Semantics of Programming Languages
Exercise Sheet 11

**Exercise 11.1** Hoare Logic

In this exercise, you shall 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** $\text{MAX} :: \text{com where}$

For the next task, you will need the following lemmas. Hint: Sledgehammering may be a good idea.

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

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

by auto

Show that $\text{MAX}$ satisfies the following Hoare-triple:

**lemma** $\vdash \{λs. \text{True}\} \text{MAX} \{λs. s "c" = \text{max} (s "a") (s "b")\}$

Now define a program $\text{MUL}$ that returns the product of $x$ and $y$ in variable $z$. You may assume that $y$ is not negative.

**definition** $\text{MUL} :: \text{com where}$

Prove that $\text{MUL}$ does the right thing.

**lemma** $\vdash \{λs. 0 \leq s "y"\} \text{MUL} \{λs. s "z" = s "x" * s "y"\}$

**Hints** You may want to use the lemma $\text{algebra\_simpls}$, that contains some useful lemmas like distributivity.

Note that we use a backward assignment rule. This implies that the best way to do proofs is also backwards, i.e., 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 $\text{Seq}$-rule is about $c_1$. Hence, you may want to use the $\text{rotated}$-attribute, that rotates the premises of a lemma:

**lemmas** $\text{Seq\_bwd} = \text{Seq}[\text{rotated}]$
lemmas hoare_rule[intro?] = Seq_bwd Assign Assign' If

Note that our specifications still have a problem, as programs are allowed to overwrite arbitrary variables.
For example, regard the following (wrong) implementation of \( \text{MAX} \):

\[
\begin{aligned}
\text{definition} \quad \text{MAX}_{\text{wrong}} = ("a" := N 0;;"b" := N 0;;"c" := N 0)
\end{aligned}
\]

Prove that \( \text{MAX}_{\text{wrong}} \) also satisfies the specification for \( \text{MAX} \):

What we really want to specify is, that \( \text{MAX} \) 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 \( \text{MAX} \):

\[
\begin{aligned}
\text{lemma} \quad \vdash \{ \lambda s. a = s''a'' \land b = s''b'' \} \quad \text{MAX} \quad \{ \lambda s. s''c'' = \max a b \land a = s''a'' \land b = s''b'' \}
\end{aligned}
\]

The specification for \( \text{MUL} \) has the same problem. Fix it!

**Exercise 11.2** Forward Assignment Rule

Think up and prove a forward assignment rule, i.e., a rule of the form \( \vdash \{ P \} x := a \{ \ldots \} \), where \( \ldots \) is some suitable postcondition. Hint: To prove this rule, use the completeness property, and prove the rule semantically.

lemmas fwd.Assign' = weaken_post[OF fwd.Assign]

Redo the proofs for \( \text{MAX} \) and \( \text{MUL} \) from the previous exercise, this time using your forward assignment rule.

\[
\begin{aligned}
\text{lemma} \quad \vdash \{ \lambda s. \text{True} \} \text{MAX} \quad \{ \lambda s. s''c'' = \max (s''a'') (s''b'') \}
\end{aligned}
\]

\[
\begin{aligned}
\text{lemma} \quad \vdash \{ \lambda s. 0 \leq s''y'' \} \text{MUL} \quad \{ \lambda s. s''z'' = s''x'' \ast s''y'' \}
\end{aligned}
\]

**Homework 11.1** Hoare Logic OR

*Submission until Tuesday, January 24, 2017, 10:00am.*

Extend IMP with a new command \( c_1 \text{ OR } c_2 \) that is a nondeterministic choice: it may execute either \( c_1 \) or \( c_2 \). Add the constructor

\[
\text{Or com com \quad ("_ OR/ _" [60, 61] 60)}
\]
to datatype \textit{com} in theory \textit{Com}, adjust the definition of the big-step semantics in theory \textit{Big Step}, add a rule for OR to the Hoare logic in theory \textit{Hoare}, and adjust the soundness and completeness proofs in theory \textit{Hoare Complete}.

All these changes should be quite minimal and very local if you have got the definitions right.

\textbf{Homework 11.2} Fixed point reasoning

\textit{Submission until Tuesday, January 24, 2017, 10:00am.}

In the lecture, you have seen the Knaster-Tarski least fixed point theorem. The relevant constant is \texttt{lfp} :: (′\texttt{a} ⇒ ′\texttt{a}) ⇒ ′\texttt{a}, which assumes a complete lattice order \(≤\) on ′\texttt{a} and returns, for each monotonic operator \texttt{f} :: ′\texttt{a} ⇒ ′\texttt{a}, its least fixed point \texttt{lfp f}.

In the lectures as well as in this exercise, one only deals with the case where ′\texttt{a} is ′\texttt{b set} (the type of sets over an arbitrary type ′\texttt{b}) and \(\leq\) is \(\subseteq\) (set inclusion). In this exercise, you will prove a different kind of fixed point theorem. It says that if there are two injective functions, one from ′\texttt{a} to ′\texttt{b}, and one the other way round, then there also exists an bijection between ′\texttt{a} and ′\texttt{b}:

\begin{verbatim}
theorem
  assumes “inj (f :: ′a ⇒ ′b)” and “inj (g :: ′b ⇒ ′a)”
  shows “∃ h :: ′a ⇒ ′b. inj h ∧ surj h”

This is a fixed point theorem because we will use a least fixed point for the construction of \textit{h}. Use the provided template and follow the proof outline below to finish the proof.

theorem
  assumes “inj (f :: ′a ⇒ ′b)” and “inj (g :: ′b ⇒ ′a)”
  shows “∃ h :: ′a ⇒ ′b. inj h ∧ surj h”

proof
  def \texttt{S} ≡ “lfp (λX. − (g ′ (− (f ′ X))))”
  let \texttt{?g}′ = “inv g”
  def \texttt{h} ≡ “λz. if z ∈ S then f z else ??g′ z”

  have “S = − (g ′ (− (f ′ S)))”
  have *: “??g′ (− S) = − (f ′ S)”

  show “inj h ∧ surj h”
  proof
    from * show “surj h”
    have “inj_on f S”
    moreover have “inj_on ??g′ (− S)”
    moreover { fix a b
      assume “a ∈ S” “b ∈ − S” and eq: “f a = ??g′ b”
      have False } 
  
\end{verbatim}
ultimately show \textit{“inj } h\textit{”}

qed

ded