Semantics of Programming Languages
Exercise Sheet 5

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 $\text{Or}$:

\[
\text{definition Or :: } \text{“bexp } \Rightarrow \text{bexp } \Rightarrow \text{bexp” where}
\]

“\text{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 $\text{OR}$.

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  Step-Index Semantics

Submission until Wednesday, November 30, 2011, 12:00 (noon).

Note: In order to save you some typing, we provide a template for this homework on the lecture’s homepage.

In this homework, a denotational semantics for while-programs will be defined, i.e., a function that takes a command and a state, and returns the result state.

In order to make this function well-defined even for non-terminating programs, it is parameterized with an additional number, that indicates the maximum number of steps to make. If the program has not yet terminated after this many steps, \( \text{None} \) is returned.

\[
\text{fun } \text{si} :: \text{"com } \Rightarrow \text{ state } \Rightarrow \text{ nat } \Rightarrow \text{ state option" where}
\]

\[
\text{si}_{\text{None}}: \text{"si } s 0 = \text{None" |}
\]

\[
\text{si}_{\text{SKIP}}: \text{"si } \text{SKIP } s \text{ (Suc } i) = \text{Some } s" |
\]

\[
\text{si}_{\text{ASS}}: \text{"si } (x:=v) s \text{ (Suc } i) = \text{Some } (s(x:=\text{aval } v s))" |
\]

\[
\text{si}_{\text{SEMI}}: \text{"si } (c1;c2) s \text{ (Suc } i) = (\text{case } \text{si } c1 s i \text{ of } \text{None } \Rightarrow \text{None } | \text{Some } s’ \Rightarrow \text{si } c2 s’ i)" |
\]

\[
\text{si}_{\text{IF}}: \text{"si } (\text{IF } b \text{ THEN } c1 \text{ ELSE } c2) s \text{ (Suc } i) = (\text{if } b\text{val } b s \text{ then } \text{si } c1 s i \text{ else } \text{si } c2 s i)" |
\]

\[
\text{si}_{\text{WHILE}}: \text{"si } (\text{WHILE } b \text{ DO } c) s \text{ (Suc } i) = (\text{if } b\text{val } b s \text{ then } (\text{case } \text{si } c s i \text{ of}\n\text{None } \Rightarrow \text{None } |
\text{Some } s’ \Rightarrow \text{si } (\text{WHILE } b \text{ DO } c) s’ i)
\text{else Some } s)"
\]

Prove the equivalence of the big-step and the step-index semantics, i.e., show that

\[
(\exists i. \text{si } c s i = \text{Some } s’) \iff \text{big-step } (c,s) s’
\]

As this proof is more complicated than any proof in homeworks so far, we will give a bit of guidance:

The two directions are proved separately. The proof of the first direction should be quite straightforward, and is left to you.

\[
\text{lemma } \text{si_imp_bigstep: } \text{"si } c s i = \text{Some } s’ \implies \text{big-step } (c,s) s’"
\]

For the other direction, it is useful to prove a monotonicity lemma first. If the step-index semantics yields a result for index \( i \), it yields the same result for any \( i’ \geq i \).

\[
\text{lemma } \text{si_mono: } \text{"si } c s i = \text{Some } s’ \implies \text{si } c s (i+k) = \text{Some } s’"\]

\[
\text{proof (induction } c s i \text{ arbitrary: } s’\n\text{ rule: si.induct[case_names None SKIP ASS SEMI IF WHILE]})
\]

\[
\text{case } (\text{WHILE } b c s i s’) \text{ thus } \text{?case
}\
\]

Only the WHILE-case requires some effort. Hint: Make a case distinction on the value of the condition \( b \).

\[
\text{qed (auto split: option.split option.split_asm)}
\]
The main lemma is proved by induction over the big-step semantics. Remember the adapted induction rule $\text{big\_step\_induct}$ that nicely handles the pattern $\text{big\_step\ (c,s) s'}$.

**Lemma** $\text{bigstep\_imp\_si}$:

$\text{big\_step\ (c,s) s'} \implies \exists i. s_i c s i = \text{Some s''}$

**Proof** (induct rule: $\text{big\_step\_induct}$)

We demonstrate the skip, while-true and sequential composition case here. The other cases are left to you!

**Case** (Skip s) have “si SKIP s 1 = Some s” by auto

thus ?case by blast

next

case (WhileTrue b s1 c s2 s3)

then obtain i1 i2 where “si c s1 i1 = Some s2” and “si (WHILE b DO c) s2 i2 = Some s3” by auto

with si_mono[of c s1 i1 s2 i2] si_mono[of “WHILE b DO c” s2 i2 s3 i1] have “si c s1 (i1+i2) = Some s2” and “si (WHILE b DO c) s2 (i2+i1) = Some s3” by auto

hence “si (WHILE b DO c) s1 (Suc (i1+i2)) = Some s3” using :bval b s1 by (auto simp add: add_ac)

thus ?case by blast

next

case (Semi c1 s1 s2 c2 s3)

then obtain i1 i2 where “si c1 s1 i1 = Some s2” and “si c2 s2 i2 = Some s3” by auto

with si_mono[of c1 s1 i1 s2 i2] si_mono[of c2 s2 i2 s3 i1] have “si c1 s1 (i1+i2) = Some s2” and “si c2 s2 (i2+i1) = Some s3” by auto

hence “si (c1;c2) s1 (Suc (i1+i2)) = Some s3” by (auto simp add: add_ac)

thus ?case by blast

Finally, prove the main theorem of the homework:

**Theorem** $\text{si\_equiv\_bigstep}$: “$\exists i. s_i c s = \text{Some s''}$”