Homework 2 – Due Thursday, September 24, 2015

Please refer to the course website for the full homework policy and options.

Reminders

- Your solutions are due in class. Please also submit a soft copy on the dropbox on Owlspace. Late homework will not be accepted.

- Collaboration is permitted, but you must write the solutions by yourself without assistance, and be ready to explain them orally to a member of the course staff if asked. You must also identify your collaborators. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

- To facilitate grading, please write down your solution to each problem on a separate sheet of paper. Make sure to include all identifying information and your collaborators on each sheet.

- Several exercises in this assignment involve inductive correctness proofs of programs. Please use Dafny (http://rise4fun.com/dafny) to complete these assignments. In each of the proof problems, your objective is to come up with a suitable ranking function (decreases clause) and loop invariant. In some cases, you may have to add some assumptions on the input (a requires clause) to make the proof go through.

1. Give a Turing machine that accepts the following set of input words:

   \{ w : w contains twice as many 0s as 1s \}

   The input alphabet here is assumed to be \{0, 1\}.

   (The ideal solution to this exercise is a Turing machine defined in full mathematical detail. However, you will get partial credit for an English-language description of the machine.)

2. Say that a write-once Turing machine is a single-tape Turing machine that can alter each tape cell at most once. Show that this variant of Turing machines is equivalent to the ordinary Turing machine model that we studied in class. (In this exercise, an English-language description of the construction will give you full credit.)

3. In class we briefly mentioned strongest postconditions, but we didn’t do much with them. Briefly, for properties \( P \) and program statements \( S \), the strongest postcondition of \( P \) with respect to \( S \) (written as \( sp(P, S) \)) is the formula \( Q \) that represents the set of terminal states of \( S \), when run from an initial state satisfying \( P \).

   Now suppose \( S \) is of the form:

   \[
   x := x + y;
   y := x - y.
   \]

   Suppose \( P \) is a property \((x + y \geq 0)\). What is the value of \( sp(P, S) \)?
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;
    }
}

Figure 1: Product

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;
    }
}

Figure 2: Integer division

4. Consider the Dafny specification and code in Fig. 1 which is supposed to calculate the product of two numbers. Give an annotated version of the code that Dafny can prove correct.

5. Consider the Dafny specification and code for the algorithm for integer division in Fig. 2. Give an annotated version of the code that Dafny can prove correct.

6. Consider the implementation of Linear Search in Fig. 3. Give an annotated version of the code that Dafny can prove correct.

7. Fig. 4 shows the functional definition of the factorial of a number, 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. As in the Fibonacci example shown in class, you may refer to `Factorial(n)` in the annotations.
method LinSearch(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;
  }
}

Figure 3: Linear search

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;
  }
}

Figure 4: Additive factorial