Homework 8: Verification, SMT, and Synthesis

17-355/17-665/17-819: Program Analysis
Jonathan Aldrich & Rohan Padhye

Due: Thursday, April 22, 2021 11:59 pm EDT

100 points total, plus 25 points of the project

Assignment Objectives:

- Propose your project.
- Demonstrate understanding of specifications for verifying programs.
- Reason about satisfiability and EUF theory and implement a solution in SMT-LIB format.
- Develop an SMT formulation and implement the solution for an Inductive Synthesis problem.

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

Handin Instructions. Please submit your assignment via Gradescope by pointing it to your GitHub repository by the due date. You should replace `project.pdf` with your project proposal, replace `answers.pdf` with your written answers to questions 2 and 3 and update the `problem.py` file to complete the coding for question 3.

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, Project Proposal, *(25 project points).* Consult the project handout and complete the project proposal component. You should replace the `project.pdf` file with your project proposal.

Question 2, SMT with EUF, *(25 points).*
a) (15 points) Show, using the congruence closure, whether the following Equality Logic with Uninterpreted Functions (EUF) formula is satisfiable. Show each step merging equivalent terms.

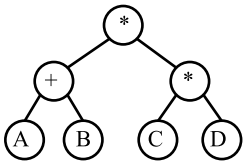
$$f(g(0)) = g(f(0)) \wedge f(g(f(y))) = 0 \wedge f(y) = 0 \wedge g(f(0)) \neq 0$$

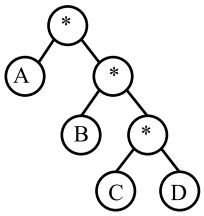
b) (10 points) Give the SMT-LIB formula (i.e., a valid Z3 program that you can run online at `https://rise4fun.com/z3`) to prove your answer to (a) is correct. Put your code in the answer, and show the output of Z3.

Question 3, Inductive Synthesis, *(75 points).*

This task asks you to write a simple synthesizer for an expression that satisfies a collection of input/output pairs. The instruction set consists of only four binary operators for bitvectors: `multiply`, `add`, `left bit shift`, and `bitwise or`. Your expression has four inputs `A`, `B`, `C`, `D`. Your task is to discover the right operators using the inputs such that it evaluates to the output.

Example. Suppose we have the input A = 2, B = 2, C = 1, D = 1, and the output 4. Two solutions are possible, and represented with different trees ((A + B) * (C * D)) and (A * (B * (C * D))):





More generally, we want to discover the operators and expression tree such that the inputs evaluate to the output.

a) (25 points) Develop a strategy to encode the general problem with SMT. Describe your approach in `answers.pdf` in your repository.

b) (50 points) Implement your solution (using your SMT formulation) to synthesize the expression satisfying the input/output pairs using Z3. The input/output pairs are provided in the starter repository. Inputs correspond to the format (A, B, C, D).

You are free to use any programming language that interfaces with Z3. APIs exist for C, Java, Python, OCaml, etc. See here. We provide starter code using Python in the provided repository.

Description. You may assume that each input is used only *once* in the expression, and there are four inputs. I.e., the solution is guaranteed to use each of the inputs A, B, C, and D, and will only use each of these values once. All values (inputs and outputs) are 16-bits wide. Note that inputs can occur in any order (`B << A` is a possible subexpression).

Test. Check your answer from your Z3 solution by making the `test.py` file (in the same directory) pass. That is: translate your synthesized expression into Python and add it to the test function. If you choose not to use Python, write a function like `test.py` that tests your answer against all the input/output pairs, and include instructions to run your solution in your `answers.pdf` writeup.

Guidelines. Permutations of binary trees correspond to the set of possible expression trees that can be generated. Leaf nodes correspond to inputs, and we know there are four leaf nodes since we have four inputs. Non-leaf nodes correspond to operators. Can you think of a way to encode permutations of binary trees with the operators in a way that Z3 can tell you which combination of operators satisfy the output?

Consider using boolean variables that correspond to operators which toggle whether an operator is used or not in an expression. If you set up the constraints the right way, Z3 can tell you which operators should be used to satisfy the solution (toggled "on") and which ones should be left out. This way, the Z3 solution consists of flags that correspond to the right operator for satisfying the input/output relation.