Exercise 5.1  Program Equivalence

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

1. $\text{IF } \text{And } b_1 b_2 \text{ THEN } c_1 \text{ ELSE } c_2 \sim \text{IF } b_1 \text{ THEN } \text{IF } b_2 \text{ THEN } c_1 \text{ ELSE } c_2 \text{ ELSE } c_2$

2. $\text{WHILE } \text{And } b_1 b_2 \text{ DO } c \sim \text{WHILE } b_1 \text{ DO } \text{WHILE } b_2 \text{ DO } c$

3. $\text{WHILE } \text{And } b_1 b_2 \text{ DO } c \sim \text{WHILE } b_1 \text{ DO } c;; \text{WHILE } b_1 b_2 \text{ DO } c$

4. $\text{WHILE } \text{Or } b_1 b_2 \text{ DO } c \sim \text{WHILE } \text{Or } b_1 b_2 \text{ DO } c;; \text{WHILE } b_1 \text{ DO } c$

Hint: Use the following definition for $\text{Or}$:

\begin{align*}
\text{definition } \text{Or} :: & \"bexp \Rightarrow bexp \Rightarrow bexp\" \text{ where} \\
& \"\text{Or } b_1 b_2 = \text{Not } (\text{And } (\text{Not } b_1) (\text{Not } b_2))\" \\
\end{align*}

Exercise 5.2  Nondeterminism

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

1. Modify the datatype $\text{com}$ to include the new commands $\text{OR}$ and $\text{ASSUME}$.

2. Adapt the big-step semantics to include rules for the new commands.

3. Prove that $c_1 \text{ OR } c_2 \sim c_2 \text{ OR } c_1$.

4. Prove: $(\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2) \sim ((\text{ASSUME } b; c_1) \text{ OR } (\text{ASSUME } (\text{Not } b); c_2))$

5. Optional (relies on material from next week): 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  Dijkstra’s Guarded Command Language (12 points)

Submission until Monday 08.06.2015 10:01 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE5]” in the subject of your email. If your name is Fornavn Etternavn, please call your theory file Etternavn.Fornavn.thy.

Unless indicated otherwise, please give all your proofs in Isar, not apply style.

In the 1970s, Edsger Dijkstra introduced the guarded command language (GCL), a nondeterministic programming language featuring a nondeterministic if command with the syntax

```
if b1 --> c1
| b2 --> c2
...
| bN --> cN
fi
```

where b1, b2, ..., bN are Boolean conditions and c1, c2, ..., cN are commands. When executing the statement, an arbitrary branch with a condition that evaluates to true is selected. If no condition is true, execution simply blocks.

To keep things simple, we will have no looping command in our language. Here is the Isabelle datatype:

```
datatype gcom =
  Skip
| Ass vname aexp
| Sq gcom gcom
| IfBlock "((bexp × gcom) list"
```

First, define the big-step semantics with infix syntax $\Rightarrow g$:

```
inductive
  big_stepg :: "gcom × state ⇒ state ⇒ bool" (infix "⇒ g" 55)
```

Use ~~/src/HOL/IMP/Big_Step.thy as an inspiration. Remember to give names to your introduction rules, so that you can refer to each rule by name in your proofs by rule induction.

Like in ~~/src/HOL/IMP/Big_Step.thy, we declare the introduction rules as intro rules for auto, blast, fast, fastforce, force, and extreme_violence. We also do some magic with the induction rule to make it more suitable—this is necessary only because the first argument to $⇒ g$ is tupled.

```
declare big_stepg.intros [intro]
```

```
lemmas big_stepg_induct = big_stepg_induct[split_format(complete)]
```

This will come in handy later:
inductive_cases SqE: "(Sq c1 c2, s) ⇒ g t"
inductive_cases IfBlockE: "(IfBlock Gs, s) ⇒ g t"

thm SqE IfBlockE

A useful lemma. Prove it:

lemma IfBlock_subset_big_stepg:
assumes "Gs: "(IfBlock Gs, s) ⇒ g t" and 
Gs': "set Gs ⊆ set Gs'"
shows "(IfBlock Gs', s) ⇒ g t"

Write various schematic lemmas in the style of ~/src/HOL/IMP/Big_Step.thy and try out your big-step semantics on them. In particular, try to take both branches of a two-way if block. For this part, you are allowed (indeed, encouraged) to write your proofs in apply style.

schematic_lemma ex1: "(Sq (Ass "x" (N 5)) (Ass "y" (V "x'")), s) ⇒ g s'"

....
thm ex1[simplified]

schematic_lemma ex2: "(IfBlock [(Less (N 4) (N 5), Ass "x" (N 2))], s) ⇒ g s'"

....
thm ex2[simplified]

schematic_lemma ex3a: 
"(IfBlock [(Less (N 4) (N 5), Ass "x" (N 2)), (Less (N 6) (N 7), Ass "x" (N 3))], s) ⇒ g s'"

....
thm ex3a[simplified]

schematic_lemma ex3b: 
"(IfBlock [(Less (N 4) (N 5), Ass "x" (N 2)), (Less (N 6) (N 7), Ass "x" (N 3))], s) ⇒ g s'"

....
thm ex3b[simplified]

Is the language deterministic? Prove or disprove.

lemma "(c, s) ⇒ g s' ⇒ (c, s) ⇒ g s'" ⇒ s' = s"

Is the language total? Prove or disprove.

lemma "∃ s'. (c, s) ⇒ g s'"
Next, define the semantics as a function. In cases where several guard conditions evaluate to true, it can arbitrarily select a true branch (e.g., the first true branch). If the program blocks, the function returns None.

\[
\text{fun big_stepgf :: } \text{"gcom } \Rightarrow \text{ state } \Rightarrow \text{ state option"}
\]

Specify names for the cases of the induction rule generated for the above function, to enhance the readability of Isar proofs (i.e., replace \ldots with appropriate names):

\[
\text{lemmas big_stepgf_induct } = \text{ big_stepgf_induct[case_names } \ldots ]
\]

A useful lemma. Prove it:

\[
\text{lemma big_stepgf_Some_imp_ex_inter: "big_stepgf } (\text{Sq } c \ c') \ s = \text{ Some } s'' \implies \exists s'. \ big_stepgf c \ s = \text{ Some } s' \land \ big_stepgf c' \ s' = \text{ Some } s''"
\]

Prove or disprove that the function \text{big_stepgf} is sound with respect to the inductive predicate \text{op } \Rightarrow g:\n
\[
\text{theorem big_stepgf_sound: "big_stepgf } c \ s = \text{ Some } s' \implies (c, s) \Rightarrow g \ s'"
\]

Hint: If you go for a proof, make sure to use the most appropriate induction principle, and ask yourself whether you need \text{arbitrary}:

Finally, prove or disprove completeness:

\[
\text{lemma big_stepgf_complete: "(c, s) } \Rightarrow g \ s' \implies \exists t. \ big_stepgf c \ s = \text{ Some } t"
\]

Homework 5.2 More GCL (8 points)

Submission until Monday 08.06.2015 10:01 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE5]” in the subject of your email. If your name is Fornavn Etternavn, please call your theory file Etternavn_Fornavn.thy.

Please give all your proofs in Isar, not apply style.

This is a continuation of the previous homework exercise.

Define a notion of program equivalence for GCL:

\[
\text{abbreviation equiv_cg :: } \text{"gcom } \Rightarrow \text{ gcom } \Rightarrow \text{ bool" (infix } \sim g\text{ 50)}
\]

Show that \sim g is an equivalence relation:

\[
\text{lemma reflp_equiv_cg: } \text{"reflp } (\text{op } \sim g\text{)"}
\]
\[
\text{lemma symp_equiv_cg: } \text{"symp } (\text{op } \sim g\text{)"}
\]
\[
\text{lemma transpp_equiv_cg: } \text{"transp } (\text{op } \sim g\text{)"}
\]

Prove the congruence lemma for \text{Sq}:
lemma Sq_cong:
assumes
c1: “c1 ∼ g c1’” and
c2: “c2 ∼ g c2’”
shows “Sq c1 c2 ∼ g Sq c1’ c2’”

Homework 5.3  Even more GCL (5 bonus points)

Submission until Monday 08.06.2015 10:01 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE5]” in the subject of your email. If your name is Fornavn Etternavn, please call your theory file Etternavn,Fornavn.thy.

Please give all your proofs in Isar, not apply style.

This is a continuation of the previous homework exercise. **Warning:** This is a difficult exercise. Do not spend too much time on it.

Prove the congruence lemma for IfBlock. There are many ways of stating it. Use whichever style you prefer, including these:

**lemma IfBlock_cong_v1:**
assumes allc: “∀ ( b , ( c , c’ ) ) ∈ set GGs. c ∼ g c’”
shows  “IfBlock (map (λ( b , ( c , ) ). ( b , c ) ) GGs)
        ∼ g IfBlock (map (λ(b , ( , c’ ) ). (b , c’ ) ) GGs)”

**lemma IfBlock_cong_v2:**
assumes len: “length bs = n”  “length cs = n”  “length cs’ = n”
assumes alli: “∀ i . i < n ⇒ cs ! i ∼ g cs’ ! i”
shows “IfBlock (zip bs cs) ∼ g IfBlock (zip bs cs’)”

The ! operator is syntactic sugar for nth, defined in List. It returns the (n−1)st element of a list (not the nth!).

Finally, show that the order of the elements in the guard block list and any duplicates are irrelevant:

**lemma IfBlock_set_eq_cong:**
assumes set: “set Gs = set Gs’”
shows “IfBlock Gs ∼ g IfBlock Gs’”

The end is the beginning is the end