Formal Methods in Software Development
Exercise 5 (December 5)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

November 16, 2011

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,
   - the deliverables requested in the description of the exercise,
   - a (nicely formatted) copy of the *.java*/*.theory* file(s) used in 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 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 5: Sorting an Array

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

```
class Exercise5
{
  // sort array a in ascending order
  public static void sort(int[] a)
  {
    int n = a.length;
    int i = 0;
    while (i < n)
    {
      int m = min(a, i);
      int e = a[i];
      a[i] = a[m];
      a[m] = e;
      i = i+1;
    }
  }

  // find position of minimum of array segment a[i...]
  public static int min(int[] a, int i)
  {
    ...
  }
}
```

In more detail, first derive suitable specifications of `sort` and `min` and reason about the behavior of `sort`; verify its correctness with respect to its specification. Deliver for `sort` the same results as requested in Exercise 4[1].

Afterward provide a suitable implementation for `min`, reason about its behavior, and verify its correctness with respect to its specification. Deliver for `min` the same results as requested in Exercise 4.

In the specification of `sort`, it does not suffice to state that the resulting array a' is sorted and has the same length n as a; one also needs to state that a' has the same elements as the original array a, i.e., that a' is a permutation of a. For this, it suffices to state that there exists an array p of length n that contains all values $\{0, \ldots, n-1\}$ that $a'[p[i]] = a[i]$ for every $i \in \{0, \ldots, n-1\}$.

This permutation p can be modeled as a value of type `ARRAY INT OF INT` which supports operations `[i]` (the element at index i) and `WITH [i] := e` (the array that is identical to the

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

original one except that value e is stored at index i). Furthermore, `(ARRAY(i:Base.int):i)` denotes the array that holds value i at index i.

The corresponding predicate "a_0 is a permutation of a_1" can be thus formulated as

```
isPermutation: PREDICATE(Base.IntArray, Base.IntArray) =
  PRED(a0: Base.IntArray, a1: Base.IntArray):
    LET n = a0.length IN
    n = a1.length AND
    (EXISTS(p: ARRAY INT OF INT):
      (FORALL(i: INT): 0 <= i AND i < n =>
        0 <= p[i] AND p[i] < n AND
        a0.value[i] = a1.value[p[i]]) AND
      (FORALL(j: INT): 0 <= j AND j < n =>
        (EXISTS(i: INT): 0 <= i AND i < n AND p[i] = j)));
```

Use this predicate in the specification and in the loop invariant.

To achieve full credit for this exercise, you only need to expand this predicate in the proof that the invariant initially holds; you have then just to show that there exists the trivial permutation for $a' = a$. In the proof that the invariant is preserved by every loop iteration, you can leave the part open where you have to show the existence of a new permutation after the iteration.

Optional Bonus Exercise (20 points): complete the proof by showing the existence of the permutation after the loop iteration. This amounts to show that

- if there exists a permutation p_0 between the original array a and the array a_0 at the begin of the loop iteration,
- and the new array a_1 is constructed from a_0 by swapping the two elements at positions m and i.
- that there also exists a permutation p_1 between a and a_1.

Make a drawing to understand how p_1 is related to p_0; p_1 can be described with the help of the construct `ARRAY(i:INT): ...`.

In the proof that p_1 is indeed the correct permutation, you need a couple of `case` distinctions. You may use `decompose`/`split` rather than `scatter` to keep track of the intuition of the individual parts of the proof.