Dr. Matthias Heizmann
Dominik Klumpp

Hand in until 10:00 on June 29, 2020
Discussion: June 29, 2020

Tutorial for Program Verification
Exercise Sheet 12

Exercise 1: Arrays and Havoc 2 Points

Consider a program $P = (V, \mu, \mathcal{T})$ whose set of variables contains three integer variables `i`, `j`, `x` and an array variable `a` whose indices and values are integers. Use a formula over primed and unprimed variables to write down the following relation.

$$[\![\texttt{if(a[i]>0)}\{\texttt{havoc a;}\}\texttt{ else }\{\texttt{havoc x; a[j]:=x;}\}]\!]$$

Exercise 2: Minimum 2 Points

The following Boogie program iterates through a two-dimensional array and finds the minimum value within the given bounds `lo` and `hi`.

```
procedure findmin(a : [int, int]int, lo : int, hi : int) returns (min : int)
  requires lo <= hi;
  ensures (forall i, j : int :: lo <= i && i <= hi && lo <= j && j <= hi
    ==> a[i, j] >= min);
{
  var i, j : int;

  i := lo;
  min := a[lo, lo];

  while (i <= hi) {
    j := lo;
    while (j <= hi) {
      if (a[i, j] < min) {
        min := a[i, j];
      }

      j := j + 1;
    }

    i := i + 1;
  }
}
```

Find inductive loop invariants for the two while loops of the program that are strong enough to prove that the program satisfies the given precondition-postcondition pair (the formulas after `requires` and `ensures`, respectively). You can use Ultimate Referee to check your solution.

Exercise 3: Selection Sort

2 Points

The following boogie procedure implements the selection sort algorithm that sorts a given array in ascending order.

```
procedure SelectionSort(lo : int, hi : int, a : [int]int) returns (ar : [int]int)
    requires lo <= hi;
    ensures (forall i1, i2 : int :: lo <= i1 && i1 <= i2 && i2 <= hi
                                    ==> ar[i1] <= ar[i2]);
{
    var i, k, min, tmp : int;
    ar := a;

    k := lo;
    while (k <= hi) {
        // Find the index of the minimal element between k and hi (inclusive)
        min := k;
        i := k + 1;
        while (i <= hi) {
            if (ar[i] < ar[min]) { min := i; }
            i := i + 1;
        }

        // Swap ar[k] and ar[min]
        tmp := ar[k];
        ar[k] := ar[min];
        ar[min] := tmp;

        k := k + 1;
    }
}
```

Find inductive loop invariants for the two while loops that are strong enough to prove that the program satisfies the given precondition-postcondition pair. You can use Ultimate Referee to check your solution.