J. Hoenicke
J. Christ

11.12.2012
Hand in solutions via email to
christj@informatik.uni-freiburg.de
until 18.12.2012 (only Java sources and PDFs accepted)

Tutorials for "Formal methods for Java"
Exercise sheet 8

Exercise 1: Friendship

Consider an implementation of a list in Java that provides iterators to access the elements of the list in the order in which they were inserted. The list can serve as backend to multiple iterators that traverse the list independent of each other.
To ensure correctness of the traversal order, all Java library classes provide *fast-fail* iterators. These iterators throw a `ConcurrentModificationException` if the collection traversed by the iterator is changed while iterating over it. To implement this behavior, the list keeps track of a modification counter and every iterator has a local copy of the value of the modification counter. Every operation that modifies the content of the list first increments the modification counter before changing the content of the list. If the value of the copy in the iterator and the counter in the list differ, iteration is not guaranteed to be correct anymore, and an exception is thrown.
To successfully apply this pattern, the modification counter may only be increased and never decreased. The desired invariant for the list iterator is shown in the following listing.

```
class ListIter extends Iterator {
  //@ghost JMLObjectSequence theContent;
  List theList;
  int expectedModCount;
  /*@ invariant expectedModCount <= theList.modCount &&
                expectedModCount == theList.modCount ==>
                  theContent.equals(theList.theContent);
    @*/
    // Further fields and methods
}
```

(a) Describe using Slide 20 from Lecture 13 why the ownership model cannot be used to realize this invariant in the given setting.

(b) State the two parties, their roles, the granted fields, and the update guards such that the friendship model can be used to check this invariant. Modify the invariant to comply to the friendship model.

Exercise 2: 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 3: 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 $[\![F]\!] = \{\, s \mid s \models F \,\}$.