Formal Methods in Software Development
Exercise 3 (May 11)

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

April 5, 2006

The result is to me submitted to me by **May 11** (hard deadline) as a paper (handed out to me in class) or as a single PDF file (sent to me per email), in both cases with a cover sheet that contains your name and "Matrikelnummer".

1 Binary Search (Verification)

Take the Hoare triple

$$
\begin{array}{l}
\{olda = a \land oldx = x \land (\forall i : 0 \leq i < length(a) - 1 \Rightarrow a[i] \leq a[i+1]) \land \\
\quad r = -1 \land low = 0 \land high = length(a) - 1\} \\
\textbf{while } r = -1 \land low \leq high \textbf{ do} \\
\quad mid := \lfloor (low + high)/2 \rfloor \\
\quad \textbf{if } a[mid] = x \textbf{ then} \\
\quad\quad r := mid \text{ // B1} \\
\quad \textbf{else if } a[mid] < x \textbf{ then} \\
\quad\quad low = mid + 1 \text{ // B2} \\
\quad \textbf{else} \\
\quad\quad high = mid - 1 \text{ // B3} \\
\quad \textbf{end} \\
\textbf{end} \\
\{a = olda \land x = oldx \land \\
\quad ((r = -1 \land (\forall i : 0 \leq i < length(a) \Rightarrow a[i] \neq x)) \lor \\
\quad\; (0 \leq r < length(a) \land a[r] = x))\}
\end{array}
$$

which describes the core proof obligation for a program that applies the "binary search" method to determine the index r of an element x in a sorted array a.

This Hoare triple can be verified with the help of the following loop invariant (take your time to understand especially the range conditions on *low* and *high*):

$$
\begin{aligned}
&olda = a \wedge oldx = x \wedge (\forall i : 0 \leq i < length(a) - 1 \Rightarrow a[i] \leq a[i+1]) \wedge \\
&-1 \leq r < length(a) \wedge (r \neq -1 \Rightarrow a[r] = x) \wedge \\
&0 \leq low \leq length(a) \wedge -1 \leq high < length(a) \wedge low \leq high + 1 \wedge \\
&(\forall i : 0 \leq i < length(a) \wedge i < low \Rightarrow a[i] < x) \wedge \\
&(\forall i : 0 \leq i < length(a) \wedge high < i \Rightarrow x < a[i])
\end{aligned}
$$

With this information, you can produce the five 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, three for showing that the invariant is preserved for each of the three possible execution paths in the loop body). Please note that the program term $\lfloor x \rfloor$ is written in PVS as `floor`$(x/2)$. Make sure that your definitions type-check correctly by using the PVS command `tcp` (no type checking condition should remain unproved).

Your task is now to verifiy these five conditions in PVS in the style of the proof of the "linear search" algorithm presented in class. For this purpose, write a PVS theory of the following structure

```
binarysearch: THEORY
BEGIN
  IMPORTING arrays[int]

  // program variables
  a, olda: arr
  x, oldx: int
  low, high, mid: int
  r: int

  // quantified variables
  i, j: VAR nat
  ...
END binarysearch
```

where the PVS file for theory `arrays` is posted on the Web site.

Define three predicates `Input`, `Output`, and `Invariant`, where (as shown in class) `Invariant` should be parameterized over the program variables. Then define five formulas `A`, `B1`, `B2`, `B3`, `C` describing the five verification conditions and prove these.

Please note that the program variables denoting indices are declared as `int` (*r* and *high* might become negative) such that all range restrictions on these variables have to be explicitly described. The quantified variables however may be declared as `nat` such that non-negativeness has not to be stated explicitly.

The following hints describe how the five conditions can be proved. The descriptions assume that you have expanded all predicate definitions.

A is a simple quantifier proof that can be performed using `split`, `flatten`, `skolem!`, `assert`.

B1 is even simpler than A.

B2 and B3 The proofs of both formulas are very similar. All but one branches of the proof run through automatically, but this one branch requires your attention. It needs for its proof an additional lemma

$$(\forall i : 0 \leq i < \mathit{length}(a) - 1 \Rightarrow a[i] \leq a[i+1]) \Rightarrow \\ (\forall i, j : 0 \leq i \leq j < \mathit{length}(a) \Rightarrow a[i] \leq a[j]).$$

Formulate this lemma as an additional formula `L` (which you need not prove) and introduce it into the proof by the use of the `lemma` command. Then apply the `split` command on this formula, which gives you one branch which is trivially fulfilled (because the antecedent of the lemma is satisfied), and another branch, where the consequent of the lemma is stated which you can use in the remaining proof. Apart from that, the proof only requires `split`, `flatten`, `skolem!`, `assert`, `inst`.

C requires a case distinction (command `case`) on r according to the two possible outcomes of the search. Otherwise it is a straight-forward quantifier proof which requires `split`, `flatten`, `skolem!`, `assert`, `inst` only.

I suggest that you first prove `A`, `B1`, and `C` and then `B2` and `B3`. All proofs have been checked, thus there is no reason that they should not work for you. However, if some proof does not work out completely, just submit the partial proof as your exercise result.

2 Binary Search (JML Specification)

Write a JML header specification for the method

```
class Searching
{
  static int binarySearch(int[] a, int x)
  {
    int low = 0;
    int high = a.length-1;
    while (low <= high)
    {
      int mid = (low + high)/2;
      if (a[mid] == x) return mid;
      if (x > a[mid])
        low = mid+1;
      else
        high = mid-1;
    }
    return -1;
  }
}
```

which looks in an array a which is sorted in ascending order for a value x and returns its index r (-1, if x does not occur in a).

Make this specification as strong as possible using the Hoare triple from the previous exercise as a hint (but also think about extra problems that might arise in the Java method).

Run your specification through `jml` and `escjava2` and include the output of these runs in the result of this exercise.