Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Peter Lammich, Johannes Hölzl

WS 2013/14
14. 1. 2013

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 "$aa \equiv ''a''$" **abbreviation** "$bb \equiv ''b''$" **abbreviation** "$cc \equiv ''c''$"
abbreviation "$dd \equiv ''d''$" **abbreviation** "$qq \equiv ''q''$" **abbreviation** "$rr \equiv ''r''$"

Some useful simplification rules:

declare *algebra_simps*[*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 "$\vdash_t \{P1\}\ c_1\ \{Q\}$" **and** "$\vdash_t \{P2\}\ c_2\ \{Q\}$"
shows "$\vdash_t \{\lambda s.\ (bval\ b\ s \longrightarrow P1\ s) \wedge (\neg\ bval\ b\ s \longrightarrow P2\ s)\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{Q\}$"
oops

A convenient loop construct:

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

abbreviation "$\{b\}\ FOR\ v\ FROM\ a1\ TO\ a2\ DO\ c \equiv$
$v ::= a1\ ;;\ \{b\}\ WHILE\ (Less\ (V\ v)\ a2)\ DO\ (c\ ;;\ v ::= Plus\ (V\ v)\ (N\ 1))$"

Multiplication. Consider the following program *MULT* for performing multiplication and the following assertions *P_MULT* and *Q_MULT*:

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* ≡ $\lambda s.\ s\ aa = i \wedge s\ bb = j \wedge 0 \leq i$"

definition *Q_MULT* :: "*int* ⇒ *int* ⇒ *assn*" **where**
  "*Q_MULT i j* ≡ $\lambda s.\ s\ cc = i * j \wedge s\ aa = i \wedge s\ bb = j$"

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* ≡ *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*"

lemmas *MULT_defs* = *MULT2_def MULT_def P_MULT_def Q_MULT_def iMULT_def AMULT2_def AMULT_def*

lemma *strip_AMULT*: "*strip* (*AMULT i j*) = *MULT*"
  oops

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*}"
  oops

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

definition *mMULT* :: "*state* ⇒ *nat*" **where**
  "*mMULT* ≡ *undefined*"

lemma *MULT_totally_correct*: "$\vdash_t$ {*P_MULT i j*} *MULT* {*Q_MULT i j*}"
  oops

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 ≡ qq ::= N 0 ;; rr ::= N 0"*

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

definition *"DIV2 ≡ rr ::= Plus (V rr) (N 1) ;; DIV_IF"*

definition *DIV* :: *com* **where**
"DIV ≡ 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 ≡ λs. s aa = i ∧ s bb = j ∧ 0 ≤ i ∧ 0 < j"

definition *Q_DIV* :: *"int ⇒ int ⇒ assn"* **where**
"Q_DIV i j ≡
λ 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 ≡ undefined"

lemma *strip_ADIV*: *"strip (ADIV i j) = DIV"*
oops

lemma *DIV_correct*: *"⊢ {P_DIV i j} DIV {Q_DIV i j}"*
oops

definition *mDIV* :: *"state ⇒ nat"* **where** — Measure function:
"mDIV ≡ undefined"

lemma *DIV_totally_correct*: *"$\vdash_t$ {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 ≡ bb ::= N 0 ;; cc ::= N 1"*

definition *SQR2* :: *com* **where**
"SQR2 ≡
bb ::= Plus (V bb) (N 1);;
cc ::= Plus (V cc) (V bb);;
cc ::= Plus (V cc) (V bb);;
cc ::= Plus (V cc) (N 1)"

definition *SQR* :: *com* **where**
"$SQR \equiv SQR1\ ;;\ (WHILE\ (Not\ (Less\ (V\ aa)\ (V\ cc)))\ DO\ SQR2)$"

definition *P_SQR* :: "$int \Rightarrow assn$" **where**
"$P\_SQR\ i \equiv \lambda s.\ s\ aa = i \wedge 0 \le i$"

definition *Q_SQR* :: "$int \Rightarrow assn$" **where**
"$Q\_SQR\ i \equiv \lambda s.\ s\ aa = i \wedge (s\ bb)\hat{\ }2 \le i \wedge i < (s\ bb + 1)\hat{\ }2$"

lemma *SQR_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 $step :: state\ set \Rightarrow state\ set\ acom \Rightarrow state\ set\ acom$) from the lecture. We reproduce the definition of *step* here for easy reference. (Recall that *post c* simply returns the right-most annotation from command *c*.)

$$
\begin{aligned}
&step\ S\ (SKIP\ \{\_\}) = SKIP\ \{S\}\\
&step\ S\ (x ::= e\ \{\_\}) = x ::= e\ \{\{s(x := aval\ e\ s) \mid s.\ s \in S\}\}\\
&step\ S\ (c_1\ ;;\ c_2) = step\ S\ c_1\ ;;\ step\ (post\ c_1)\ c_2\\
&step\ S\ (IF\ b\ THEN\ \{P_1\}\ c_1\ ELSE\ \{P_2\}\ c_2\ \{\_\}) =\\
&\qquad IF\ b\ THEN\ \{\{s{\in}S.\ bval\ b\ s\}\}\ step\ P_1\ c_1\\
&\qquad ELSE\ \{\{s{\in}S.\ \neg\ bval\ b\ s\}\}\ step\ P_2\ c_2\\
&\qquad \{post\ c_1 \cup post\ c_2\}\\
&step\ S\ (\{I\}\ WHILE\ b\ DO\ \{P\}\ c\ \{\_\}) =\\
&\qquad \{S \cup post\ c\}\\
&\qquad WHILE\ b\ DO\ \{\{s{\in}I.\ bval\ b\ s\}\}\ step\ P\ c\\
&\qquad \{\{s{\in}I.\ \neg\ bval\ b\ s\}\}
\end{aligned}
$$

In this exercise you must evaluate the collecting semantics on the example program below by repeatedly applying the *step* function.

$$
\begin{aligned}
&c = (IF\ x < 0\\
&\qquad\qquad THEN\ \{A_1\}\\
&\qquad\qquad\qquad \{A_2\}\ WHILE\ 0 < y\ DO\ \{A_3\}\ (y := y + x\ \{A_4\})\ \{A_5\}\\
&\qquad\qquad ELSE\ \{A_6\}\ SKIP\ \{A_7\})\ \{A_8\}
\end{aligned}
$$

Let S be $\{\langle -2{,}3\rangle{,}\langle 1{,}2\rangle\}$, abbreviated $-2{,}3 \mid 1{,}2$. Calculate column $n{+}1$ in the table below by evaluating $step\ S\ c$ with the annotations for c taken from column n. For conciseness, we use "$\langle i,\, j\rangle$" as notation for the state $<''x'':=i,\ ''y'':=j>$.

	0	1	2	3	4	5	6	7	8	9	10
A_1	$\emptyset$										
A_2	$\emptyset$										
A_3	$\emptyset$										
A_4	$\emptyset$										
A_5	$\emptyset$										
A_6	$\emptyset$										
A_7	$\emptyset$										
A_8	$\emptyset$										

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 $\bigsqcup S = \bigsqcap\{u.\ \forall\, s{\in}S.\ s \le u\}$ 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.