Concrete Semantics with Isabelle/HOL
Exercise Sheet 11

Exercise 11.1 Denotational Semantics for Repeat–Until

Define a denotational semantics for ‘repeat’ loops, and show its equivalence to the big-step semantics. Start by copying the Com, Big_Step, and theory Denotational theory files. The new command language is enriched with

| Repeat com bexp \((\text{"(REPEAT \_ / UNTIL \_)" \[0, 61\] 61})\)

Exercise 11.2 Hoare Logic: MAX

In this exercise, you will prove correct some Hoare triples.

First, write a program that stores the maximum of the values of variables \(a\) and \(b\) in variable \(c\).

definition MAX :: com

For the next task, you will probably need the following lemmas.

\[
\text{lemma [simp]: "}(a :: int) < b \implies \max a b = b"
\]

\[
\text{lemma [simp]: "} \neg (a :: int) < b \implies \max a b = a"
\]

Show that MAX satisfies the following Hoare triple:

\[
\text{lemma \quad \{\lambda s. True\} MAX \{\lambda s. s"c" = \max (s"a") (s"b")\}}
\]

Hints. You may want to use the lemma collection algebra_simps, which contains some useful facts, e.g., distributivity.

Since the assignment rule is backward, the best way to do proofs is also backwards: On a semicolon \(c_1::c_2\), you first continue the proof for \(c_2\), thus instantiating the intermediate assertion, and then do the proof for \(c_1\). However, the first premise of the Seq rule is about \(c_1\). Hence, you may want to use the rotated attribute, which rotates the premises of a lemma:

\[
\text{lemmas Seq_bwd = Seq[rotated]}
\]

\[
\text{lemmas hoare_rule[intro?] = Seq_bwd Assign Assign' If}
\]
Our specification still has a problem: Programs are allowed to overwrite arbitrary variables. Consider the following (wrong) implementation of $MAX$:

\[
\begin{align*}
definition \text{MAX\_wrong :: com where} \\
& "a\" ::= N 0 ;; \\
& "b\" ::= N 0 ;; \\
& "c\" ::= N 0
\end{align*}
\]

Prove that $\text{MAX\_wrong}$ also satisfies the specification for $MAX$.

What we really want to specify is that $MAX$ computes the maximum of the values of $a$ and $b$ in the initial state. Moreover, we may require that $a$ and $b$ are not changed.

For this, we can use logical variables in the specification. Prove the following more accurate specification for $MAX$:

\[
\begin{align*}
\text{lemma} & \quad \{ \lambda s. a = s "a" \land b = s "b" \} \\
& \quad \text{MAX} \\
& \quad \{ \lambda s. "c" = \max a b \land a = s "a" \land b = s "b" \}
\end{align*}
\]

Homework 11.1 Duality of Least and Greatest Fixpoints (5 points)

Submission until Monday 27.07.2015 10:00 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE11]” in the subject of your email. If your name is Eponymikos Patronymikos, please call your theory file Patronymikos_Eponymikos.thy.

Least and greatest fixpoints are dual with respect to set complement. Prove this.

Hint: The theorems \texttt{lfp\_def}, \texttt{gfp\_def}, \texttt{lfp\_lowerbound}, \texttt{gfp\_upperbound}, \texttt{gfp\_least}, and \texttt{lfp\_greatest} may be useful.

\[
\begin{align*}
\text{theorem} & \quad \texttt{lfp\_dual\_gfp:} "lfp f = \neg gfp (\lambda A :: 'a set. \neg f (- A))" \\
\text{theorem} & \quad \texttt{gfp\_dual\_lfp:} "gfp f = \neg lfp (\lambda A :: 'a set. \neg f (- A))"
\end{align*}
\]

Homework 11.2 Kleene for Greatest Fixpoints (10 points)

Submission until Monday 27.07.2015 10:00 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE11]” in the subject of your email. If your name is Eponymikos Patronymikos, please call your theory file Patronymikos_Eponymikos.thy.

Prove the greatest fixpoint version of the Kleene theorem (Theorem 11.12 in Nipkow and Klein, Denotational.lfp.if_cont).

Hint: There are many ways to skin this cat. Laziness can be a useful strategy here.
definition dual_chain :: "(nat ⇒ 'a set) ⇒ bool" where
dual_chain S = (∀ i. S (Suc i) ⊆ S i)"

definition dual_cont :: "('a set ⇒ 'b set) ⇒ bool" where
dual_cont f = (∀ S. dual_chain S → f (∩ n. S n) = (∩ n. f (S n)))"

theorem gfp_if_cont:
  assumes dc: "dual_cont f"
  shows "gfp f = (∩ n. (f " " n) UNIV)"