Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Andrei Popescu, Peter Lammich

WS 2012/13
13. 11. 2012

Semantics of Programming Languages

Exercise Sheet 5

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* $\Rightarrow$ *bexp* $\Rightarrow$ *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 want to include a command c_1 *OR* c_2, which expresses the nondeterministic choice between two commands. That is, when executing c_1 *OR* c_2 either c_1 or c_2 may be executed, and it is not specified which one.

1. Modify the datatype *com* to include a new constructor *OR*.
2. Adapt the big step semantics to include rules for the new construct.
3. Prove that c_1 *OR* $c_2 \sim c_2$ *OR* c_1.
4. 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 Nondeterminism

Submission until Tuesday, November 20, 2012, 10:00am.

Note: We will provide a template for this homework on the course webpage

In this homework, we will explore various nondeterministic commands. A nondetermistic command may have multiple (or no) results.
We will define *nondeterministic assignment* ($x ::= *$), that assigns some arbitrary value to x; *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.

In the following we provide the syntax for the new commands:

```
datatype
  com = SKIP
      | Assign vname aexp      ("_ ::= _" [1000, 61] 61)
      | Ndet vname             ("_ ::= *" [1000] 61)
      | Semi   com  com        ("_;/ _"  [60, 61] 60)
      | If     bexp com com    ("(IF _/ THEN _/ ELSE _)"  [0, 0, 61] 61)
      | While  bexp com        ("(WHILE _/ DO _)"  [0, 61] 61)
      | Or com com             ("_ OR _" [57,58] 59)
      | ASSUME bexp
```

Task 1 Extend the big-step semantics by rules for the new commands:

```
inductive
  big_step :: "com × state ⇒ state ⇒ bool" (infix "⇒" 55)
where— Add your rules here
```

Task 2 As a warm-up, show that *OR* is commutative

lemma *or_comm*: *"c1 OR c2 ∼ c2 OR c1"*

A similar property also holds for chained nondeterminsitic assignments. Prove it!

lemma *ndet_semi_comm*: *"x::=*; y::=* ∼ y::=*; x::=*"*

Hint: You may need an auxiliary lemma that allows you to swap updates of the state

Task 3 If-commands can be translated to assumption and nondeterministic choice as follows:

lemma *sim_if_or*:
  "(IF b THEN c1 ELSE c2) ∼ ((ASSUME b; c1) OR (ASSUME (Not b); c2))"

Prove this lemma!

Finally, define a function that eliminates all if-commands in a given command, and prove that it preserves the semantics:

fun *cnv* :: *"com ⇒ com"* **where**
lemma *"cnv c ∼ c"* **oops**

Hint: You may need auxiliary lemmas like the following one, that we have already proved for you.

lemma *sim_commute_while*:
  assumes *SIM*: *"c∼c'"*
  shows *"WHILE b DO c ∼ WHILE b DO c'"*