Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Peter Lammich, Johannes Hölzl

WS 2013/14
12. 11. 2013

Semantics of Programming Languages

Exercise Sheet 5

Due to SVV, there has been no tutorial on November 12th. These exercises will be discussed in the tutorial on November 19th. The homework is due on November 26th. You may want to wait until the tutorial before attempting the homework. Note that the bonus homework is independent of the material discussed in the next tutorial.

Exercise 5.1 Program Equivalence

Prove or disprove (by giving counterexamples) the following program equivalences.

1. *IF And b1 b2 THEN c1 ELSE c2* $\sim$ *IF b1 THEN IF b2 THEN c1 ELSE c2 ELSE c2*
2. *WHILE And b1 b2 DO c* $\sim$ *WHILE b1 DO WHILE b2 DO c*
3. *WHILE And b1 b2 DO c* $\sim$ *WHILE b1 DO c*;; *WHILE And b1 b2 DO c*
4. *WHILE Or b1 b2 DO c* $\sim$ *WHILE Or b1 b2 DO c*;; *WHILE b1 DO c*

Hint: Use the following definition for *Or*:

definition *Or* :: *"bexp ⇒ bexp ⇒ bexp"* **where**
"Or b1 b2 = Not (And (Not b1) (Not b2))"

Exercise 5.2 Nondeterminism

In this exercise we extend our language with nondeterminism. We will define *nondeterministic choice* (c_1 *OR* c_2), that decides nondeterministically to execute c_1 or c_2; and *assumption* (*ASSUME b*), that behaves like *SKIP* if *b* evaluates to true, and returns no result otherwise.

1. Modify the datatype *com* to include the new commands *OR* and *ASSUME*.
2. Adapt the big step semantics to include rules for the new commands.
3. Prove that c_1 *OR* $c_2 \sim c_2$ *OR* c_1.
4. Prove: (*IF b THEN c1 ELSE c2*) $\sim$ ((*ASSUME b*; *c1*) *OR* (*ASSUME* (*Not b*); *c2*))
5. Adapt the small step semantics, and the equivalence proof of big and small step semantics.

Note: It is easiest if you take the existing theories and modify them.

Homework 5.1 Exceptions

Submission until Tuesday, November 26, 10:00am.

Extend IMP with exceptions. Add two constructors *THROW* and *TRY c1 CATCH c2* to datatype *com*.

Use the template we provide on the web-page! Only submit a single file!

Command *THROW* throws an exception. The only command that can catch an exception is *TRY c1 CATCH c2*: if an exception is thrown by *c1*, execution continues with *c2*, otherwise *c2* is ignored. Adjust the definitions of big-step and small-step semantics as follows.

The big-step semantics is now of type *com* × *state* ⇒ *com* × *state*. In a big step $(c,s) \Rightarrow (x,t)$, x can only be *SKIP* (signalling normal termination) or *THROW* (signalling that an exception was thrown but not caught).

Adjust the proof of theorem *big_step_determ*, showing that big-step semantics is deterministic.

The small-step semantics is of the same type as before. There are two final configurations now, (*SKIP*, t) and (*THROW*, t). Exceptions propagate upwards until an enclosing handler is found. That is, until a configuration (*TRY THROW CATCH* c, s) is reached and *THROW* can be caught.

Adjust the statement *final* $(c,s) \longleftrightarrow c = SKIP$ and its proof.

Adjust the equivalence proof between the two semantics such that you obtain $cs \Rightarrow cs' \longleftrightarrow (cs \rightarrow\!* \ cs' \wedge final\ cs')$.

Homework 5.2 Paths in Graphs

Submission until Tuesday, November 26, 10:00am. This homework is worth 5 bonus points.

In Homework 4.2, we formulated paths in graphs. Now prove, that for any path from u to v, there is also a path from u to v that contains each node at most once.

Hint: Theorems *not_distinct_decomp* and *length_induct* may help.

```
inductive is_path :: "('v×'v) set ⇒ 'v ⇒ 'v list ⇒ 'v ⇒ bool"
for E where
  NilI: "is_path E u [] u"
| ConsI: "⟦ (u,v)∈E; is_path E v l w ⟧ ⟹ is_path E u (u#l) w"

lemma path_distinct:
  assumes "is_path E u p v"
  shows "∃p'. distinct p' ∧ is_path E u p' v"
```