Homework 8: Verification, SMT, and Synthesis

17-355/17-665/17-819O: Program Analysis
Claire Le Goues and Jonathan Aldrich
clegoues@cs.cmu.edu, aldrich@cs.cmu.edu

Due: Thursday, April 5 11:59 pm

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.
- Develop an SMT formulation and implement the solution for an Inductive Synthesis problem.

Handin Instructions. Please submit the written assignment (**Q1** and **Q2**) on Canvas as a **PDF** by the due date. Name it **[your-andrew-id]-hw8.pdf**. Submit your solution to **Q3** in a folder called **hw8** in your GitHub repository.

Question 1, Verifying with Dafny, *(30 points).* Dafny is a programming language with built-in specification constructs. For example, Dafny lets you specify pre- and post conditions on methods, and will verify that your code meets the specification. Underneath the hood, Dafny discharges SMT formulas based on the program and specifications, and validates correctness using, e.g., Z3. The online tutorial for Dafny is a good resource for examples and getting started: https://rise4fun.com/Dafny/tutorial/Guide. *Note, however, that you do not need to write or understand much Dafny to complete this question, which primarily concerns specification/verification.*

Consider the BubbleSort program written in Dafny at https://rise4fun.com/Dafny/1xSS. By writing specifications in Dafny, we can verify the correctness of bubble sort (i.e., that it always returns a sorted list). Take some time to understand the program and the existing specifications, then answer the following questions.

a) (5 points) The predicate `sorted` is incomplete. What should be substituted for `__FIXME__` on line 5?

b) (10 points) After adding the condition for part *a)*, run Dafny again. Dafny still unable to prove the program correct due to a loop invariant. It gives two errors: `This loop invariant might not hold on entry` and `This loop invariant might not be maintained by the loop`.

Correct the reported loop invariant so that Dafny no longer reports the case where the `loop invariant might not be maintained by the loop`. Write out the code/invariant you changed in your assignment, and explain in prose why the original loop invariant was insufficient.

c) (15 points) After fixing the loop invariant in part *b)*, Dafny still reports that the correct loop invariant `might not hold on entry`. Explain in prose why this is the case.

Dafny will verify the complete implementation with some changes that deal with the condition on loop entry. One way is to add an additional invariant. Another way is to change the program so that Dafny infers stronger conditions on variable(s).

Either add a single invariant *or* make a small change the program so that Dafny verifies the program. Rerun Dafny and confirm that it verifies the program with no warnings. Describe the change you made.

Question 2, SMT with EUF, *(20 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.

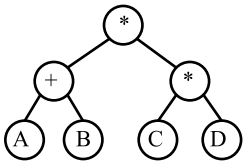
$$f(g(0)) = g(f(0)) \wedge f(g(f(y))) = 0 \wedge f(y) = 0 \wedge g(f(0)) \neq 0$$

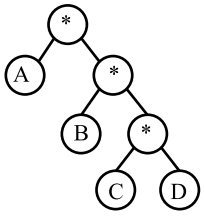
b) (10 points) Give the SMT-LIB formula (i.e., a valid Z3 program that you can run online at `https://rise4fun.com/z3`) to prove your answer to (a) is correct. Put your code in the answer, and show the output of Z3.

Question 3, Inductive Synthesis, *(50 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) (15 points) Develop a strategy to encode the general problem with SMT. Describe your approach in a `README.md` in the `hw8` folder in your repository.

b) (35 points) Implement your solution (using your SMT formulation) to synthesize the expression satisfying the input/output pairs using Z3. The input/output pairs is available online, and also in the starter code (see **Setup**, next). Inputs correspond to the format (A, B, C, D).

Setup. 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 online. Please refer to instructions in the recitation for Z3 (here) to set up Z3 with the Python API.

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 `README.md`.

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.