Exercise 6.1  Small step equivalence

We define an equivalence relation ⇑ on programs that uses the small-step semantics. Unlike with ∼, we also demand that the programs take the same number of steps.

The following relation is the n-steps reduction relation:

\[
\text{inductive } n\text{-steps} :: \text{"com} \times \text{state} \Rightarrow \text{nat} \Rightarrow \text{com} \times \text{state} \Rightarrow \text{bool} \\
\text{where} \\
\text{zero_steps: } \text{"cs} \rightarrow \text{"0 cs" } | \\
\text{one_step: } \text{"cs} \rightarrow \text{cs}^\prime \Rightarrow \text{cs}^\prime \rightarrow \text{"n cs"'} \Rightarrow \text{cs} \rightarrow \text{"(Suc n) cs"’}
\]

Prove the following lemmas:

\text{lemma small_steps_n:} \quad \text{"cs} \rightarrow \ast \text{cs}^\prime \Rightarrow (\exists n. \text{cs} \rightarrow \text{"n cs"’})

\text{lemma n_small_steps:} \quad \text{"cs} \rightarrow \text{"n cs"'} \Rightarrow \text{cs} \rightarrow \ast \text{cs}^\prime

The equivalence relation is defined as follows:

\text{definition small_step_equiv :: } \text{"com} \Rightarrow \text{com} \Rightarrow \text{bool} \text{ (infix "\approx" 50) where} \\
\text{"c} \approx \text{c}' \equiv (\forall s t n. (c,s) \rightarrow \text{"n (SKIP, t)}) = (c', s) \rightarrow \text{"n (SKIP, t)})"

Prove the following lemma:

\text{lemma small_eqv_implies_big_eqv:} \quad \text{"c} \approx \text{c}' \Rightarrow \text{c} \sim \text{c}'

How about the reverse implication?

Exercise 6.2  A different instruction set architecture

We consider a different instruction set which evaluates boolean expressions on the stack, similar to arithmetic expressions:

- The boolean value \text{False} is represented by the number 0, the boolean value \text{True} is represented by any number not equal to 0.
• For every boolean operation exists a corresponding instruction which, similar to 
arithmetic instructions, operates on values on top of the stack.
• The new instruction set introduces a conditional jump which pops the top-most 
element from the stack and jumps over a given amount of instructions, if the 
popped value corresponds to False, and otherwise goes to the next instruction.

Modify the theory Compiler by defining a suitable set of instructions, by adapting the 
execution model and the compiler and by updating the correctness proof.

Homework 6  Micro-Step Semantics

Submission until Wednesday, December 7, 2011, 12:00 (noon).

In the lectures you have seen big-step and small-step semantics for the IMP language, 
and how to prove that they are equivalent. In this homework you will formalize a new 
micro-step semantics, and show that it is also equivalent to big-step.

The micro-step semantics relates pairs consisting of a list of commands together with 
a state. A single micro-step consists of executing the command at the head of the list, 
just like the small-step semantics—unless that command is a sequence \((c_1; c_2)\). In that 
case a micro-step consists of putting \(c_1\) and \(c_2\) back onto the list separately, without 
executing either one.

\[
\begin{align*}
\text{inductive } \text{micro\_step} &:: \text{“com list } \times \text{ state } \Rightarrow \text{ com list } \times \text{ state } \Rightarrow \text{ bool”} \text{ where} \\
\text{ms\_skip} &:: “\text{micro\_step (SKIP } \# l, s) (l, s)” \mid \\
\text{ms\_assign} &:: “\text{micro\_step } ((x ::= a) \# l, s) (\text{SKIP } \# l, s(x := \text{aval a s}))” \mid \\
\text{ms\_semi} &:: “\text{micro\_step } ((c_1; c_2) \# l, s) (c_1 \# c_2 \# l, s)” \mid \\
\text{ms\_ift} &:: “\text{bval b s }\Rightarrow \text{ micro\_step } ((\text{IF b THEN c}_1 \text{ ELSE c}_2) \# l, s) (c_1 \# l, s)” \mid \\
\text{ms\_iff} &:: “\text{¬ bval b s }\Rightarrow \text{ micro\_step } ((\text{IF b THEN c}_1 \text{ ELSE c}_2) \# l, s) (c_2 \# l, s)” \mid \\
\text{ms\_while} &:: “\text{micro\_step } ((\text{WHILE b DO c}) \# l, s) \\
((\text{IF b THEN c}; \text{WHILE b DO c ELSE SKIP}) \# l, s)”
\end{align*}
\]

We define \textit{micro\_steps} as an abbreviation for the reflexive, transitive closure of \textit{micro\_step}. (Recall that \textit{star} is defined in \textit{Star.thy}.)

\textbf{abbreviation}  “\textit{micro\_steps} ≡ \textit{star micro\_step}”

Because these are relations on pairs, we will need to generate new induction rules for 
them using the \texttt{split\_format} attribute.

\textbf{lemmas} \textit{micro\_step\_induct} =
\textit{micro\_step\_induct[split\_format(complete)]}

\textbf{lemmas} \textit{micro\_steps\_induct} =
\textit{star\_induct [where r=\textit{micro\_step}, split\_format(complete)]}

Your assignment is to prove that the micro-step semantics is equivalent to the big-step 
semantics:

\textbf{theorem}  “\textit{micro\_steps } ([c], s) ([], s’) \leftrightarrow (c, s) \Rightarrow s’”
You should prove implications in each direction separately. The following lemma states the right-to-left direction:

**lemma** big_step_imp_micro_steps: 

\[(c, s) \implies s' \implies \text{micro_steps} ([c], s) ([], s')\]

Hint: Proving big_step_imp_micro_steps may require additional lemmas; alternatively, you may find that it is easier to prove a generalization of this lemma instead. Also, note that you will probably need to use the rule star_trans (from Star.thy) in your proof.

To help with the left-to-right direction, we recommend defining a function seq that combines a list of commands into a single command. Then you can prove a lemma like micro_steps_imp_big_step_seq below:

**fun** seq :: “com list ⇒ com” where

“seq [] = SKIP” |
“seq (c #: l) = (c; seq l)”

**lemma** micro_steps_imp_big_step_seq:

“micro_steps (cs, s) (cs', s') \implies \forall t. (seq cs, s) \implies t \leftarrow (seq cs', s') \implies t”

Together with big_step_imp_micro_steps, you should then be able to prove the final theorem:

**theorem** “micro_steps ([c], s) ([], s') \leftrightarrow (c, s) \implies s'”