Formal Methods in Software Development
Exercise 8 (December 18)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

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,
   - a section for each part of the exercise with the requested deliverables and optionally any explanations or comments you would like to make;
2. the JML-annotated `.java` file(s) used in the exercise,
3. the proof files generated by the KeY prover (use the menu option "Save").

Email submissions are *not* accepted.

Exercise 8: JML Verification with KeY

Take from Exercise 7 the JML-annotated program functions maximumPosition, maximumElement1, maximumElement2, insert, replace, and subtract1. Perform for each of these functions the tasks described below (you may use a single Java class file for all functions).

Annotate every loop in the function with an appropriate invariant (loop_invariant) and termination term (decreases) and check these with escjava2/openjmlesc. It is recommended to use multiple loop_invariant statements for each conjunct of the invariant; then it is easier to determine which part of an invariant failed. In the case of a for loop, do not forget to add the range condition for the loop variable to the invariant. If an array is modified, do not forget to specify which part of the array has remained unchanged so far.

When you are confident about these annotations, provide the loop also with an assignable clause (which is not standard JML but needed by the KeY prover) that lists all variables/array contents changed in the loop; in the case of a for loop, do not forget to add the loop variable to this clause. Then verify the method with KeY.

If your annotations are correct and sufficiently strong, the proofs should run through automatically with a few invocations of the KeY prover (be sure that in tab "Proof Search Strategy" the "Defaults" options are selected; you may want to reduce the "Max. Rule Applications" to speed up the proof search). After each proof search, you may also attempt to apply an SMT Solver (I recommend Z3) to close some proof obligations. If you cannot complete the proof, investigate the proof tree to find out what went wrong and reconsider your annotations (they may be wrong, i.e, too strong, or too weak); for this purpose, you may unselect the option "Hide intermediate proof steps" in the context menu of the proof tree in order to see all simplification steps performed by the prover. If you cannot complete the proof, explain in detail which part of the verification failed and what you believe is the reason for the failure.

Optional: you may validate your specifications/loop invariants by translating the Java functions to RISCAL procedures, equip them with specifications and loop annotations, and additionally check these (this is recommended, if a KeY proof fails). For each such RISCAL specification/check, you get 5P bonus, 10P in case of insert and replace, and 15P in case of subtract. Please note that the RISCAL versions of replace and subtract do not modify their argument arrays but return a corresponding result array (additionally to the result of the Java function, i.e., the RISCAL procedure returns a Tuple value).

The deliverables of this exercise consist of

- a nicely formatted copy of the JML-annotated Java code (the version with the assignable clauses used for running KeY),
- the output of `jml -Q` on the class,
- the output of `escjava2 -NoCautions/openjmlesc` on the class,
- for each function, an explicit statement where you say whether you could complete the verification or not (and how many proof branches have remained open)

- for each function, a screenshot of the KeY prover when the proof has been completed (or got stuck) illustrating the generated proof tree (without the intermediate steps) with emphasis of the still open proof branches (if any),
- for each open proof branch a screenshot of the proof obligation, an explanation of the role of this obligation in the overall verification, and your conjecture why the proof failed.

Please also report any observations or insights you have gained.