Homework 7: Symbolic and Concolic Execution

17-355/17-665/17-819: Program Analysis
Rohan Padhye

Due: Tuesday, April 5 11:59 pm EST

100 points total

Assignment Objectives:

- Demonstrate understanding of specifications for verifying programs.
- Reason about satisfiability and EUF theory and implement a solution in SMT-LIB format.
- Understand Concolic Execution and implement constraint-collecting logic on behalf of a concolic execution engine

Setup. Clone the starter repository here: `https://classroom.github.com/a/hJ_Zq_kL`

Handin Instructions. Please submit your assignment via Gradescope by pointing it to your GitHub repository by the due date. You should replace `written-answers.pdf` with your written answers to question 1 and update `signed.py`, `f1.py`, `f2.py`, and `sum.py` to complete question 2.

Note: *We will not look at your homework repository directly, but will only see what you have submitted to Gradescope!* ***Make sure that you (re)submit after you have completed all parts; Gradescope does not automatically pull new commits from GitHub.***

Question 1, SMT with EUF, *(25 points).*
a) (10 points) Show, using the congruence closure, whether the following Equality Logic with Uninterpreted Functions (EUF) formula is satisfiable. Show each step merging equivalent terms.

$$f(g(0)) = g(f(0)) \wedge f(g(f(y))) = 0 \wedge f(y) = 0 \wedge g(f(0)) \neq 0$$

b) (15 points) Give the SMT-LIB formula (i.e., a valid Z3 program) to prove your answer to (a) is correct. Put your code in the answer, and show the output of Z3.

Question 2, Concolic Execution, *(75 points).*

For this task, you will concolically execute the following programs provided in the repository

- `signed.py` *(0 points)*
- `f1.py` *(25 points)*
- `f2.py` *(25 points)*
- `sum.py` *(25 points)*

The starter repository already provides a full concolic execution driver in `concolic.py`, which executes a subject program, gets back constraints, invokes a SAT/SMT solver, figures out the next path to execute, and keeps doing this until all feasible paths are exhausted. However, the repository does not have an instrumentation engine that collects constraints for subject programs.

Your task will be to simulate the instrumentation engine by hard-coding logic within the subject programs to collect path constraints on behalf of the concolic execution engine. Once you add the proper constraint-collecting logic, the engine will be able to successfully execute all paths and report bugs if any are found.

To complete and submit this task, please refer to the **Concolic Execution** section of the README in the starter repository for detailed instructions on how to setup and use the provided code, how to modify the appropriate files with your implementation, and how to test and prepare your implementation for submission.