

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

typed decl *state*

type-synonym *bexp = state \Rightarrow bool*

datatype *com = Do (state \Rightarrow state set)*

<i>Semi com com</i>	(<i>-; - [60, 60] 10</i>)
<i>Cond bexp com com</i>	(<i>IF - THEN - ELSE - 60</i>)
<i>While bexp com</i>	(<i>WHILE - DO - 60</i>)
<i>Local (state \Rightarrow state) com (state \Rightarrow state \Rightarrow state)</i>	(<i>LOCAL -; -; - [0,0,60] 60</i>)

Semantics of abstract while-language

inductive

$exec :: state \Rightarrow com \Rightarrow state \Rightarrow bool \text{ (-/ ---\to/ - [50,0,50] 50)}$

where

$t \in f s \Longrightarrow s -Do f \rightarrow t$

| $\llbracket s0 -c1 \rightarrow s1; s1 -c2 \rightarrow s2 \rrbracket \Longrightarrow s0 -c1;c2 \rightarrow s2$

| $\llbracket b s; s -c1 \rightarrow t \rrbracket \Longrightarrow s -IF b THEN c1 ELSE c2 \rightarrow t$

| $\llbracket \neg b s; s -c2 \rightarrow t \rrbracket \Longrightarrow s -IF b THEN c1 ELSE c2 \rightarrow t$

| $\neg b s \Longrightarrow s -WHILE b DO c \rightarrow s$

| $\llbracket b s; s -c \rightarrow t; t -WHILE b DO c \rightarrow u \rrbracket \Longrightarrow s -WHILE b DO c \rightarrow u$

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

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

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

Rules for partial correctness

inductive

hoare :: *assn* \Rightarrow *com* \Rightarrow *assn* \Rightarrow *bool* (\vdash ($\{(1-)\}$ / $(-)$ / $\{(1-)\}$) 50)

where

$\vdash \{\lambda s. \forall t \in f s. P t\} Do f \{P\}$

| $\llbracket \vdash \{P\} c1 \{Q\}; \vdash \{Q\} c2 \{R\} \rrbracket \Longrightarrow \vdash \{P\} c1;c2 \{R\}$

| $\llbracket \vdash \{\lambda s. P s \wedge b s\} c1 \{Q\}; \vdash \{\lambda s. P s \wedge \neg b s\} c2 \{Q\} \rrbracket$
 $\Longrightarrow \vdash \{P\} IF b THEN c1 ELSE c2 \{Q\}$

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

| $\llbracket \forall s. P' s \longrightarrow P s; \vdash \{P\} c \{Q\}; \forall s. Q s \longrightarrow Q' s \rrbracket \Longrightarrow \vdash \{P'\} c \{Q'\}$

| $\llbracket \bigwedge s. P s \Longrightarrow P' s (f s); \forall s. \vdash \{P' s\} c \{Q \circ (g s)\} \rrbracket \Longrightarrow$
 $\vdash \{P\} LOCAL f;c;g \{Q\}$

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

$termi :: com \Rightarrow state \Rightarrow bool$ (**infixl** \downarrow 50)

where

$f\ s \neq \{\}$ \Longrightarrow $Do\ f \downarrow s$

| $\llbracket c_1 \downarrow s_0; \forall s_1. s_0 -c_1 \rightarrow s_1 \longrightarrow c_2 \downarrow s_1 \rrbracket \Longrightarrow (c_1; c_2) \downarrow s_0$

| $\llbracket b\ s; c_1 \downarrow s \rrbracket \Longrightarrow IF\ b\ THEN\ c_1\ ELSE\ c_2 \downarrow s$

| $\llbracket \neg b\ s; c_2 \downarrow s \rrbracket \Longrightarrow IF\ b\ THEN\ c_1\ ELSE\ c_2 \downarrow s$

| $\neg b\ s \Longrightarrow WHILE\ b\ DO\ c \downarrow s$

| $\llbracket b\ s; c \downarrow s; \forall t. s -c \rightarrow t \longrightarrow WHILE\ b\ DO\ c \downarrow t \rrbracket \Longrightarrow WHILE\ b\ DO\ c \downarrow s$

| $c \downarrow f\ s \Longrightarrow LOCAL\ f; c; g \downarrow s$

Formulas and validity for total correctness

definition

hoare-tvalid :: *assn* \Rightarrow *com* \Rightarrow *assn* \Rightarrow *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* \Rightarrow *com* \Rightarrow *assn* \Rightarrow *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*: $\llbracket \vdash_t \{ P \} c \{ Q \}; \vdash_t \{ Q \} d \{ R \} \rrbracket \Longrightarrow \vdash_t \{ P \} c; d \{ R \}$
- | *If*: $\llbracket \vdash_t \{ \lambda s. P s \wedge b s \} c \{ Q \}; \vdash_t \{ \lambda s. P s \wedge \neg b s \} d \{ Q \} \rrbracket \Longrightarrow$
 $\vdash_t \{ P \} IF b THEN c ELSE d \{ Q \}$
- | *While*:
 $\llbracket wf r; \forall s'. \vdash_t \{ \lambda s. P s \wedge b s \wedge s' = s \} c \{ \lambda s. P s \wedge (s, s') \in r \} \rrbracket$
 $\Longrightarrow \vdash_t \{ P \} WHILE b DO c \{ \lambda s. P s \wedge \neg b s \}$
- | *Conseq*: $\llbracket \forall s. P' s \longrightarrow P s; \vdash_t \{ P \} c \{ Q \}; \forall s. Q s \longrightarrow Q' s \rrbracket \Longrightarrow$
 $\vdash_t \{ P' \} c \{ Q' \}$
- | *Local*: $(\llbracket \lambda s. P s \Longrightarrow P' s (f s) \rrbracket) \Longrightarrow \forall ps. \vdash_t \{ P' ps \} c \{ Q o (g ps) \} \Longrightarrow$
 $\vdash_t \{ P \} LOCAL f; c; g \{ Q \}$

Language with one procedure, no parameters

typed decl *state*

type-synonym *bexp* = *state* \Rightarrow *bool*

datatype *com* = *Do* (*state* \Rightarrow *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* \Rightarrow *state*) *com* (*state* \Rightarrow *state* \Rightarrow *state*)
- (*LOCAL -; -; - [0,0,60] 60*)

consts *body* :: *com*

Semantics

inductive

$exec :: state \Rightarrow com \Rightarrow state \Rightarrow bool \quad (-/ \dashrightarrow/ - [50,0,50] 50)$

where

$Do: \quad t \in fs \Longrightarrow s - Do f \rightarrow t$

| $Semi: \quad \llbracket s0 - c1 \rightarrow s1; s1 - c2 \rightarrow s2 \rrbracket$
 $\Longrightarrow s0 - c1; c2 \rightarrow s2$

| $IfTrue: \quad \llbracket b s; s - c1 \rightarrow t \rrbracket \Longrightarrow s - IF b THEN c1 ELSE c2 \rightarrow t$

| $IfFalse: \quad \llbracket \neg b s; s - c2 \rightarrow t \rrbracket \Longrightarrow s - IF b THEN c1 ELSE c2 \rightarrow t$

| $WhileFalse: \quad \neg b s \Longrightarrow s - WHILE b DO c \rightarrow s$

| $WhileTrue: \quad \llbracket b s; s - c \rightarrow t; t - WHILE b DO c \rightarrow u \rrbracket$
 $\Longrightarrow s - WHILE b DO c \rightarrow u$

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

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

Fine-grain semantics

inductive

$execn :: state \Rightarrow com \Rightarrow nat \Rightarrow state \Rightarrow bool \quad (-/ \dashrightarrow/ - [50,0,0,50] 50)$

where

$t \in fs \Longrightarrow s -Do\ f-n \rightarrow t$

$| \llbracket s0 -c1-n \rightarrow s1; s1 -c2-n \rightarrow s2 \rrbracket \Longrightarrow s0 -c1;c2-n \rightarrow s2$

$| \llbracket b\ s; s -c1-n \rightarrow t \rrbracket \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2-n \rightarrow t$

$| \llbracket \neg b\ s; s -c2-n \rightarrow t \rrbracket \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2-n \rightarrow t$

$| \neg b\ s \Longrightarrow s -WHILE\ b\ DO\ c-n \rightarrow s$

$| \llbracket b\ s; s -c-n \rightarrow t; t -WHILE\ b\ DO\ c-n \rightarrow u \rrbracket \Longrightarrow s -WHILE\ b\ DO\ c-n \rightarrow u$

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

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

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 $'a \text{ assn} = 'a \Rightarrow \text{state} \Rightarrow \text{bool}$

type-synonym $'a \text{ cntxt} = ('a \text{ assn} \times \text{com} \times 'a \text{ assn})\text{set}$

definition

$\text{valid} :: 'a \text{ assn} \Rightarrow \text{com} \Rightarrow 'a \text{ assn} \Rightarrow \text{bool}$ ($\models \{(1-)\} / (-) / \{(1-)\} 50$) **where**
 $\models \{P\}c\{Q\} \longleftrightarrow (\forall s t. s \text{ --}c\text{--} t \longrightarrow (\forall z. P z s \longrightarrow Q z t))$

definition

$\text{valids} :: 'a \text{ cntxt} \Rightarrow \text{bool}$ ($\|\models - 50$) **where**
 $[\text{simp}]: \|\models C \equiv (\forall (P,c,Q) \in C. \models \{P\}c\{Q\})$

definition

$\text{cvalid} :: 'a \text{ cntxt} \Rightarrow 'a \text{ assn} \Rightarrow \text{com} \Rightarrow 'a \text{ assn} \Rightarrow \text{bool}$ ($- \models / \{(1-)\} / (-) / \{(1-)\} 50$) **where**
 $C \models \{P\}c\{Q\} \longleftrightarrow \|\models C \longrightarrow \models \{P\}c\{Q\}$

Formulas and validity for fine-grain semantics

definition

$nvalid :: nat \Rightarrow 'a\ assn \Rightarrow com \Rightarrow 'a\ assn \Rightarrow bool (\models - \{ (1-) \} / (-) / \{ (1-) \} 50)$

where

$\models_n \{P\}c\{Q\} \equiv (\forall s\ t. s \text{ -- } c \text{ -- } n \rightarrow t \longrightarrow (\forall z. P\ z\ s \longrightarrow Q\ z\ t))$

definition

$nvalids :: nat \Rightarrow 'a\ cntxt \Rightarrow bool (\models' - / - 50)$ **where**

$\models'_n C \equiv (\forall (P,c,Q) \in C. \models_n \{P\}c\{Q\})$

definition

$cnvalid :: 'a\ cntxt \Rightarrow nat \Rightarrow 'a\ assn \Rightarrow com \Rightarrow 'a\ assn \Rightarrow bool (- \models - / \{ (1-) \} / (-) / \{ (1-) \} 50)$ **where**

$C \models_n \{P\}c\{Q\} \iff \models'_n C \longrightarrow \models_n \{P\}c\{Q\}$

Rules

inductive

hoare :: 'a cntxt \Rightarrow 'a assn \Rightarrow com \Rightarrow 'a assn \Rightarrow bool (- \vdash / ($\{(1-)\}$ / (-) / $\{(1-)\}$) 50) **where**

- $C \vdash \{\lambda z s. \forall t \in f s. P z t\} \text{ Do } f \{P\}$
- $\llbracket C \vdash \{P\} c1 \{Q\}; C \vdash \{Q\} c2 \{R\} \rrbracket \Longrightarrow C \vdash \{P\} c1; c2 \{R\}$
- $\llbracket C \vdash \{\lambda z s. P z s \wedge b s\} c1 \{Q\}; C \vdash \{\lambda z s. P z s \wedge \neg b s\} c2 \{Q\} \rrbracket \Longrightarrow C \vdash \{P\} \text{ IF } b \text{ THEN } c1 \text{ ELSE } c2 \{Q\}$
- $C \vdash \{\lambda z s. P z s \wedge b s\} c \{P\} \Longrightarrow C \vdash \{P\} \text{ WHILE } b \text{ DO } c \{\lambda z s. P z s \wedge \neg b s\}$
- $\llbracket C \vdash \{P'\} c \{Q'\}; \forall s t. (\forall z. P' z s \longrightarrow Q' z t) \longrightarrow (\forall z. P z s \longrightarrow Q z t) \rrbracket \Longrightarrow C \vdash \{P\} c \{Q\}$
- $\{(P, \text{CALL}, Q)\} \vdash \{P\} \text{ body} \{Q\} \Longrightarrow \{\} \vdash \{P\} \text{ CALL} \{Q\}$
- $\{(P, \text{CALL}, Q)\} \vdash \{P\} \text{ CALL} \{Q\}$
- $\llbracket \forall s'. C \vdash \{\lambda z s. P z s' \wedge s = f s'\} c \{\lambda z t. Q z (g s' t)\} \rrbracket \Longrightarrow C \vdash \{P\} \text{ LOCAL } f; c; g \{Q\}$

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"
```