Test Case Generation

As we saw with `symcc` and `klee`, a common use of Symbolic Execution is for generating test cases that are guaranteed to cover a path that branches off

We can generate branching inputs by solving for *negated* path constraints

Case Generation

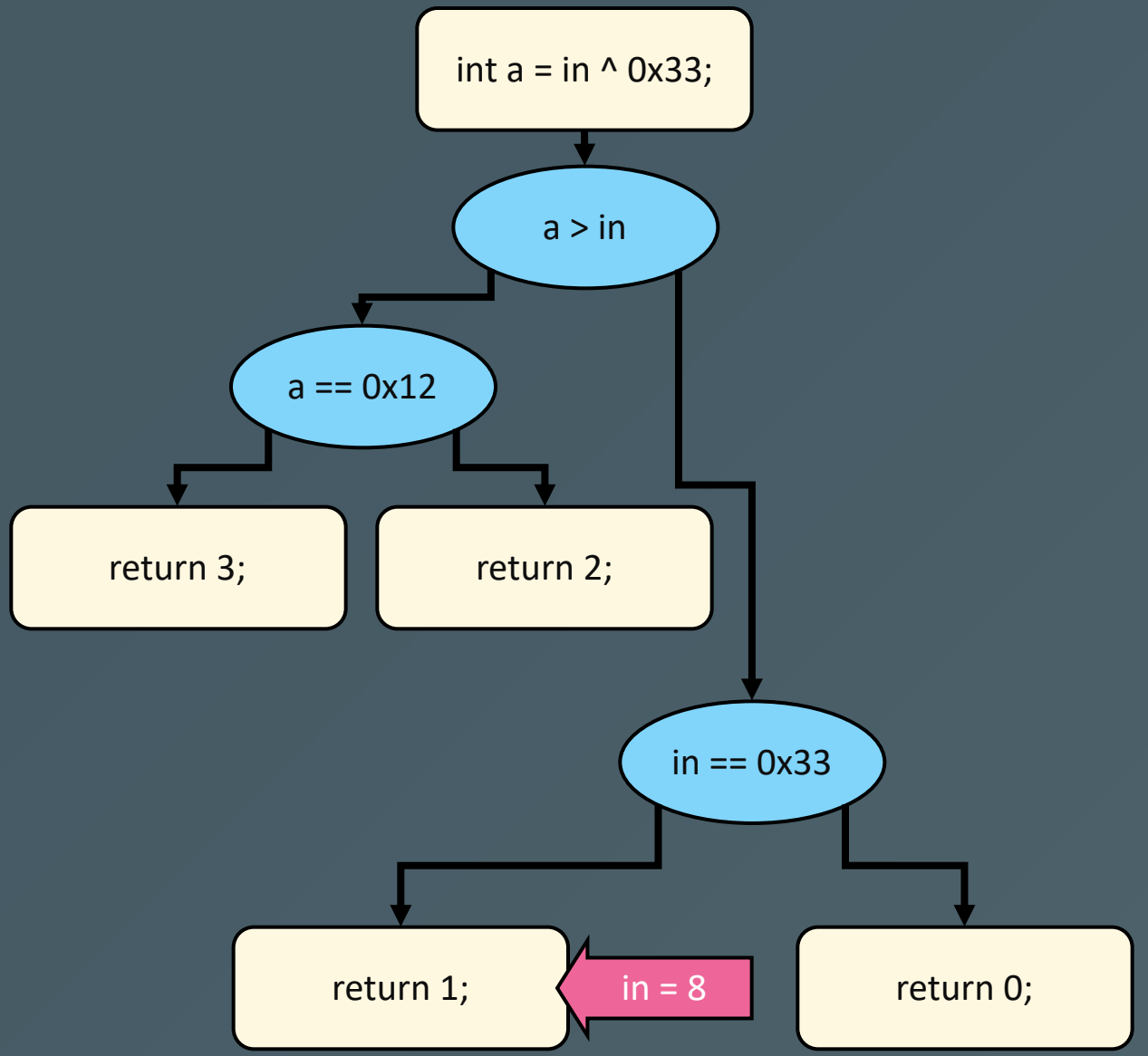
```
int func(int in) {
  int a = in ^ 0x33;

  if (a > in) {
    if (in == 0x33) {
      return 0;
    }

    return 1;
  }

  if (a == 0x12) {
    return 2
  }

  return 3;
}
```

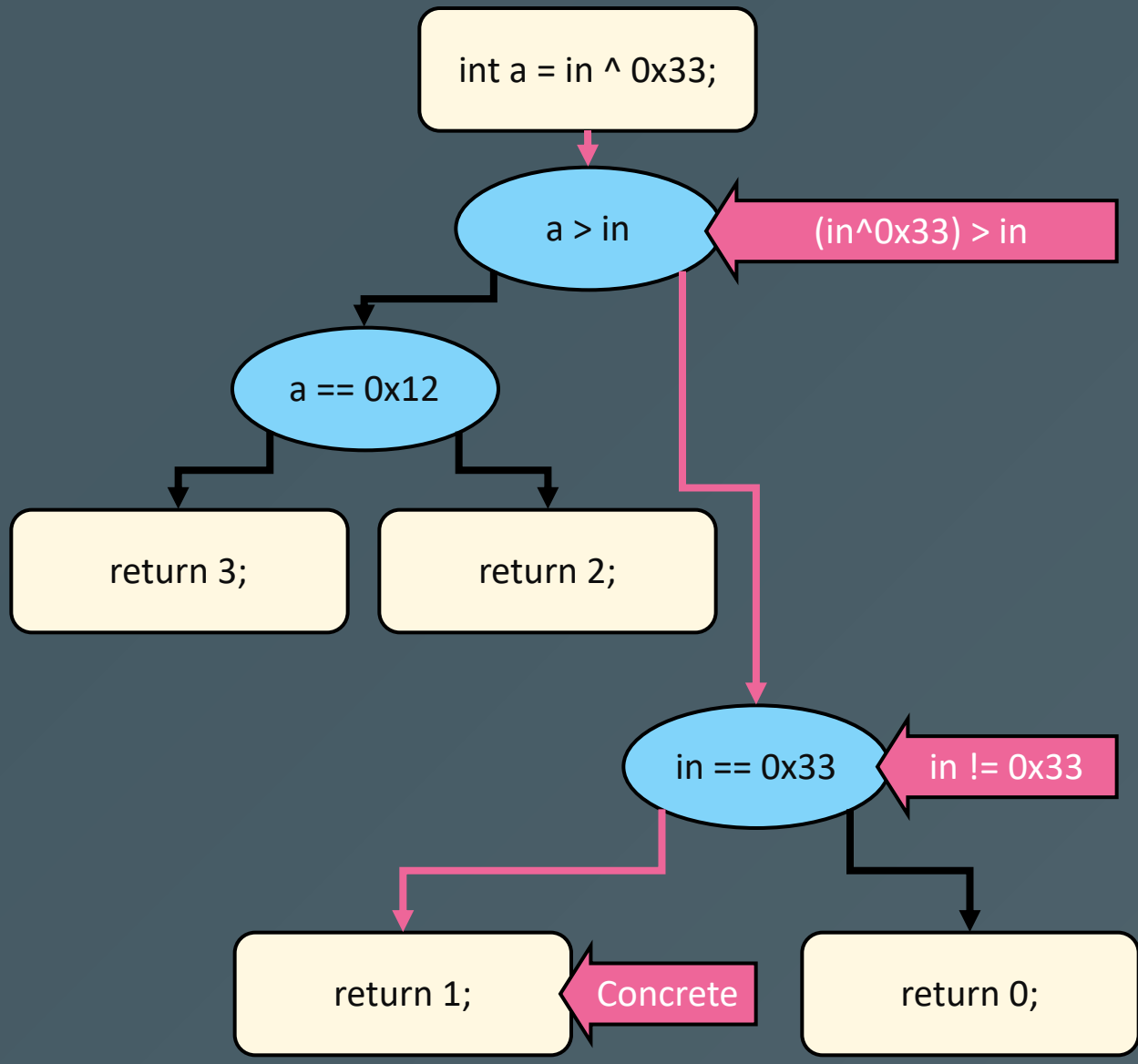


Case Generation

We can track the
symbolic constraints
along our concrete path

- Constraints on our symbolic variables to follow the path
- Not every branch adds a symbolic constraint

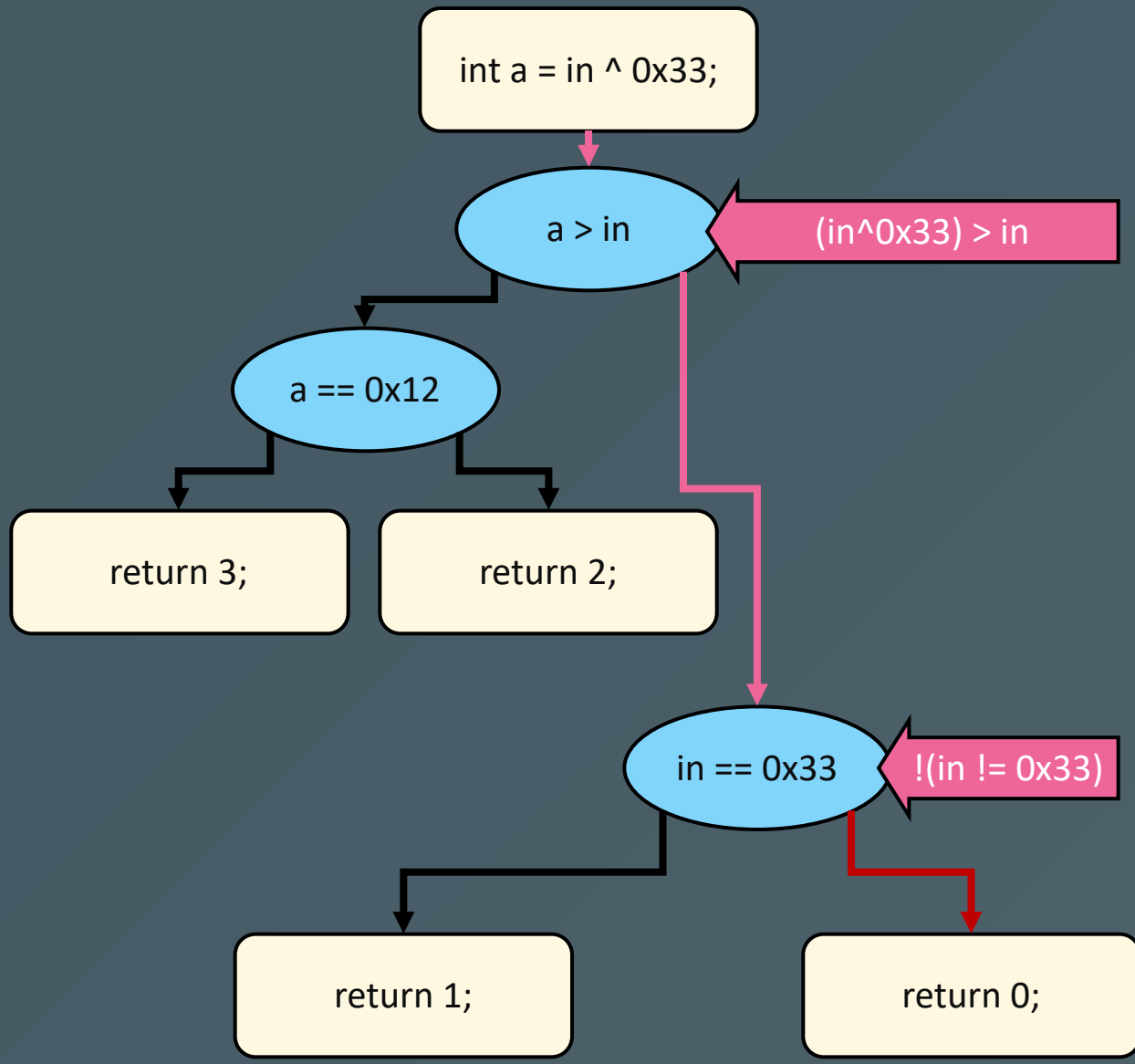
The constraints can be useful for Reverse Engineering



Case Generation

We can negate constraints to solve for inputs that would take our execution down other forks

```
i = Var(32, "in")
solver.add((i ^ 0x33) > i)
solver.add((i != 0x33).invert())
solver.check()
```

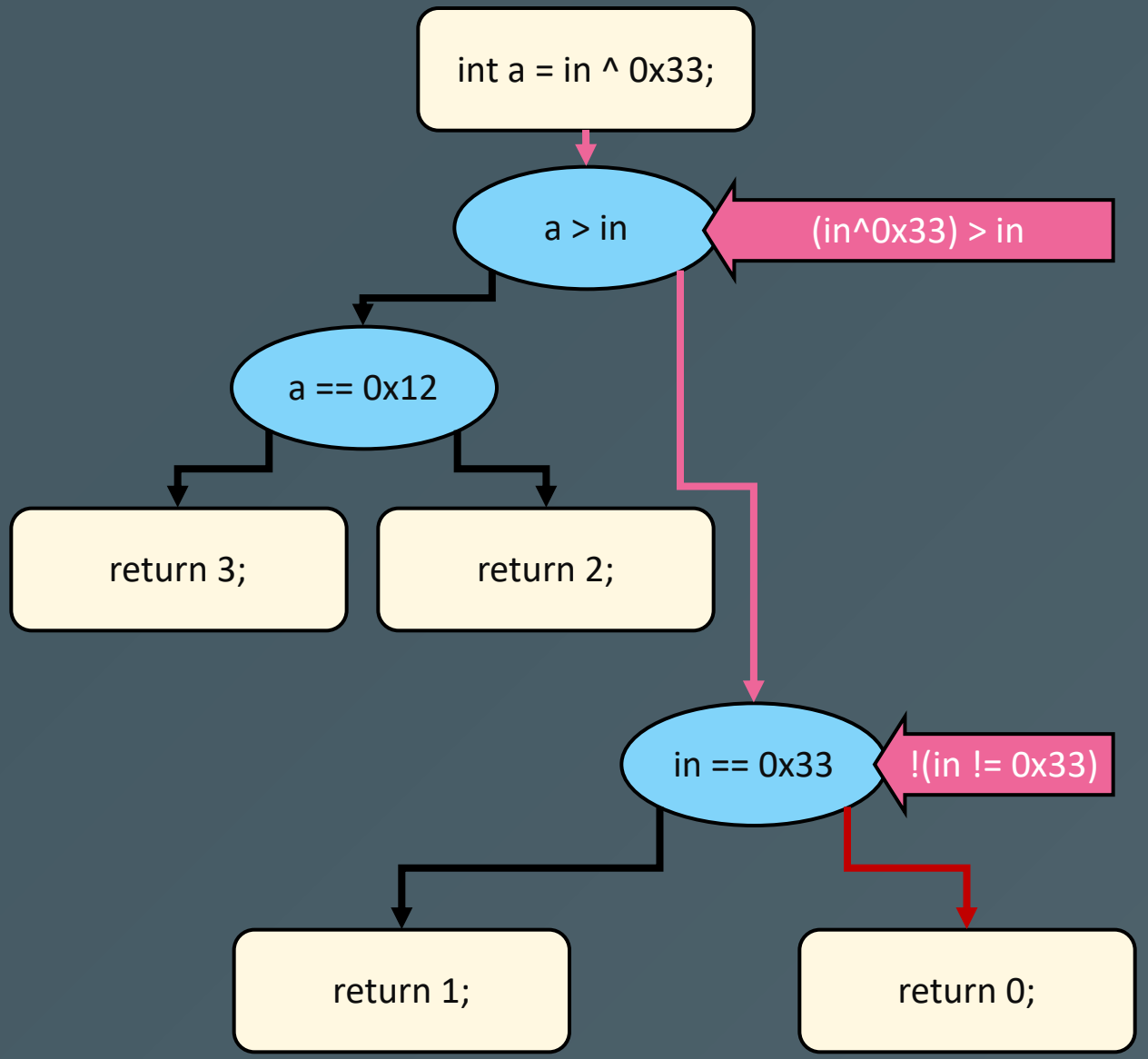


Case Generation

```
solver.check() # False
```

Sometime it is impossible to reach a path

- Dead code
- Depends on an input that was not made symbolic

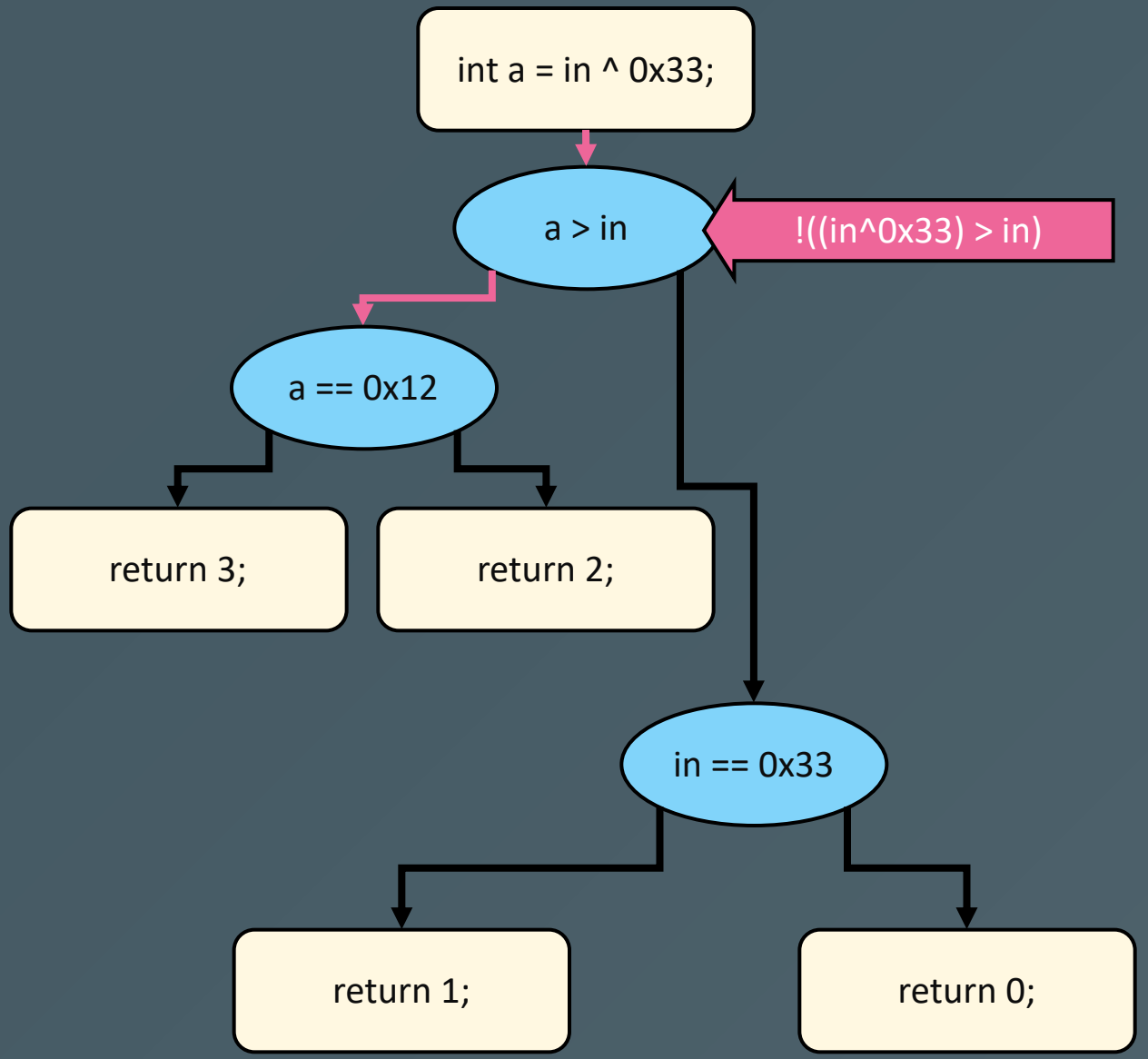


Case Generation

```
i = Var(32, "in")
solver.add(((i ^ 0x33) > i).invert())
solver.check()
solver.get_model()
```

We probably can't get full code coverage by generating test cases based on a single concrete path

The generated input here could either reach `return 3` or `return 2`



Case Generation -- MAAT example

```
eng.run()
cons = eng.path.constraints()

s = Solver()
# generate cases based on the each path branch
for bi in range(len(cons)):
  s.reset()
  # add all proceeding constraints
  for i in range(bi):
    s.add(cons[i])

  # invert final condition
  s.add(cons[bi].invert())

  if s.check(): # check if reachable
    # generate input given a model
    gen_input_from_model(s.get_model())
```

Case Generation -- MAAT hook example

```
def path_hook(eng): # a EVENT.PATH, WHEN.BEFORE hook
  s = Solver()

  # Add constraints to get up to this branch
  for c in eng.path.constraints():
    s.add(c)

  # add a constraint to take the other fork from this branch
  if eng.info.branch.taken:
    s.add(eng.info.branch.cond.invert())
  else:
    s.add(eng.info.branch.cond)

  if s.check(): # False means unreachable path
    gen_new_input(eng, s.get_model()) # given new vals for our vars, gen a new input case
```

Test Case Generation

DEMO: `testcase_demo`

Let's generate some branching inputs given an example input

Using `./testcase_demo/guessnum -1` generate alternate inputs
One of the alternate inputs should result in an output of `Win!`

LAB: Crashables

Given `./nearmiss/nearmiss < ./nearmiss/case0`, generate alternate input files

At least one of the alternate inputs should result in a segmentation fault

- *BONUS*
  - get the path constraints for the crashing inputs
    - is the crash exploitable?
  - Can you find other issues?