugins.unicorn_engine.SimOSEnum attribute*), 287
SIMOS_OTHER (*angr.state_plugins.unicorn_engine.SimOSEnum attribute*), 288
SimOSEnum (*class in angr.state_plugins.unicorn_engine*), 287
SimPackets (*class in angr*), 192
SimPackets (*class in angr.storage.file*), 321
SimPacketsSlots (*class in angr.storage.file*), 332
SimPacketsStream (*class in angr*), 196
SimPacketsStream (*class in angr.storage.file*), 323
SimpleInterfaceMixin (*class in angr.storage.memory_mixins.simple_interface_mixin*), 342
SimpleSolver (*class in angr.analyses.typehoon.simple_solver*), 828
SimplificationMixin (*class in angr.storage.memory_mixins.simplification_mixin*), 347
simplified_data_graph (*angr.analyses.ddg.DDG property*), 747
simplified_graph (*angr.analyses.data_dep.data_dependency_analysis.D property*), 866
SimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.engine_base*), 709
SimplifierAILState (*class in angr.analyses.decompiler.optimization_passes.engine_base*), 709
simplify() (*angr.sim_state.SimState method*), 226
simplify() (*angr.SimState method*), 182
simplify() (*angr.state_hierarchy.StateHierarchy method*), 389
simplify() (*angr.state_plugins.solver.SimSolver method*), 262
simplify() (*angr.StateHierarchy method*), 180
simplify_condition() (*angr.analyses.decompiler.condition_processor.ConditionProcess static method*), 696
simplify_condition_deprecated() (*angr.analyses.decompiler.condition_processor.ConditionProcess static method*), 696
simplify_else_scope (*angr.analyses.decompiler.structured_codegen.c.CIfElse attribute*), 728
simplify_lowered_switches() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl*), 721
simplify_lowered_switches_core() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl*), 721
simplify_switch_clusters() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl*), 721
SimPosixError, 897
SimProcedure (*class in angr*), 157
SimProcedure (*class in angr.sim_procedure*), 469
simprocedure_name (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
simprocedure_name (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
SimProcedureArgumentError, 898
SimProcedureError, 898
SimReferenceArgument (*class in*

angr.calling_conventions), 486
SimRegArg (*class in angr.calling_conventions*), 484
SimRegionMapError, 896
SimRegisterVariable (*class in angr.sim_variable*), 504
SimRegNameView (*class in angr.state_plugins.view*), 309
SimReliftException, 898
SimSegfaultError (*in module angr.errors*), 899
SimSegfaultException, 899
SimShadowStackError, 898
SimSlicer (*class in angr.slicer*), 871
SimSlicerError, 899
SimSolver (*class in angr.state_plugins.solver*), 254
SimSolverError, 897
SimSolverModeError, 897
SimSolverOptionError, 897
SimStackArg (*class in angr.calling_conventions*), 485
SimStackVariable (*class in angr.sim_variable*), 506
SimState (*class in angr*), 180
SimState (*class in angr.sim_state*), 224
SimStateCGC (*class in angr.state_plugins.cgc*), 271
SimStateError, 896
SimStateGlobals (*class in angr.state_plugins.globals*), 278
SimStateHistory (*class in angr.state_plugins.history*), 267
SimStateJNIReferences (*class in angr.state_plugins.jni_references*), 296
SimStateLibc (*class in angr.state_plugins.libc*), 236
SimStateLog (*class in angr.state_plugins.log*), 262
SimStateLoopData (*class in angr.state_plugins.loop_data*), 291
SimStatementError, 898
SimStateOptions (*class in angr.sim_state_options*), 228
SimStateOptionsError, 899
SimStatePlugin (*class in angr*), 161
SimStatePlugin (*class in angr.state_plugins.plugin*), 231
SimStatePreconstrainer (*class in angr.state_plugins.preconstrainer*), 282
SimStateScratch (*class in angr.state_plugins.scratch*), 280
SimStruct (*class in angr.sim_type*), 516
SimStructArg (*class in angr.calling_conventions*), 486
SimStructValue (*class in angr.sim_type*), 516
SimSuccessors (*class in angr.engines.successors*), 428
SimSymbolicFilesystemError, 897
SimSymbolizer (*class in angr.state_plugins.symbolizer*), 307
SimSyscallLibrary (*class in angr.procedures.definitions*), 480
SimSystemPosix (*class in angr.state_plugins.posix*), 244
SimTemporaryVariable (*class in angr.sim_variable*), 504
SimTranslationError, 898
SimType (*class in angr.sim_type*), 507
simtype2tc() (*angr.analyses.typehoon.translator.TypeTranslator method*), 828
SimTypeArray (*class in angr.sim_type*), 512
SimTypeBool (*class in angr.sim_type*), 511
SimTypeBottom (*class in angr.sim_type*), 509
SimTypeChar (*class in angr.sim_type*), 510
SimTypeCppFunction (*class in angr.sim_type*), 514
SimTypeDouble (*class in angr.sim_type*), 515
SimTypeFd (*class in angr.sim_type*), 511
SimTypeFixedSizeArray (*in module angr.sim_type*), 512
SimTypeFloat (*class in angr.sim_type*), 515
SimTypeFunction (*class in angr.sim_type*), 513
SimTypeInt (*class in angr.sim_type*), 510
SimTypeLength (*class in angr.sim_type*), 514
SimTypeLong (*class in angr.sim_type*), 510
SimTypeLongLong (*class in angr.sim_type*), 510
SimTypeNum (*class in angr.sim_type*), 509
SimTypeNumOffset (*class in angr.sim_type*), 518
SimTypePointer (*class in angr.sim_type*), 511
SimTypeReference (*class in angr.sim_type*), 511
SimTypeReg (*class in angr.sim_type*), 509
SimTypeShort (*class in angr.sim_type*), 510
SimTypeString (*class in angr.sim_type*), 512
SimTypeTempRef (*class in angr.analyses.typehoon.translator*), 828
SimTypeTop (*class in angr.sim_type*), 509
SimTypeWideChar (*class in angr.sim_type*), 511
SimTypeWString (*class in angr.sim_type*), 513
SimUCManager (*class in angr.state_plugins.uc_manager*), 279
SimUCManagerAllocationError, 899
SimUCManagerError, 899
simulation_manager() (*angr.factory.AngrObjectFactory method*), 219
SimulationManager (*class in angr*), 171
SimulationManager (*class in angr.sim_manager*), 382
SimulationManagerError, 894
SimUnicornError, 899
SimUnicornSymbolic, 899
SimUnicornUnsupport, 899
SimUninitializedAccessError, 898
SimUnion (*class in angr.sim_type*), 516
SimUnionValue (*class in angr.sim_type*), 517
SimUnsatError, 897
SimUnsupportedError, 897
SimUserland (*class in angr.simos.userland*), 878
SimValueError, 897
SimVariable (*class in angr.sim_variable*), 502

SimVariableSet (*class in angr.sim_variable*), 507
SimWindows (*class in angr.simos.windows*), 879
SimZeroDivisionException, 899
single_valued() (*angr.state_plugins.solver.SimSolver method*), 262
SingleNodeGraphVisitor (*class in angr.analyses.forward_analysis.visitors.single_node_graph*), 627
size (*angr.analyses.cfg.segment_list.Segment property*), 670
size (*angr.analyses.propagator.values.Top attribute*), 752
size (*angr.analyses.propagator.vex_vars.VEXMemVar attribute*), 752
size (*angr.analyses.propagator.vex_vars.VEXReg attribute*), 752
size (*angr.analyses.reaching_definitions.Atom attribute*), 766
size (*angr.analyses.reaching_definitions.Definition property*), 770
SIZE (*angr.analyses.typehoon.typeconsts.Double attribute*), 835
SIZE (*angr.analyses.typehoon.typeconsts.Float attribute*), 834
SIZE (*angr.analyses.typehoon.typeconsts.Int1 attribute*), 834
SIZE (*angr.analyses.typehoon.typeconsts.Int128 attribute*), 834
SIZE (*angr.analyses.typehoon.typeconsts.Int16 attribute*), 834
SIZE (*angr.analyses.typehoon.typeconsts.Int32 attribute*), 834
SIZE (*angr.analyses.typehoon.typeconsts.Int64 attribute*), 834
SIZE (*angr.analyses.typehoon.typeconsts.Int8 attribute*), 834
SIZE (*angr.analyses.typehoon.typeconsts.TypeConstant attribute*), 834
size (*angr.analyses.typehoon.typeconsts.TypeConstant property*), 834
size (*angr.Block attribute*), 170
size (*angr.block.Block attribute*), 221
size (*angr.block.CapstoneInsn property*), 221
size (*angr.block.DisassemblerInsn property*), 220
size (*angr.block.SootBlock property*), 222
size (*angr.codenode.CodeNode attribute*), 873
size (*angr.engines.pcode.lifter.IRSB property*), 438
size (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 435
size (*angr.keyed_region.RegionObject attribute*), 613
size (*angr.keyed_region.StoredObject attribute*), 613
size (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
size (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
size (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 543
size (*angr.knowledge_plugins.cfg.MemoryData attribute*), 525
size (*angr.knowledge_plugins.functions.function.Function property*), 555
size (*angr.knowledge_plugins.key_definitions.atoms.Atom attribute*), 584
size (*angr.knowledge_plugins.key_definitions.Definition property*), 583
size (*angr.knowledge_plugins.key_definitions.definition.Definition property*), 590
size (*angr.procedures.stubs.format_parser.FormatSpecifier attribute*), 474
size (*angr.sim_type.SimStruct property*), 516
size (*angr.sim_type.SimType property*), 507
size (*angr.sim_type.SimTypeArray property*), 512
size (*angr.sim_type.SimTypeFunction property*), 514
size (*angr.sim_type.SimTypeInt property*), 510
size (*angr.sim_type.SimTypeLength property*), 515
size (*angr.sim_type.SimTypePointer property*), 511
size (*angr.sim_type.SimTypeReference property*), 512
size (*angr.sim_type.SimTypeString property*), 513
size (*angr.sim_type.SimTypeWString property*), 513
size (*angr.sim_type.SimUnion property*), 516
size (*angr.sim_type.TypeRef property*), 508
size (*angr.sim_variable.SimVariable attribute*), 503
size (*angr.SimFile property*), 190
size (*angr.SimFileBase property*), 189
size (*angr.SimPackets property*), 192
size (*angr.state_plugins.unicorn_engine.RegisterValue attribute*), 285
size (*angr.storage.file.SimFile property*), 317
size (*angr.storage.file.SimFileBase property*), 316
size (*angr.storage.file.SimPackets property*), 321
size (*angr.storage.file.SimPacketsSlots property*), 332
size() (*angr.calling_conventions.AllocHelper method*), 483
size() (*angr.SimFileDescriptor method*), 198
size() (*angr.SimFileDescriptorDuplex method*), 201
size() (*angr.storage.file.SimFileDescriptor method*), 327
size() (*angr.storage.file.SimFileDescriptorBase method*), 326
size() (*angr.storage.file.SimFileDescriptorDuplex method*), 330
size() (*angr.storage.memory_object.SimMemoryObject method*), 334
SizeConcretizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*), 343
SizeNormalizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*),

343
skip_stmts (*angr.engines.pcode.lifter.Lifter attribute*), 440
skip_stmts (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
slice (*angr.Blade property*), 168
slice (*angr.blade.Blade property*), 871
slice_callgraph() (*in module angr.analyses.cfg_slice_to_sink.graph*), 815
slice_cfg_graph() (*in module angr.analyses.cfg_slice_to_sink.graph*), 815
slice_function_graph() (*in module angr.analyses.cfg_slice_to_sink.graph*), 816
slice_graph() (*angr.analyses.decompiler.region_identifier.RegionIdentifier static method*), 714
Slicecutor (*class in angr.exploration_techniques*), 391
Slicecutor (*class in angr.exploration_techniques.slicecutor*), 417
SlottedMemoryMixin (*class in angr.storage.memory_mixins.slotted_memory*), 374
SmartFindMixin (*class in angr.storage.memory_mixins.smart_find_mixin*), 339
SMod() (*angr.state_plugins.sim_action_object.SimActionObject method*), 468
snippet() (*angr.factory.AngrObjectFactory method*), 216
solution() (*angr.state_plugins.solver.SimSolver method*), 258
solve() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 828
solver (*angr.sim_state.SimState attribute*), 225
solver (*angr.SimState attribute*), 181
soot (*angr.block.SootBlock property*), 222
soot_block (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
soot_block (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
SootBlock (*class in angr.block*), 222
SootBlockNode (*class in angr.codenode*), 874
SootBlockProcessor (*class in angr.analyses.loop_analysis*), 837
SootClassHierarchy (*class in angr.analyses.soot_class_hierarchy*), 636
SootClassHierarchyError, 636
SootExpression (*class in angr.analyses.disassembly*), 848
SootExpressionInvoke (*class in angr.analyses.disassembly*), 848
SootExpressionStaticFieldRef (*class in angr.analyses.disassembly*), 848
SootExpressionTarget (*class in angr.analyses.disassembly*), 848
SootFunction (*class in angr.knowledge_plugins.functions.soot_function*), 560
SootMixin (*class in angr.engines.soot.engine*), 431
SootStatement (*class in angr.analyses.disassembly*), 848
sort (*angr.analyses.cfg.segment_list.Segment attribute*), 670
sort (*angr.analyses.decompiler.structuring.structurer_nodes.LoopNode attribute*), 687
sort (*angr.knowledge_plugins.cfg.memory_data.MemoryData attribute*), 543
sort (*angr.knowledge_plugins.cfg.MemoryData attribute*), 525
sort (*angr.sim_type.SimTypeDouble attribute*), 515
sort (*angr.sim_type.SimTypeFloat attribute*), 515
sort() (*angr.exploration_techniques.spiller.PickledStatesBase method*), 410
sort() (*angr.exploration_techniques.spiller.PickledStatesDb method*), 411
sort() (*angr.exploration_techniques.spiller.PickledStatesList method*), 411
sort_nodes() (*angr.analyses.forward_analysis.visitors.call_graph.CallGr method*), 623
sort_nodes() (*angr.analyses.forward_analysis.visitors.function_graph.Fu method*), 624
sort_nodes() (*angr.analyses.forward_analysis.visitors.graph.GraphVisito method*), 624
sort_nodes() (*angr.analyses.forward_analysis.visitors.loop.LoopVisitor method*), 626
sort_nodes() (*angr.analyses.forward_analysis.visitors.single_node_graph method*), 627
sources (*angr.analyses.reaching_definitions.function_handler.FunctionEff attribute*), 798
sources_defns (*angr.analyses.reaching_definitions.function_handler.Func attribute*), 798
sp_delta (*angr.knowledge_plugins.functions.function.Function attribute*), 553
sp_delta (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
sp_offset() (*angr.engines.light.engine.SimEngineLightMixin static method*), 751
spec_type (*angr.procedures.stubs.format_parser.FormatSpecifier property*), 474
Special (*angr.analyses.disassembly.SootExpressionInvoke attribute*), 848
SPECIAL_THUNKS (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 653
SpecialFillerMixin (*class in angr.storage.memory_mixins.default_filler_mixin*), 340
Spiller (*class in angr.exploration_techniques*), 401
Spiller (*class in angr.exploration_techniques.spiller*), 411

split() (*angr.keyed_region.RegionObject method*), 614
split() (*angr.sim_manager.SimulationManager method*), 388
split() (*angr.SimulationManager method*), 177
split_arm_op_string() (*angr.analyses.disassembly.Instruction static method*), 848
split_op_string() (*angr.analyses.disassembly.Instruction static method*), 848
split_operands() (*in module angr.analyses.reassembler*), 851
SpOffset (*class in angr.engines.light.data*), 750
squash_array_reference() (*in module angr.analyses.decompiler.structured_codegen.c*), 724
src_block_id (*angr.analyses.vfg.PendingJob attribute*), 840
src_func_addr (*angr.analyses.cfg.cfg_fast.FunctionEdge attribute*), 650
src_ins_addr (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
src_ins_addr (*angr.analyses.vfg.PendingJob attribute*), 840
src_node (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
src_node (*angr.analyses.cfg.cfg_fast.FunctionCallEdge attribute*), 650
src_node (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge attribute*), 650
src_node (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute*), 650
src_stmt_idx (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
src_stmt_idx (*angr.analyses.vfg.PendingJob attribute*), 840
src_type (*angr.analyses.decompiler.structured_codegen.c.CTypeCast attribute*), 735
sse_extend() (*angr.calling_conventions.SimRegArg method*), 485
st_atime (*angr.state_plugins.filesystem.Stat attribute*), 248
st_atimensec (*angr.state_plugins.filesystem.Stat attribute*), 248
st_blksize (*angr.state_plugins.filesystem.Stat attribute*), 248
st_blocks (*angr.state_plugins.filesystem.Stat attribute*), 248
st_ctime (*angr.state_plugins.filesystem.Stat attribute*), 248
st_ctimensec (*angr.state_plugins.filesystem.Stat attribute*), 248
st_dev (*angr.state_plugins.filesystem.Stat attribute*), 248
st_gid (*angr.state_plugins.filesystem.Stat attribute*), 248
st_ino (*angr.state_plugins.filesystem.Stat attribute*), 248
st_mode (*angr.state_plugins.filesystem.Stat attribute*), 248
st_mtime (*angr.state_plugins.filesystem.Stat attribute*), 248
st_mtimensec (*angr.state_plugins.filesystem.Stat attribute*), 248
st_nlink (*angr.state_plugins.filesystem.Stat attribute*), 248
st_rdev (*angr.state_plugins.filesystem.Stat attribute*), 248
st_size (*angr.state_plugins.filesystem.Stat attribute*), 248
st_uid (*angr.state_plugins.filesystem.Stat attribute*), 248
StableVarExprHasher (*class in angr.analyses.decompiler.optimization_passes.lowered_switch_si*
707
stack (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
stack (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsSta property*), 809
stack (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
stack (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefiniti attribute*), 594
stack (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
stack (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mix property*), 376
stack_actions (*angr.state_plugins.history.SimStateHistory property*), 270
stack_addr_from_offset() (*angr.analyses.variable_recovery.variable_recovery_base.Variab method*), 822
stack_address() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 759
stack_address() (*angr.analyses.reaching_definitions.rd_state.ReachingD method*), 808
stack_address() (*angr.analyses.reaching_definitions.ReachingDefinition method*), 778
stack_address() (*angr.analyses.variable_recovery.variable_recovery_ba method*), 821
stack_address() (*angr.knowledge_plugins.key_definitions.live_definition method*), 595
stack_address() (*angr.knowledge_plugins.key_definitions.LiveDefinition method*), 574
STACK_ALIGNMENT (*angr.calling_conventions.SimCC attribute*), 488
STACK_ALIGNMENT (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
STACK_ALIGNMENT (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
STACK_ALIGNMENT (*angr.SimCC attribute*), 185
stack_base (*angr.storage.memory_mixins.regioned_memory.region_data. property*), 369

stack_definitions (*angr.analyses.reaching_definitions.LiveDefinitions property*), 758
stack_definitions (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions property*), 594
stack_definitions (*angr.knowledge_plugins.key_definitions.LiveDefinitions property*), 574
stack_id() (*angr.storage.memory_mixins.regioned_memory.stack_region_mapper.StackRegionMapper method*), 368
stack_loc() (*angr.calling_conventions.AllocHelper class method*), 483
stack_offset (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 589
stack_offset_to_stack_addr() (*angr.analyses.reaching_definitions.LiveDefinitions method*), 760
stack_offset_to_stack_addr() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 596
stack_offset_to_stack_addr() (*angr.knowledge_plugins.key_definitions.LiveDefinitions method*), 575
stack_offsets (*angr.slicer.SimLightState attribute*), 871
stack_pointer_as_atom() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler static method*), 805
stack_pointer_as_atom() (*angr.analyses.reaching_definitions.FunctionHandler static method*), 785
stack_pop() (*angr.sim_state.SimState method*), 227
stack_pop() (*angr.SimState method*), 184
stack_push() (*angr.sim_state.SimState method*), 227
stack_push() (*angr.SimState method*), 183
stack_read() (*angr.sim_state.SimState method*), 227
stack_read() (*angr.SimState method*), 184
stack_region (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal attribute*), 562
stack_space() (*angr.calling_conventions.SimCC method*), 489
stack_space() (*angr.SimCC method*), 186
stack_suffix() (*angr.state_plugins.callstack.CallStack method*), 266
stack_suffix_to_string() (*angr.state_plugins.callstack.CallStack static method*), 265
stack_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
stack_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 809
stack_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
stack_uses (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594
stack_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
StackAllocationMixin (*class in angr.storage.memory_mixins.paged_memory.stack_allocation_mixin*), 358
STACKARG_SP_BUFF (*angr.calling_conventions.SimCC attribute*), 488
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCN64 attribute*), 498
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCO32 attribute*), 497
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCPowerPC attribute*), 499
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCPowerPC64 attribute*), 500
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCS390X attribute*), 501
STACKARG_SP_BUFF (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 465
STACKARG_SP_BUFF (*angr.SimCC attribute*), 185
STACKARG_SP_DIFF (*angr.calling_conventions.SimCC attribute*), 488
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCCdecl attribute*), 491
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 493
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 492
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 494
STACKARG_SP_DIFF (*angr.engines.pcode.cc.SimCCM68k attribute*), 464
STACKARG_SP_DIFF (*angr.SimCC attribute*), 185
StackCanarySimplifier (*class in angr.analyses.decompiler.optimization_passes.stack_canary_simplifier*), 704
StackLocationAnnotation (*class in angr.analyses.variable_recovery.annotations*), 818
StackPointerTracker (*class in angr.analyses.stack_pointer_tracker*), 818
StackPointerTrackerState (*class in angr.analyses.stack_pointer_tracker*), 817
STAGE (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePointerSaveSimplifier attribute*), 705
STAGE (*angr.analyses.decompiler.optimization_passes.const_derefs.ConstantDereferencesSimplification attribute*), 701
STAGE (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifier attribute*), 705
STAGE (*angr.analyses.decompiler.optimization_passes.eager_returns.EagerReturnsSimplifier attribute*), 702
STAGE (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.ExprOpSwapper attribute*), 710

STAGE (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 706
STAGE (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier attribute*), 707
STAGE (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 708
STAGE (*angr.analyses.decompiler.optimization_passes.multi_simplifier.MultiSimplifier attribute*), 708
STAGE (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute*), 703
STAGE (*angr.analyses.decompiler.optimization_passes.optimization_pass.StructuringOptimizationPass attribute*), 704
STAGE (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifier attribute*), 711
STAGE (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 711
STAGE (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 704
STAGE (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier attribute*), 711
start (*angr.analyses.cfg.segment_list.Segment attribute*), 670
start (*angr.analyses.decompiler.structured_codegen.base.StatementBlockSingleBlockStream attribute*), 723
start (*angr.keyed_region.RegionObject attribute*), 613
start (*angr.keyed_region.StoredObject attribute*), 613
start() (*angr.distributed.worker.Worker method*), 901
start() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
startpoint (*angr.knowledge_plugins.functions.function.Function attribute*), 552
startpoint (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
stash (*angr.exploration_techniques.spiller_db.PickledState attribute*), 412
stash() (*angr.sim_manager.SimulationManager method*), 387
stash() (*angr.SimulationManager method*), 176
stashed (*angr.sim_manager.SimulationManager attribute*), 383
stashed (*angr.SimulationManager attribute*), 172
stashes (*angr.sim_manager.SimulationManager property*), 383
stashes (*angr.SimulationManager property*), 172
Stat (*class in angr.state_plugins.filesystem*), 248
state (*angr.analyses.propagator.engine_ail.SimEnginePropagatorAIL attribute*), 753
state (*angr.analyses.propagator.engine_vex.SimEnginePropagatorVEX attribute*), 753
state (*angr.analyses.reaching_definitions.engine_ail.SimEngineRDAIL attribute*), 814
state (*angr.analyses.variable_recovery.engine_ail.SimEngineVRAIL attribute*), 826
state (*angr.analyses.variable_recovery.engine_base.SimEngineVRBase attribute*), 827
state (*angr.analyses.variable_recovery.engine_vex.SimEngineVRVEX attribute*), 826
state (*angr.analyses.vfg.PendingJob attribute*), 840
state (*angr.engines.vex.engine.SimEngineVEX attribute*), 427
state (*angr.procedures.stubs.format_parser.FormatParser attribute*), 475
state (*angr.procedures.stubs.format_parser.FormatString property*), 473
state (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 476
state (*angr.sim_procedure.SimProcedure attribute*), 471
state (*angr.SimFileBase attribute*), 189
state (*angr.SimFileDescriptorDuplex attribute*), 203
state (*angr.SimFileStreamBase attribute*), 196
state (*angr.SimHeapBrk attribute*), 206
state (*angr.SimHeapPTMalloc attribute*), 208
state (*angr.SimHostFilesystem attribute*), 204
state (*angr.SimMount attribute*), 203
state (*angr.SimPackets attribute*), 194
state (*angr.SimPacketsStream attribute*), 198
state (*angr.SimProcedure attribute*), 159
state (*angr.state_plugins.callstack.CallStack attribute*), 264
state (*angr.state_plugins.cgc.SimStateCGC attribute*), 273
state (*angr.state_plugins.concrete.Concrete attribute*), 294
state (*angr.state_plugins.debug_variables.SimDebugVariablePlugin attribute*), 309
state (*angr.state_plugins.filesystem.SimConcreteFilesystem attribute*), 253
state (*angr.state_plugins.filesystem.SimFilesystem attribute*), 251
state (*angr.state_plugins.filesystem.SimHostFilesystem attribute*), 254
state (*angr.state_plugins.filesystem.SimMount attribute*), 251
state (*angr.state_plugins.gdb.GDB attribute*), 271
state (*angr.state_plugins.globals.SimStateGlobals attribute*), 279
state (*angr.state_plugins.heap.heap_base.SimHeapBase attribute*), 298
state (*angr.state_plugins.heap.heap_brk.SimHeapBrk attribute*), 300
state (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist attribute*), 301
state (*angr.state_plugins.heap.heap_libc.SimHeapLibc attribute*), 302
state (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc attribute*), 306
state (*angr.state_plugins.history.SimStateHistory*

attribute), 270
state (*angr.state_plugins.inspect.SimInspector attribute*), 236
state (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader attribute*), 296
state (*angr.state_plugins.jni_references.SimStateJNIReferences attribute*), 297
state (*angr.state_plugins.libc.SimStateLibc attribute*), 240
state (*angr.state_plugins.light_registers.SimLightRegisters attribute*), 267
state (*angr.state_plugins.log.SimStateLog attribute*), 263
state (*angr.state_plugins.loop_data.SimStateLoopData attribute*), 292
state (*angr.state_plugins.plugin.SimStatePlugin attribute*), 232
state (*angr.state_plugins.posix.PosixDevFS attribute*), 242
state (*angr.state_plugins.posix.PosixProcFS attribute*), 243
state (*angr.state_plugins.posix.SimSystemPosix attribute*), 248
state (*angr.state_plugins.preconstrainer.SimStatePreconstrainer attribute*), 284
state (*angr.state_plugins.scratch.SimStateScratch attribute*), 280
state (*angr.state_plugins.solver.SimSolver attribute*), 260
state (*angr.state_plugins.symbolizer.SimSymbolizer attribute*), 307
state (*angr.state_plugins.trace_additions.ChallRespInfo attribute*), 276
state (*angr.state_plugins.trace_additions.ZenPlugin attribute*), 278
state (*angr.state_plugins.uc_manager.SimUCManager attribute*), 280
state (*angr.state_plugins.unicorn_engine.Unicorn attribute*), 291
state (*angr.state_plugins.view.SimMemView attribute*), 312
state (*angr.state_plugins.view.SimRegNameView attribute*), 310
state (*angr.storage.file.SimFile attribute*), 319
state (*angr.storage.file.SimFileBase attribute*), 317
state (*angr.storage.file.SimFileDescriptor attribute*), 329
state (*angr.storage.file.SimFileDescriptorBase attribute*), 327
state (*angr.storage.file.SimFileDescriptorDuplex attribute*), 332
state (*angr.storage.file.SimFileStream attribute*), 321
state (*angr.storage.file.SimPackets attribute*), 323
state (*angr.storage.file.SimPacketsSlots attribute*), 333
state (*angr.storage.file.SimPacketsStream attribute*), 325
state (*angr.storage.memory_mixins.AbstractMemory attribute*), 339
state (*angr.storage.memory_mixins.actions_mixin.ActionsMixinHigh attribute*), 342
state (*angr.storage.memory_mixins.actions_mixin.ActionsMixinLow attribute*), 342
state (*angr.storage.memory_mixins.address_concretization_mixin.Address attribute*), 346
state (*angr.storage.memory_mixins.bvv_conversion_mixin.DataNormaliza attribute*), 341
state (*angr.storage.memory_mixins.clouseau_mixin.InspectMixinHigh attribute*), 346
state (*angr.storage.memory_mixins.conditional_store_mixin.ConditionalM attribute*), 346
state (*angr.storage.memory_mixins.convenient_mappings_mixin.Convenie attribute*), 348
state (*angr.storage.memory_mixins.default_filler_mixin.DefaultFillerMixi attribute*), 340
state (*angr.storage.memory_mixins.default_filler_mixin.ExplicitFillerMixi attribute*), 340
state (*angr.storage.memory_mixins.default_filler_mixin.SpecialFillerMixi attribute*), 340
state (*angr.storage.memory_mixins.dirty_addrs_mixin.DirtyAddrsMixin attribute*), 344
state (*angr.storage.memory_mixins.FastMemory attribute*), 338
state (*angr.storage.memory_mixins.hex_dumper_mixin.HexDumperMixin attribute*), 341
state (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mix attribute*), 378
state (*angr.storage.memory_mixins.JavaVmMemory attribute*), 339
state (*angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory_ attribute*), 376
state (*angr.storage.memory_mixins.KeyValueMemory attribute*), 339
state (*angr.storage.memory_mixins.label_merger_mixin.LabelMergerMixi attribute*), 347
state (*angr.storage.memory_mixins.MemoryMixin attribute*), 338
state (*angr.storage.memory_mixins.multi_value_merger_mixin.MultiValue attribute*), 352
state (*angr.storage.memory_mixins.name_resolution_mixin.NameResoluti attribute*), 339
state (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.C attribute*), 357
state (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.C attribute*), 357
state (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.D attribute*), 358
state (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin attribute*), 355
state (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin*

attribute), 356
state (angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin attribute), 356
state (angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagesMixin attribute), 356
state (angr.storage.memory_mixins.paged_memory.paged_memory_mixin.ResetMixin attribute), 356
state (angr.storage.memory_mixins.paged_memory.paged_memory_mixin.SimMixin attribute), 357
state (angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.HistoryTrackingMixin attribute), 361
state (angr.storage.memory_mixins.paged_memory.pages.ispo_mixin.ISPOMixin attribute), 361
state (angr.storage.memory_mixins.paged_memory.pages.list_page.ListPage attribute), 363
state (angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage attribute), 350
state (angr.storage.memory_mixins.paged_memory.pages.page_base.PageBase attribute), 359
state (angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.PermissionsMixin attribute), 360
state (angr.storage.memory_mixins.paged_memory.pages.refcount_mixin.RefcountMixin attribute), 360
state (angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage attribute), 365
state (angr.storage.memory_mixins.paged_memory.privileged_mixin.PrivilegedPagingMixin attribute), 359
state (angr.storage.memory_mixins.paged_memory.stack_allocation_mixin.StackAllocationMixin attribute), 358
state (angr.storage.memory_mixins.regioned_memory.abstract_region_mapper.AbstractRegionMapper attribute), 373
state (angr.storage.memory_mixins.regioned_memory.region_category_mixin.RegionCategoryMixin attribute), 370
state (angr.storage.memory_mixins.regioned_memory.region_meta_mixin.MemoryRegionMetaMixin attribute), 373
state (angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin.RegionedAddressConcretizationMixin attribute), 374
state (angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin attribute), 368
state (angr.storage.memory_mixins.regioned_memory.static_find.StaticFindMixin attribute), 371
state (angr.storage.memory_mixins.simple_interface_mixin.SimpleInterfaceMixin attribute), 342
state (angr.storage.memory_mixins.simplification_mixin.SimplificationMixin attribute), 347
state (angr.storage.memory_mixins.size_resolution_mixin.SizeConcretizationMixin attribute), 344
state (angr.storage.memory_mixins.size_resolution_mixin.SizeNormalizationMixin attribute), 343
state (angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin attribute), 375
state (angr.storage.memory_mixins.smart_find_mixin.SmartFindMixin attribute), 340
state (angr.storage.memory_mixins.symbolic_merger_mixin.SymbolicMergerMixin attribute), 342
state (angr.storage.memory_mixins.top_merger_mixin.TopMergerMixin attribute), 352
state (angr.storage.memory_mixins.underconstrained_mixin.UnderconstrainedMixin attribute), 342
state (angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin attribute), 347
state_blank() (angr.SimOS method), 168
state_blank() (angr.simos.cgc.SimCGC method), 878
state_blank() (angr.simos.javavm.SimJavaVM method), 881
state_blank() (angr.simos.linux.SimLinux method), 877
state_blank() (angr.simos.simos.SimOS method), 875
state_blank() (angr.simos.windows.SimWindows method), 879
state_call() (angr.SimOS method), 169
state_call() (angr.simos.javavm.SimJavaVM method), 881
state_call() (angr.simos.simos.SimOS method), 875
state_entry() (angr.SimOS method), 168
state_entry() (angr.simos.cgc.SimCGC method), 878
state_entry() (angr.simos.javavm.SimJavaVM method), 881
state_entry() (angr.simos.linux.SimLinux method), 877
state_entry() (angr.simos.simos.SimOS method), 875
state_entry() (angr.simos.windows.SimWindows method), 879
state_full_init() (angr.SimOS method), 168
state_full_init() (angr.simos.linux.SimLinux method), 877
state_full_init() (angr.simos.simos.SimOS method), 875
state_priority() (angr.exploration_techniques.Spiller method), 402
state_priority() (angr.exploration_techniques.spiller.Spiller method), 412
StateHierarchy (class in angr), 180
StateHierarchy (class in angr.state_hierarchy), 389
statement_location() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionFolding method), 715
StatementMigration (class in angr.analyses.decompiler.region_simplifiers.expr_folding), 715
statements (angr.analyses.decompiler.structured_codegen.c.CFunction attribute), 726
statements (angr.analyses.decompiler.structured_codegen.c.CStatements attribute), 726
statements (angr.engines.pcode.lifter.IRSB property), 439
StateOption (class in angr.sim_state_options), 228
StaticAnalysis (class in angr.analyses.disassembly.SootExpressionInvoke

attribute), 848
STATIC (*angr.simos.windows.SecurityCookieInit attribute*), 879
static_exits() (*angr.sim_procedure.SimProcedure method*), 472
static_exits() (*angr.SimProcedure method*), 159
StaticFindMixin (*class in angr.storage.memory_mixins.regioned_memory.static_find_mixin*), 370
StaticHooker (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
StaticHooker (*class in angr.analyses.static_hooker*), 859
StaticObjectFinder (*class in angr.analyses.find_objects_static*), 846
status() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 608
step() (*angr.distributed.worker.BadStatesDropper method*), 900
step() (*angr.distributed.worker.ExplorationStatusNotifier method*), 901
step() (*angr.exploration_techniques.DFS method*), 398
step() (*angr.exploration_techniques.dfs.DFS method*), 408
step() (*angr.exploration_techniques.Director method*), 400
step() (*angr.exploration_techniques.director.Director method*), 420
step() (*angr.exploration_techniques.driller_core.DrillerCore method*), 416
step() (*angr.exploration_techniques.DrillerCore method*), 393
step() (*angr.exploration_techniques.ExplorationTechnique method*), 390
step() (*angr.exploration_techniques.Explorer method*), 397
step() (*angr.exploration_techniques.explorer.Explorer method*), 409
step() (*angr.exploration_techniques.LengthLimiter method*), 398
step() (*angr.exploration_techniques.lengthlimiter.LengthLimiter method*), 409
step() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 410
step() (*angr.exploration_techniques.ManualMergepoint method*), 402
step() (*angr.exploration_techniques.memory_watcher.MemoryWatcher method*), 426
step() (*angr.exploration_techniques.MemoryWatcher method*), 405
step() (*angr.exploration_techniques.Spiller method*), 402
step() (*angr.exploration_techniques.spiller.Spiller method*), 412
step() (*angr.exploration_techniques.stochastic.StochasticSearch method*), 423
step() (*angr.exploration_techniques.StochasticSearch method*), 403
step() (*angr.exploration_techniques.Suggestions method*), 407
step() (*angr.exploration_techniques.suggestions.Suggestions method*), 427
step() (*angr.exploration_techniques.Symbion method*), 404
step() (*angr.exploration_techniques.symbion.Symbion method*), 425
step() (*angr.exploration_techniques.Threading method*), 397
step() (*angr.exploration_techniques.threading.Threading method*), 413
step() (*angr.exploration_techniques.Timeout method*), 407
step() (*angr.exploration_techniques.timeout.Timeout method*), 408
step() (*angr.exploration_techniques.Tracer method*), 396
step() (*angr.exploration_techniques.tracer.Tracer method*), 416
step() (*angr.exploration_techniques.unique.UniqueSearch method*), 423
step() (*angr.exploration_techniques.UniqueSearch method*), 403
step() (*angr.ExplorationTechnique method*), 178
step() (*angr.sim_manager.SimulationManager method*), 385
step() (*angr.sim_state.SimState method*), 226
step() (*angr.SimState method*), 183
step() (*angr.SimulationManager method*), 174
step_back() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 790
step_state() (*angr.exploration_techniques.ExplorationTechnique method*), 391
step_state() (*angr.exploration_techniques.Slicecutor method*), 392
step_state() (*angr.exploration_techniques.slicecutor.Slicecutor method*), 417
step_state() (*angr.exploration_techniques.Symbion method*), 404
step_state() (*angr.exploration_techniques.symbion.Symbion method*), 425
step_state() (*angr.exploration_techniques.Tracer method*), 396
step_state() (*angr.exploration_techniques.tracer.Tracer method*), 416
step_state() (*angr.exploration_techniques.Veritesting method*), 399
step_state() (*angr.exploration_techniques.veritesting.Veritesting method*), 413

step_state() (*angr.ExplorationTechnique method*), 179
step_state() (*angr.sim_manager.SimulationManager method*), 386
step_state() (*angr.SimulationManager method*), 175
stmt (*angr.analyses.decompiler.structured_codegen.c.CUnsupportedStatement attribute*), 731
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationMultiStmtBase attribute*), 712
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase attribute*), 712
stmt_comments (*angr.angrdb.models.DbStructuredCode attribute*), 678
stmt_idx (*angr.analyses.cfg.cfg_fast.FunctionEdge attribute*), 650
stmt_idx (*angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionLocation attribute*), 715
stmt_idx (*angr.analyses.decompiler.region_simplifiers.expr_folding.StatementLocation attribute*), 715
stmt_idx (*angr.analyses.disassembly.SootStatement property*), 848
stmt_idx (*angr.code_location.CodeLocation attribute*), 612
stmt_idx (*angr.engines.UberEngine attribute*), 427
stmt_idx (*angr.errors.SimError attribute*), 896
stmt_idx (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 548
stmt_idx (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
stmt_idx (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 610
stmt_idx (*angr.state_plugins.unicorn_engine.VEXStmtDetails attribute*), 285
stmt_observe() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 793
stmt_observe() (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis method*), 773
stmts (*angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression attribute*), 737
stmts (*angr.codenode.SootBlockNode attribute*), 874
stmts_to_instrument (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessorSimAnalysis attribute*), 663
stmts_used (*angr.engines.pcode.lifter.IRSB property*), 438
StochasticSearch (*class in angr.exploration_techniques*), 403
StochasticSearch (*class in angr.exploration_techniques.stochastic*), 423
STOP (*class in angr.state_plugins.unicorn_engine*), 286
stop() (*angr.distributed.server.Server method*), 900
stop() (*angr.Server method*), 210
STOP_ERROR (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_EXECNONE (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_HLT (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
stop_message (*angr.state_plugins.unicorn_engine.STOP attribute*), 287
STOP_NODECODE (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_NORMAL (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_NOSTART (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
stop_reason (*angr.state_plugins.unicorn_engine.StopDetails attribute*), 287
STOP_SEGFAULT (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_STOPPOINT (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_SYMBOLIC_BLOCK_EXIT_CONDITION (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_SYMBOLIC_BLOCK_EXIT_TARGET (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_SYMBOLIC_PC (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_SYMBOLIC_READ_ADDR (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_SYMBOLIC_READ_SYMBOLIC_TRACKING_DISABLED (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_SYMBOLIC_WRITE_ADDR (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_SYSCALL (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_SYSCALL_ARM (*angr.state_plugins.unicorn_engine.STOP attribute*), 287
STOP_UNKNOWN_MEMORY_WRITE_SIZE (*angr.state_plugins.unicorn_engine.STOP attribute*), 287
STOP_UNSUPPORTED_EXPR_GETI (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_UNSUPPORTED_EXPR_UNKNOWN (*angr.state_plugins.unicorn_engine.STOP attribute*), 287
STOP_UNSUPPORTED_STMT_CAS (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_UNSUPPORTED_STMT_DIRTY (*angr.state_plugins.unicorn_engine.STOP

attribute), 286
STOP_UNSUPPORTED_STMT_LLSC (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_UNSUPPORTED_STMT_LOADG (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_UNSUPPORTED_STMT_PUTI (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_UNSUPPORTED_STMT_STOREG (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_UNSUPPORTED_STMT_UNKNOWN (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_VEX_LIFT_FAILED (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_X86_CPUID (*angr.state_plugins.unicorn_engine.STOP attribute*), 287
STOP_ZERO_DIV (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
STOP_ZEROPAGE (*angr.state_plugins.unicorn_engine.STOP attribute*), 286
StopDetails (*class in angr.state_plugins.unicorn_engine*), 287
stopped (*angr.distributed.server.Server property*), 900
stopped (*angr.Server property*), 211
storage (*angr.state_plugins.sim_action.SimActionData property*), 468
Store (*class in angr.analyses.typehoon.typevars*), 832
store() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 818
store() (*angr.sim_type.SimCppClass method*), 517
store() (*angr.sim_type.SimStruct method*), 516
store() (*angr.sim_type.SimTypeArray method*), 512
store() (*angr.sim_type.SimTypeBool method*), 511
store() (*angr.sim_type.SimTypeChar method*), 510
store() (*angr.sim_type.SimTypeFloat method*), 515
store() (*angr.sim_type.SimTypeNum method*), 510
store() (*angr.sim_type.SimTypeNumOffset method*), 518
store() (*angr.sim_type.SimTypeReg method*), 509
store() (*angr.sim_type.SimTypeWideChar method*), 511
store() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 308
store() (*angr.state_plugins.light_registers.SimLightRegisters method*), 267
store() (*angr.state_plugins.view.SimMemView method*), 314
store() (*angr.storage.memory_mixins.actions_mixin.ActionsMixin method*), 342
store() (*angr.storage.memory_mixins.actions_mixin.ActionsMixinLow method*), 342
store() (*angr.storage.memory_mixins.address_concretization_mixin.AddrConcretizationMixin method*), 346
store() (*angr.storage.memory_mixins.bvv_conversion_mixin.DataNormalizationMixin method*), 341
store() (*angr.storage.memory_mixins.clouseau_mixin.InspectMixinHigh method*), 346
store() (*angr.storage.memory_mixins.conditional_store_mixin.ConditionalStoreMixin method*), 346
store() (*angr.storage.memory_mixins.convenient_mappings_mixin.ConvenientMappingsMixin method*), 348
store() (*angr.storage.memory_mixins.dirty_addrs_mixin.DirtyAddrsMixin method*), 344
store() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mixins.JavaVMMemoryMixin method*), 376
store() (*angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory_mixins.KeyValueMemoryMixin method*), 375
store() (*angr.storage.memory_mixins.MemoryMixin method*), 336
store() (*angr.storage.memory_mixins.name_resolution_mixin.NameResolutionMixin method*), 339
store() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixins.PagedMemoryMixin method*), 353
store() (*angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixins.HistoryTrackingMixin method*), 361
store() (*angr.storage.memory_mixins.paged_memory.pages.list_page.ListPage method*), 362
store() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 349
store() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 364
store() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.RegionMetaMixin method*), 372
store() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixins.RegionedMemoryMixin method*), 366
store() (*angr.storage.memory_mixins.simple_interface_mixin.SimpleInterfaceMixin method*), 342
store() (*angr.storage.memory_mixins.simplification_mixin.SimplificationMixin method*), 347
store() (*angr.storage.memory_mixins.size_resolution_mixin.SizeConcretizationMixin method*), 344
store() (*angr.storage.memory_mixins.size_resolution_mixin.SizeNormalizationMixin method*), 343
store() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method*), 375
store() (*angr.storage.memory_mixins.underconstrained_mixin.UnderconstrainedMixin method*), 342
store() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 347
store() (*angr.vaults.Vault method*), 617
store() (*angr.vaults.VaultDirShelf method*), 618
store_array_element() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mixins.JavaVMMemoryMixin method*), 376

store_array_elements() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_mixin.JavaVmMemoryMixin method*), 376
store_tmp() (*angr.state_plugins.scratch.SimStateScratch method*), 280
store_variable() (*angr.analyses.decompiler.optimization_passes.duplication_splitting.DuplicationSplitter method*), 709
stored_objects (*angr.keyed_region.RegionObject attribute*), 613
StoredObject (*class in angr.keyed_region*), 613
StoreHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 664
StoreStatementFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 717
str_to_c_str() (*angr.analyses.decompiler.structured_codegen.c.CConstant static method*), 736
Strict (*angr.exploration_techniques.tracer.TracingMode attribute*), 414
strict_block_end (*angr.engines.pcode.lifter.Lifter attribute*), 440
strict_block_end (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
String (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 863
String (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 542
String (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
string (*angr.procedures.stubs.format_parser.FormatSpecifier attribute*), 474
string (*angr.state_plugins.debug_variables.SimDebugVariable property*), 308
string_escape() (*in module angr.analyses.reassembler*), 850
string_references() (*angr.knowledge_plugins.functions.function.Function method*), 555
StringProxiNode (*class in angr.analyses.proximity_graph*), 864
STRONGREF_STATE (*angr.SimStatePlugin attribute*), 161
STRONGREF_STATE (*angr.state_plugins.history.SimStateHistory attribute*), 267
STRONGREF_STATE (*angr.state_plugins.plugin.SimStatePlugin attribute*), 231
struct (*angr.sim_type.SimStructValue property*), 516
struct (*angr.state_plugins.view.SimMemView attribute*), 313
Struct (*class in angr.analyses.typehoon.typeconsts*), 835
struct_name() (*angr.analyses.typehoon.translator.TypeTranslator method*), 828
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCCdecl attribute*), 492
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCMicrosoftCdecl attribute*), 492
struct_type (*angr.analyses.decompiler.structured_codegen.c.CStructField attribute*), 731
StructMode (*class in angr.state_plugins.view*), 314
structured_code (*angr.angrdb.models.DbKnowledgeBase attribute*), 677
structured_code (*angr.knowledge_base.knowledge_base.KnowledgeBase attribute*), 520
structured_code (*angr.KnowledgeBase attribute*), 211
structured_node_is_simple_return() (*in module angr.analyses.decompiler.utils*), 743
StructuredCodeGenerator (*in module angr.analyses.decompiler.structured_codegen.c*), 741
StructuredCodeManager (*class in angr.knowledge_plugins.structured_code.manager*), 570
StructuredCodeManagerSerializer (*class in angr.angrdb.serializers.structured_code*), 683
structurer_class_from_name() (*in module angr.analyses.decompiler.structuring*), 684
StructurerBase (*class in angr.analyses.decompiler.structuring.structurer_base*), 689
STRUCTURING (*angr.analyses.decompiler.optimization_passes.lowered_switch_builder.LoweredSwitchBuilder attribute*), 707
STRUCTURING (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 703
Sub (*angr.engines.light.data.ArithmeticExpression attribute*), 750
Sub (*class in angr.analyses.typehoon.typevars*), 830
sub_graph (*angr.analyses.data_dep.data_dependency_analysis.DataDependencyAnalysis property*), 866
sub_type (*angr.analyses.typehoon.typevars.Subtype attribute*), 830
subgraph() (*angr.knowledge_plugins.functions.function.Function method*), 558
subgraph_between_nodes() (*in module angr.utils.graph*), 887
subject (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 794
subject (*angr.analyses.reaching_definitions.ReachingDefinitionsAnalysis property*), 773
Subject (*class in angr.analyses.reaching_definitions.subject*), 813
SubjectType (*class in angr.analyses.reaching_definitions.subject*), 813
SubN (*class in angr.analyses.typehoon.typevars*), 832
subscribe_actions() (*angr.state_plugins.history.SimStateHistory method*), 269

Subtype (*class in angr.analyses.typehoon.typevars*), 829
successor_func() (*angr.annocfg.AnnotatedCFG method*), 873
successors (*angr.analyses.decompiler.graph_region.GraphRegion attribute*), 700
successors (*angr.engines.UberEngine attribute*), 427
successors (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 545
successors (*angr.knowledge_plugins.cfg.CFGNode property*), 526
successors() (*angr.analyses.forward_analysis.visitors.call_graph.CallGraphVisitor method*), 622
successors() (*angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor method*), 623
successors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 624
successors() (*angr.analyses.forward_analysis.visitors.loop.LoopVisitor method*), 626
successors() (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor method*), 627
successors() (*angr.codenode.CodeNode method*), 873
successors() (*angr.exploration_techniques.Bucketizer method*), 406
successors() (*angr.exploration_techniques.bucketizer.Bucketizer method*), 426
successors() (*angr.exploration_techniques.ExplorationTechnique method*), 391
successors() (*angr.exploration_techniques.local_loop_seer.LocalLoopSeer method*), 422
successors() (*angr.exploration_techniques.LocalLoopSeer method*), 406
successors() (*angr.exploration_techniques.loop_seer.LoopSeer method*), 421
successors() (*angr.exploration_techniques.LoopSeer method*), 394
successors() (*angr.exploration_techniques.Oppologist method*), 399
successors() (*angr.exploration_techniques.oppologist.Oppologist method*), 420
successors() (*angr.exploration_techniques.Slicecutor method*), 392
successors() (*angr.exploration_techniques.slicecutor.Slicecutor method*), 418
successors() (*angr.exploration_techniques.Threading method*), 398
successors() (*angr.exploration_techniques.threading.Threading method*), 413
successors() (*angr.ExplorationTechnique method*), 179
successors() (*angr.factory.AngrObjectFactory method*), 216
successors() (*angr.sim_manager.SimulationManager method*), 386
successors() (*angr.SimulationManager method*), 175
successors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 545
successors_and_jumpkinds() (*angr.knowledge_plugins.cfg.CFGNode method*), 526
SuccessorsMixin (*class in angr.engines.engine*), 428
Suggestions (*class in angr.exploration_techniques*), 407
Suggestions (*class in angr.exploration_techniques.suggestions*), 426
super_type (*angr.analyses.typehoon.typevars.Subtype attribute*), 830
SUPPORTS_CONCRETE_LOAD (*angr.storage.memory_mixins.MemoryMixin attribute*), 336
SUPPORTS_CONCRETE_LOAD (*angr.storage.memory_mixins.paged_memory.paged_memory_mi attribute*), 353
SUPPORTS_CONCRETE_LOAD (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.U attribute*), 363
switch (*angr.analyses.decompiler.structured_codegen.c.CSwitchCase attribute*), 729
switch_expr (*angr.analyses.decompiler.structuring.structurer_nodes.Swit attribute*), 688
switch_expr_cmp_bounds() (*in module angr.analyses.decompiler.utils*), 742
switch_variable (*angr.analyses.decompiler.structuring.structurer_nodes attribute*), 689
SwitchCaseNode (*class in angr.analyses.decompiler.structuring.structurer_nodes*), 688
SwitchCaseRegion (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl 719*
SwitchClusterFinder (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl 720*
SwitchClusterReplacer (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl 720*
SwitchExpressionSimplifier (*class in angr.analyses.decompiler.region_simplifiers.switch_expr_simplifi 722*
Symbion (*class in angr.exploration_techniques*), 404
Symbion (*class in angr.exploration_techniques.symbion*), 424
symbol (*angr.analyses.reaching_definitions.function_handler.FunctionCall attribute*), 800
symbol (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
symbol (*angr.knowledge_plugins.functions.function.Function*

property), 556
symbol_hooked_by() (*angr.Project method*), 166
symbol_hooked_by() (*angr.project.Project method*), 215
symbol_name (*angr.analyses.reassembler.NotypeLabel property*), 851
symbol_name (*angr.analyses.reassembler.ObjectLabel property*), 851
symbolic (*angr.analyses.reaching_definitions.MemoryLocation property*), 769
symbolic (*angr.engines.light.data.RegisterOffset property*), 750
symbolic (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation property*), 588
SYMBOLIC (*angr.simos.windows.SecurityCookieInit attribute*), 879
symbolic (*angr.storage.memory_object.SimMemoryObject property*), 334
symbolic() (*angr.state_plugins.solver.SimSolver method*), 262
symbolic_stop_reasons (*angr.state_plugins.unicorn_engine.STOP attribute*), 287
symbolic_vex_stmts (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 285
symbolic_vex_stmts_count (*angr.state_plugins.unicorn_engine.BlockDetails attribute*), 286
SymbolicMergerMixin (*class in angr.storage.memory_mixins.symbolic_merger_mixin*), 342
symbolize() (*angr.analyses.reassembler.Reassembler method*), 858
symbolized (*angr.analyses.reassembler.Operand property*), 852
SymbolManager (*class in angr.analyses.reassembler*), 851
sympy_expr_to_claripy_ast() (*angr.analyses.decompiler.condition_processor.ConditionProcessor static method*), 696
sync() (*angr.state_plugins.concrete.Concrete method*), 294
SyncController (*class in angr.knowledge_plugins.sync.sync_controller*), 607
syscall (*angr.analyses.cfg.cfg_fast.CFGJob attribute*), 652
syscall (*angr.analyses.cfg.cfg_fast.FunctionCallEdge attribute*), 650
syscall (*angr.knowledge_plugins.cfg.cfg_node.CFGENode attribute*), 547
syscall (*angr.knowledge_plugins.cfg.CFGENode attribute*), 528
syscall() (*angr.SimOS method*), 169
syscall() (*angr.simos.simos.SimOS method*), 876
syscall() (*angr.simos.userland.SimUserland method*), 878
syscall_abi() (*angr.SimOS method*), 169
syscall_abi() (*angr.simos.linux.SimLinux method*), 877
syscall_abi() (*angr.simos.simos.SimOS method*), 876
syscall_abi() (*angr.simos.userland.SimUserland method*), 878
syscall_cc() (*angr.SimOS method*), 169
syscall_cc() (*angr.simos.simos.SimOS method*), 876
syscall_cc() (*angr.simos.userland.SimUserland method*), 878
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 499
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 498
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 500
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 499
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCSyscall attribute*), 493
syscall_from_addr() (*angr.SimOS method*), 169
syscall_from_addr() (*angr.simos.simos.SimOS method*), 876
syscall_from_addr() (*angr.simos.userland.SimUserland method*), 879
syscall_from_number() (*angr.SimOS method*), 169
syscall_from_number() (*angr.simos.simos.SimOS method*), 876
syscall_from_number() (*angr.simos.userland.SimUserland method*), 879
syscall_hook() (*in module angr.state_plugins.trace_additions*), 274
syscall_name (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 546
syscall_name (*angr.knowledge_plugins.cfg.CFGNode attribute*), 527
syscall_num() (*angr.calling_conventions.SimCCAArch64LinuxSyscall static method*), 497
syscall_num() (*angr.calling_conventions.SimCCAMD64LinuxSyscall static method*), 495
syscall_num() (*angr.calling_conventions.SimCCAMD64WindowsSyscall static method*), 495
syscall_num() (*angr.calling_conventions.SimCCARMLinuxSyscall static method*), 496

syscall_num() (*angr.calling_conventions.SimCCN64LinuxSyscall static method*), 499
syscall_num() (*angr.calling_conventions.SimCCO32LinuxSyscall static method*), 498
syscall_num() (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall static method*), 500
syscall_num() (*angr.calling_conventions.SimCCPowerPCLinuxSyscall static method*), 499
syscall_num() (*angr.calling_conventions.SimCCRISCV64LinuxSyscall static method*), 497
syscall_num() (*angr.calling_conventions.SimCCS390XLinuxSyscall static method*), 502
syscall_num() (*angr.calling_conventions.SimCCSyscall static method*), 493
syscall_num() (*angr.calling_conventions.SimCCX86LinuxSyscall static method*), 494
syscall_num() (*angr.calling_conventions.SimCCX86WindowsSyscall static method*), 494
SyscallNode (*class in angr.codenode*), 874

T

T (*angr.sim_state.SimState attribute*), 226
T (*angr.SimState attribute*), 182
tag (*angr.analyses.cfg.cfg_base.CFGBase attribute*), 645
tag (*angr.analyses.cfg.cfg_emulated.CFGEmulated attribute*), 641
tag (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 653
Tag (*class in angr.knowledge_plugins.key_definitions.tag*), 604
tags (*angr.analyses.decompiler.structured_codegen.c.CAssignment attribute*), 729
tags (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp attribute*), 734
tags (*angr.analyses.decompiler.structured_codegen.c.CBreak attribute*), 729
tags (*angr.analyses.decompiler.structured_codegen.c.CConstant attribute*), 735
tags (*angr.analyses.decompiler.structured_codegen.c.CContinue attribute*), 729
tags (*angr.analyses.decompiler.structured_codegen.c.CDoWhileLoop attribute*), 727
tags (*angr.analyses.decompiler.structured_codegen.c.CFakeVariable attribute*), 732
tags (*angr.analyses.decompiler.structured_codegen.c.CForLoop attribute*), 728
tags (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall attribute*), 730
tags (*angr.analyses.decompiler.structured_codegen.c.CGoto attribute*), 731
tags (*angr.analyses.decompiler.structured_codegen.c.CIfBreak attribute*), 728
tags (*angr.analyses.decompiler.structured_codegen.c.CIfElse attribute*), 728
tags (*angr.analyses.decompiler.structured_codegen.c.CITE attribute*), 736
tags (*angr.analyses.decompiler.structured_codegen.c.CLabel attribute*), 731
tags (*angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpr attribute*), 737
tags (*angr.analyses.decompiler.structured_codegen.c.CRegister attribute*), 736
tags (*angr.analyses.decompiler.structured_codegen.c.CReturn attribute*), 730
tags (*angr.analyses.decompiler.structured_codegen.c.CStructField attribute*), 732
tags (*angr.analyses.decompiler.structured_codegen.c.CSwitchCase attribute*), 729
tags (*angr.analyses.decompiler.structured_codegen.c.CTypeCast attribute*), 735
tags (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp attribute*), 734
tags (*angr.analyses.decompiler.structured_codegen.c.CVariable attribute*), 732
tags (*angr.analyses.decompiler.structured_codegen.c.CWhileLoop attribute*), 727
tags (*angr.analyses.reaching_definitions.Definition attribute*), 770
tags (*angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute*), 798
tags (*angr.knowledge_plugins.functions.function.Function attribute*), 553
tags (*angr.knowledge_plugins.functions.soot_function.SootFunction attribute*), 560
tags (*angr.knowledge_plugins.key_definitions.Definition attribute*), 583
tags (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 590
TAINT_ENTITY_MEM (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 284
TAINT_ENTITY_NONE (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 284
TAINT_ENTITY_REG (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 284
TAINT_ENTITY_TMP (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 284
TaintEntityEnum (*class in angr.state_plugins.unicorn_engine*), 284
taken (*angr.exploration_techniques.spiller_db.PickledState attribute*), 412
tally() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 608
tally() (*angr.sim_state_options.SimStateOptions method*), 231
target (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 815
target (*angr.analyses.decompiler.optimization_passes.lowered_switch_sim attribute*), 707

target (*angr.analyses.decompiler.structured_codegen.c.CGoto attribute*), 730
target (*angr.analyses.decompiler.structuring.structurer_nodes.BaseNode attribute*), 687
target (*angr.analyses.decompiler.structuring.structurer_nodes.ContinueNode attribute*), 688
target (*angr.analyses.reaching_definitions.call_trace.CallTrace attribute*), 790
target (*angr.analyses.reaching_definitions.dep_graph.FunctionCallRelationship attribute*), 794
target (*angr.analyses.reaching_definitions.GuardUse attribute*), 769
target (*angr.knowledge_plugins.key_definitions.atoms.GuardUse attribute*), 586
target_idx (*angr.analyses.decompiler.optimization_passes.hundred_gr.knowledge_plugins.cfg.CFGNode attribute*), 707
target_idx (*angr.analyses.decompiler.structured_codegen.c.CGoto attribute*), 730
tc2simtype() (*angr.analyses.typehoon.translator.TypeTranslator method*), 828
teardown_callsite() (*angr.calling_conventions.SimCC method*), 490
teardown_callsite() (*angr.SimCC method*), 187
TechniqueBuilder (*class in angr.exploration_techniques*), 402
TechniqueBuilder (*class in angr.exploration_techniques.tech_builder*), 424
tell() (*angr.SimFileDescriptor method*), 198
tell() (*angr.SimFileDescriptorDuplex method*), 201
tell() (*angr.storage.file.SimFileDescriptor method*), 327
tell() (*angr.storage.file.SimFileDescriptorBase method*), 326
tell() (*angr.storage.file.SimFileDescriptorDuplex method*), 330
TemporaryNode (*class in angr.utils.graph*), 888
temps (*angr.slicer.SimLightState attribute*), 871
terminate_execution() (*angr.engines.soot.engine.SootMixin static method*), 432
terminate_execution() (*angr.Project method*), 166
terminate_execution() (*angr.project.Project method*), 216
test_empty_condition_node() (*angr.analyses.decompiler.structuring.structurer_nodes.BaseNode static method*), 685
test_empty_node() (*angr.analyses.decompiler.structuring.structurer_nodes.BaseNode static method*), 685
test_unsupported_overlap() (*angr.knowledge_plugins.debug_variables.DebugVariable method*), 569
text (*angr.analyses.decompiler.structured_codegen.c.CArrayType attribute*), 737
Threading (*class in angr.exploration_techniques*), 397
Threading (*class in angr.exploration_techniques.threading*), 413
THUMB (*angr.analyses.cfg.cfg_fast.ARMDecodingMode attribute*), 648
thumb (*angr.Block attribute*), 170
thumb (*angr.block.Block attribute*), 221
thumb (*angr.block.DisassemblerBlock attribute*), 220
thumb (*angr.codenode.CodeNode attribute*), 873
thumb (*angr.engines.pcode.lifter.PcodeDisassemblerBlock attribute*), 435
thumb (*angr.knowledge_plugins.cfg.cfg_node.CFGNode attribute*), 545
thumb (*angr.knowledge_plugins.cfg.CFGNode attribute*), 526
thumb (*angr.sim_state.SimState property*), 228
thumb (*angr.SimState property*), 184
tidy_data_references() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 541
tidy_data_references() (*angr.knowledge_plugins.cfg.CFGModel method*), 534
timed_function() (*in module angr.state_plugins.solver*), 254
Timeout (*class in angr.exploration_techniques*), 407
Timeout (*class in angr.exploration_techniques.timeout*), 407
timestamp (*angr.exploration_techniques.spiller_db.PickledState attribute*), 412
timethis() (*in module angr.utils.timing*), 892
TLSMixin (*class in angr.engines.engine*), 428
TLSProperty (*class in angr.engines.engine*), 428
Tmp (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 867
tmp (*angr.analyses.propagator.vex_vars.VEXTmp attribute*), 752
TMP (*angr.analyses.reaching_definitions.AtomKind attribute*), 766
TMP (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 584
TMP (*angr.state_plugins.sim_action.SimAction attribute*), 466
Tmp (*class in angr.analyses.reaching_definitions*), 769
Tmp (*class in angr.knowledge_plugins.key_definitions.atoms*), 586
tmp_deps (*angr.state_plugins.sim_action.SimAction property*), 466
tmp_deps (*angr.state_plugins.sim_action.SimActionData property*), 468
tmp_expr() (*angr.state_plugins.scratch.SimStateScratch method*), 280
tmp_id (*angr.sim_variable.SimTemporaryVariable*

attribute), 504
tmp_idx (*angr.analyses.reaching_definitions.Tmp attribute*), 769
tmp_idx (*angr.knowledge_plugins.key_definitions.atoms.Tmp attribute*), 587
tmp_idx (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 589
tmp_uses (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
tmp_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 809
tmp_uses (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
tmp_uses (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594
tmp_uses (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 574
TmpDepNode (*class in angr.analyses.data_dep.dep_nodes*), 869
tmps (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
tmps (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 809
tmps (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
tmps (*angr.engines.UberEngine attribute*), 427
tmps (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594
tmps (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
TmpvarFinder (*class in angr.analyses.propagator.tmpvar_finder*), 755
to_acyclic_graph() (*in module angr.utils.graph*), 887
to_ail_supergraph() (*in module angr.analyses.decompiler.utils*), 742
to_bits (*angr.analyses.typehoon.typevars.ConvertTo attribute*), 832
to_bits (*angr.analyses.typehoon.typevars.ReinterpretAs attribute*), 832
to_claripy() (*angr.state_plugins.sim_action_object.SimActionObject method*), 468
to_codenode() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 546
to_codenode() (*angr.knowledge_plugins.cfg.CFGNode method*), 527
to_engine() (*angr.engines.concrete.SimEngineConcrete method*), 433
to_outside (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge attribute*), 650
to_string() (*angr.knowledge_plugins.xrefs.xref_types.XRefTypes static method*), 611
to_type (*angr.analyses.typehoon.typevars.ReinterpretAs attribute*), 832
to_valueset() (*angr.storage.memory_mixins.regioned_memory.region_d method*), 369
TOLOWER_LOC_ARRAY (*angr.state_plugins.libc.SimStateLibc attribute*), 238
top (*angr.state_plugins.callstack.CallStack property*), 385
Top (*class in angr.analyses.propagator.values*), 751
top() (*angr.analyses.reaching_definitions.LiveDefinitions static method*), 758
top() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsSta method*), 807
top() (*angr.analyses.reaching_definitions.ReachingDefinitionsState method*), 777
top() (*angr.analyses.variable_recovery.variable_recovery_base.VariableR static method*), 821
top() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinit static method*), 595
top() (*angr.knowledge_plugins.key_definitions.LiveDefinitions static method*), 574
TopCheckerMixin (*class in angr.analyses.propagator.top_checker_mixin*), 757
TopMergerMixin (*class in angr.storage.memory_mixins.top_merger_mixin*), 351
TopType (*class in angr.analyses.typehoon.typeconsts*), 824
TOUPPER_LOC_ARRAY (*angr.state_plugins.libc.SimStateLibc attribute*), 238
traceback (*angr.knowledge_plugins.cfg.cfg_node.CFGNodeCreationFailu attribute*), 544
traceflags (*angr.engines.pcode.lifter.Lifter attribute*), 440
traceflags (*angr.engines.pcode.lifter.PcodeLifter attribute*), 442
Tracer (*class in angr.exploration_techniques*), 394
Tracer (*class in angr.exploration_techniques.tracer*), 414
TracerDesyncError, 414
TracerEnvironmentError, 896
TracingMode (*class in angr.exploration_techniques.tracer*), 414
track_tmps (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
track_tmps (*angr.knowledge_plugins.key_definitions.live_definitions.LiveI attribute*), 594
track_tmps (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 573
transition_graph (*angr.knowledge_plugins.functions.function.Function attribute*), 552
transition_graph (*angr.knowledge_plugins.functions.soot_function.Soot attribute*), 560
transition_graph_ex() (*angr.knowledge_plugins.functions.function.Function*

method), 557
transitions (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 814
transitions_as_tuples (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 814
transitive_closure() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 796
translate() (*angr.calling_conventions.AllocHelper method*), 483
TRANSMIT_RECORD (*class in angr.state_plugins.unicorn_engine*), 284
TreeIter (*class in angr.state_plugins.history*), 270
trim() (*angr.state_plugins.history.SimStateHistory method*), 268
true_node (*angr.analyses.decompiler.structuring.structurer_nodes.ConditionNode attribute*), 686
Truncation (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransferringTypes attribute*), 662
try_unpack_const() (*angr.engines.light.data.ArithmeticExpression static method*), 750
ty_ptr() (*angr.sim_procedure.SimProcedure method*), 473
ty_ptr() (*angr.SimProcedure method*), 160
tyenv (*angr.engines.pcode.lifter.IRSB property*), 438
type (*angr.analyses.data_dep.dep_nodes.BaseDepNode property*), 868
type (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp property*), 734
type (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 736
type (*angr.analyses.decompiler.structured_codegen.c.CDirtyExpression property*), 737
type (*angr.analyses.decompiler.structured_codegen.c.CExpression property*), 726
type (*angr.analyses.decompiler.structured_codegen.c.CFakeVariable property*), 732
type (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall property*), 730
type (*angr.analyses.decompiler.structured_codegen.c.CIndexedVariable property*), 733
type (*angr.analyses.decompiler.structured_codegen.c.CITE property*), 736
type (*angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression property*), 737
type (*angr.analyses.decompiler.structured_codegen.c.CRegister property*), 736
type (*angr.analyses.decompiler.structured_codegen.c.CStructureCo property*), 732
type (*angr.analyses.decompiler.structured_codegen.c.CTypeCast property*), 735
type (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp property*), 734
type (*angr.analyses.decompiler.structured_codegen.c.CVariable property*), 733
type (*angr.analyses.decompiler.structured_codegen.c.CVariableField property*), 733
type (*angr.analyses.loop_analysis.AnnotatedVariable attribute*), 837
type (*angr.analyses.reaching_definitions.subject.Subject property*), 813
type (*angr.angrdb.models.DbComment attribute*), 679
type (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump attribute*), 549
type (*angr.knowledge_plugins.cfg.IndirectJump attribute*), 529
type (*angr.knowledge_plugins.xrefs.xref.XRef attribute*), 610
type (*angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory_m attribute*), 375
type_ (*angr.analyses.typehoon.typevars.Existence attribute*), 829
type_0 (*angr.analyses.typehoon.typevars.Add attribute*), 830
type_0 (*angr.analyses.typehoon.typevars.Sub attribute*), 830
type_1 (*angr.analyses.typehoon.typevars.Add attribute*), 830
type_1 (*angr.analyses.typehoon.typevars.Sub attribute*), 830
type_a (*angr.analyses.typehoon.typevars.Equivalence attribute*), 829
type_b (*angr.analyses.typehoon.typevars.Equivalence attribute*), 829
type_constraints (*angr.analyses.decompiler.decompilation_cache.Deco attribute*), 697
type_constraints (*angr.analyses.variable_recovery.engine_base.RichR attribute*), 827
type_parser_singleton() (*in module angr.sim_type*), 518
type_r (*angr.analyses.typehoon.typevars.Add attribute*), 830
type_r (*angr.analyses.typehoon.typevars.Sub attribute*), 830
type_string (*angr.knowledge_plugins.xrefs.xref.XRef property*), 610
type_to_c_repr_chunks() (*in module angr.analyses.decompiler.structured_codegen.c*), 725
type_var (*angr.analyses.typehoon.typevars.DerivedTypeVariable attribute*), 831
TypeConstant (*class in angr.analyses.typehoon.typeconsts*), 834
TypeConstraint (*class in angr.analyses.typehoon.typevars*), 829
TypedVariable (*class in angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory*

375
Typehoon (*class in angr.analyses.typehoon.typehoon*), 833
TypeLifter (*class in angr.analyses.typehoon.lifter*), 827
TypeRef (*class in angr.sim_type*), 508
types (*angr.knowledge_base.knowledge_base.KnowledgeBase attribute*), 520
types (*angr.KnowledgeBase attribute*), 211
types (*angr.sim_state_options.StateOption attribute*), 228
types (*angr.state_plugins.view.SimMemView attribute*), 311
TypesStore (*class in angr.knowledge_plugins.types*), 549
TypeTranslator (*class in angr.analyses.typehoon.translator*), 828
typevar (*angr.analyses.variable_recovery.engine_base.RichR attribute*), 827
TypeVariable (*class in angr.analyses.typehoon.typevars*), 831
TypeVariableReference (*class in angr.analyses.typehoon.typeconsts*), 835
TypeVariables (*class in angr.analyses.typehoon.typevars*), 831

U

UberEngine (*class in angr.engines*), 427
UberEnginePcode (*class in angr.engines*), 427
uc (*angr.state_plugins.unicorn_engine.Unicorn property*), 290
UC_CONFIG (*angr.state_plugins.unicorn_engine.Unicorn attribute*), 288
UltraPage (*class in angr.storage.memory_mixins.paged_memory.pages.ultra_page*), 363
UltraPagesMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 357
uncache_region() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
unconstrained (*angr.sim_manager.SimulationManager attribute*), 383
unconstrained (*angr.SimulationManager attribute*), 172
Unconstrained() (*angr.state_plugins.solver.SimSolver method*), 255
Undefined (*class in angr.knowledge_plugins.key_definitions.undefined*), 605
UnderconstrainedMixin (*class in angr.storage.memory_mixins.underconstrained_mixin*), 341
unfreeze() (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerState method*), 817
unhook() (*angr.Project method*), 165
unhook() (*angr.project.Project method*), 215
unhook_symbol() (*angr.Project method*), 166
unhook_symbol() (*angr.project.Project method*), 216
UnicodeString (*angr.knowledge_plugins.cfg.memory_data.MemoryDataS attribute*), 542
UnicodeString (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
Unicorn (*class in angr.state_plugins.unicorn_engine*), 288
unified_local_vars (*angr.analyses.decompiler.structured_codegen.c.CH attribute*), 726
unified_variable (*angr.analyses.decompiler.structured_codegen.c.CVar attribute*), 732
unified_variable() (*angr.knowledge_plugins.variables.variable_manag method*), 567
unify_arch_name() (*in module angr.calling_conventions*), 502
unify_variables() (*angr.knowledge_plugins.variables.variable_manage method*), 566
uninit_read_base (*angr.analyses.cfg.indirect_jump_resolvers.jumptable. attribute*), 662
UninitReadMeta (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 662
union() (*angr.state_plugins.sim_action_object.SimActionObject method*), 468
unique() (*angr.state_plugins.solver.SimSolver method*), 261
unique_type_name() (*angr.knowledge_plugins.types.TypesStore method*), 549
UniqueSearch (*class in angr.exploration_techniques*), 403
UniqueSearch (*class in angr.exploration_techniques.unique*), 423
Uniwrapper (*class in angr.state_plugins.unicorn_engine*), 288
Unknown (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 863
Unknown (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 547
Unknown (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 529
Unknown (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 542
Unknown (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
Unknown (*class in angr.analyses.cfg.cfb*), 637
UnknownProxiNode (*class in angr.analyses.proximity_graph*), 864
UnknownSize (*class in angr.knowledge_plugins.key_definitions.unknown_size*), 605
UnknownSizeTag (*class in angr.knowledge_plugins.key_definitions.tag*), 605

unlinks (*angr.state_plugins.filesystem.SimFilesystem property*), 249
unmap_by_address() (*angr.storage.memory_mixins.regioned_memory.region_data.RegionMap method*), 370
unmap_region() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 346
unmap_region() (*angr.storage.memory_mixins.MemoryMixin method*), 337
unmap_region() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PageMemoryMixin method*), 354
unmatched_blocks (*angr.analyses.bindiff.FunctionDiff property*), 631
unmatched_functions (*angr.analyses.bindiff.BinDiff property*), 632
UnmatchedStatementsException, 629
unmount() (*angr.state_plugins.filesystem.SimFilesystem method*), 250
unpack_array() (*in module angr.analyses.decompiler.structured_codegen.c*), 724
unpack_pointer() (*in module angr.analyses.decompiler.structured_codegen.c*), 724
unpack_typeref() (*in module angr.analyses.decompiler.structured_codegen.c*), 724
unqualified_name() (*angr.sim_type.NamedTypeMixin method*), 509
unreachable_history() (*angr.state_hierarchy.StateHierarchy method*), 390
unreachable_history() (*angr.StateHierarchy method*), 180
unreachable_state() (*angr.state_hierarchy.StateHierarchy method*), 390
unreachable_state() (*angr.StateHierarchy method*), 180
unresolvables (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 644
unresolved_indirect_jumps (*angr.knowledge_base.knowledge_base.KnowledgeBase property*), 520
unresolved_indirect_jumps (*angr.KnowledgeBase property*), 211
unroll_loops() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 643
unsat (*angr.sim_manager.SimulationManager attribute*), 383
unsat (*angr.SimulationManager attribute*), 172
unsat_core() (*angr.state_plugins.solver.SimSolver method*), 259
unset_stack_address_mapping() (*angr.storage.memory_mixins.regioned_memory.region_memory_mixin.RegionedMemoryMixin method*), 367
UnsignedExtension (*angr.analyses.cfg.indirect_jump_resolvers.jumptable attribute*), 662
unsilence_logger() (*in module angr.state_plugins.concretization_mixin*), 302
Unspecified (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 542
Unspecified (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 524
unstash() (*angr.sim_manager.SimulationManager method*), 387
unstash() (*angr.SimulationManager method*), 176
unsupported_reasons (*angr.state_plugins.unicorn_engine.STOP attribute*), 287
UnsupportedCCallError, 898
UnsupportedDirtyError, 898
UnsupportedIRExprError, 897
UnsupportedIROpError, 897
UnsupportedIRStmtError, 898
UnsupportedNodeTypeError, 899
UnsupportedSyscallError (*in module angr.errors*), 898
UnwrapperMixin (*class in angr.storage.memory_mixins.unwrapper_mixin*), 347
update() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 608
update() (*angr.procedures.definitions.SimLibrary method*), 476
update() (*angr.procedures.definitions.SimSyscallLibrary method*), 480
update() (*angr.sim_state_options.SimStateOptions method*), 230
update_dbinfo() (*angr.angrdb.db.AngrDB method*), 675
update_resolved_addrs() (*angr.knowledge_plugins.indirect_jumps.IndirectJumps method*), 549
update_switch_case_list() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl* 720
update_variable_types() (*angr.analyses.typehoon.typehoon.Typehoon method*), 833
UpdateArgumentsOption (*class in angr.analyses.calling_convention*), 633
UpdateWhenCCHasNoArgs (*angr.analyses.calling_convention.UpdateArgumentsOption attribute*), 633
use_plugin_preset() (*angr.misc.plugins.PluginHub method*), 223
use_technique() (*angr.SimulationManager*

method), 384
use_technique() (*angr.SimulationManager method*), 173
UsercallArgSession (*class in angr.calling_conventions*), 487
users() (*angr.knowledge_plugins.sync.sync_controller.SyncController method*), 608
uses (*angr.analyses.decompiler.region_simplifiers.expr_folding.Expression.UseFinder attribute*), 716
Uses (*class in angr.knowledge_plugins.key_definitions*), 581
Uses (*class in angr.knowledge_plugins.key_definitions.uses*), 605
uses_by_codeloc (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 758
uses_by_codeloc (*angr.analyses.reaching_definitions.rd_state.RedefinitionDefinitionsState property*), 809
uses_by_codeloc (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 779
uses_by_codeloc (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 594
uses_by_codeloc (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 574

V

va_arg() (*angr.sim_procedure.SimProcedure method*), 472
va_arg() (*angr.SimProcedure method*), 160
val (*angr.analyses.stack_pointer_tracker.Constant attribute*), 817
value (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.Case attribute*), 707
value (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.ConditionalRegion attribute*), 719
value (*angr.analyses.decompiler.structured_codegen.c.CConstant attribute*), 735
value (*angr.analyses.reaching_definitions.ConstantSrc attribute*), 769
value (*angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute*), 798
value (*angr.angrdb.models.DbInformation attribute*), 676
value (*angr.knowledge_plugins.key_definitions.atoms.ConstantSrc attribute*), 586
value (*angr.knowledge_plugins.key_definitions.heap_address.HeapAddress property*), 591
value (*angr.sim_variable.SimConstantVariable attribute*), 503
value (*angr.state_plugins.unicorn_engine.MemoryValue attribute*), 285
value (*angr.state_plugins.unicorn_engine.RegisterValue attribute*), 285
value (*angr.storage.memory_mixins.keyvalue_memory.keyvalue_memory_mixin.TypedVariable attribute*), 375
Value (*class in angr.analyses.disassembly*), 849
value_tuple() (*angr.analyses.data_dep.dep_nodes.BaseDepNode method*), 868
values() (*angr.knowledge_plugins.patches.PatchManager method*), 522
vClues() (*angr.state_plugins.globals.SimStateGlobals method*), 279
values() (*angr.storage.memory_mixins.paged_memory.pages.multi_value. method*), 351
var_collections (*angr.angrdb.models.DbKnowledgeBase attribute*), 677
var_to_typevar (*angr.analyses.decompiler.decompilation_cache.Decomp attribute*), 697
VarDepNode (*class in angr.analyses.data_dep.dep_nodes*), 869
variable (*angr.analyses.decompiler.region_simplifiers.switch_cluster_sim attribute*), 719
variable (*angr.analyses.decompiler.region_simplifiers.switch_cluster_sim attribute*), 720
variable (*angr.analyses.decompiler.structured_codegen.c.CVariable attribute*), 732
variable (*angr.analyses.loop_analysis.AnnotatedVariable attribute*), 837
Variable (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 863
variable (*angr.analyses.variable_recovery.engine_base.RichR attribute*), 827
variable (*angr.knowledge_plugins.key_definitions.definition.DefinitionMa attribute*), 589
variable (*angr.knowledge_plugins.variables.variable_access.VariableAcc attribute*), 561
variable_hash (*angr.analyses.decompiler.optimization_passes.lowered_s attribute*), 707
variable_key_prefix (*angr.storage.memory_mixins.MemoryMixin property*), 336
variable_list_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 726
variable_manager (*angr.analyses.decompiler.structured_codegen.c.CFun attribute*), 726
variable_manager (*angr.analyses.variable_recovery.variable_recovery_b property*), 822
variable_manager (*angr.knowledge_plugins.key_definitions.definition.De attribute*), 589
variable_type (*angr.analyses.decompiler.structured_codegen.c.CVariabl attribute*), 732
VariableAccess (*class in angr.knowledge_plugins.variables.variable_access*), 561
VariableAccessSort (*class in angr.knowledge_plugins.variables.variable_access*), 561
VariableAnnotation (*class in*

angr.analyses.variable_recovery.variable_recovery_base), 819
VariableManager (*class in angr.knowledge_plugins.variables.variable_manager*), 567
VariableManagerInternal (*class in angr.knowledge_plugins.variables.variable_manager*), 562
VariableManagerSerializer (*class in angr.angrdb.serializers.variables*), 682
VariableProxiNode (*class in angr.analyses.proximity_graph*), 863
VariableRecovery (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 620
VariableRecovery (*class in angr.analyses.variable_recovery.variable_recovery*), 825
VariableRecoveryBase (*class in angr.analyses.variable_recovery.variable_recovery_base*), 820
VariableRecoveryFast (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 620
VariableRecoveryFast (*class in angr.analyses.variable_recovery.variable_recovery_fast*), 824
VariableRecoveryFastState (*class in angr.analyses.variable_recovery.variable_recovery_fast*), 823
VariableRecoveryState (*class in angr.analyses.variable_recovery.variable_recovery*), 825
VariableRecoveryStateBase (*class in angr.analyses.variable_recovery.variable_recovery_base*), 820
variables (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase property*), 822
variables (*angr.knowledge_base.knowledge_base.KnowledgeBase attribute*), 520
variables (*angr.KnowledgeBase attribute*), 211
variables (*angr.storage.memory_object.SimMemoryObject property*), 334
variables() (*angr.state_plugins.solver.SimSolver method*), 262
variables_in_use (*angr.analyses.decompiler.structured_codegen.CStructuredCodeGenerator attribute*), 726
VariableSourceAnnotation (*class in angr.analyses.variable_recovery.annotations*), 819
VariableType (*class in angr.knowledge_plugins.variables.variable_manager*), 561
VariableTypes (*class in angr.analyses.loop_analysis*), 837
Vault (*class in angr.vaults*), 617
VaultDict (*class in angr.vaults*), 618
VaultDir (*class in angr.vaults*), 618
VaultDirShelf (*class in angr.vaults*), 618
VaultPickler (*class in angr.vaults*), 617
VaultShelf (*class in angr.vaults*), 618
VaultUnpickler (*class in angr.vaults*), 617
VendorPreset (*class in angr.misc.plugins*), 224
Veritesting (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
Veritesting (*class in angr.analyses.veritesting*), 838
Veritesting (*class in angr.exploration_techniques*), 398
Veritesting (*class in angr.exploration_techniques.veritesting*), 413
VeritestingError, 838
VERSION (*angr.angrdb.db.AngrDB attribute*), 674
vex (*angr.Block property*), 170
vex (*angr.block.Block property*), 221
vex_nostmt (*angr.Block property*), 170
vex_nostmt (*angr.block.Block property*), 221
VEXIRSBScanner (*class in angr.analyses.variable_recovery.irsb_scanner*), 827
VEXMemVar (*class in angr.analyses.propagator.vex_vars*), 752
VEXReg (*class in angr.analyses.propagator.vex_vars*), 752
VEXStmtDetails (*class in angr.state_plugins.unicorn_engine*), 285
VEXTmp (*class in angr.analyses.propagator.vex_vars*), 752
VEXVariable (*class in angr.analyses.propagator.vex_vars*), 752
VFG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
VFG (*class in angr.analyses.vfg*), 842
VFGJob (*class in angr.analyses.vfg*), 839
VFGNode (*class in angr.analyses.vfg*), 842
Virtual (*angr.analyses.disassembly.SootExpressionInvoke attribute*), 848
visited_blocks (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 801
visited_blocks (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 787
visited_blocks (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 793
visited_blocks (*angr.analyses.reaching_definitions.ReachingDefinitionsState property*), 772
visitor (*angr.analyses.reaching_definitions.subject.Subject property*), 813
VSA_DDG (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 619
VSA_DDG (*class in angr.analyses.vsa_ddg*), 844
Vtable (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType*

attribute), 547
Vtable (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 529
Vtable (*class in angr.analyses.vtable*), 845
VtableFinder (*class in angr.analyses.vtable*), 845

W

walk() (*angr.analyses.decompiler.ailgraph_walker.AILGraphWalker method*), 692
walk() (*angr.analyses.decompiler.optimization_passes.const_derefs.BlockWalker method*), 701
walk() (*angr.analyses.decompiler.region_walker.RegionWalker method*), 722
walk() (*angr.analyses.decompiler.sequence_walker.SequenceWalker method*), 723
walk_node() (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.BlockLocator method*), 706
walk_node() (*angr.analyses.decompiler.region_walker.RegionWalker method*), 722
whitelist (*angr.analyses.veritesting.CallTracingFilter attribute*), 838
widen() (*angr.sim_state.SimState method*), 227
widen() (*angr.SimFile method*), 191
widen() (*angr.SimFileDescriptor method*), 200
widen() (*angr.SimFileDescriptorDuplex method*), 202
widen() (*angr.SimHeapBrk method*), 206
widen() (*angr.SimHeapPTMalloc method*), 208
widen() (*angr.SimPackets method*), 194
widen() (*angr.SimState method*), 183
widen() (*angr.SimStatePlugin method*), 162
widen() (*angr.state_plugins.callstack.CallStack method*), 265
widen() (*angr.state_plugins.cgc.SimStateCGC method*), 273
widen() (*angr.state_plugins.concrete.Concrete method*), 293
widen() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 253
widen() (*angr.state_plugins.filesystem.SimFilesystem method*), 250
widen() (*angr.state_plugins.globals.SimStateGlobals method*), 278
widen() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 299
widen() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 306
widen() (*angr.state_plugins.history.SimStateHistory method*), 268
widen() (*angr.state_plugins.inspect.SimInspector method*), 235
widen() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 295
widen() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 297
widen() (*angr.state_plugins.libc.SimStateLibc method*), 239
widen() (*angr.state_plugins.log.SimStateLog method*), 263
widen() (*angr.state_plugins.loop_data.SimStateLoopData method*), 292
widen() (*angr.state_plugins.plugin.SimStatePlugin method*), 233
widen() (*angr.state_plugins.posix.PosixDevFS method*), 241
widen() (*angr.state_plugins.posix.PosixProcFS method*), 243
widen() (*angr.state_plugins.posix.SimSystemPosix method*), 247
widen() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 283
widen() (*angr.state_plugins.scratch.SimStateScratch method*), 281
widen() (*angr.state_plugins.sim_action_object.SimActionObject method*), 468
widen() (*angr.state_plugins.solver.SimSolver method*), 257
widen() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 275
widen() (*angr.state_plugins.trace_additions.ZenPlugin method*), 277
widen() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
widen() (*angr.state_plugins.view.SimMemView method*), 314
widen() (*angr.state_plugins.view.SimRegNameView method*), 310
widen() (*angr.storage.file.SimFile method*), 319
widen() (*angr.storage.file.SimFileDescriptor method*), 329
widen() (*angr.storage.file.SimFileDescriptorDuplex method*), 331
widen() (*angr.storage.file.SimPackets method*), 323
widen() (*angr.storage.file.SimPacketsSlots method*), 333
widen() (*angr.storage.memory_mixins.javavm_memory.javavm_memory_m method*), 378
widen() (*angr.storage.memory_mixins.MemoryMixin method*), 337
widen() (*angr.storage.memory_mixins.regioned_memory.region_meta_mix method*), 372
widened_jobs (*angr.analyses.forward_analysis.job_info.JobInfo property*), 622
width (*angr.analyses.data_dep.dep_nodes.MemDepNode property*), 868
width() (*angr.analyses.disassembly.DisassemblyPiece method*), 846
with_arch() (*angr.sim_type.SimType method*), 508
with_arch() (*angr.sim_type.TypeRef method*), 508
with_condition (*angr.sim_state.SimState property*),

228
with_condition (*angr.SimState property*), 184
with_type() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 308
with_type() (*angr.state_plugins.view.SimMemView method*), 313
work() (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis method*), 635
Worker (*class in angr.distributed.worker*), 901
WRITE (*angr.knowledge_plugins.variables.variable_access.VariableAccessSort attribute*), 561
Write (*angr.knowledge_plugins.xrefs.xref_types.XRefType attribute*), 611
WRITE (*angr.state_plugins.sim_action.SimActionData attribute*), 467
write() (*angr.SimFile method*), 190
write() (*angr.SimFileBase method*), 189
write() (*angr.SimFileStream method*), 195
write() (*angr.SimPackets method*), 193
write() (*angr.SimPacketsStream method*), 196
write() (*angr.storage.file.SimFile method*), 318
write() (*angr.storage.file.SimFileBase method*), 316
write() (*angr.storage.file.SimFileDescriptorBase method*), 325
write() (*angr.storage.file.SimFileStream method*), 320
write() (*angr.storage.file.SimPackets method*), 322
write() (*angr.storage.file.SimPacketsSlots method*), 332
write() (*angr.storage.file.SimPacketsStream method*), 324
write_data() (*angr.SimFileDescriptor method*), 198
write_data() (*angr.SimFileDescriptorDuplex method*), 201
write_data() (*angr.storage.file.SimFileDescriptor method*), 327
write_data() (*angr.storage.file.SimFileDescriptorBase method*), 326
write_data() (*angr.storage.file.SimFileDescriptorDuplex method*), 330
write_msr() (*angr.state_plugins.unicorn_engine.Unicorn method*), 290
write_pos (*angr.SimFileDescriptor property*), 199
write_pos (*angr.SimFileDescriptorDuplex property*), 201
write_pos (*angr.storage.file.SimFileDescriptor property*), 328
write_pos (*angr.storage.file.SimFileDescriptorBase property*), 326
write_pos (*angr.storage.file.SimFileDescriptorDuplex property*), 330
write_storage (*angr.SimFileDescriptor property*), 199
write_storage (*angr.SimFileDescriptorDuplex property*), 201
write_storage (*angr.storage.file.SimFileDescriptor property*), 328
write_storage (*angr.storage.file.SimFileDescriptorBase property*), 326
write_storage (*angr.storage.file.SimFileDescriptorDuplex property*), 330
write_to() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 563

X

X86ElfPicPltResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt*), 661
X86GccGetPcSimplifier (*class in angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_sim*), 711
X86PeIatResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat*), 659
Xor (*angr.engines.light.data.ArithmeticExpression attribute*), 750
XRef (*class in angr.knowledge_plugins.xrefs.xref*), 610
XRefManager (*class in angr.knowledge_plugins.xrefs.xref_manager*), 611
XRefs (*angr.analyses.analysis.KnownAnalysesPlugin attribute*), 620
xrefs (*angr.angrdb.models.DbKnowledgeBase attribute*), 676
xrefs (*angr.knowledge_base.knowledge_base.KnowledgeBase attribute*), 520
xrefs (*angr.knowledge_plugins.functions.function.Function property*), 554
xrefs (*angr.KnowledgeBase attribute*), 211
XRefsAnalysis (*class in angr.analyses.xrefs*), 862
XRefsSerializer (*class in angr.angrdb.serializers.xrefs*), 682
XRefType (*class in angr.knowledge_plugins.xrefs.xref_types*), 611

Z

zen_hook() (*in module angr.state_plugins.trace_additions*), 276
zen_memory_write() (*in module angr.state_plugins.trace_additions*), 276
zen_register_write() (*in module angr.state_plugins.trace_additions*), 276
ZenPlugin (*class in angr.state_plugins.trace_additions*), 276