Section 8.5

Reasoning about more expressive languages

Overview

- Abstract while-language with nondeterminism and local variables
- Total correctness
- Recursive procedure

Literature

- T. Nipkow: Hoare Logics in Isabelle/HOL
- T. Nipkow: Hoare Logics for Recursive Procedures and Unbounded Nonterminism
- T. Nipkow: Abstract Hoare Logics (Archive of formal proofs)

Abstract while-language

- Nondeterminism and local variables
- Shallow embedding of expressions and state transformers

typedecl *state*

type-synonym *bexp* = *state* ⇒ *bool*

datatype *com* = *Do* (*state* ⇒ *state set*)
| *Semi com com* (-; - [*60*, *60*] *10*)
| *Cond bexp com com* (*IF - THEN - ELSE - 60*)
| *While bexp com* (*WHILE - DO - 60*)
| *Local* (*state* ⇒ *state*) *com* (*state* ⇒ *state* ⇒ *state*)
(*LOCAL -; -; - [0,0,60] 60*)

Semantics of abstract while-language

inductive
$exec :: state \Rightarrow com \Rightarrow state \Rightarrow bool$ (-/ $-\!\!\rightarrow$/ - [50,0,50] 50)
where
$t \in f\ s \Longrightarrow s -Do\ f\rightarrow t$

$\mid [\![\ s0 -c1\rightarrow s1;\ s1 -c2\rightarrow s2\ ]\!] \Longrightarrow s0 -c1;c2\rightarrow s2$

$\mid [\![\ b\ s;\ s -c1\rightarrow t\ ]\!] \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2\rightarrow t$
$\mid [\![\ \neg b\ s;\ s -c2\rightarrow t\ ]\!] \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2\rightarrow t$

$\mid \neg b\ s \Longrightarrow s -WHILE\ b\ DO\ c\rightarrow s$
$\mid [\![\ b\ s;\ s -c\rightarrow t;\ t -WHILE\ b\ DO\ c\rightarrow u\ ]\!] \Longrightarrow s -WHILE\ b\ DO\ c\rightarrow u$

$\mid\ f\ s -c\rightarrow t \Longrightarrow s -LOCAL\ f;\ c;\ g\rightarrow g\ s\ t$

Discussion

- abstractness of the language allows to realize different concrete language constructs
- the LOCAL command can handle local variable declarations:

$$LOCAL(\lambda\ s.s(x := a\ s)); c; (\lambda\ s\ t.t(x := s\ x))$$

Formulas and validity for partial correctness

type-synonym $assn = state \Rightarrow bool$

definition
$hoare\text{-}valid :: assn \Rightarrow com \Rightarrow assn \Rightarrow bool$ ($\models$ {(1-)}/ (-)/ {(1-)} 50) **where**
$\models \{P\}c\{Q\} \longleftrightarrow (\forall\ s\ t.\ s -c\rightarrow t \longrightarrow P\ s \longrightarrow Q\ t)$

Rules for partial correctness

inductive
$hoare :: assn \Rightarrow com \Rightarrow assn \Rightarrow bool$ ($\vdash$ ({(1-)}/ (-)/ {(1-)}) 50)
where
$\vdash \{\lambda s.\ \forall\ t \in f\ s.\ P\ t\}\ Do\ f\ \{P\}$

$\mid [\![\vdash \{P\}c1\{Q\};\ \vdash \{Q\}c2\{R\}\ ]\!] \Longrightarrow \vdash \{P\}\ c1;c2\ \{R\}$

$\mid [\![\vdash \{\lambda s.\ P\ s \wedge b\ s\}\ c1\ \{Q\};\ \vdash \{\lambda s.\ P\ s \wedge \neg b\ s\}\ c2\ \{Q\}\ ]\!]$
$\Longrightarrow \vdash \{P\}\ IF\ b\ THEN\ c1\ ELSE\ c2\ \{Q\}$

$\mid \vdash \{\lambda s.\ P\ s \wedge b\ s\}\ c\ \{P\} \Longrightarrow \vdash \{P\}\ WHILE\ b\ DO\ c\ \{\lambda s.\ P\ s \wedge \neg b\ s\}$

$\mid [\![\forall s.\ P'\ s \longrightarrow P\ s;\ \vdash \{P\}c\{Q\};\ \forall s.\ Q\ s \longrightarrow Q'\ s\ ]\!] \Longrightarrow \vdash \{P'\}c\{Q'\}$

$\mid [\![\bigwedge s.\ P\ s \Longrightarrow P'\ s\ (f\ s);\ \forall s. \vdash \{P'\ s\}\ c\ \{Q \circ (g\ s)\}\ ]\!] \Longrightarrow$
$\vdash \{P\}\ LOCAL\ f;c;g\ \{Q\}$

Soundness and completeness

theorem *hoare-sound*: $\vdash \{P\}c\{Q\} \implies \models \{P\}c\{Q\}$

theorem *hoare-relative-complete*: $\models \{P\}c\{Q\} \implies \vdash \{P\}c\{Q\}$

Termination

inductive
termi :: *com* ⇒ *state* ⇒ *bool* (**infixl** ↓ *50*)
where
$f\,s \neq \{\} \implies Do\,f \downarrow s$

| $[\![\ c_1 \downarrow s_0;\ \forall s_1.\ s_0 -c_1\rightarrow s_1 \longrightarrow c_2 \downarrow s_1\ ]\!] \implies (c_1;c_2) \downarrow s_0$

| $[\![\ b\,s;\ c_1 \downarrow s\ ]\!] \implies IF\ b\ THEN\ c_1\ ELSE\ c_2 \downarrow s$
| $[\![\ \neg b\,s;\ c_2 \downarrow s\ ]\!] \implies IF\ b\ THEN\ c_1\ ELSE\ c_2 \downarrow s$

| $\neg b\,s \implies WHILE\ b\ DO\ c \downarrow s$
| $[\![\ b\,s;\ c \downarrow s;\ \forall t.\ s -c\rightarrow t \longrightarrow WHILE\ b\ DO\ c \downarrow t\ ]\!] \implies WHILE\ b\ DO\ c \downarrow s$

| $c \downarrow f\,s \implies LOCAL\ f;c;g \downarrow s$

Formulas and validity for total correctness

definition
hoare-tvalid :: *assn* ⇒ *com* ⇒ *assn* ⇒ *bool* ($\models_t$ {(1-)}/ (-)/ {(1-)} *50*) **where**
$\models_t \{P\}c\{Q\} \longleftrightarrow \models \{P\}c\{Q\} \wedge (\forall s.\ P\,s \longrightarrow c{\downarrow}s)$

Rules for total correctness

inductive
thoare :: *assn* ⇒ *com* ⇒ *assn* ⇒ *bool* ($\vdash_t$ ({(1-)}/ (-)/ {(1-)}) *50*)
where
Do: $\vdash_t \{\lambda s.\ (\forall t \in f\,s.\ P\,t) \wedge f\,s \neq \{\}\}\ Do\,f\,\{P\}$
| *Semi*: $[\![\ \vdash_t \{P\}c\{Q\};\ \vdash_t \{Q\}d\{R\}\ ]\!] \implies \vdash_t \{P\}\ c;d\ \{R\}$
| *If*: $[\![\ \vdash_t \{\lambda s.\ P\,s \wedge b\,s\}c\{Q\};\ \vdash_t \{\lambda s.\ P\,s \wedge \neg b\,s\}d\{Q\}\ ]\!] \implies$
$\vdash_t \{P\}\ IF\ b\ THEN\ c\ ELSE\ d\ \{Q\}$
| *While*:
$[\![wf\,r;\ \forall s'.\ \vdash_t \{\lambda s.\ P\,s \wedge b\,s \wedge s' = s\}\ c\ \{\lambda s.\ P\,s \wedge (s,s') \in r\}]\!]$
$\implies \vdash_t \{P\}\ WHILE\ b\ DO\ c\ \{\lambda s.\ P\,s \wedge \neg b\,s\}$
| *Conseq*: $[\![\ \forall s.\ P'\,s \longrightarrow P\,s;\ \vdash_t \{P\}c\{Q\};\ \forall s.\ Q\,s \longrightarrow Q'\,s\ ]\!] \implies$
$\vdash_t \{P'\}c\{Q'\}$

| *Local*: $(!!s.\ P\,s \implies P'\,s\,(f\,s)) \implies \forall ps.\ \vdash_t \{P'\,ps\}\ c\ \{Q\ o\ (g\ ps)\} \implies$
$\vdash_t \{P\}\ LOCAL\ f;c;g\ \{Q\}$

Language with one procedure, no parameters

typedecl *state*

type-synonym *bexp* = *state* ⇒ *bool*

datatype *com* = *Do* (*state* ⇒ *state set*)
| *Semi com com* (-; - [*60*, *60*] *10*)
| *Cond bexp com com* (*IF* - *THEN* - *ELSE* - *60*)
| *While bexp com* (*WHILE* - *DO* - *60*)
| *CALL*
| *Local* (*state* ⇒ *state*) *com* (*state* ⇒ *state* ⇒ *state*)
(*LOCAL* -; -; - [*0*,*0*,*60*] *60*)

consts *body* :: *com*

Semantics

inductive
exec :: *state* ⇒ *com* ⇒ *state* ⇒ *bool* (-/ −-→/ - [*50*,*0*,*50*] *50*)
where
Do: $t \in f\ s \Longrightarrow s -Do\ f\rightarrow t$

| *Semi*: $[\![s0 -c1\rightarrow s1;\ s1 -c2\rightarrow s2]\!] \Longrightarrow s0 -c1;c2\rightarrow s2$

| *IfTrue*: $[\![b\ s;\ s -c1\rightarrow t]\!] \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2\rightarrow t$
| *IfFalse*: $[\![\neg b\ s;\ s -c2\rightarrow t]\!] \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2\rightarrow t$

| *WhileFalse*: $\neg b\ s \Longrightarrow s -WHILE\ b\ DO\ c\rightarrow s$
| *WhileTrue*: $[\![b\ s;\ s -c\rightarrow t;\ t -WHILE\ b\ DO\ c\rightarrow u]\!] \Longrightarrow s -WHILE\ b\ DO\ c\rightarrow u$

| $s -body\rightarrow t \Longrightarrow s -CALL\rightarrow t$

| *Local*: $f\ s -c\rightarrow t \Longrightarrow s -LOCAL\ f;\ c;\ g\rightarrow g\ s\ t$

Fine-grain semantics

inductive
execn :: *state* ⇒ *com* ⇒ *nat* ⇒ *state* ⇒ *bool* (-/ −-*n*→/ - [*50*,*0*,*0*,*50*] *50*)
where
$t \in f\ s \Longrightarrow s -Do\ f-n\rightarrow t$

| $[\![s0 -c1-n\rightarrow s1;\ s1 -c2-n\rightarrow s2]\!] \Longrightarrow s0 -c1;c2-n\rightarrow s2$

| $[\![b\ s;\ s -c1-n\rightarrow t]\!] \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2-n\rightarrow t$
| $[\![\neg b\ s;\ s -c2-n\rightarrow t]\!] \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2-n\rightarrow t$

| $\neg b\ s \Longrightarrow s -WHILE\ b\ DO\ c-n\rightarrow s$
| $[\![b\ s;\ s -c-n\rightarrow t;\ t -WHILE\ b\ DO\ c-n\rightarrow u]\!] \Longrightarrow s -WHILE\ b\ DO\ c-n\rightarrow u$

| $s -body-n\rightarrow t \Longrightarrow s -CALL-Suc\ n\rightarrow t$

| $f\ s -c-n\rightarrow t \Longrightarrow s -LOCAL\ f;\ c;\ g-n\rightarrow g\ s\ t$

Example program and adaptation

- Simple recursive program:

  `proc = IF i=0 THEN SKIP ELSE i:=i-1; CALL; i:=i+1`

- Specification:

  `{ i=N } CALL { i=N }`

- Adaptation rules; how can we deduce

  `{ i=N-1 } CALL { i=N-1 }`

  from specification?

Formulas and validity

type-synonym *'a assn* = *'a* ⇒ *state* ⇒ *bool*

type-synonym *'a cntxt* = (*'a assn* × *com* × *'a assn*)*set*

definition
valid :: *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (⊨ {(1-)}/ (-)/ {(1-)} 50) **where**
⊨ {*P*}*c*{*Q*} ⟷ (∀ *s t*. *s* −*c*→ *t* ⟶ (∀ *z*. *P z s* ⟶ *Q z t*))

definition
valids :: *'a cntxt* ⇒ *bool* (||⊨ - 50) **where**
[*simp*]: ||⊨ *C* ≡ (∀ (*P*,*c*,*Q*) ∈ *C*. ⊨ {*P*}*c*{*Q*})

definition
cvalid :: *'a cntxt* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (- ⊨/ {(1-)}/ (-)/ {(1-)} 50) **where**
C ⊨ {*P*}*c*{*Q*} ⟷ ||⊨ *C* ⟶ ⊨ {*P*}*c*{*Q*}

Formulas and validity for fine-grain semantics

definition
nvalid :: *nat* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (⊨- {(1-)}/ (-)/ {(1-)} 50)
where
⊨*n* {*P*}*c*{*Q*} ≡ (∀ *s t*. *s* −*c*−*n*→ *t* ⟶ (∀ *z*. *P z s* ⟶ *Q z t*))

definition
nvalids :: *nat* ⇒ *'a cntxt* ⇒ *bool* (||⊨'--/ - 50) **where**
||⊨-*n C* ≡ (∀ (*P*,*c*,*Q*) ∈ *C*. ⊨*n* {*P*}*c*{*Q*})

definition
cnvalid :: *'a cntxt* ⇒ *nat* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (- ⊨-/ {(1-)}/ (-)/ {(1-)} 50) **where**
C ⊨*n* {*P*}*c*{*Q*} ⟷ ||⊨-*n C* ⟶ ⊨*n* {*P*}*c*{*Q*}

Rules

inductive
hoare :: *'a cntxt* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (- ⊢/ ({(1-)}/ (-)/ {(1-)}) 50) **where**
C ⊢ {λ*z s*. ∀ *t* ∈ *f s* . *P z t*} *Do f* {*P*}
| ⟦ *C* ⊢ {*P*}*c1*{*Q*}; *C* ⊢ {*Q*}*c2*{*R*} ⟧ ⟹ *C* ⊢ {*P*} *c1*;*c2* {*R*}

| ⟦ *C* ⊢ {λ*z s*. *P z s* ∧ *b s*}*c1*{*Q*}; *C* ⊢ {λ*z s*. *P z s* ∧ ¬*b s*}*c2*{*Q*} ⟧
⟹ *C* ⊢ {*P*} *IF b THEN c1 ELSE c2* {*Q*}

| *C* ⊢ {λ*z s*. *P z s* ∧ *b s*} *c* {*P*}
⟹ *C* ⊢ {*P*} *WHILE b DO c* {λ*z s*. *P z s* ∧ ¬*b s*}

| ⟦ *C* ⊢ {*P'*}*c*{*Q'*}; ∀ *s t*. (∀ *z*. *P' z s* ⟶ *Q' z t*) ⟶ (∀ *z*. *P z s* ⟶ *Q z t*) ⟧
⟹ *C* ⊢ {*P*}*c*{*Q*}

| {(*P*,*CALL*,*Q*)} ⊢ {*P*}*body*{*Q*} ⟹ {} ⊢ {*P*} *CALL* {*Q*}

| {(*P*,*CALL*,*Q*)} ⊢ {*P*} *CALL* {*Q*}
| ⟦ ∀ *s'*. *C* ⊢ {λ*z s*. *P z s'* ∧ *s* = *f s'*} *c* {λ*z t*. *Q z* (*g s' t*)} ⟧ ⟹
C ⊢ {*P*} *LOCAL f*;*c*;*g* {*Q*}

Final remarks on language extensions

- Nondeterminism and many other aspects can be handled without conceptual extensions
- Termination proofs need well-founded orderings
- Procedures need assumptions/contexts, logical variables and adaptation rules

Section 8.6

Verifying procedural, heap-manipulating programs

General approach 1

- Specify programming language syntax and semantics in an interactive theorem prover for HOL
- Use HOL as specification language for program properties
- Use the HOL proof system for verification

Example: Verifying Simpl programs

Simpl (by N. Schirmer [Archives of formal proofs]):

A sequential imperative programming language:

- mutually recursive procedures
- abrupt termination and exceptions
- runtime faults
- local and global variables
- pointers and heaps
- expressions with side-effects
- pointers to procedures
- partial application and closures
- dynamic method invocation
- unbounded nondeterminism

Example program: Quicksort on heap lists

Illustrating:

- Handling of programming language aspects:
  - recursive procedures
  - local and global variables
  - pointers and heaps
- Handling of specification aspects:
  - heap-manipulation using abstraction
  - frame properties
- Problems of embedding

Modeling of a program specific state

Heap for singly-linked lists (sll-heaps):

```
record globals_heap =
  next_' :: "ref ⇒ ref"
  cont_' :: "ref ⇒ nat"
```

A predicate to abstract sll-heaps:

```
primrec List :: ref ⇒ (ref⇒ref) ⇒ ref list ⇒ bool
where
  List x h []     = (x = Null) |
  List x h (p#ps) = (x = p ∧ x ≠ Null ∧ List (h x) h ps)
```


Modeling of a program specific state (2)

The variables for procedures `append` and `quickSort`

```
record 'g vars = "'g state" +
  p_'    :: "ref"
  q_'    :: "ref"
  le_'   :: "ref"
  gt_'   :: "ref"
  hd_'   :: "ref"
  tl_'   :: "ref"
```


Implementation and specification of procedure `append`

```
procedures  append(p,q|p) =
  "IF ´p=Null THEN ´p :=´q
   ELSE ´p→´next :== CALL append(´p→´next,´q) FI"

  append_spec:
   "∀σ Ps Qs. Γ⊢
      { σ. List ´p ´next Ps ∧ List ´q ´next Qs ∧
        set Ps ∩ set Qs = {} }
          ´p :== PROC append(´p,´q)
      { List ´p ´next (Ps@Qs) ∧
        (∀x. x∉set Ps ⟶ ´next x =  σnext x) }"

  append_modifies:
   "∀σ. Γ⊢
      { σ }       ´p :== PROC append(´p,´q)
      { t. t may_only_modify_globals σ in [next] }"
```


Implementation of procedure `quickSort`

```
procedures quickSort(p|p) =
 "IF ´p=Null THEN SKIP
  ELSE ´tl :== ´p→´next;; ´le :== Null;; ´gt :== Null;;
       WHILE ´tl≠Null DO
         ´hd :== ´tl;;  ´tl :== ´tl→´next;;
         IF ´hd→´cont ≤ ´p→´cont
            THEN ´hd→´next :== ´le;; ´le :== ´hd
            ELSE ´hd→´next :== ´gt;; ´gt :== ´hd
         FI
       OD;;
       ´le :== CALL quickSort(´le);;
       ´gt :== CALL quickSort(´gt);;
       ´p→´next :== ´gt;;
       ´le :== CALL append(´le,´p);;
       ´p :== ´le
  FI"
```