ECE653
Software Testing, Quality Assurance, and Maintenance
Assignment 3 (70 Points), Version 1

Instructor: Arie Gurfinkel
Release Date: Nov 13, 2023

Due: 10:00 PM, Dec 5, 2023
Submit: An electronic copy on GitLab

Any source code and test cases for the assignment will be released in the skeleton repository at `https://git.uwaterloo.ca/stqam-1239/skeleton`.

I expect each of you to do the assignment independently. I will follow UW's Policy 71 for all cases of plagiarism.

Submission Instructions:

Please read the following instructions carefully. **If you do not follow the instructions, you may be penalized up to 5 points.** Illegible answers receive no points.

Submit by pushing your changes to the `main` branch of your GitLab repository in directory `a3`. Make sure to use a web browser to check that your changes have been committed! The submission must contain the following:

- a `user.yml` file with your UWaterloo user information;
- a single pdf file called `a3_sub.pdf`. The first page must include your full name, 8-digit student number and your uwaterloo email address;
- a directory `dafny` that includes your code for Questions 2 and 3; and
- a directory `wlang` that includes your code for Question 4.

Question 1 (5 points)

The following program computes a sum of telescoping series[1] of powers of two in a variable r:

```
1  assume (n >= 0);
2  r := 0;
3  i := 0;
4  p := 1;
5  while not (i = n) do
6  {
7    r = r - p;
8    p = 2 * p;
9    r = r + p;
10   i = i + 1;
11 }
```

Show that the program is correct using the rules of Hoare logic and the fact that

$$\sum_{i=1}^{n}(2^i - 2^{i-1}) = 2^n - 2^0$$

That is, let P denote the statements in lines 5–10. Construct a derivation to show validity of the following Hoare triple:

$$\{n \geq 0 \wedge r = 0 \wedge i = 0 \wedge p = 1\}\ P\ \{r = 2^n - 1\}$$

Hint: you can use $p = 2^i \wedge r = 2^i - 1 \wedge i \leq n$ as an inductive invariant for the while loop.

You can use the combined rule of assignment and consequence to shorten the proof

$$\frac{P \implies Q[e/x]}{\{P\}\ x := e\ \{Q\}}\text{ASGN} + \text{CONS}$$

Recall that the notation $Q[e/x]$ stands for "an expression Q in which e replaces all occurrences of x".

[1] https://en.wikipedia.org/wiki/Telescoping_series.

Question 2 (10 points)

For this question, you will have to verify two small programs in Dafny. You can install Dafny locally on your machine following the instructions in the lecture notes.

All files for this questions are contained in `dafny` folder of the repository.

(a) **`CalcTerm.dfy`**: The file contains a method

```
CalcTerm(m:int, n:nat) returns (res:int)
```

The method computes $5 \times m - 3 \times n$ using a loop and operations to add 3, and subtract 1 and 2.

Provide the missing invariants and decreases annotations so that the method verifies.

(b) **`SlowMax.dfy`**: The file contains a method

```
slow_max(a:nat, b:nat) returns (z:nat)
```

that computes $z = \max(a, b)$ via a series of increments and decrements.

Provide the missing invariants and decreases annotations so that the method verifies.

Question 3 (30 Points)

For this question, you will have to verify two implementation of sorting methods in Dafny. You can install Dafny locally on your machine following the instructions in the lecture notes.

All files for this questions are contained in `dafny` folder of the repository.

(a) Pancake sorting[2] is a sorting technique that uses only flip operation in which elements of an array are reversed (flipped) at some position. The directory `pancakesort` contains an implementation of pancake sorting algorithm in Dafny.

- Specify and verify method `findMax` in `findmax.dfy`
- Specify and verify method `flip` in `flip.dfy`
- Specify and verify method `pancakeSort` in `PancakeSort.dfy`

(b) Quicksort[3] is an efficient sorting technique developed by Tony Hoare. The directory `quicksort` contains an implementation of quick sort in Dafny.

- Write a loop invariant to complete the verification of `partition` method in `part.dfy`
- *(Bonus)* Complete and verify the implementation of `qsort` method in `QuickSort.dfy`. You might find Chapter 6 of Calculus of Computation useful for this problem.

[2] `https://en.wikipedia.org/wiki/Pancake_sorting`
[3] `https://en.wikipedia.org/wiki/Quicksort`

Question 4 (25 points)

In this question, you have to extend your symbolic execution engine from Assignment 2 to a verification engine. Recall that your symbolic execution engine is located in directory `wlang`. You can execute the engine using the following command:

```
(venv) $ python -m wlang.sym wlang/test1.prg
```

A sample program is provided for your convinience in `wlang/test1.prg`

The syntax of the WHILE language has been extended with annotation for

$$\textbf{while } b \textbf{ inv } inv \textbf{ do } s$$

where b is a Boolean loop condition, inv is a Boolean invariant, and s a statement. Recall (from the lecture notes) that execution of an annonated while-loop is equivalent to the following loop-free program that checks taht inv is indeed a loop invariant:

```
assert inv;
havoc V;
assume inv;
if b then { s ; assert inv; assume false }
```

where V is the set of all variabled modified (i.e., defined) in the loop body s.

(a) Extend your symbolic execution engine to execute while-loops annotated with loop invariants. Your engine should execute the loop according to the scheme above and report any assertion failures (i.e., whether inv fails initiation or invariance).

Hint: You can use your *Undefined Use Visitor* from Assignment 1 to find all variables that are modified/defined by the loop body.

(b) Provide the missing inductive invariants and use your implementation to verify the following program:

```
havoc x, y;
assume y >= 0;
c := 0;
r := x;
while c < y
inv [MISSING]
do
{
  r := r + 1;
  c := c + 1
};
assert r = x + y
```

(c) Extend the test suite `test_sym.py` to achieve 100% branch coverage of your implementation. Recall that you can run the test suite using

```
(venv) $ python -m wlang.test
```

and measure coverage of the test suite using

```
(venv) $ coverage run -m wlang.test
(venv) $ coverage html
```

(d) Construct a program with at least one loop and inductive invariants and verify it using your system. You can use a variant of a program from part (b) of this question, or create a different one. Save the program in the file called `q4d.prg` at the top of the repository.