Formal Methods in Software Development
Exercise 8 (January 14)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

December 14, 2012

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").

Exercise 8: JML Specification & Verification of Insertionsort

Formalize the specifications of the methods `sort` and `insert` in the attached class `Exercise8` in the JML *heavy-weight* format by a precondition (`requires`), frame condition (`assignable`), and postcondition (`ensures`). Furthermore, give the class a function `main` that allows you to test the code by some calls of method `sort`. 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.

First use `jml` to type-check the specification. Then use the JML runtime assertion compiler `jmlc` and assertion checker `jmlrac` (respectively `jml4c` and `jml4crun`) to validate the specification by at least five calls of method `sort` including a null array, an array of length zero, an array of length one, an array of length two, and a longer array. 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).

When you are confident with your specifications, provide the loop in method `sort` with an appropriate invariant (`loop_invariant`) and termination term (`decreases`) and repeat the checks above. When you are confident about the correctness of these annotations, provide the loop also with an `assignable` clause (which is not standard JML but needed by the KeY prover). Then verify the methods with KeY (verification condition `EnsuresPost`, i.e. the postcondition of the method body and the termination of the method).

Recommendation: you may split a conjunction invariant into multiple `loop_invariant` statements; then it is easier to determine which part of an invariant failed.

If your annotations are correct and sufficiently strong, the proofs should run through with less than five iterations of “Run” and “Run Z3, Simplify, . . . ” (use all available SMT solvers simultaneously, not only “Simplify”). Be sure that in tab “Proof Search Strategy” the options “Loop treatment: Invariant” and “Method treatment: Contracts” are selected (and don’t fiddle with the options otherwise). If you cannot complete the proof, investigate the proof tree to find out what went wrong and reconsider your specification/invariant/termination terms. Please be aware that for this proof it is critical that `insert` is specified as strongly as possible.

Then provide also the loop in method `insert` with the required annotations, check them, and verify the correctness of the postcondition of the method. This proof is independent of the proof of `sort`, so it may succeed, even if the other one failed (and vice versa).

The deliverables of this exercise consist

- a nicely formatted copy of the JML-annotated Java code used for the following checks,
- the output of running `jml -Q` on the class,
- the output of running `jmlrac`/`jml4run` on the class,
- the output of running `escjava2 -NoCautions` on the class,
- a nicely formatted copy of the JML-annotated Java code used for running the KeY prover,

- for each proof, a screenshot of the KeY prover when the proof has been completed (respectively with an open state if you could not complete the proof),
- for each 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.

Bonus (20 points): State in an extended version of the specification (separate from the original one) that the sorted array is a permutation of the original one, i.e. that there is an index permutation that maps all indices of the old array to indices with the same values in the new array; also adapt the loop invariant correspondingly. An index permutation is an integer array of length n where each of the indices $0 \ldots n-1$ occurs exactly once. Attempt a KeY proof for this specification, show by a screenshot how far the proof attempt went (it will not succeed automatically), and based on an investigation of the proof tree explain why you think the prover failed.