CS 520
Extra Credit
Formal Reasoning

High-level goal

The high-level goal of this exercise is to learn to use bit vectors with theorem provers to reason about code and mutants.

Submission

This extra credit exercise is an extension of the in-class 4 exercise. This exercise is a group submission. This means that each group only needs to submit their solution once and also that every student in a group will get the same grade. The deadline for submitting the solution is **Friday, Dec 15, 2017, 11:55 PM EST**. Submit via Moodle.

Grading

Each solution will be graded on a scale of $\{0, 1, 2\}$.

Forming groups

Team up in groups of size 2–4 and create a new group on Moodle (Extra credit exercise : group selection), and add all group members.

Background

Do you recall in-class 4, in which we used Z3 to find test cases that differentiate mutants? This exercise will build on that one, asking you to differentiate one more pair of mutants. You may refer to that exercise if you need to be reminded about how to use Z3: Z3.

But first, we need to introduce bit vectors.

Bit Vectors

Z3 models `int`s as unbounded integers. This means that computer arithmetic works differently than Z3 Integer arithmetic. For example, let's ask Z3 if there is a biggest number.

```
1 (declare-const a Int)
2 (assert (forall ((x Int)) (>= a x)))
3 (check-sat)
```

to which Z3 replies '`unsat`'.

So how do we model computer arithmetic? We use bit vectors.

We can model signed 32-bit `int` here. Note that we have replaced `Int` with `(_ BitVec 32)` and the `>=` operator with the ugly looking `bvsge` which stands for Bit Vector Greater than or Equal To.

```
(declare-const my-java-int (_ BitVec 32))
(assert (forall ((x (_ BitVec 32))) (bvsge my-java-int x)))
(check-sat)
(get-model)
```

Z3 happily returns

```
sat
(model
  (define-fun my-byte () (_ BitVec 32)
    #x7fffffff)
)
```

Great! We have just proven that there is a largest Java `int` and it has hex value `0x7fffffff`!

You can read more about bit vectors here: https://rise4fun.com/Z3/tutorialcontent/guide#h25

Modeling Control Flow

Now let's recall an example of how ot model control flow from in-class exercise 4. Say we have the following Java method and we want to know if it can ever return 3.

```
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;
}
```

Z3 can do this for us, no problem! Here are the constraints.

```
(define-sort JInt () (_ BitVec 32))   ; For convenience we alias the 32 bit vector
(declare-const a JInt)
(declare-const b JInt)
(declare-const c JInt)

(assert (not (= c #x00000000)))
(assert (not (= c #x00000004)))
(assert (not (bvslt (bvadd a b) c)))
(assert (not (bvsgt (bvadd a b) c)))
(assert (= (bvmul a b) c))

(check-sat)
(get-model)
```

So why does this ask Z3 if our method ever returns 3? Well, returning 3 is equivalent to saying that none of the other if statement conditionals were `true` but that the if-statement conditional `a*b == c` *was* `true`. Thus we assert that `c != 0`, `c != 4`, that `a + b > c`, that `a + b < c`, and finally that `a * b == c`.

Cool, let's see Z3's answer:

```
sat
(model
  (define-fun c () (_ BitVec 32)
    #x2da77000)
  (define-fun a () (_ BitVec 32)
    #x252cc8c0)
  (define-fun b () (_ BitVec 32)
    #x087aa740)
)
```

While this is a valid answer with Java `int`s, it is actually an overflow error. To verify this, try the same Z3 script with all instances of JInts replaced with Ints (and replace the operators as well). This will return `unknown` which means that Z3 can't find an answer.

Questions

Download one pair of programs:
https://people.cs.umass.edu/~rjust/courses/2017Fall/CS520/extraCredit/programs.zip

For the pair, you'll find three files:

1. A program (`Triangle.java`).
2. A mutant (`Mutant.java`). Note that javac won't compile this class cause of the filename.
3. A starter script with helpful Z3 commands encoding constraints you'll need (`Z3startercode.smt2`). Look for the

   ```
   ;;;;;;;;;;;;;;;;; START STUDENT CODE ;;;;;;;;;;;;;;;
   ;;;;;;;;;;;;;;;;; END STUDENT CODE   ;;;;;;;;;;;;;;;
   ```

   tags for where to write your code. You won't have to edit anything outside of these tags, except possibly uncommenting the penultimate `;; (get-model)` line.

For the pair, your job is to determine if there exists a test that produces different outputs on the two programs in the pair (and, if so, what that test is), or if the programs are equivalent (and thus no such test exists). Start with `Z3startercode.smt2` and fill in the space that's labeled for you to fill in.

Deliverables

Submit a single archive with 2 files with your solutions:

1. `Z3startercode.smt2`
2. `writeup.txt`

The `Z3startercode.smt2` file should contain your Z3 code solution. Verify that pasting the **entire file** into the Z3 interface produces the output you expect.

The `writeup.txt` file should contain an explanation of either how you know the mutants are equivalent, or a test case that demonstrates that they are not equivalent.