Formal Methods in Software Development
Exercise 5 (December 6)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

November 5, 2010

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 ProofNavigator file used in the exercise,
   - for each proof of a formula F, a readable screenshot of the RISC ProofNavigator after executing the command `proof` F,
   - an explicit statement whether the proof succeeded,
   - optionally any explanations or comments you would like to make;
2. the RISC ProofNavigator (.pn) file(s) used in the exercise;
3. the proof directories generated by the RISC ProofNavigator.

Exercise 5: Insertion Sort Core

Let a be an integer array of length greater than n such that a is sorted in ascending order in range $0 \ldots n-1$. We consider the problem of inserting $a[n]$ into that position of a such that a is sorted in range $0 \ldots n$.

1. (20P) Give a formal specification of the problem by a pair of pre- and post-condition. Please note that the post-condition must not only state that the updated array is sorted; it also must describe the relationship of the elements of the updated array to the elements of the original array (i.e. that there exists a suitable position p where the value has been inserted, that all elements before p have remained unchanged and that all elements after p have been shifted by one position). Please also note that the only program variables that appear in the problem statement are a and n.
2. (10P) The problem is expected to be solved by the following piece of code (the core of the insertion sort algorithm):

   ```
   i = n;
   t = a[i];
   while (i > 0 && a[i-1] > t)
   {
     a[i] = a[i-1];
     i = i-1;
   }
   a[i] = t;
   ```

   Assume that you are given a suitable loop invariant I and termination term T. Using these, derive those conditions that have to pe proved to verify the *total* correctness of the code (use the notation $P[e/x]$ to denote phrase P with variable x substituted by term e).
3. (20P) Give a suitable definition of I and T. The invariant must apparently express the information available about all positions that have already been processed (i.e. the positions greater than i) as well as the information about all positions that have not yet been processed (i.e. the positions less than or equal i). Validate I and T by constructing variable traces for at least three example inputs (including one where $a[n]$ is inserted at the beginning of the array and one where it stays where it is).
4. (10P) Formalize the proof obligations as a theory of the RISC ProofNavigator.
5. (40P) Prove the obligations with the RISC ProofNavigator.