CITS5501 Software Testing and Quality Assurance
Semester 1, 2020
Week 10 Workshop – Formal methods

Reading

It is strongly suggested you complete the recommended readings for weeks 1-9 *before* attempting this lab/workshop.

1. Applicability of formal methods

Discuss the following scenarios. Would you use formal methods for any of the following systems? If so, which systems, and why?

a. The next version of *Confectionery Crush Story*, a web– and mobile-app–based puzzle video game your company develops. The game is available for both Android and iOS mobile devices, and the previous version grossed over $US 1 billion in revenue.

b. *Exemplarex*, software produced for Windows and MacOS operating systems and licensed to educational institutions. The software semi-automatically invigilates exams set by the institutions: machine-learning techniques are used to analyse audio and video of exam candidates to identify possible academic misconduct.

c. The online banking website provided by a major Australian bank, EastPac. Over 5 million customers use the website to perform banking transactions on personal or business bank accounts.

d. A radiation therapy system used to treat cancer patients. The system has two principal components:

   - A radiation therapy machine that delivers controlled doses of radiation to tumour sites, controlled by an embedded software system.
   - A treatment database that includes details of the treatment given to each patient.

Notes on exercise 1

The aim of the exercise is just to get you to think about what factors might make formal methods appropriate.

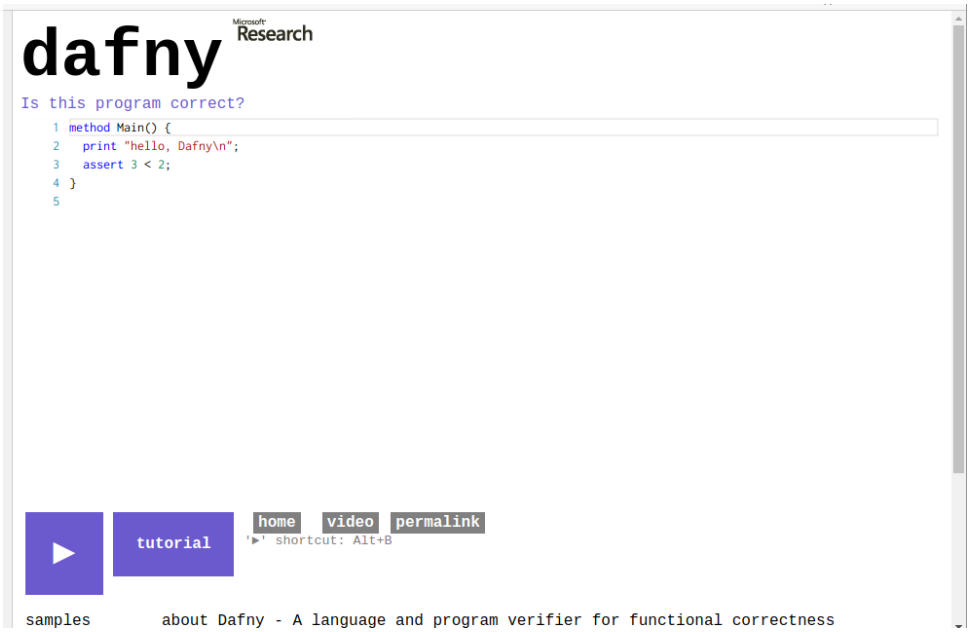
However, some possible answers are:

a. Probably not, because speed-to-market is more important for the games industry
b. Possibly; but if the market doesn't *demand* formal assurance that the software works as specified, it is probably not worth the effort of doing so. In any case, the end result of a machine-learning technique will be some sort of model used for prediction/classification, and formal methods probably can't prove that the classifier is "good".
c. Possibly lightweight formal methods (e.g. advanced type systems) might be appropriate for use in securing web applications. In general, banks apply sufficient security techniques to bring losses through fraud etc to a tolerable level – the cost of fully verifying systems is typically not worth the return.
d. Yes, since this is critical software – lives could be endangered if it malfunctions – the use of formal methods may well be appropriate. Any of the methods we have looked at (advanced type systems, formal verification, model-based methods) could be applied.
   [Note that in practice, however, most medical software – even for critical applications – is not, in fact, developed using formal methods.]

2. Introduction to Dafny

Dafny is a "verification-aware" programming language. The compiler for the language incorporates a program verifier – as you type in a program, the verifier constantly checks to see whether what you have written can be proved correct, and flags any errors.

A web-based interface to the language is provided at https://rise4fun.com/Dafny/.



If you press the "compile" button (marked with a "play"/triangle symbol), your Dafny code will be checked and compiled – and, if it contains a method called "`Main()`", that method will be run. Try compiling the code that is initially contained in the code box:

```
1 method Main() {
2   print "hello, Dafny\n";
3   assert 3 < 2;
4 }
```

You should see that the Dafny compiler reports an *assertion violation* in line 3; the assertion "$3 < 2$" is not true.

Now comment out the assertion (using double forward-slashes – "`//`" – the same as Java.) If you try re-compiling, you should see the code now compiles and runs.

Dafny resources

The paper introducing Dafny is available here.

Documentation for the language (including detailed tutorials and reference material) is available at:

https://dafny-lang.github.io/dafny/

3. Preconditions

Try compiling the following code:

```
1 method Main() {
2   PrintInt(-1);
3 }
4
5 method PrintInt(x : int)
6 {
7   print "the int is: \n";
8   print x;
9 }
```

The int -1 should be printed with a message. Now add a precondition to PrintInt, "`requires x >= 0`":

```
1 method Main() {
2   PrintInt(-1);
3 }
4
```

```
5  method PrintInt(x : int)
6    requires x >= 0
7  {
8    print "the int is: \n";
9    print x;
10 }
```

If you try compiling again, you will see that Dafny won't permit you to call the `PrintInt` method, unless it can verify all the preconditions hold. Change the `int` being passed from -1 to 1, and you should see the code now compiles.

4. Using Dafny features

The following Dafny code is intended to find the position of the largest element of an array. It is only guaranteed to produce a result if the array is *non-empty*, however. We won't compile this (yet), but look at the following code:

```
method FindMax(arr: array<int>) returns (r: int)
{
  var max_val : int := arr[0];
  var max_idx : int := 0;

  var i : int       := 1;

  while (i < arr.Length)
  {
    if arr[i] > max_val
      {
        max_idx := i;
        max_val := arr[i];
      }
    i := i + 1;
  }
  return max_idx;
}
```

At what points in the code might we insert the following, and what Dafny keywords would be used?

- preconditions
- postconditions
- loop invariants
- assertions

See if you can state what the preconditions and postconditions are (in English is fine).

Solution:

Correct locations would be:

```
method FindMax(arr: array<int>) returns (r: int)
   requires /* **preconditions go here** */
   ensures /* **postconditions go here** */
{

  var max_val : int := arr[0];
  var max_idx : int := 0;

  var i : int       := 1;

  /* **assertions could go anywhere in the body** */

  while (i < arr.Length)
    invariant /* **loop invariants go here** */
  {
    if arr[i] > max_val
      {
        max_idx := i;
        max_val := arr[i];
      }
    i := i + 1;
  }
  return max_idx;
}
```

Preconditions (in English):

- The array is of length 1 or more.

Postcondition (in English):

- The return value `r` is greater than or equal to every element of the array.

Challenge exercise: Try verifying the above code using the online Dafny verifier. You will probably want to try the rest of the exercises in this worksheet first, and work through the online Dafny tutorial. This will explain how to use the `forall` keyword, which we have not covered, but is needed for the challenge.

Sample solution:

Fully verified code:

```
method FindMax(arr: array<int>) returns (r: int)
   requires arr.Length > 0
   ensures (0 <= r < arr.Length) && (forall k :: 0 <= k < arr.Length
   ↪ ==> arr[r] >= arr[k])
{

  var max_val : int := arr[0];
  var max_idx : int := 0;

  var i : int       := 1;

  assert i >= 1;
  assert max_idx <= i;

  while (i < arr.Length)
    // probably could get away with fewer loop invariants
    invariant 1 <= i <= arr.Length
    invariant 0 <= max_idx  <= arr.Length
    invariant 0 <= max_idx  <= i
    invariant forall k :: 0 <= k <= i-1 ==> max_val >= arr[k]
    invariant (0 <= max_idx < arr.Length) && (max_val == arr[max_idx])
   ↪ && (forall k :: 0 <= k <= i-1 ==> arr[max_idx] >= arr[k])
  {
    if arr[i] > max_val
      {
        max_idx := i;
        max_val := arr[i];
      }
    i := i + 1;
  }
  return max_idx;
}
```

5. Writing Dafny methods

The following code does not yet compile:

```
method Max(a: int, b:int) returns (c: int)
  ensures c >= a
{
  return 0;
}
```

Write the body of the method Max. It should take two integer parameters and return their maximum. Add appropriate annotations (one is already provided) and make sure your code verifies. Refer to the documentation and tutorials, if necessary.

```
method Max(a: int, b:int) returns (c: int)
  ensures c >= a
  ensures c >= b
{
  if (b > a) {
    return b;
  } else {
    return a;
  }
}
```