Program Verification: Hoare Logic and Dafny

Rigorous Software Development
EUROPEAN MASTER IN SOFTWARE ENGINEERING/ MASTER IN SOFTWARE AND SYSTEMS
Universidad Politécnica de Madrid/IMDEA Software

October 16, 2019

To be turned in by November 2, 2019. Send them to me by mail to `jmarino@fi.upm.es`.

Exercise 1. Give a formal proof for the validity of the following two derived rules in Hoare logic:

$$\frac{\{\psi\}P\{\phi\} \qquad \varphi_1 \implies \psi \wedge \phi \implies \varphi_2}{\{\varphi_1\}P\{\varphi_2\}}$$

$$\frac{\{\varphi \wedge B\}P_1\{\psi\} \qquad \{\varphi \wedge \neg B\}P_2\{\phi\}}{\{\varphi\}\text{ if }(B)\text{ then }P_1\text{ else }P_2\{\psi \wedge \phi\}}$$

Exercise 2. Complete the formal proof in Hoare logic of the Java code for the factorial method shown below. This is similar to the *squares* method proven in the classroom – and, in more detail, in the slides. The key is in finding the right invariant for the loop.

```
public static final int factorial (int n) {
  int r = 1;
  int t = n;
  while (t>0) {
    r = r*t;
    t = t−1;
  }
  return r;
}
```

Exercise 3. Take the Dafny tutorial at `http://www.rise4fun.com/Dafny/tutorial/` and use the server to verify a Dafny version of the factorial method shown above. That is, you must:

- Define a recursive factorial function that serves as the mathematical intended meaning, and
- translate the code above into a Dafny method that ensures that the value returned is actually the same obtained with the recursive definition.

You can use the following skeleton as starting point:

```
function fact(n: nat): nat {
  // COMPLETE
}
method Factorial (n: nat) returns (r: int)
  ensures r == fact(n);
{
  // COMPLETE
}
```

Exercise 4. A typical introductory example on formal methods, when trying to explain the limitations of plain testing, is a program to sort a list of integers. Among other issues, it shows that leaving relevant details out of a formal specification is relatively easy.

I am not asking you to fully verify a sorting algorithm, but just *partly* verifying *part* of a sorting algorithm. Concretely, your challenge is to verify that the InsertSorted method that takes a sorted array of integers and creates a new array by inserting a given element at the right position, returns an array that is sorted. That method would typically be part of an InsertionSort routine.

You can use the following skeleton as starting point:

```
function sorted(a: array<int>): bool
   requires a != null;
   reads a;
{
   forall i :: 0 <= i < (a.Length - 1) ==> a[i] <= a[i+1]
}

method InsertSorted(a: array<int>, key: int) returns (b: array<int>)
   requires a != null && sorted(a);
   ensures b != null;
   ensures sorted(b);
{
   // COMPLETE
}
```

Observe that the sorted predicate shown above is slightly different from the one used in the Dafny tutorial. You are free to use the other definition, although my perception is that applying it to this problem would require introducing some auxiliary lemmas.