Exercise 5.1  Program Equivalence

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

1. \[ \text{IF And } b_1 b_2 \text{ THEN } c_1 \text{ ELSE } c_2 \sim \text{IF } b_1 \text{ THEN IF } b_2 \text{ THEN } c_1 \text{ ELSE } c_2 \text{ ELSE } c_2 \]
2. \[ \text{WHILE And } b_1 b_2 \text{ DO } c \sim \text{WHILE } b_1 \text{ DO WHILE } b_2 \text{ DO } c \]
3. \[ \text{WHILE And } b_1 b_2 \text{ DO } c \sim \text{WHILE } b_1 \text{ DO } c; \text{WHILE And } b_1 b_2 \text{ DO } c \]
4. \[ \text{WHILE Or } b_1 b_2 \text{ DO } c \sim \text{WHILE Or } b_1 b_2 \text{ DO } c; \text{WHILE } b_1 \text{ DO } c \]

Hint: Use the following definition for Or:

\[
\text{definition Or :: } \text{“bexp } \Rightarrow \text{ bexp } \Rightarrow \text{ bexp” where “Or } b_1 b_2 = \text{Not (And (Not } b_1 \text{) (Not } b_2\text{))”}
\]

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 \( \text{com} \) 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.

\textbf{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 nondeterministic 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 
\[ \begin{array}{l}
com = SKIP \\
| Assign\ vname\ aexp\ ("::="\ [1000, 61]) \\
| Ndet\ vname\ ("::="\ [1000, 61]) \\
| Semi\ com\ com\ ("/"\ [60, 61]) \\
| If\ bexp\ com\ com\ ("(IF \ THEN \ ELSE \))\ [0, 0, 61]) \\
| While\ bexp\ com\ ("(WHILE \ DO \))\ [0, 61]) \\
| Or\ com\ com\ ("\ OR \")\ [57, 58, 59]) \\
| ASSUME\ bexp
\end{array} \]

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

inductive 
\[ \text{big\_step} :: \ "\ com \times\ state \Rightarrow\ state \Rightarrow\ bool\" \quad (\text{infix} \ "\Rightarrow\"\ 55) \]
where—Add your rules here.

Task 2 As a warm-up, show that \( OR \) is commutative

\[ \text{lemma or\_comm} : \ "c1 \ OR \ c2 \sim c2 \ OR \ c1" \]

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

\[ \text{lemma ndet\_semi\_comm} : \ "x::=*; y::=* \sim y::=*; x::=*" \]

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:

\[ \text{lemma sim\_if\_or} : \ "(IF\ b\ THEN\ c1\ ELSE\ c2) \sim \ ((ASSUME\ b; c1) \ OR \ (ASSUME\ (Not\ b); c2))" \]

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’”