Prof. Dr. Andreas Podelski
Dr. Matthias Heizmann
Christian Schilling

Preference deadline: July 24, 2017
Discussion: July 26, 2017

Tutorial for Program Verification

Exercise Sheet 12

We know that you have worked very hard during the semester and that you feel exhausted. This is why we have tried very hard to come up with exercises that should be fun and should not take too much time.

Exercise 1: Transition invariants — 2 Points

Let R be a transition relation. We say that a binary relation T is *inductive* if $R \subseteq T$ and $R \circ T \subseteq T$. We can adapt the definition of *inductiveness* to a set of abstract transitions $\{T_1, \dots, T_n\}$ in the following two ways.

Definition 1 We call $\{T_1, \dots, T_n\}$ *inductive* if there exists some j such that $R \subseteq T_j$ and for all i there exists some j such that $R \circ T_i \subseteq T_j$.

Definition 2 We call $\{T_1, \dots, T_n\}$ *inductive* if $R \subseteq T_1 \cup \dots \cup T_n$ and $R \circ (T_1 \cup \dots \cup T_n) \subseteq T_1 \cup \dots \cup T_n$.

(a) Are both definitions equivalent? If not, give a counterexample.

(b) For which of the two defintions above is the set of abstract transitions $P^\#$ computed by the TPA algorithm inductive? Prove your claims.

Exercise 2: Completeness of termination proofs — 1 Point

In the lecture we discussed the statement that both

- ranking functions and
- well-founded relations

are "complete", i.e., if a program is terminating, we can always find a ranking function and we can find always find a well-founded relation that can be used to prove termination of the program.

Prove this statement.

Exercise 3: Transitive closure — 1 Point

In the lecture we implicitly used the following fact.

For any relations R_1, R_2 over the same domain the following holds.

$$R_1 \subseteq R_2 \land R_2 = R_2^+ \implies R_1^+ \subseteq R_2$$

As usual, R^+ denotes the transitive closure of a relation R.

Show that this fact holds.

Exercise 4: Program transformation 1 Point

In the lecture we have discussed a transformation of programs.
For a variable `x` we introduced a variable `'x` (called "old `x`") and in front of each assignment to `x` we added the following command: `if (*) 'x := x`.
One must be careful how to generalize this transformation to the case where the program has more than one variable. Assume that we have two variables `x` and `y`.
(Strictly speaking, going from the case of one to two variables is not a generalization but rather an extension.)

(a) Show that the following generalization does not work.

Insert the following command in front of every assignment to `x` *and* `y`:

```
if (*) 'x := x;
if (*) 'y := y
```

Hint: The problem is related to the Cartesian product.

(b) Show that the following generalization does not work.

Insert the command `if (*) 'x := x` *in front of every assignment to* `x`*, and insert the command* `if (*) 'y := y` *in front of every assignment to* `y`.

(c) Are the above versions equivalent?

(d) Describe a generalization from the case of one to two variables that works.

Exercise 5: Abstract transition 1 Point

In the lecture we said that an abstract transition can be more precise (as an abstraction of a binary relation over states) than a transition between abstract states. Give an example for this claim.

Hint: The problem is related to the Cartesian product.