Formal Methods in Software Development Exercise 6 (December 17)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

December 6, 2012

The result is to be submitted by the deadline stated above *via the Moodle interface* of the course as a *.zip or .tgz* file which contains

1. a PDF file with
   - a cover page with the course title, your name, Matrikelnummer, and email address,
   - a (nicely formatted) copy of the *.java/.theory* file(s) used in the exercise,
   - the deliverables requested in the description of the exercise,
   - for each program method, a screenshot of the "Analysis" view of the RISC ProgramExplorer with the specification/implementation of the method and the (expanded) tree of all (non-optional) tasks generated from the method,
   - for each program method, a screenshot of the corresponding "Semantics" view and an informal interpretation of the method semantics;
   - for each task an explicit statement whether the goal of the task was achieved or not and, if yes, how (fully automatic proof, immediate completion after starting an interactive proof, complete or incomplete interactive proof),
   - for each truly interactive proof, a screenshot of the corresponding "Verify" view with the proof tree,
   - optionally any explanations or comments you would like to make;
2. the *.java/.theory* file(s) used in the exercise,
3. the task directory (*.PETASKS**) generated by the RISC ProgramExplorer.

Email submissions are *not* accepted.

Exercise 6: Merging two Arrays

Use the RISC ProgramExplorer to specify the following program, reason about its behavior, and verify its correctness with respect to its specification:

```
class Exercise6
{
  // merges two sorted arrays a and b into a sorted result array c
  // ("sorted" means "sorted in ascending order")
  public static int[] merge(int[] a, int [] b)
  {
    int n = a.length+b.length;
    int[] c = new int[n];
    int i = 0;
    int j = 0;
    int k = 0;
    while (k < n)
    {
      boolean aisnext = append(a, b, c, i, j, k);
      if (aisnext)
        i = i+1;
      else
        j = j+1;
      k = k+1;
    }
    return c;
  }

  // writes into c[k] either a[i] or b[j], whatever is smaller
  // returns true iff a[i] was written
  public static boolean append(int[] a, int[] b, int[] c, int i, int j, int k)
  {
  }
}
```

First, create a separate directory in which you place the file *Exercise6.java*, `cd` to this directory, and start `ProgramExplorer&` from there. The task directory *.PETASKS** is then generated as a subdirectory of this directory.

Then perform the following tasks:

1. Derive suitable specifications of `merge` and `next` and reason about the behavior of `merge`; verify its correctness with respect to its specification. Deliver for `merge` the same results as requested in Exercise 5[1].

   Please note that in the specification of `merge` is does not suffice to say that the result array is sorted, it is also necessary to establish some relationship between

[1]If you cannot show all required verification conditions for `merge`, you may nevertheless continue with the second part of the exercise.

the elements of the input arrays and the elements of the result array. Since the exact specification (and its verification) is a bit complicated, for the purpose of this exercise it suffices to state that every element in the result array occurs in one of the input arrays and vice versa.

2. Provide a suitable implementation of `next`, reason about its behavior, and verify its correctness with respect to its specification. Deliver for `next` the same results as requested in Exercise 5.

Bonus (20 Points): give an exact specification of `merge` (no verification is required).

Hint: in addition to specify that the result array is sorted, you must specify, for each input array, the existence of a mapping of indices from the input array to strictly increasing indices of the result array (such mappings are just values of type `ARRAY INT OF INT`). The two mappings must not overlap and must be exhaustive.