Formal Methods in Software Development
Exercise 7 (December 19)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

November 27, 2011

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,
2. the JML-annotated Java files developed in the exercise,
3. the proof files generated by the KeY prover (use the menu option "Save").

Email submissions are *not* accepted.

Exercise 7a: Merging Sorted Arrays

Formalize the specification of the method `merge` in the attached class `Exercise7a` using the JML *heavy-weight* format by giving a precondition (`requires`), frame condition (`assignable`), and postcondition (`ensures`). Also give the loop an appropriate invariant (`loop_invariant`) and termination term (`decreases`). Furthermore, give the class a function `main` that allows you to test the code by some calls of `merge`.

Now use `jml` to type-check the specification. Then use the JML runtime assertion compiler `jml4c` (or `jmlc`) and assertion checker `jml4crun` (or `jmlrac`) to validate the specification by at least five calls involving the null array, an array of length zero, an array of length one, an array of length two, and longer arrays. Then use the extended static checker `escjava2` to further validate your specification (which may or may not give warnings which you may or may not ignore).

If you are confident with your specifications, provide the loop also with an `assignable` clause (which is not standard JML but needed by KeY). Then verify the method with the KeY prover (verification condition `EnsuresPost`, i.e. the postcondition of the method body and the termination of the method). If your specification is correct, the proof should run through with less than ten iterations of automatic proof search and simplification (use all available SMT solvers simultaneously, not only "Simplify"). If you cannot complete the proof, investigate the proof tree to find out what went wrong and reconsider your specification/invariant/termination terms.

The deliverables of this exercise consist

- a nicely formatted copy of the JML-annotated Java code used for running the KeY prover (including the `assignable` clause for the loop),
- the output of running `jml4c`/`jml -Q` on the class,
- the output of running `jml4crun`/`jmlrac` on the class,
- the output of running `escjava2` on the class and your interpretation of that output,
- a screenshot of the KeY prover when the proof has been completed (respectively with an open state if you could not complete the proof),
- an explicit statement where you say whether you could complete the KeY proof or not (and how many states have remained open) and optionally any explanations or comments you would like to make.

Make sure that in the KeY "Proof Search Strategy" the options loop reasoning by invariants and method reasoning by contracts are selected (and do not fiddle with the options otherwise).

Exercise 7b: QuickSort

Proceed in the style of Exercise 7a by

1. specifying both methods `sort` and `partition` in the attached class `Exercise7b`,
2. validating the specifications by runtime assertion checking and extended static checking, and
3. verifying the correctness of methods `sort` (both) by formal proof.

For the purpose of this exercise, in the specification of `sort` it suffices to say that the result array is sorted; you need not state that the result array is a permutation of the original array.

Then provide `partition` with an appropriate invariant and termination term and verify also its correctness.