Lab 3

Automated Verifier : Three Easy Pieces

December 3, 2019

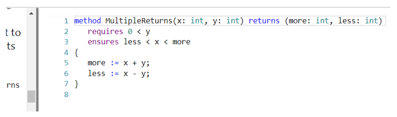
Deadline

Due: December 31, 2019, 23:59. Good luck!

Prerequisite

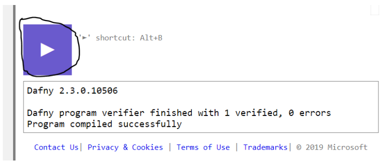
Pink words below are hyperlinks. You can click on them to open websites.

1. In this lab, you need to use Dafny to formally verify real programs.
2. Microsoft research provides a tutorial. You need to read it to learn how to use Dafny. To implement this lab, you need to understand section 1, 2, 3, 6, 8, 9 and 10.
3. There are some exercises in the tutorial. You don't need to implement them. If you are interested in these exercises, their solutions are here.
4. In this lab, you don't need to install any software. Because you can write code on the right of tutorial. Then push the violet button and Dafny will verify and compile your program. After an interval, Dafny will tell you if the program has bugs. There is an example in Figure 1 and Figure 2.
5. If you want to install Dafny in your computer, you can read guide of CMU.



```
method MultipleReturns(x: int, y: int) returns (more: int, less: int)
   requires 0 < y
   ensures less < x < more
{
   more := x + y;
   less := x - y;
}
```

Figure 1: Write code.



'►' shortcut: Alt+B
Contact Us| Privacy & Cookies | Terms of Use | Trademarks| © 2019 Microsoft

```
Dafny 2.3.0.10506

Dafny program verifier finished with 1 verified, 0 errors
Program compiled successfully
```

Figure 2: Push button and verify code.

1 Problem

There are 4 problems in lab3. You need to write preconditions, invariants and predicates.

1.1 method1

```
method method1(x: int, y: int) returns (z: int)
// Add a precondition here.
   ensures z > 0
{
   if x < 0
     { return y; }
   else
     { return x; }
}
```

Write a proper precondition to make sure the program returns a positive number.

1.2 method2

```
method method2(a: array<int>, v: int) returns (b: int)
// Add a precondition here.
{
   return a[v] / v;
}
```

Write a proper precondition to make sure the program has no exception, such as division by zero and access an array out of bounds.

1.3 method3

```
predicate notzero(a: array<int>)
  reads a
{
// Add a predicate here.
}

method method3(a : array<int>, n : int) returns (b : int)
  requires n == a.Length && notzero(a)
  ensures b == 0;
{
  var i := 0;
  while i < n
      invariant 0 <= i <= a.Length
      invariant n == a.Length
      invariant forall k :: 0 <= k < i ==> a[k] != 0
  {
    if a[i] == 0
    { return 1; }

    i := i + 1;
  }
  return 0;
}
```

Write a proper predicate to make sure the program returns 0.

1.4 method4

```
method method4(a: array<int>) returns (min: int)
  requires 0 < a.Length
  ensures forall k :: 0 < k < a.Length ==> min <= a[k]
  ensures exists k :: (0 < k < a.Length ==> min == a[k])
{
  var index := 0;
  min := a[0]; // assume first is min
  while index < a.Length
// Add invariants here
  {
    if a[index] <= min { min := a[index]; }
    index := index + 1;
  }
}
```

This method returns the minimum number in the array. Write the proper loop invariants to help Dafny prove the program. Dafny is smart and sometimes you don't need to give very detailed invariants. The invariants are very similar to that of exercise 12 in Dafny exercises. Read the exercise and you will find that writing invariants is not difficult.

2 Framework

The four problems are in method1.dfy - method4.dfy. You should write your answers in lab3.txt like this:

```
requires xxx
#
requires xxx
#
xxx
#
invariant xxx
#
```

There are four lines having only one # in lab3.txt. They are necessary. Don't delete them and don't insert more #. But you can insert some blank lines. After writing lab3.txt, you should input "make run" to generate answer1.dfy - answer4.dfy. Copy them into the right square frame of Dafny tutorial website and push the purple button. If Dafny outputs 0 errors, your answers are right. Otherwise, your answers are not correct and you need to change them.