Formal Methods in Software Development
Exercise 2 (October 28)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

September 18, 2015

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 and email address,
   - the deliverables requested in the description of the exercise,
2. the JML-annotated Java files developed in the exercise.

Email submissions are *not* accepted.

Exercise 2: Verifying JML Specifications

Annotate the method given in the file `Exercise2.java` by a JML specification in the style of Exercise 1 (it is not necessary to give a main function). Additionally, annotate the loop with an invariant (clause `loop_invariant`) and termination term (clause `decreases`).

Hint: the loop invariant should contain all information that you have about the variables:

- the (non-)nullness status of the array,
- the range of the loop variable,
- the content of the array before the position indicated by the loop variable (i.e, the already processed part),
- the content of the array starting with the position indicated by the loop variable (i.e., the not yet processed part).

First, type-check the specification with `jml` and check it with `escjava2`/`openjmlesc` in the style of Exercise 1.

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; 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 proof should run through automatically with a single of the KeY prover (be sure that in tab "Proof Search Strategy" the "Defaults" options are selected). If the proof does not run through, 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.

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,
- an explicit statement where you say whether you could complete the KeY verification or not (and how many proof branches have remained open)
- a screenshot of the KeY prover when the proof has been completed (or got stuck) illustrating the generated proof tree,
- 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.