Techniques for Improving Software Productivity

16/17 Fall Semester

Homework Assignment 4: Dafny

The code skeleton for the exercise can be found in: https://bitbucket.org/tausigplan/soft-prod16 under exercises/ex4/, and the code from the demos can be found there under demos/.

1. Consider the Dafny specification and code below, which is supposed to calculate the product of two numbers. Annotate the code so that it checks.

```
method Product (m: nat, n: nat) returns (res:nat)
  ensures res == m * n;
{
  var m1: nat := m; res := 0;
  while (m1 != 0)
  {
    var n1: nat := n;
    while (n1 != 0)
    {
      res := res + 1;
      n1 := n1 - 1;
    }
    m1 := m1 - 1;
  }
}
```

2. Prove, using Dafny, that the following algorithm satisfies its postconditions.

```
method Divide(x : nat, y : nat) returns (q : nat, r : nat)
  requires y > 0;
  ensures q * y + r == x && r >= 0 && r < y;
{
  q := 0;
  r := x;
  while (r >= y)
  {
    r := r - y;
    q := q + 1;
  }
}
```

3. Prove the correctness of the following program, which performs bitwise addition of two numbers a0 and b0. You have to supply the appropriate invariant and ranking function.

```
method Bitwise_add(a0 : int, b0 : int) returns (c : int)
  requires a0 >= 0 && b0 >= 0;
  ensures c == a0+b0;
{
  c := 0;
  var a, b, g, m := a0, b0, 1, 0;
  while (a > 0 || b > 0)
  {
    m := m + a % 2 + b % 2;
    a := a / 2;
    b := b / 2;
    c := c + g * (m % 2);
    g := 2 * g;
    m := m / 2;
  }
  c := c + g * m;
}
```

4. Prove the correctness of the following implementation of Linear Search.

```
method find(a : array<int>, key : int) returns (index : int)
  requires a != null;
  ensures 0 <= index <= a.Length;
  ensures index < a.Length ==> a[index] == key;
{
  index := 0;
  while (index < a.Length && a[index] != key)
  {
    index := index + 1;
  }
}
```

5. Below, we show the functional definition of the factorial function, and a loopy program that supposedly computes the factorial of a number as well. Write suitable annotations to show that the latter in fact computes the factorial function.

```
function Factorial(n: nat): nat
{
  if n == 0 then 1 else n * Factorial(n-1)
}
method AdditiveFactorial(n: nat) returns (u: nat)
  ensures u == Factorial(n);
{
  u := 1;
  var r := 0;
  while (r < n)
  {
    var v := u;
    var s := 1;
    while (s <= r)
    {
      u := u + v;
      s := s + 1;
    }
    r := r + 1;
  }
}
```

6. Consider the following implementation of Binary Search, annotated with pre and post-conditions. As you will find by pasting this code into Dafny, there is something wrong with either the annotations or the code. Explain what the problem is.

```
method BinarySearch(a: array<int>, value: int) returns (index: int)
  requires a != null && 0 <= a.Length;
  requires forall j, k :: 0 <= j < k < a.Length ==> a[j] <= a[k];
  ensures 0 <= index ==> index < a.Length && a[index] == value;
  ensures index < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != value;
{
  var low, high := 0, a.Length;
  while (low < high)
    invariant 0 <= low <= high <= a.Length;
    invariant forall i :: 0 <= i < a.Length && !(low <= i < high) ==> a[i] != value;
  {
    var mid := (low + high) / 2;
    if (a[mid] < value)
    {
      low := mid;
    }
    else if (value < a[mid])
    {
      high := mid - 1;
    }
    else
    {
      return mid;
    }
  }
return -1;
}
```