

Welcome back



In-Class Exercise: Reasoning About Mutants

- Today we'll learn how to use Z3, a formal theorem prover
- And we'll use it to help us create tests

Z3

- Online interface: <https://rise4fun.com/Z3>
- Tutorial: <https://rise4fun.com/Z3/tutorial/guide>
- In-class assignment:
<https://people.cs.umass.edu/~brun/class/2018Fall/CS520/in-class4.pdf>

Z3's language

- Z3 uses a kind of programming language
- Can declare variables and functions, define constraints, print things to the screen, etc.

Z3's language

```

1 (echo "starting Z3...")
2 (declare-const a Int)
3 (declare-fun f (Int Bool) Int)
4 (assert (> a 10))
5 (assert (< (f a true) 100))
6 (check-sat)

```

This code prints "starting Z3..." to the screen, declares a constant **a**
 declares a function **Int f (Int Bool)**
 makes 2 assertions: **a > 10** and **f(a, true) < 100**
 asks "is this possible?"

Encoding programs in constraints

Given a program **P** and a question about **P**, encode them into constraints and ask Z3 to answer the question!

```

int P(int a, int b) {
    return a + b;
P:
}

```

Question: Can **P** ever return 0?

```

1 (declare-const a Int)
2 (declare-const b Int)
3 (assert (= (+ a b) 0)) ; We want a + b to be 0
4 (check-sat) ; Find out if this is satisfiable
5 (get-model) ; It is, so let's get a satisfying model

```

Modeling Control Flow

```
int doesStuff(int a, int b, int c){
    if (c == 0) return 0;
    if (c == 4) return 0;
    if (a + b < c) return 1;
    if (a + b > c) return 2;
    if (a * b == c) return 3; // Does this ever happen??
    return 4;
}
```

To ask if `doesStuff` ever returns 3, encode:

$$\begin{array}{lll} !(c == 0) & !(c == 4) & !(a + b < c) \\ !(a + b > c) & (a * b == c) & \end{array}$$

Modeling Control Flow

```
int doesStuff(int a, int b, int c){
    if (c == 0) return 0;
    if (c == 4) return 0;
    if (a + b < c) return 1;
    if (a + b > c) return 2;
    if (a * b == c) return 3; // Does this ever happen??
    return 4;
}
1 (define-sort JInt () (BitVec 32))
2 (declare-const a JInt)
3 (declare-const b JInt)
4 (declare-const c JInt)
5
6 (assert (not (= c #x00000000)))
7 (assert (not (= c #x00000004)))
8 (assert (not (bvslt (bvadd a b) c)))
9 (assert (not (bvsgt (bvadd a b) c)))
10 (assert (= (bvmul a b) c))
11
12 (check-sat)
13 (get-model)
```

Z3 for Mutation Testing

```
int normal_sum(int a, int b) {
    return a + b;
}

int mutant_sum(int a, int b) {
    return a * b;
}

1 (declare-const a Int)
2 (declare-const b Int)
3 (assert (= (+ a b) (* a b)))
4 (check-sat)
5 (get-model)
```

We have to frame the question in terms of
“Does there exist an input such that...”

- If two functions are identical, then for all inputs, they act the same.
- We can ask if two functions are **NOT** identical.

“Does there exist an input for which they differ?”

```
1 (declare-const a Int)
2 (declare-const b Int)
3 (assert (not (= (+ a b) (* a b))))
4 (check-sat)
5 (get-model)
```

Now, you drive!

- In-class assignment: <https://people.cs.umass.edu/~brun/class/2018Fall/CS520/in-class4.pdf>
- Online Z3 interface: <https://rise4fun.com/Z3>
- Tutorial: <https://rise4fun.com/Z3/tutorial/guide>