Formal Methods in Software Development
Exercise 2 (November 23)

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

October 30, 2009

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,
  - the derivation of the verification conditions (not only the finally generated conditions),
  - the definition of the loop invariant,
  - a listing 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) file used for the proof;
- the proof directory generated by the RISC ProofNavigator.

Exercise 2: Merging Two Arrays

Take the following Hoare triple which describes how two sorted arrays a and b of length n are merged into a sorted array c of length $2n$ (consider a, b, c as arrays of natural numbers).

$$\{a = olda \wedge b = oldb \wedge n = oldn \wedge \\ length(a) = n \wedge length(b) = n \wedge length(c) = 2n \wedge \\ sorted(a, n) \wedge sorted(b, n)\}$$

```
i = 0; j = 0; k = 0;
while (k < 2*n) {
  if (i >= n) {
    c[k] = b[j]; j = j+1;
  }
  else if (j >= n) {
    c[k] = a[i]; i = i+1;
  }
  else if (a[i] <= b[j]) {
    c[k] = a[i]; i = i+1;
  }
  else {
    c[k] = b[j]; j = j+1;
  }
  k = k+1;
}
```

$$\{a = olda \wedge b = oldb \wedge n = oldn \wedge length(c) = 2n \wedge \\ sorted(c, 2n) \wedge contains(c, a, b, n, n)\}$$

The predicate $sorted(a, n)$ expresses "array a is sorted in the first n positions":

$$sorted(a, n) :\Leftrightarrow \forall i, j : 0 \leq i \leq j < n \Rightarrow a[i] \leq a[j]$$

The predicate $contains(c, a, b, i, j)$ expresses "array c contains in the first $i + j$ positions the first i elements of array a and the first j elements of array b" (we omit its formal definition). Furthermore, we define a predicate

$$isBounded(c, a, b, i, j, n) :\Leftrightarrow \\ \forall k, l : k < i + j \Rightarrow \\ (i \leq l \wedge l < n \Rightarrow c[k] \leq a[l]) \wedge (j \leq l \wedge l < n \Rightarrow c[k] \leq b[l])$$

which expresses that all elements of c at positions less than $i+j$ are less than or equal to all elements of a from position i on and less than or equal to all elements of b from position j on (draw some figures to understand the definitions).

Now for your tasks: first, assume you are given a loop invariant I. Using I, derive the six conditions for verifying the partial correctness of the triple (among these four for the conditional branches in the loop body). Please note that

you have to derive from the precondition that is stated in above triple the precondition that holds at the start of the loop.

Second, give a suitable definition of I using above predicates.

Hint: first, study the loop's precondition to determine which parts can be transferred to the invariant without change and which parts have to be modified such that they hold before every loop iteration. Second, consider the postcondition and investigate which parts can be transferred to the invariant and which parts have to be modified such that they hold after every loop iteration. Finally, give suitable range conditions respectively relationships among the local variables i, j, k and among the arrays c, a, b that hold before and after every loop iteration. For this purpose use the predicates given above.

Third, develop a RISC ProofNavigator theory that includes the "array" theory presented in class, definitions of the predicates *sorted* and *isBounded*, a declaration of predicate *contains* (which needs not be defined), definitions of the predicates for the loop's precondition, its invariant, and its postcondition, and finally the six verification conditions.

Finally, prove *three* of the verification conditions: the condition that the loop's precondition implies the invariant, the condition that the invariant on termination of the loop implies the postcondition, and the condition that the invariant is preserved by the *last conditional branch* in the loop. Whenever in a proof a subproof of goal `contains(...)` or `isBounded(...)`is required, the corresponding subproof may remain open (the proof is then not fully closed).

All proofs only require expanding definitions, scattering, splitting (always split disjunctions/implications in the knowledge), case distinctions (see below), and manual/automatic instantiations; some proof branches can be terminated by expanding (simultaneously) the definitions of the functions `put`, `get`, `length`, and `content` in the theory of arrays (do not expand these definitions, unless they immediately close the proof).

In the third proof, there is only one difficult branch in which you have to prove a goal of form *sorted*(...). Here, use `lemma` to import the propositions `get1` and `get2` from the theory of arrays, Then, (after expanding/scattering/splitting) proof goals with occurrences of terms $get(put(c, x, e), y)$ (for some subterms x, y, e) appear. Use `case` $x = y$ to split the proof and instantiate in each branch one of `get1` or `get2` to simplify the subterm. Proceed in the same way until no more subterm of this form is present in the goal. The rest of the proof proceeds by further splitting and instantiation of some universally quantified knowledge.

Hint: the first two proofs are rather short. As for the third condition, there exists a proof of moderate size (less than 20 command invocations). If you cannot complete some proof, submit the partial proof you were able to construct.

Bonus (25%): also prove the goal of form `isBounded`(...). The proof proceeds by scattering, splitting, case distinction, expansion of definitions, and instantiations. It may be sometimes wise to apply the command `goal`.