Formal Methods in Software Development
Exercise 5 (December 14)

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

November 24, 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,
  - for each exercise, a section with the number and name of the exercise, the JML-annotated Java code, a copy of the output of an `escjava2` check of that code, and a screen shot 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;
- the JML-annotated Java files developed in the exercise,
- the proof files generated by the KeY prover (use the menu option "Save").

5a (50 points): JML specification and KeY Proof

Consider the function `invert` in class `Exercise5a` (put the attached source file into a subdirectory `exercises` and check/verify it with the *parent directory* as the current working directory).

Write for this function a JML header specifications that is as expressive as possible and provide the loop with a suitable invariant and termination term (but no `assignable` clause yet). In the loop invariant use the dummy variable `olda` rather than the term `\old(a)` to refer to the old value of a (because KeY does not allow the use of `\old` in invariants).

Type-check the specification with `jml -Q` (which must not give an error) and then statically check it with `escjava2`. If this gives a warning, reconsider your specification (perhaps you have made an error, but perhaps the warning is also superfluous).

If you are confident with your specification, provide the loop also with an assignable clause and verify the specification with the KeY prover (verification condition `EnsuresPost`, i.e. the postcondition of the method body and the termination of the method). If your specifications are correct, the proofs should run through with repeated applications of "Run" (and optionally "Simplify"). If you cannot complete the proof, reconsider your specification.

5b (50 points): JML specification and KeY Proof

Proceed as for Exercise 5a with the function `merge` in class `Exercise5b` (for this task it is not necessary to use the dummy variable); this proof proceeds by repeated applications of "Run" (the application of "Simplify" is not needed and not recommended, since it may here take a long time).

In the method specification, you do not have to specify that the result array contains the same elements as the parameter arrays a and b, only that it is sorted. As for the property of being sorted, I recommend to use the definition "an array a is sorted, if for any array position i prior to another array position j, array element $a[i]$ is less than or equal array element $a[j]$.

In the loop invariant, it is necessary to formulate

- all the knowledge you have about a and b,
- the information you have about m, n, and the size of c,
- the ranges of i and j, and the relationship between i, j, and k,
- the information about the already processed part of c (up to position k),
- the relationship between the already processed part of c and the not yet processed part of a (from position i on),
- the relationship between the already processed part of c and the not yet processed part of b (from position j on).

Do not forget to state the requirement that all array pointers are not null.