Hoare Logic

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 :

$\{ \#S>0 \}$ getMax(S) $\{ return \in S \ /\backslash \ (\forall x: x \in S : return \geq x) \}$

- *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

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

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

IF-rule

- Deterministic

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

- Non-deterministic if

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

This version does not allow a situation where none of the guards is true. We can drop this restriction, but first you need to define what the statement should do in that case. E.g. if you say that it will then just do a skip, then we need to replace the third condition with P /\ ~g /\ ~h ==> Q.

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:

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

Loop

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

$$\frac{P \Rightarrow I \qquad \{ g \wedge I \}\ S\ \{ I \} \qquad I \wedge \neg g \Rightarrow Q}{\{ P \}\ \textbf{while}\ g\ \textbf{do}\ S\ \{ Q \}}$$

- E.g. a trivial loop:

$\{ i \geq 0 \}$ **while** $i>0$ **do** $i=i-1$ $\{ i=0 \}$

Loop, few more examples

{ i>0 /\ even(i) } **while** i<N **do** { i = i*2 } { even(i) /\ i>N }

assuming N≥0

{ true } i:=0; **while** i<#a **do** { a[i]=3 ; i++ }

{ all elements of a is 3 }

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

Loop, few more examples

- A program to check if an array b contains ‘true’ :

{ true }

i = 0 ;
found = false ;
while i<#b /\ ¬found **do** { found=b[i] ; i=i+1}

{ found = (∃ j: 0≤j<#b: b[j]) }

Rule for proving termination (of loop)

- Extend the previous rule to:

$P \Rightarrow I$ // setting up I

$\{ g \wedge I \}\ \ S\ \ \{ I \}$ // invariance

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

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

$I \wedge g \Rightarrow\ m > 0$ // 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.”

- Prove that any sequence that Bob’s actions will terminate.

Example

- We model Bob's actions with a non-deterministic program:

```
{ r ≥0  /\  b ≥0 }

while  r+b>0  do {
   if  r>0  →  r:=r-1
   []  b>0 → { b:=b-1;  r:=r+2 }
   fi
   }

{ r=0   /\  b=0 }
```

non-deterministic if …

- Sufficient to prove the correctness of this model.

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 “consistent” pre-condition from which S ends up in Q.

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

Using suf-pre …

- That is, we can have this inference rule :

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

- 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

- Can be formally defined/characterized by:

wp Stmt Q = { s | executing Stmt in s ends up in Q }

{ P } Stmt { Q } = P $\Rightarrow$ **wp** Stmt Q

- But these are not constructive definitions.

WP

- **wp** (x:=e) Q = Q[e/x]
- **wp** (S_1 ; S_2) Q = **wp** S_1 (**wp** S_2 Q)
- **wp** (if g then S_1 else S_2) Q
  =
  (g $\Rightarrow$ **wp** S_1 Q) $\wedge$ ($\neg$g $\Rightarrow$ **wp** S_2 Q)
- No constructive definition for loop nor recursion!

Example

- Prove:

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

- We do this with the help of intermediate assertions:

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

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

Abstract model, overview

- *Abstract model* of a program S: abstractly models the effect of a program.
- To convince yourself that H.L. is *sound*, in particular when you need to extend H.L.
- We will just look at a simplistic model, for the purpose of illustration.
- The elements:
  - Models of: state, expression, and predicate
  - Model of statement/program
  - Semantic of Hoare triple in terms of those models

Model of states

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

{ x#0 , y#9 }

- Or, with a function *VarName* $\rightarrow$ *Value*
- In any case, those models are *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.

Simplistic model of expression & predicate

- An expression is modeled by a function $\Sigma \rightarrow$ val
- We do not take exception and non-termination of the expr into account
- A (state) predicate is an expression that returns a bool; so it is a function $\Sigma \rightarrow$ bool
  e.g.

  x>y is modeled by $(\lambda s.\ s.x > s.y)$

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

Simplistic model of a program

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

$$T : \Sigma \rightarrow \Sigma$$

- We can only model deterministic programs in this way.

Simplistic model of a program

- Instead, we'll model a program by a function :

$$T : \Sigma \rightarrow \text{pow}(\Sigma)$$

such that for any state s, T s gives us the set of all possible end-states if T is executed in s.

- We will assume T terminates.

Specification

- Now we have enough to define what a specification means:

$$\{\, P \,\}\quad Stmt\quad \{\; Q \;\}$$
$$=$$
$$(\forall s ::\; P\, s \;\Rightarrow (\forall t : t \in Stmt\, s \Rightarrow Q\, t))$$

We assume S terminates (so, we use the partial correctness interpretation of Hoare triple).

How to proceed to prove the soundness of the logic?

- You need to provide the semantic of every programming constructs (assignment, if-else, loop, etc) in terms of our abstract semantic.

- Then you can prove all the inference rules you saw before.

Example, rule of SEQ

- If $f : A \to \text{pow } B$, we *lift* it to pow A, *overloading* :

$$f\ V\ =\ (\cup x :\ x \in V : f\ x)$$

- Semantic of ";" $\quad (S_1 ; S_2)\ s\ =\ S_2\ (S_1\ s)$

- From this we have enough formalism to prove :

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

Refining the semantic

- How to express non-termination → e.g. by modeling a program by:

$$Stmt : \Sigma_\perp \rightarrow \text{pow}(\Sigma_\perp)$$

where $\Sigma_\perp = \Sigma \cup \{\perp\}$, and $\perp$ represents non-termination. Furthermore, $Stmt\ \perp = \{\perp\}$.

- Specification {P} Stmt {Q} now means:

$$\begin{aligned}(\forall s: s \neq \perp:\ P\ s\ \Rightarrow\ & \perp \notin Stmt\ s \\ & \text{and } Stmt\ s \neq \phi \\ & \text{and } (\forall t : t \in Stmt\ s \Rightarrow Q\ t)\quad)\end{aligned}$$

Refining the semantic

- How to express exception and abort → by modeling a state a pair (*s,flag*) where *s* is a record describing the values of our variables as before, and *flag* is {**N**,**E**,**A**} indicating whether the state is normal, exceptional, or aborted.
- Post-condition is now a triple ($Q_{\mathbf{N}}$,$Q_{\mathbf{E}}$,$Q_{\mathbf{A}}$)
- A specification {*P*} *Stmt* {$Q_{\mathbf{N}}$,$Q_{\mathbf{E}}$,$Q_{\mathbf{A}}$} now means:

$$
\begin{aligned}
(\forall s::\ P\ s\ \Rightarrow\ (\forall t,f : (t,f) \in Stmt\ (s,\mathbf{N})\ &\Rightarrow \\
&\text{if } f=\mathbf{N} \text{ then } Q_{\mathbf{N}}\ t \\
&\text{else if } f=\mathbf{E} \text{ then } Q_{\mathbf{E}}\ t \\
&\text{else } Q_{\mathbf{A}}\ t\ ))
\end{aligned}
$$