Hands-on session UT: Dafny

Big Software on the Run Winter School 2016

October 27, 2016

You are encouraged to work in pairs.

Name 1:	
Name 2:	

Please note that you should not modify any of the provided implementations (unless stated otherwise), but only add annotations (in terms of pre- and post-conditions, invariants and assertions) to the code. There are multiple correct solutions to the challenges, we encourage you to design the most compact specifications. If you have any questions, don't hesitate to raise your hand and ask the tutorial organizers.

We encourage you to provide the solutions as permalinks. Please hand in the solutions when you are done so we can check them and determine a winner.

Introduction to Dafny

Dafny is a tool for verifying functional correctness of programs. A program, written in the Dafny language, is annotated with a specification to describe the desired behaviour of each method. Dafny then tries to verify if the implementation is correct.

The following example illustrates the main components of the specification language:

```
0  // view online at: http://rise4fun.com/Dafny/dBwb
1  // return the index if the value is in a, otherwise return a negative value
2  method Find(a: array<int>, value: int) returns (index: int)
3    requires a != null
4    ensures index >= 0 ==> index < a.Length && a[index] == value
5    ensures index < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != value
6  {
7    index := 0;
8    while index < a.Length
9      invariant index <= a.Length
10     invariant forall k :: 0 <= k < index ==> a[k] != value
11   {
12     if a[index] == value {
13       return;
14     }
15     index := index + 1;
16   }
17   index := -1;
18 }
```

The `Find()` method checks the array `a` and looks if it contains the element `value`. If this is the case, it returns the corresponding index of the array and it returns a negative value otherwise.

- The code is annotated with the *pre-condition* `requires a != null`, indicating that the implementation should only be correct for the cases where the array `a` exists.
- The two `ensures` statements are called the *post-condition* and indicate the properties that should hold after the method returns.
  - The first statement specifies that if the returned `index` is positive, it must be smaller than the array length and `a[index]` should point to `value`.

– The second statement specifies that of the returned `index` is negative, there should not be any element `value` in the array `a`. Thus, we use `forall` to indicate that every element in the array does not equal to `value`.

The while-loop is annotated with two *loop-invariants*. A loop-invariant is a property that is preserved by each iteration of the loop (thus it also hold before and after the loop). Verification with loops is difficult, therefore Dafny often requires such loop-invariants to help the verification process.

– The first invariant simply states that `index` will never grow beyond `a.Length`. This invariant is necessary for Dafny to ensure that the next invariant won't go out of range (even though it may seem obvious that `index` won't grow larger than `a.Length`).

– The second invariant states that for every value `k` that we have seen 'up to this moment', we have `a[k] != value`. This invariant is created to help verifying the post-condition of the method; consider the case when `index=a.Length`.

The following code shows the use of predicates in Dafny. A `sorted` predicate is defined that checks whether the array `a` is sorted. A predicate can be seen as a boolean function; it returns true if the predicate holds, and false otherwise.

```
0  // view online at: http://rise4fun.com/Dafny/p7jg
1  // returns true if the array is sorted
2  predicate sorted(a: array<int>)
3     requires a != null
4     reads a
5  {
6     forall j, k :: 0 <= j < k < a.Length ==> a[j] <= a[k]
7  }
8
9  // test method
10 method m()
11 {
12   var a := new int[3];
13   a[0] := 1;  a[1] := 2;  a[2] := 3;
14   assert sorted(a);
15 }
```

When verifying a program, you should be aware that the verification process is *modular*. This means that if a program calls a method and this method returns, it only knows what is specified in the method specification, i.e. we know the that the post-condition holds after the method returns. The verification process can be described in three steps:

1. Prove termination; verify the program without pre- and post-conditions and check what needs to be specified, to ensure the method terminates (normally, without exceptions).

2. Specify a post-condition; this is the part that you want to prove.

3. Verify that the post-condition holds; use Dafny to find out what additional information is required to prove that the post-condition is true.

For more information on Dafny, please check: http://rise4fun.com/Dafny/tutorial/Guide

Challenge 1

Show that the following method `m()` correctly returns the maximum value over the set $\{a, b, a + b\}$ by providing a suitable specification in Dafny.

```
// view online at: http://rise4fun.com/Dafny/YS7l
// returns the maximum over the values a, b, and a+b
method m(a: int, b: int) returns (m : int)
{
  if a <= b {
    if a > 0 {
      return a + b;
    }
    return b;
  }
  if b > 0 {
    return a + b;
  }
  return a;
}
```

Please provide the specification below (and indicate where it is inserted), or provide a permalink to your Dafny implemenation:

Challenge 2a

The following function `max()` returns an index of the array `a` that points to its maximum value. First check if Dafny can verify the program without any post-conditions, and provide specification where required. Then, provide a post-condition that correctly describes the requirement for the method and check what is necessary for Dafny to verify.

```
// view online at: http://rise4fun.com/Dafny/DSxc
// return the index of the maximum element
method max(a: array<int>) returns (x: int)
{
    x := 0;
    var i := 0;
    while i < a.Length
    {
        if a[x] < a[i] {
            x := i;
        }
        i := i + 1;
    }
    return x;
}
```

Please provide the specification below (and indicate where it is inserted), or provide a permalink to your Dafny implemenation:

Challenge 2b

The following program computes the same result as Challenge 2a, only here the implementation is a bit more complicated. Show with Dafny that this program is also correctly implemented.

```
// view online at: http://rise4fun.com/Dafny/8gqY9
// return the index of the maximum element
method max(a: array<int>) returns (x: int)
{
    x := 0;
    var y := a.Length - 1;
    while x != y
    {
        if a[x] <= a[y] {
            x := x + 1;
        } else {
            y := y - 1;
        }
    }
    return x;
}
```

Please provide the specification below (and indicate where it is inserted), or provide a permalink to your Dafny implemenation:

Challenge 3

The following program sorts an array. Show that the sorting algorithm is correctly implemented. It may be helpful to use a modified version of the `sorted` predicate.

```
// view online at: http://rise4fun.com/Dafny/pl1Ty
// sorts the array a
method sort(a: array<int>)
{
    var m := 0;
    while (m != a.Length)
    {
        var k := m;
        while (0 <= k-1 && a[k-1] > a[k])
        {
            a[k-1],a[k] := a[k],a[k-1];
            k := k-1;
        }
        m := m+1;
    }
}
```

Please provide the specification below (and indicate where it is inserted), or provide a permalink to your Dafny implemenation: