Homework 1 - Loop invariants

February 28, 2011

Recall that we say I is an inductive loop invariant, written as

```
void p()
  requires Pre
  ensures Post
{
  s1;
  while (e)
    invariant I
  {
   s2;
  }
  s3;
}
```

if the following three conditions hold:

1. I holds initially: in all states satisfying Pre, when execution reaches loop entry, I holds
2. I is preserved: if we assume that I and the loop condition (e) hold, then we can prove that I will hold again after executing s2
3. I is strong enough: if we assume that I holds and the loop condition (e) does not hold, then we can prove that Post holds after executing s3

1 Problem 1

This question is a pen and paper question. You are expected to give a full proof similar to the one done in class.(Spec# does not seem to be able to handle this problem. Should you find a solution that Spec# is able to prove, you're welcome to submit it as well.)
Assume that an exponentiation operator (^) is available (at least for the specification and analysis part) and consider the following code for fast exponentiation:

```
int exp(int x, int y)
  requires x >= 0 && y >= 0;
  ensures result == x^y;
{
  int res = 1;
  int a = x;
  int n = y;

  while(n > 0) {
    if(n % 2 == 1) {
      res = ...;
      n = ...;
    }

    a = ...;
    n = ...;
  }
  return res;
}
```

Complete the code, find the loop invariant and show (on paper) that the three conditions for a loop invariant are satisfied. In your write-up you can use the usual notation of exponentiation (x^y)
Hint: A similar algorithm works for fast multiplication, potentially the invariant is easier to see there...

2 Problem 2

This question is to be done with (the web-interface of) Spec#. Write and test your code there and provide it as the solution. In case Spec# crashes or times out, but you still believe that your specifications are correct, provide a detailed explanation or proof.

Consider a function that takes as argument two 2D integer arrays of the same dimensions and outputs the number of places where the entries match. Write the code including the complete contract and invariant(s) in Spec# for this function (you need to enclose it in a class). Provide all preconditions so that the function works correctly under all valid inputs (instead of having checks in the code). The non-null requirement is already by default taken care of by Spec#, since types are by default non-null.

2D arrays are best represented by multidimensional arrays in Spec#. These are basically $n \times m$ matrices, whose individual elements can be accessed as `a[1, 3]`. The lengths in each dimension are obtained by `a.GetLength(0)` and `a.GetLength(1)` for n and m respectively. The header of the function you should write is then written as `int count(int[,] a, int[,] b)`.

Hint: The `sum` function in Spec# may come in handy. The syntax and some examples of its use can be found in the Spec# presentation slides linked on the course webpage.