ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

J. Hoenicke
J. Christ

11.01.2012
Hand in solutions via email to
christj@informatik.uni-freiburg.de
until 18.01.2012 (only Java sources, KeY
proofs, and PDFs accepted).
Paper submissions possible after the lecture.

Tutorials for "Formal methods for Java"
Exercise sheet 9

Exercise 1: Relational semantics of loop free guarded commands
In the lecture we defined the semantics of loop free guarded commands in terms of weakest preconditions. Alternatively we can give a relational semantics that defines guarded commands as relations on states:

$$\begin{array}{lcl} [\![x := e]\!] & = & \{\, (s, s') \mid s' = s[x := s[\![e]\!]] \,\} \\ [\![\mathsf{havoc}(x)]\!] & = & \{\, (s, s') \mid \exists v. s' = s[x := v] \,\} \\ [\![\mathsf{assert}(G)]\!] & = & \{\, (s, s_{err}) \mid s \not\models G \,\} \cup \{\, (s, s) \mid s \models G \,\} \\ [\![c_1 \,\square\, c_2]\!] & = & [\![c1]\!] \cup [\![c_2]\!] \end{array}$$

Hereby s_{err} is a special error state that does not satisfy any assertion, i.e. $s_{err} \not\models F$ for any assertion F. Give proper relational semantics to the missing guarded commands.

Exercise 2: Weakest preconditions
We can define weakest preconditions in terms of the relational semantics of loop free guarded commands:

$$\mathsf{wp}(c, S) = \{\, s \mid \forall s'. (s, s') \in [\![c]\!] \rightarrow s' \in S \,\}$$

Prove that the weakest precondition semantics given in the lecture is correct with respect to the relational semantics for the cases: $\mathsf{assert}(\mathsf{G})$ and $\mathsf{c_1} \,\square\, \mathsf{c_2}$, i.e. prove:

(a) $\mathsf{wp}(\mathsf{assert}(G), [\![F]\!]) = [\![G \wedge F]\!]$, and

(b) $\mathsf{wp}(c_1 \,\square\, c_2, [\![F]\!]) = \mathsf{wp}(c_1, [\![F]\!]) \cap \mathsf{wp}(c_2, [\![F]\!])$

where $[\![F]\!] = \{\, s \mid s \models F \,\}$.

Exercise 3: Translation to loop free guarded commands
Translate the following Java code fragment into loop free guarded commands:

```
result = 0;
s = 0;
while /*: inv "s = result * result & n > 0 & result >= 0 & s >=0 &
               n > s - 2 * result" */
 (s < n) {
  result = result + 1;
  s = s + 2 * result - 1;
}
return (s == n ? result : result - 1);
```