Homework 7: Symbolic and Concolic Execution

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

Due: Tuesday, April 13 11:59 pm EST

100 points total

Assignment Objectives:

- Demonstrate understanding of specifications for verifying programs.
- 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/WacxDnCJ`

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, 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 bubble sort program written in Dafny in Figure 1 (also at `https://rise4fun.com/Dafny/1xSS` and in the `bubblesort.dfy` file in the homework repository). 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.

```
0  predicate sorted(a: array?<int>, l: int, u: int)
1    reads a
2    requires a ≠ null
3  {
4    ∀ i, j • 0 ≤ l ≤ i ≤ j ≤ u < a.Length ⟹  __FIXME__
5  }
6
7  predicate partitioned(a: array?<int>, i: int)
8    reads a
9    requires a ≠ null
10 {
11   ∀ k, k' • 0 ≤ k ≤ i < k' < a.Length ⟹ a[k] ≤ a[k']
12 }
13
14 method BubbleSort(a: array?<int>)
15   modifies a
16   requires a ≠ null
17   ensures sorted(a, 0, a.Length-1)
18 {
19   var i := a.Length - 1;
20   while(i > 0)
21     invariant 0 < i < a.Length
22     invariant sorted(a, i, a.Length-1)
23     invariant partitioned(a, i)
24   {
25     var j := 0;
26     while (j < i)
27       invariant 0 < i < a.Length ∧ 0 ≤ j ≤ i
28       invariant sorted(a, i, a.Length-1)
29       invariant partitioned(a, i)
30       invariant ∀ k • 0 ≤ k ≤ j ⟹ a[k] ≤ a[j]
31     {
32       if(a[j] > a[j+1])
33       {
34         a[j], a[j+1] := a[j+1], a[j];
35       }
36       j := j + 1;
37     }
38     i := i -1;
39   }
40 }
41
42 method Main() {
43   var a := new int[5];
44   a[0], a[1], a[2], a[3], a[4] := 9, 4, 6, 3, 8;
45   BubbleSort(a);
46   var k := 0;
47   while(k < 5) { print a[k], "\n"; k := k + 1; }
48 }
```

Figure 1: Incomplete Dafny bubble sort.

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, Concolic Execution, *(70 points).*

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

- `signed.py` *(5 points)*
- `f1.py` *(20 points)*
- `f2.py` *(20 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.