Exercise 11.1 Forward Assignment Rule

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

**lemmas** \( \textit{fwd\_Assign}' = \textit{weaken\_post}[\textit{OF \textit{fwd\_Assign}}] \)

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

**Definition** \( \text{MAX} :: \text{com} \) where

\[
\text{"MAX } \equiv \text{ IF (Less (V "a") (V "b")) THEN } \\
\quad "c":= V "b" \\
\text{ ELSE } \\
\quad "c":= V "a"
\]

**Lemma** \( \vdash \{ \lambda s. \text{True} \} \text{MAX} \{ \lambda s. \quad "c" = \text{max (s "a") (s "b")} \} \)

**Definition** \( \text{MUL} :: \text{com} \) where

\[
\text{"MUL } \equiv \text{ "z":=N 0;; } \\
\quad "c":=N 0;; \\
\text{ WHILE (Less (V "c") (V "y")) DO ( } \\
\quad "z":=\text{Plus (V "z") (V "x");; } \\
\quad "c":=\text{Plus (V "c") (N 1))}"
\]

**Lemma** \( \vdash \{ \lambda s. \quad 0 \leq s "y" \} \text{MUL} \{ \lambda s. \quad "z" = s "x" \ast s "y" \} \)

Exercise 11.2 Using the VCG

For each of the three programs given here, you must prove partial correctness. You should first write an annotated program, and then use the verification condition generator from \( \text{VCG.thy} \).
Some abbreviations, freeing us from having to write double quotes for concrete variable names:

abbreviation "aa ≡ "a""  abbreviation "bb ≡ "b""
abbreviation "cc ≡ "c""
abbreviation "dd ≡ "d""
abbreviation "ee ≡ "e""
abbreviation "ff ≡ "f""
abbreviation "pp ≡ "p""
abbreviation "qq ≡ "q""
abbreviation "rr ≡ "r"

Some useful simplification rules:

declare algebra_simps[flip]  declare power2_eq_square[flip]

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

definition Q_MULT :: "int ⇒ int ⇒ assn" where
  "Q_MULT i j ≡ λs cc = i * j ∧ s aa = i ∧ s bb = j"

Define an annotated program AMULT i j, so that when the annotations are stripped away, it yields MULT. (The parameters i and j will appear only in the loop annotations.)

Hint: The program AMULT i j will be essentially MULT with an invariant annotation iMULT i j at the FOR loop, which you have to define:

definition iMULT :: "int ⇒ int ⇒ assn" where
  "iMULT i j ≡ (cc ::= N 0) ;; {iMULT i j} FOR dd FROM (N 0) TO (V aa) DO (cc ::= Plus (V cc) (V bb))"
lemmas MULTdefs = MULT_def P_MULT_def Q_MULT_def iMULT_def AMULT_def

lemma strip_AMULT: “strip (AMULT i j) = MULT”

Once you have the correct loop annotations, then the partial correctness proof can be done in two steps, with the help of lemma vc_sound’.

lemma MULT_correct: “{- P_MULT i j} MULT {Q_MULT i j}”

**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 DIV :: com where “DIV ≡
qq ::= N 0 ;;
rr ::= N 0 ;;
FOR cc FROM (N 0) TO (V aa) DO (  
  rr ::= Plus (V rr) (N 1) ;;  
  IF Less (V rr) (V bb) THEN  
    Com.SKIP  
  ELSE (  
    rr ::= N 0 ;;  
    qq ::= Plus (V qq) (N 1))  
)"

definition P_DIV :: “int ⇒ int ⇒ assn” where
“P_DIV i j ≡ \( \lambda s. s aa = i \land s bb = j \land 0 \leq i \land 0 < j \)”

definition Q_DIV :: “int ⇒ int ⇒ assn” where
“Q_DIV i j ≡ \( \lambda s. i = s qq \ast j + s rr \land 0 \leq s rr \land s rr < j \land s aa = i \land s bb = j \)”

definition iDIV :: “int ⇒ int ⇒ assn” where
“iDIV i j ≡
qq ::= N 0 ;;
rr ::= N 0 ;;
{iDIV i j} FOR cc FROM (N 0) TO (V aa) DO (  
  rr ::= Plus (V rr) (N 1) ;;  
  IF Less (V rr) (V bb) THEN  
    SKIP  
  ELSE (  
    rr ::= N 0 ;;  
    qq ::= Plus (V qq) (N 1))  
)"

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

lemma DIV_correct: “{- P_DIV i j} DIV {Q_DIV i j}”
**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** \(SQR :: \text{com where} \quad \text{“}SQR \equiv \text{\begin{verbatim}
n 0 \\
1
\end{verbatim}} \quad \text{WHILE} \ (\text{Not} \ (\text{Less} \ (V \ aa) \ (V \ cc))) \ DO \ \text{\begin{verbatim}
bb ::= \text{Plus} \ (V \ bb) \ (N \ 1) \\
cc ::= \text{Plus} \ (V \ cc) \ (\text{Plus} \ (V \ bb) \ (\text{Plus} \ (V \ bb) \ (N \ 1)))
\end{verbatim}} \) \n
**definition** \(P_{SQR} :: \text{\begin{verbatim}
“int ⇒ assn” where \text{\begin{verbatim}
λs. s aa = i ∧ 0 ≤ i
\end{verbatim}}
\end{verbatim}} \n
**definition** \(Q_{SQR} :: \text{\begin{verbatim}
“int ⇒ assn” where \text{\begin{verbatim}
λs. s aa = i ∧ (s bb)² ≤ i ∧ i < (s bb + 1)²
\end{verbatim}}
\end{verbatim}} \n
**Homework 11.1** Program Verification

*Submission until Tuesday, 20 January 2015, 10:00am.*

Define an annotated command \(C_{\text{diff}}\) that subtracts \(x\) from \(y\) and prove

**lemma** “\(\vdash \{\lambda s. s \ ‘x’ = x \land s \ ‘y’ = y \land 0 \leq x\} \ \text{strip}(C_{\text{diff}} \ x \ y) \ \{\lambda t. t \ ‘y’ = y - x\}’’”

**Homework 11.2** Hoard Logic OR

*Submission until Tuesday, 20 January 2015, 10:00am.*

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

\[ \text{Or com com} \quad (\_ \ OR/ \ _\) \quad \text{[60, 61] 60} \]

to datatype \text{com} in theory \text{Com}, adjust the definition of the big-step semantics in theory \text{Big_Sem}, add a rule for \text{OR} to the Hoare logic in theory \text{Hoare}, and adjust the soundness and completeness proofs in theory \text{Hoare_Sound_Complete}.

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