Dear students,

Below are the most frequently asked questions of this week's lab (week 3):

Question 1: Which functions are we supposed to implement?

Answer:

To get symbolic execution working for Lab 2, you must first translate all the expressions found in the actual problem file into the symbolic expressions that Z3 can use. This is done in the following methods: *createInput, createBoolExpr, createIntExpr, createStringExpr,* and *assign*.

Once you have implemented these methods, you can implement the main functionality of calling the solver in *encounteredNewBranch*. Then you can proceed to *newSatisfiableInput* and implement your logic for dealing with the traces returned by the solver.

Question 2: Does the solver return on a single input or multiple inputs?

Answer:

The solver always returns a trace that can be used to trigger a particular branch; the opposite branch that you are trying to solve within the *encounteredNewBranch*. Running this trace should trigger the branch.

Tip 1: The solver wraps each individual of the input within a quote (“), make sure you escape these characters as they can cause strings comparison to fail within the if-statement (even though it is the right character)

Tip 2: Add one or multiple extra symbols into the returned trace for exploration; the trace would only bring you to the branch that you called the solver for but there could be more branches after it.

Question 3: Where can we find the documentation of the Z3 API?

Answer:

This can be found at:
https://z3prover.github.io/api/html/namespacecom_1_1microsoft_1_1z3.html

Question 4: Why do we need to keep track of assignments in the path constraints?

Answer:

This is because variables could be assigned with a different value during a run of a trace. Tracking the assignments in the path constraint makes sure that the solution found by the solver is only possible following the sequence of variable assignments that occurred before reaching the branch that you want to solve (making the solution stricter).

Question 5: What is the *createInput* method used for?

Answer:

This method is used to create symbolic mapping for the input of the program. Once the solver has found a solution, it will assign the concrete values to the input variable and thus creating the trace for triggering the targeted branch.

Question 5: Do we need to use the *fuzz* method?

Answer:

It is not required, but you can use it for your symbolic fuzzer. As an example, you can use it to select the next trace (one that was found by the solver) to run. This example does assume that you are using some kind of a (priority) queue to select which trace to run. Additionally, you can also add extra character(s) to the trace that you want to run next (this refers to Tip 2 listed in the answer to question 2).

Question 6: I get the following error when trying to run the second lab, what am I doing wrong?

```
java.util.concurrent.ExecutionException: java.lang.NoClassDefFoundError: Errors
        at java.util.concurrent.FutureTask.report(FutureTask.java:122)
        at java.util.concurrent.FutureTask.get(FutureTask.java:192)
                                                                                at
nl.tudelft.instrumentation.symbolic.PathTracker.runNextFuzzedSequence(PathTracker.java:270)
                                                                                at
nl.tudelft.instrumentation.symbolic.SymbolicExecutionLab.run(SymbolicExecutionLab.java:116)
        at nl.tudelft.instrumentation.symbolic.PathTracker.run(PathTracker.java:252)
        at Problem11.main(Problem11.java:1598)
Caused by: java.lang.NoClassDefFoundError: Errors.....
```

Answer:

The instructions in the README for running the second lab are missing the current directory in the classpath. You can use the following command to run the problems for this lab.

```
java -cp target/aistr.jar:lib/com.microsoft.z3.jar:./instrumented:. Problem1
```