Formal Methods in Software Development
Exercise 2 (November 15)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

September 9, 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 2: Searching for the Maximum

Take the Hoare triple

```
{$a = olda \land n = oldn \land 1 \leq n \leq length(a) \land (\forall j : 0 \leq j < n \Rightarrow a[j] \geq 0)$}
$m := 0; i := 0$
while $i < n$ do
  if $a[i] > m$ then
    $m := a[i]$
  end
  $i := i + 1$
end
{$a = olda \land n = oldn \land$
  $(\forall j : 0 \leq j < n \Rightarrow m \geq a[j]) \land (\exists j : 0 \leq j < n \land m = a[j])$}
```

which describes the core proof obligation for a program that searches for the maximum *m* in a non-empty array *a* of non-negative integers.

This Hoare triple can be verified with the help of a loop invariant with the following structure:

$$
\begin{aligned}
&a = olda \land n = oldn \land 1 \leq n \leq length(a) \land (\forall j : 0 \leq j < n \Rightarrow a[j] \geq 0) \land \\
&(\ldots \leq i \leq \ldots) \land (i = 0 \Rightarrow m = \ldots) \land \\
&(\forall j : 0 \leq j < i \Rightarrow \ldots) \land (i > 0 \Rightarrow \exists j : 0 \leq j < i \land \ldots)
\end{aligned}
$$

Use this invariant to produce the four verification conditions for proving the partial correctness of the program (one for showing that the input condition of the loop implies the invariant, one for showing that the invariant and the negation of the loop condition implies the output condition, two for showing that the invariant is preserved for each of the two possible execution paths in the loop body). Explicitly show the derivation of the conditions (not only the final result). Please note that the input condition of the loop is not the input condition of above code fragment!

Your task is now to verify these five conditions in the style of the verification of the "linear search" algorithm presented in class with the help of the RISC ProofNavigator. For this, write a declaration file with the following structure

```
newcontext "maxsearch";

// arrays as presented in class (with ELEM set to INT)
...
ELEM: TYPE = INT;
...

// program variables and mathematical constants
a: ARR; olda: ARR;
n: INT; oldn: INT;
i: INT; m: INT;
...
```

Define predicates `Input`, `Output`, and `Invariant`, where (as shown in class) `Invariant` should be parametrized over the program variables. Then define four formulas `A`, `B1`, `B2`, `C` describing the verification conditions and prove these.

All proofs can be done with the command `expand`, `scatter`, `split`, `goal`, `auto` only (it may be sometimes wise to use the `goal` command to switch the goal formula).