Semantics of Programming Languages
Exercise Sheet 11

The following exercises are typical exam exercises. You are supposed to solve them on a sheet of paper, without using Isabelle/HOL.

Exercise 11.1 Using the VCG, Total correctness

For each of the three programs given here, you must prove partial correctness and total correctness. For the partial correctness proofs, you should first write an annotated program, and then use the verification condition generator from VCG. For the total correctness proofs, use the Hoare rules from Hoare, Total.

Some abbreviations, freeing us from having to write double quotes for concrete variables:

abbreviation "aa" ≡ "a"
abbreviation "bb" ≡ "b"
abbreviation "cc" ≡ "c"
abbreviation "dd" ≡ "d"
abbreviation "qq" ≡ "q"
abbreviation "rr" ≡ "r"

Some useful simplification rules:

declare algebra_simp[simp] declare power2_eq_square[simp]

Rotated rule for sequential composition:

lemmas SeqTR = Hoare_Total.Seq[rotated]

Prove the following syntax-directed conditional rule (for total correctness):

lemma IfT:
assumes "τ_t \{ P1 \} c_1 \{ Q \} " and "τ_t \{ P2 \} c_2 \{ Q \} "
shows "τ_t \{ λs. (bval b s ⇒ P1 s) ∧ (¬ bval b s ⇒ P2 s) \} IF b THEN c_1 ELSE c_2 \{ Q \} "
oops

A convenient loop construct:

abbreviation "FOR v FROM a1 TO a2 DO c ≡ \[ v := a1 ;; WHILE (Less (V v) a2) DO (c ;; v := Plus (V v) (N 1)) \]"

abbreviation "\{ b \} FOR v FROM a1 TO a2 DO c ≡ \[ v := a1 ;; \{ b \} WHILE (Less (V v) a2) DO (c ;; v := Plus (V v) (N 1)) \]"
Multiplication. Consider the following program \texttt{MULT} for performing multiplication and the following assertions \texttt{P\_MULT} and \texttt{Q\_MULT}:

\begin{verbatim}
definition MULT2 :: com where
  "MULT2 ≡ FOR dd FROM (N 0) TO (V aa) DO cc ::= Plus (V cc) (V bb)"

definition MULT :: com where  "MULT ≡ cc ::= N 0 ;; MULT2"

definition P\_MULT :: "int ⇒ int ⇒ assn" where
  "P\_MULT i j ≡ λs. s aa = i ∧ s bb = j ∧ 0 ≤ i"

definition Q\_MULT :: "int ⇒ int ⇒ assn" where
  "Q\_MULT i j ≡ λs. s cc = i * j ∧ s aa = i ∧ s bb = j"
\end{verbatim}

Define an annotated program \texttt{AMULT \_i \_j}, so that when the annotations are stripped away, it yields \texttt{MULT}. (The parameters \texttt{i} and \texttt{j} will appear only in the loop annotations.) Hint: The program \texttt{AMULT \_i \_j} will be essentially \texttt{MULT} with an invariant annotation \texttt{iMULT \_i \_j} at the FOR loop, which you have to define:

\begin{verbatim}
definition iMULT :: "int ⇒ int ⇒ assn" where
  "iMULT i j ≡ undefined"

definition AMULT2 :: "int ⇒ int ⇒ acom" where
  "AMULT2 i j ≡ \{iMULT i j\}
  FOR dd FROM (N 0) TO (V aa) DO cc ::= Plus (V cc) (V bb)"

definition AMULT :: "int ⇒ int ⇒ acom" where
  "AMULT i j ≡ (cc ::= N 0) ;; AMULT2 i j"
\end{verbatim}

Once you have the correct loop annotations, then the partial correctness proof can be done in two steps, with the help of lemma \texttt{vc\_sound}'.

\begin{verbatim}
lemma strip\_AMULT: "strip (AMULT \_i \_j) = MULT"

lemma MULT\_correct: "⊢ \{P\_MULT \_i \_j\} MULT \{Q\_MULT \_i \_j\}"
\end{verbatim}

The total correctness proof will look much like the Hoare logic proofs from Exercise Sheet 9, but you must use the rules from \texttt{Hoare\_Total} instead. Also note that when using rule \texttt{Hoare\_Total\_While\_fun'}, you must instantiate both the predicate \texttt{P :: state ⇒ bool} and the measure \texttt{f :: state ⇒ nat}. The measure must decrease every time the body of the loop is executed. You can define the measure first:

\begin{verbatim}
definition mMULT :: "state ⇒ nat" where
  "mMULT ≡ undefined"

lemma MULT\_totally\_correct: "⊢t \{P\_MULT \_i \_j\} MULT \{Q\_MULT \_i \_j\}"
\end{verbatim}
**Division.** Define an annotated version of this division program, which yields the quotient and remainder of \(aa/bb\) in variables "\(q\)" and "\(r\)", respectively.

**definition DIV1 :: com where** “DIV1 \(\equiv\) qq ::=: N 0 ;; rr ::= N 0”

**definition DIV_IF :: com where**
“DIV_IF \(\equiv\) (IF Less (V rr) (V bb) THEN Com.SKIP
ELSE (rr ::= N 0 ;; qq ::= Plus (V qq) (N 1)))”

**definition “DIV2 \(\equiv\) rr ::= Plus (V rr) (N 1) ;; DIV_IF”**

**definition DIV :: com where**
“DIV \(\equiv\) DIV1 ;; FOR cc FROM (N 0) TO (V aa) DO DIV2”

**lemmas DIV_defs = DIV1_def DIV_IF_def DIV2_def DIV_def**

**definition P DIV :: “int ⇒ int ⇒ assn” where**
“P DIV i j \(\equiv\) \(\lambda\) s. s aa = i ∧ s bb = j ∧ 0 ≤ i ∧ 0 < j”

**definition Q DIV :: “int ⇒ int ⇒ assn” where**
“Q DIV i j \(\equiv\) \(\lambda\) s. i = s qq ∗ j + s rr ∧ 0 ≤ s rr ∧ s rr < j ∧ s aa = i ∧ s bb = j”

**definition iDIV :: “int ⇒ int ⇒ assn” where**
“iDIV i j \(\equiv\) undefined”

**lemma strip_ADIV: “strip (ADIV i j) = DIV”**

**oops**

**lemma DIV_correct: “\(\vdash\) (P_DIV i j) DIV (Q_DIV i j)”**

**oops**

**definition mDIV :: “state ⇒ nat” where** — Measure function:
“mDIV \(\equiv\) undefined”

**lemma DIV_totally_correct: “\(\vdash\) (P_DIV i j) DIV (Q_DIV i j)”**

**oops**

**Square roots.** Define an annotated version of this square root program, which yields the square root of input \(aa\) (rounded down to the next integer) in output \(bb\).

**definition SQR1 :: com where** “SQR1 \(\equiv\) bb ::= N 0 ;; cc ::= N 1”

**definition SQR2 :: com where**
“SQR2 \(\equiv\)

\[
\begin{align*}
bb &::= Plus (V bb) (N 1);; \\
cc &::= Plus (V cc) (V bb);; \\
cc &::= Plus (V cc) (V bb);; \\
cc &::= Plus (V cc) (N 1) \\
\end{align*}
\]
definition \( SQR :: \text{com where} \)
\[ "SQR \equiv SQR1 :: (\text{WHILE (Not (Less (V aa) (V cc))) DO SQR2})" \]

definition \( P_{SQR} :: \text{int \Rightarrow assn where} \)
\[ "P_{SQR} i \equiv \lambda s. s aa = i \land 0 \leq i" \]

definition \( Q_{SQR} :: \text{int \Rightarrow assn where} \)
\[ "Q_{SQR} i \equiv \lambda s. s aa = i \land (s bb)^2 \leq i \land i < (s bb + 1)^2" \]

lemma \( SQR\text{.totally.correct} : "\vdash t\{P_{SQR} i\} SQR \{Q_{SQR} i\}" \)

Exercise 11.2 Where is the mistake in the following argument?
The natural numbers form a complete lattice because any set of natural numbers has an infimum, its least element.

Exercise 11.3 Collecting Semantics
Recall the datatype of annotated commands (type `'a acom) and the collecting semantics (function \( \text{step :: state set \Rightarrow state set acom \Rightarrow state set acom} \)) from the lecture. We reproduce the definition of \( \text{step} \) here for easy reference. (Recall that \( \text{post c} \) simply returns the right-most annotation from command \( c \).)

\( \text{step S (SKIP \{\_\}) = SKIP \{S\} } \)
\( \text{step S (x::=e \{\_\}) = x ::= e \{\{s(x:=aval e s) \mid s. s \in S\}\} } \)
\( \text{step S (c1 ::; c2) = step S c1 ::; step (post c1) c2 } \)
\( \text{step S (IF b THEN \{P1\} c1 ELSE \{P2\} c2 \{\_\}) = } \)
\[ \text{IF b THEN \{\{s \in S. bval b s\}\} step P1 c1 } \]
\[ \text{ELSE \{\{s \in S. \neg bval b s\}\} step P2 c2 } \]
\[ \{\text{post c1} \cup \text{post c2}\} \]
\( \text{step S (\{I\} WHILE b DO \{P\} c \{\_\}) = } \)
\[ \{\text{S} \cup \text{post c}\} \]
\[ \text{WHILE b DO \{\{s \in I. bval b s\}\} step P c } \]
\[ \{\{s \in I. \neg bval b s\}\} \]
In this exercise you must evaluate the collecting semantics on the example program below by repeatedly applying the \( \text{step} \) function.
\[ c = (\text{IF } x < 0 \]
\[ \text{THEN } \{A1\} \]
\[ \{A2\} \text{ WHILE } 0 < y \text{ DO } \{A3\} (y := y + x \{A4\}) \{A5\} \]
\[ \text{ELSE } \{A6\} \text{ SKIP } \{A7\} \} \{A8\} \]
Let \( S \) be \( \{(−2,3),(1,2)\} \), abbreviated \( −2,3 \ | \ 1,2 \). Calculate column \( n+1 \) in the table below by evaluating \( \text{step S c} \) with the annotations for \( c \) taken from column \( n \). For
conciseness, we use “(i, j)” as notation for the state <"x":=i, "y":=j>.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1</td>
<td>∅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_2</td>
<td>∅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_3</td>
<td>∅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_4</td>
<td>∅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_5</td>
<td>∅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_6</td>
<td>∅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_7</td>
<td>∅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_8</td>
<td>∅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Homework 11.1  P&P proof for complete lattices**

*Submission until Tuesday, 21. 1. 2013, 10:00am.*

Make a pen & paper proof for the following statement:

In a complete lattice ∪ S = ⨇ {u. ∀ s ∈ S. s ≤ u} is the least upper bound of S.

**Homework 11.2  Counterexamples**

*Submission until Tuesday, 21. 1. 2013, 10:00am.*

We know that least pre-fixpoints of monotone functions are also least fixpoints.

1. Show that leastness matters: find a (small!) partial order with a monotone function that has a pre-fixpoint that is not a fixpoint.

2. Show that the reverse implication does not hold: find a partial order with a monotone function function that has a least fixpoint that is not a least pre-fixpoint.