Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Andrei Popescu, Peter Lammich

WS 2012/13
8. 1. 2013

Semantics of Programming Languages

Exercise Sheet 11

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 *VC.thy*. For the total correctness proofs, use the Hoare rules from *HoareT.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* ≡ ''*d*''" **abbreviation** "*ff* ≡ ''*f*''"
abbreviation "*pp* ≡ ''*p*''" **abbreviation** "*qq* ≡ ''*q*''" **abbreviation** "*rr* ≡ ''*r*''"

Some useful simplification rules:

declare *algebra_simps*[*simp*] **declare** *power2_eq_square*[*simp*]

Rotated rule for sequential composition:

lemmas *SeqTR* = *HoareT.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 "$\vdash_t$ {λ*s*. (*bval b s* ⟶ *P1 s*) ∧ (¬ *bval b s* ⟶ *P2 s*)} *IF b THEN* c_1 *ELSE* c_2 {*Q*}"

A convenient loop construct:

abbreviation *For* :: "*vname* ⇒ *aexp* ⇒ *aexp* ⇒ *com* ⇒ *com*"
  ("(*FOR _/ FROM _/ TO _/ DO _*)" [*0*, *0*, *0*, *61*] *61*) **where**
  "*FOR v FROM a1 TO a2 DO c* ≡
    v ::= *a1* ; *WHILE* (*Less* (*V v*) *a2*) *DO* (*c* ; *v* ::= *Plus* (*V v*) (*N 1*))"

abbreviation *Afor* :: "*assn* ⇒ *vname* ⇒ *aexp* ⇒ *aexp* ⇒ *acom* ⇒ *acom*"
  ("({_}/ *FOR _/ FROM _/ TO _/ DO _*)" [*0*, *0*, *0*, *0*, *61*] *61*) **where**
  "{*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 *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 ≡ 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"

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**

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"*

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}"*

The total correctness proof will look much like the Hoare logic proofs from Exercise Sheet 9, but you must use the rules from *HoareT.thy* instead. Also note that when using rule *HoareT.While′*, 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**

lemma *MULT_totally_correct*: "$\vdash_t$ {*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 *DIV1* :: *com* **where** "*DIV1* ≡ *qq* ::= *N 0* ; *rr* ::= *N 0*"

definition *DIV_IF* :: *com* **where**
"*DIV_IF* ≡
IF Less (*V rr*) (*V bb*) *THEN 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* ≡ $\lambda s.\ s\ aa = i \wedge s\ bb = j \wedge 0 \le i \wedge 0 < j$"
definition *Q_DIV* :: "*int* ⇒ *int* ⇒ *assn*" **where**
"*Q_DIV i j* ≡
$\lambda\ s.\ i = s\ qq * j + s\ rr \wedge 0 \le s\ rr \wedge s\ rr < j \wedge s\ aa = i \wedge s\ bb = j$"

definition *iDIV* :: "*int* ⇒ *int* ⇒ *assn*" **where**

definition *ADIV1* :: *acom* **where** "*ADIV1* ≡ *qq* ::= *N 0* ; *rr* ::= *N 0*"

definition *ADIV_IF* :: *acom* **where**
"*ADIV_IF* ≡
IF Less (*V rr*) (*V bb*) *THEN ASKIP*
ELSE (*rr* ::= *N 0* ; *qq* ::= *Plus* (*V qq*) (*N 1*))"

definition *ADIV2* :: *acom* **where** "*ADIV2* ≡ *rr* ::= *Plus* (*V rr*) (*N 1*) ; *ADIV_IF*"

definition *ADIV* :: "*int* ⇒ *int* ⇒ *acom*" **where**
"*ADIV i j* ≡ *ADIV1* ; {*iDIV i j*} *FOR cc FROM* (*N 0*) *TO* (*V aa*) *DO ADIV2*"

lemmas *ADIV_defs* = *ADIV1_def ADIV_IF_def ADIV2_def ADIV_def*

lemma *strip_ADIV*: "*strip* (*ADIV i j*) = *DIV*"
lemma *DIV_correct*: "⊢ {*P_DIV i j*} *DIV* {*Q_DIV i j*}"

definition *mDIV* :: "*state* ⇒ *nat*" **where**

lemma *DIV_totally_correct*: "$\vdash_t$ {*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 *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 ≡ SQR1 ; (WHILE (Not (Less (V aa) (V cc))) DO SQR2)"

lemmas *SQR_defs = SQR1_def SQR2_def SQR_def*

definition *P_SQR* :: *"int ⇒ assn"* **where**
"P_SQR i ≡ λs. s aa = i ∧ 0 ≤ i"
definition *Q_SQR* :: *"int ⇒ assn"* **where**
"Q_SQR i ≡ λs. s aa = i ∧ (s bb)ˆ2 ≤ i ∧ i < (s bb + 1)ˆ2"

Homework 11 Be Original

Submission until Tuesday, 15. 1. 2013, 10:00am.

Deadline of previous homework was extended, so polish your submission a bit!