Formal Methods in Software Development
Exercise 2 (November 12)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

October 10, 2007

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

- A PDF file with
  - a cover page with the title of the course, your name, Matrikelnummer, and email-address,
  - for each exercise, a section with the number and name of the exercise and a copy of the ProofNavigator file used in the exercise,
  - for each proof of a formula F, a screenshot of the RISC ProofNavigator after executing the command `proof` F,
  - optionally any explanations or comments you would like to make;
- the RISC ProofNavigator (.pn) files used for the proofs;
- the proof directories generated by the RISC ProofNavigator.

2a(all): Searching for the Minimum

Take the Hoare triple

$\{a = olda \wedge n = oldn \wedge 0 \leq n \leq length(a) \wedge$
$\quad (\forall j : 0 \leq j < n \Rightarrow a[j] < MAX) \wedge i = 0 \wedge m = MAX\}$
while $i < n$ **do**
$\quad$ **if** $a[i] < m$ **then**
$\quad\quad m := a[i]$
$\quad$ **end**
$\quad i := i + 1$
end
$\{a = olda \wedge n = oldn \wedge$
$\quad ((n = 0 \wedge m = MAX) \vee$
$\quad\quad ((\forall j : 0 \leq j < n \Rightarrow m \leq a[j]) \wedge (\exists j : 0 \leq j < n \wedge m = a[j])))\}$

which describes the core proof obligation for a program that searches for the minumum element m in an array of integers less than MAX (which denotes an arbitrary constant).

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

$$
\begin{array}{l}
a = olda \wedge n = oldn \wedge 0 \leq n \leq length(a) \wedge \\
(\forall j : 0 \leq j < n \Rightarrow a[j] < MAX) \wedge i \leq n \wedge \\
((i = 0 \wedge m = MAX) \vee \\
\quad ((\forall j : 0 \leq j < i \Rightarrow m \leq a[j]) \wedge (\exists j : 0 \leq j < i \wedge m = a[j])))
\end{array}
$$

Use this invariant to produce the four verification conditions for proving the partial correctness of the program (one for showing that the input condition 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).

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 "minsearch";

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

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

...
```

Define predicates `Input`, `Output`, and `Invariant`, where (as shown in class) `Invariant` should be parameterized 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 (I recommend to use in some proof states the `goal` command to switch the goal formula, but this is not strictly necessary).

2b(all): Replacing Array Values

Take the Hoare triple

$\{a = olda \wedge n = oldn \wedge x = oldx \wedge y = oldy \wedge n \leq length(a)\}$
$i := 0$
while $i < n$ **do**
  if $a[i] = x$ **then**
    $a[i] := y$
  end
  $i := i + 1$
end
$\{length(a) = length(olda) \wedge n = oldn \wedge x = oldx \wedge y = oldy \wedge$
  $(\forall j : 0 \leq j < n \Rightarrow$
    $(olda[j] = x \Rightarrow a[j] = y) \wedge$
    $(olda[j] \neq x \Rightarrow a[j] = olda[j]))\}$

which describes the core proof obligation for a program that replaces in an array a every element x by y.

Find a suitable invariant, produce the necessary conditions for verifying the partial correctness of the program, and prove these conditions with the help of the RISC ProofNavigator (as in the previous exercise).