Assignment 4 — Symbolic execution

Due Date — 1PM on 7 May 2013

1 Goal

Understand and implement forward symbolic execution.

2 Requirements

Write a program `fse` that is passed a Java classfile and it generates inputs using symbolic execution of one public method in the class.

3 Details

This is standard symbolic execution using interpretation of the Java byte-code. No instrumentation is involved.

Assume that the passed Java classfile contains only a single public method with an arbitrary number of `int` parameters. These parameters are your symbolic variables.

Assume that the only method will *not* make any function calls, will be a closed deterministic system (i.e. only depends on input parameters. Only `int` variables may be created or used or returned from the function.

Assume that the only method will be using assignment, addition, subtraction, and comparison operators.

Assume that there are no loops. This means that you will always be able to completely explore the state space. You are free to choose a strategy but the easiest is depth-first search which can be implemented recursively in your interpreter.

4 Example

```
public class JavaClass {
  public int JavaMethod(int x, int y, int z) {
    int xy, xz, yz;
    xy=x+y;
    xz=x+z;
    yz=y+z;
    if (x>yz) {
      y=x;
      if (xy>z) {
        return 0;
      }
      return 1;
    }
    if (y>xz) {
      y=x;
      if (xy>z) {
        return 0;
      }
      return 1;
    }
    if (z>xy) {
```

```
      x=z;
      if (xz>y) {
        return 0;
      }
      return 1;
    }
  }
}
```

Your should output the number of paths covered and the path conditions in postfix notation. The output on your program should look like this:

```
path conditions for 6 covered paths:
(and (> x (+ y z)) (> (+ x x) z)
(and (> x (+ y z)) (not (> (+ x x) z))
(and (> y (+ x z)) (> (+ x x) z)
(and (> y (+ x z)) (not (> (+ x x) z))
(and (> z (+ x y)) (> (+ z z) y)
(and (> z (+ x y)) (not (> (+ z z) y))
```

5 Bonus

Integrate with yices to solve the path conditions and give inputs for the feasible paths. One possible output (the solver could give different correct values for the same path condition) for the above program could look like this. This is in addition to the above output.

```
inputs for feasible paths:
(= x 1) (= y 0) (= z 0)
...remaining 5 entries...
all 6 paths feasible, none infeasible.
```

The easiest approach to integrate with yices is to generate an intermediate file for every path condition in a format that is directly consumed by yices. Then execute the solver as an external process on that file and parse the output produced to extract necessary information.

For example, assume the file `test.in` contains:

```
(define x::int)
(define y::int)
(define z::int)
(assert (and (> x (+ y z)) (> (+ x y) z)))
(check)
```

Running `yices -e test.in` gives the following output

```
sat
(= x 1)
(= y 0)
(= z 0)
```

If it is unsatisfiable the output is `unsat` followed possibly by some error.

Good luck!