Hoare Logic, and its embedding in HOL

Wishnu Prasetya

wishnu@cs.uu.nl
www.cs.uu.nl/docs/vakken/pv

Hoare Logic

- Is a simple and intuitive logic to prove the correctness of a sequential imperative programs.

- Programs are specified by "Hoare triples", like :

$$\{ \#S>0 \} \quad P(S) \quad \{ return \in S \ \wedge \ (\forall x: x \in S : return \geq x) \}$$

- *partial correctness* or *total correctness* interpretation.

Examples of the proof rules

- You can weaken a post-condition :

$$\frac{\{P\}\ S\ \{Q\} \quad , \quad Q \Rightarrow R}{\{P\}\ S\ \{R\}}$$

- Analogously, you can weaken pre-condition.
- Hoare triples are conjunctive and disjunctive.

Dealing with seq

$$\frac{\{P\}\ S_1\ \{Q\}\quad ,\quad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1 ; S_2\ \{R\}}$$

Require you to come up with Q…. we'll get back to this.

IF-rule

- Deterministic

$$\frac{\{P \wedge g\}\ S1\ \{Q\}\quad,\quad \{P \wedge \neg g\}\ S2\ \{Q\}}{\{P\}\ \textbf{if}\ g\ \textbf{then}\ S1\ \textbf{else}\ S2\ \{Q\}}$$

- Non-deterministic

$$\frac{\{P \wedge g\}\ S1\ \{Q\}\quad,\quad \{P \wedge h\}\ S2\ \{Q\}\quad,\quad P \Rightarrow g \vee h}{\begin{array}{l}\{P\}\ \textbf{if}\ g \to S1 \\ \qquad\quad [\,]\ h \to S2 \\ \qquad\quad \textbf{fi}\ \{Q\}\end{array}}$$

Dealing with assignment

- By introducing fresh variable representing the new value of x :

$$\frac{P \;\wedge\; x' = e \;\Rightarrow\; Q[x'/x]}{\{\,P\,\}\;\; x := e \;\;\{\,Q\,\}}$$

- Or shorter:

$$\frac{P \;\Rightarrow\; Q[e/x]}{\{\,P\,\}\;\; x := e \;\;\{\,Q\,\}}$$

Recall the intermediate assertion problem in SEQ …

- Try to come up with an algorithm :

  pre : Statement $\rightarrow$ Predicate $\rightarrow$ Predicate

  that given a statement S and a post-condition Q constructs a pre-condition from which S ends up in Q.

- Would give you a way to calculate Q in the SEQ-rule.

Weakest pre-condition

- That is, we can have this inference rule :

$$\frac{P \quad \Rightarrow \quad \textbf{pre}\ \ S_1 \ \ (\textbf{pre}\ \ S_2\ \ R)}{\{\,P\,\}\ \ S_1\,;\,S_2\ \ \{\,R\,\}}$$

- Can be incomplete. It may give you a pre-cond that is too strong for your actual pre-cond to imply.

- Complete: ‘weakest pre-condition’ (wp)

Weakest pre-condition

- Defined/characterized by:

$$\textbf{wp}\ T\ Q \;=\; \{\ s \mid T\,s \subseteq Q\ \}$$

$$\{P\}\ T\ \{Q\} \;=\; P \Rightarrow \textbf{wp}\ T\ Q$$

- But these are not constructive definitions.

WP

- **wp** $(x := e)\ Q \quad = \quad Q[e/x]$

- **wp** $(S_1 ; S_2)\ Q \quad = \quad$ **wp** S_1 (**wp** S_2 Q)

- **wp** (if g then S_1 else S_2) Q

  $=$

  $(g \Rightarrow \textbf{wp}\ S_1\ Q) \wedge (\neg g \Rightarrow \textbf{wp}\ S_2\ Q)$

Example

- Prove:

$\{^*\ x \neq y\ ^*\}$ tmp:= x ; x:=y ; y:=tmp $\{^*\ x \neq y\ ^*\}$

- We do this with the help of intermediate assertions:

$\{^*\ x \neq y\ ^*\}$ tmp:= x { ? } ; x:=y { ? } ; y:=tmp $\{^*\ x \neq y\ ^*\}$

- Use the strategy “calculate sufficient pre-condition” to fill-in the ? assertions.

Loop

- A loop is correct if you can find an "invariant" :

$$\frac{\begin{array}{c} P \Rightarrow I \\ \{\, g \;\wedge\; I \,\} \quad S \quad \{\, I \,\} \\ I \;\wedge \neg g \;\Rightarrow\; Q \end{array}}{\{\, P \,\} \;\; \underline{\textbf{while}} \;\; g \;\; \underline{\textbf{do}} \;\; S \;\; \{\, Q \,\}}$$

- E.g. a trivial loop:

{ i=10 } **while** i>0 **do** i=i-1 { i=0 }

Examples

- A program to sum the elements of array a:

```
{ s=0 }

  i=#a ;

  while i>0  do  { i=i-1 ;    s=s+a[i] }

{   s =  SUM(a[0..#a))   }
```

Examples

- A program to search in an array :

```
{ true }

i = 0 ;
found = false ;

while i<#b /\ ¬found  do  { found=b[i] ; i=i+1}

{ found = (∃ j: 0≤j<#b: b[j]) }
```

- To note: invariant can be used as *abstraction*!

Rule for proving termination (of loop)

- Extend the previous rule to:

$P \Rightarrow I$ // setting up I

$\{ g \land I \} \; S \; \{ I \}$ // invariance

$I \land \neg g \Rightarrow Q$ // exit cond

$\{ I \land g \} \; C{:=}m; S \; \{ m<C \}$ // m decreasing

$I \land g \Rightarrow m > 0$ // m bounded below

$\{ P \}$ <u>while</u> g <u>do</u> S $\{ Q \}$

Example

- Bag of red and blue candy. Blues are delicious!

  Mom: “Bob, you can take one everyday. When it’s empty we’ll buy a new bag.”

  Bob: “Yay!”

  Mom: ”Oh, .. if you take a blue, put two new reds in the bag.”

  Bob: “But mom … the bag then will never be empty!”

  Mom: “Oh it will be.”
- Proof ?

Example

- Simulate with this non-deterministic program:

$\{ \ r \geq 0 \ \wedge \ b \geq 0 \ \}$

while $r+b>0$ **do** {
  if $r>0 \ \rightarrow \ r{:=}r-1$
  $[] \ \ b>0 \rightarrow \{ \ b{:=}b-1; \ \ r{:=}r+2 \ \}$
  fi
  }

$\{ \ r=0 \ \ \wedge \ \ b=0 \ \}$

non-deterministic if …

- Sufficient to prove the correctness of this simulation.

Automation support ?

- Custom tools like Esc/Java and Spec#
- You can also embed Hoare logic in HOL:
  - Come up with a way to model specifications (various possibilities)
  - Add axioms to represent your Hoare inference rules
  - Or safer: prove these ‘axioms’ instead

- Model of a “specification” $\leftarrow$ need these concepts :
  - program, statement, expression
  - predicate
  - state

Model of states

- Many ways to model states, with pros/cons.
- E.g. using record:

$$\boxed{\{\ x\#0\ \ ,\ \ y\#9\ \}}$$

- Or function : $\boxed{\text{type } \textit{state} = \textit{varName} \rightarrow \textit{value}}$

- This is *abstract*! Actual state of P may consists of the value of CPU registers, stacks etc.
- Let Σ denotes the space of all possible states of S.

Model of expression & predicate

- An expression is modeled by a function $\Sigma \rightarrow$ val
- A (state) predicate is an expression that returns a bool; so it is a function $\Sigma \rightarrow$ bool

  e.g. $\boxed{x > y}$ is modeled by $\boxed{(\lambda s.\ s\ x\ > s\ y)}$

- $\vdash$ P means $(\forall s: s \in \Sigma : P\ s)$

Predicate as a set

- Sets over Σ and predicates over Σ (so, $\Sigma \to$ bool) are isomorphic, with this bijection :

- $\chi(P) \quad = \quad \{\ s \mid P\, s\ \}$
- $\chi^{-1}(U) \ = \ (\lambda s.\ s \in U)$

- Standard operators on predicates ($\land$, $\lor$, $\neg$, etc) have the corresponding operators on sets ($\cap$, $\cup$, complement).
- We'll use them interchangebly.

Implication corresponds to subset

- $\boxed{\vdash \; P \Rightarrow Q}$ // $P \Rightarrow Q$ is valid

  This means: $\forall s{::}\; P\,s \Rightarrow Q\,s$

  In terms of set this is equivalent to: $\chi(P) \subseteq \chi(Q)$

- And to confuse you ☺, the often used jargon:
  - P is ***stronger*** than Q
  - Q is ***weaker*** than P
  - Observe that in term of sets, stronger means smaller!

Model of a program

- We can model a *deterministic* program *T* by a function that takes an initial state, and returns the final state:

$$\boxed{T : \Sigma \to \Sigma}$$

- *Non-deterministic* program T:

$$\boxed{T : \Sigma \to \Sigma \text{ set}}$$

(for a given state s, T s gives us the set of all possible end-states if T is executed in s)

“Embedding” in HOL

- We’ll model a statement as a relation over states:

$$\text{stmt} \;:\; \text{‘state} \rightarrow \underbrace{\text{‘state} \rightarrow \text{bool}}_{\textit{see this as a set of ‘state}}$$

type variable

Idea:

$\boxed{\textit{stmt s t} \;=\; \textit{true} \quad \textit{iff} \quad \textit{t is a possible end state when stmt is executed on s.}}$

Alternatively: stmt s gives the set of all possible end states when stmt is executed on s.

Representing Skip and Assignment

Define ` **SKIP** = $(\backslash s\ t.\ t = s)$ `

- We represent states as s : ‘*var* $\rightarrow$ ‘*value*.

  $s\ x$ → gives the value of variable x in this state

  Represent expressions as e : *state* $\rightarrow$ ‘*value*

Define ` **ASG** *var e*
=
$(\backslash s\ t.\ (\forall x.$ **if** $x = var$ **then** $t\ x = e\ s$ **else** $t\ x = s\ x))$

Examples

- What does S1 do?

  Define `S1 = ASG "x" (\s. s "x" + 1)`

- Define `*A* UNION *B* = (\*x y*. *A x y* \/ *B x y*)`

- W hat does this do?

  SKIP UNION S1

Sequencing and branching

- new_infix ("THEN", ...) ;

Define \`S_1 **THEN** S_2
=
(\\s u. (∃t. S_1 s t ∧ S_2 t u)) \` ;

Define \`**IF** g S_1 S_2
=
(\\s t. **if** g s **then** S_1 s t **else** S_2 s t) \` ;

And loop...

- **Define** \`(**ITERATE** *g* *Stmt* **0** = SKIP)
  /\
  (**ITERATE** *g* *Stmt* (**SUC *n***)
  =
  IF *g* (*Stmt* THEN (ITERATE *g* *Stmt* *n*)) SKIP)\`

Define \`**WHILE** *g* *Stmt*
=
(*s* *t*. (∃*n*. ITERATE *g* *Stmt* *n* *s* *t* /\ ~*g* *t*)) \` ;

Representing Hoare triple

$$\boxed{\{\, P \,\}\ \ Stmt\ \ \{\, Q \,\}}$$

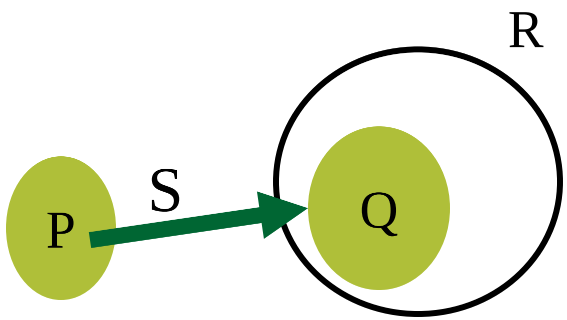
Define `**HOARE** *P* *Stmt* *Q*
=
($\forall$*s t.* *P s* $\wedge$ *Stmt s t* ==> *Q* *t*)`

Notice that this is "partial correctness" !

- Still missing : Hoare logic's inference rules
- Options:
  - introduce as axioms
  - prove them

Example, proving Post-condition weakening rule

$$\frac{\{P\}\ S\ \{Q\}\quad ,\quad \vdash Q \Rightarrow R}{\{P\}\ S\ \{R\}}$$



In HOL, prove this PostW_thm :

HOA RE P $Stmt$ Q $\wedge$ $(\forall s.\ Q\ s \Longrightarrow R\ s)$
==>
HOA RE P $Stmt$ R

But wait … that is a *theorem*! I want a rule…

Rule for sequencing

$$\frac{\{ P \}\ S_1\ \{ Q \}\quad ,\quad \{ Q \}\ S_2\ \{ R \}}{\{ P \}\ S_1 ; S_2\ \{ R \}}$$

In HOL, prove this SEQ_thm :

HOARE *P S1 Q* /\ HOARE *Q S2 R*
==>
HOARE *P (S1 THEN S2) R*

Example

{ N ≥ 0 }

n :=N ; x := 0 ;
while n>0 **do** { n-- ; x := 2+x }

{ x = 2N }

Model state by a function mapping variable name to int.

Represent variable name by int.

Define `vn=0` ;
Define `vx=1` ;

val **ProgTrivial** = --`

ASSIGN vn (\s. N) **THEN** ASSIGN vx (\s. 0)

THEN

WHILE (\s. s vn > 0) (ASSIGN vn … THEN ASSIGN vx …)

`--
;

represented a plain HOL-term, rather than a HOL definition (but you could choose to do so).

Verification

```
val ProgTrivial_Spec_thm = prove(

  --`N>=0
    ==>
    HOARE   (\s. T)   ^ProgTrivial   (\s. s vx = 2*N)  `--,

    STRIP_TAC
    THEN MATCH_MP_TAC (GEN_ALL THEN_thm)
    THEN EXISTS_TAC (--`(\s. (s vx + 2 * s vn = 2*N) /\ s vn >= 0)`--)
    THEN REPEAT CONJ_TAC
    THENL
    [ RW_TAC int_ss [HOARE_def] THEN COOPER_TAC,
      MATCH_MP_TAC (GEN_ALL WHILE_thm)
      THEN RW_TAC int_ss [HOARE_def] THEN COOPER_TAC
    ]
) ;
```