Hoare Logic

Hoare Logic is used to reason about the correctness of programs. In the end, it reduces a program and its specification to a set of verifications conditions.

Slides by Wishnu Prasetya
URL : www.cs.uu.nl/~wishnu
Course URL : www.cs.uu.nl/docs/vakken/pc

$s, \delta \xrightarrow{\text{EVALUATION SEMANTIC}} s'$

$\{P\}$ PROPERTY δ PROGRAM $\{Q\}$ PROPERTY

PARTIAL CORRECTNESS

Overview

- Hoare triples
- Basic statements // SEQ, IF, ASG
  - Composition rules for seq and if
  - Assignment
  - Weakest pre-condition
- Loops // WHILE
  - Invariants
  - Variants

Hoare triples

How do we prove our claims ?

- In Hoare logic we use inference rules.

- Usually of this form:

$$\frac{\text{premise-1} \quad , \quad \text{premise-2} \quad , \ldots}{\text{conclusion}}$$

- A proof is essentially just a series of invocations of inference rules, that produces our claim from known facts and assumptions.

Needed notions

- Inference rule:

PROGRAM

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

is this sound?

- What does a specification mean ?
- Programs
- Predicates
- States

We'll explain this in term of abstract models.

State

- In the sequel we will consider a program P with two variables: x:int , y:int.

- The state of P is determined by the value of x,y. Use record to denote a state:

  { x=0 , y=9 } // denote state where x=0 and y=9

- This notion of state is abstract! Actual state of P may consists of the value of CPU registers, stacks etc.

- Σ denotes the space of all possible states of P.

Expression

- An expression can be seen as a function $\Sigma \rightarrow$ val

$x + 1 \ \{ x=0 , y = 9 \}$	yields	1
$x + 1 \ \{ x=9 , y = 9 \}$	yields	10
etc		

- A (state) predicate is an expression that returns a boolean:

$x>0 \ \{ x=0 , y = 9 \}$	yields	false
$x>0 \ \{ x=9 , y = 9 \}$	yields	true
etc		

Viewing predicate as set

- So, a (state) predicate P is a function $\Sigma \rightarrow$ bool. It induces a set:

$$\chi_P = \{ s \mid s \models P \}$$

// the set of all states satisfying P

- P and its induced set are 'isomorphic' :

$$P(s) = s \in \chi_P$$

- Ehm … so for convenience lets just overload "P" to also denote χ_P. Which one is meant, depends on the context.

- Eg. when we say "P is an empty predicate".

Implication

- $P \Rightarrow Q$ // $P \Rightarrow Q$ is valid

  This means: $\forall s.\ s \models P \Rightarrow s \models Q$

  In terms of set this is equivalent to: $\chi_P \subseteq \chi_Q$

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

Non-termination

- What does this mean?

s Pr s' stands for $(Pr,s) \rightarrow s'$

$$\{s' \mid s \; Pr \; s'\} = \emptyset, \text{ for some state } s$$

- Can be used to model: "Pr does not terminate when executed on s".

- However, in discussion about models, we usually assume that our programs terminate.

- Expressing non-termination leads to some additional complications → not really where we want to focus now.

Hoare triples

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

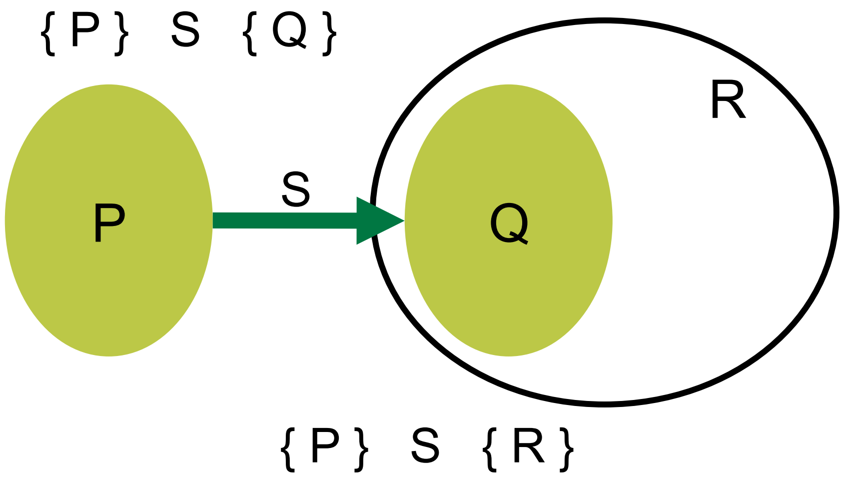
s Pr s' stands for (Pr,s)→ s'

$$\{\ P\ \}\ \ Pr\ \ \{\ Q\ \}\ \ = \ \ (\forall s.\ \ s \models P\ \Rightarrow\ (\forall s'.\ s\ Pr\ s'\ \Rightarrow s' \models Q))$$

- Since our model cannot express non-termination, we assume that Pr terminates.

- The interpretation of Hoare triple where termination is assumed is called "<u>partial correctness</u>" interpretation.
- Otherwise it is called <u>total correctness</u>.

Now we can explain …

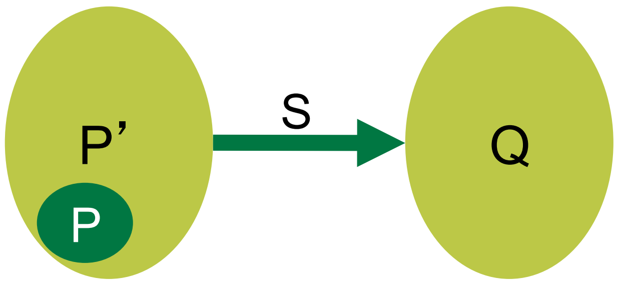


Post-condition weakening Rule:

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

And the dual

{ P' } S { Q }



{ P } S { Q }

Pre-condition strengthening Rule:

$$\frac{P \Rightarrow P' \quad , \{ P' \} \; S \; \{ Q \}}{\{ P \} \; S \; \{ Q \}}$$

Joining specifications

- Conjunction:

$$\frac{\{ P_1 \} \ S \ \{ Q_1 \} \quad , \quad \{ P_2 \} \ S \ \{ Q_2 \}}{\{ P_1 \wedge P_2 \} \ S \ \{ Q_1 \wedge Q_2 \}}$$

- Disjunction:

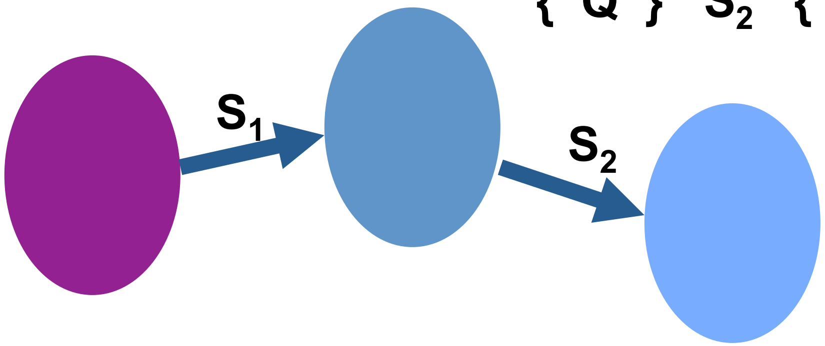
$$\frac{\{ P_1 \} \ S \ \{ Q_1 \} \quad , \quad \{ P_2 \} \ S \ \{ Q_2 \}}{\{ P_1 \vee P_2 \} \ S \ \{ Q_1 \vee Q_2 \}}$$

Reasoning about basic statements

Rule for SEQ composition

$\{ P \}\ S_1\ \{ Q \}$

$\{ Q \}\ S_2\ \{ R \}$



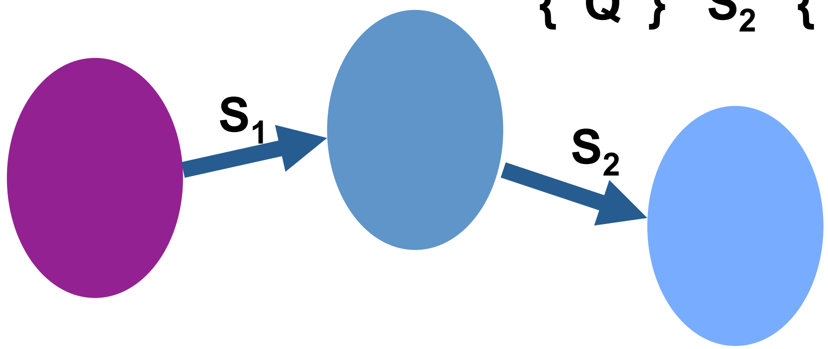
$\{ P \}\ S_1 ; S_2\ \{ R \}$

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

Rule for SEQ composition

$\{ P \}\ S_1\ \{ Q \}$

$\{ Q \}\ S_2\ \{ R \}$

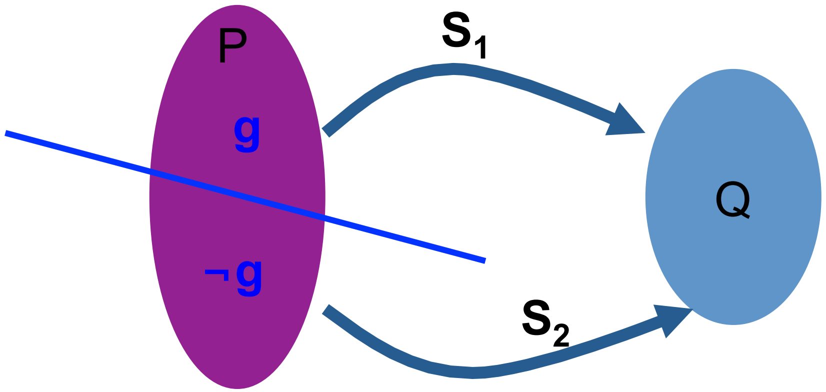


$\{ P \}\ S_1 ; S_2\ \{ R \}$

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

Rule for IF

$\{ P \wedge g \}\ S_1\ \{ Q \}$



$\{ P \}\ \underline{\text{if}}\ g\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \{ Q \}$

$\{ P \wedge \neg g \}\ S_2\ \{ Q \}$

$$\frac{\{ P \wedge g \}\ S_1\ \{ Q \}\ ,\ \{ P \wedge \neg g \}\ S_2\ \{ Q \}}{\{ P \}\ \underline{\text{if}}\ g\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \{ Q \}}$$

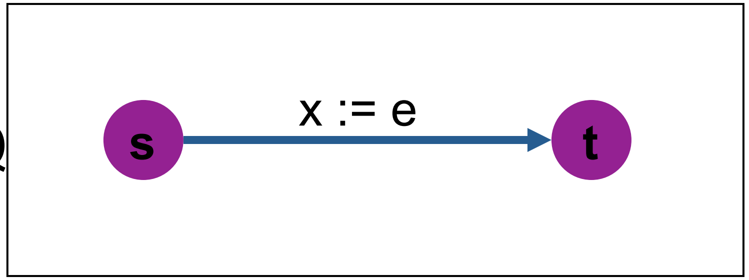
Rule for Assignment

- Let see ….
-

$$\frac{??}{\{\ P\ \}\ \ x:=e\ \ \{\ Q\ \}}$$

- Find a pre-condition W, such that, for any begin state s, and end state t:

$$s \models W \quad \Leftrightarrow \quad t \models Q$$



- Then we can equivalently prove $P \Rightarrow W$

Assignment, examples

- { 10 = y } x:=10 { x=y }

- { x+a = y } x:=x+a { x=y }

- So, W can be obtained by Q[e/x]

Assignment

- Theorem:

  Q holds after x:=e iff Q[e/x] holds before the assignment.

- Express this indirectly by:

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

- Corollary:

  { Q[e/x] } x:=e { Q } always valid.

How does a proof proceed now ?

- $\{ x \neq y \}$ tmp:= x ; x:=y ; y:=tmp $\{ x \neq y \}$

- Rule for SEQ requires you to come up with intermediate assertions:

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

- What to fill ??
  - Use the "Q[e/x]" suggested by the ASG theorem.
  - Work in <u>reverse</u> direction.
  - "Weakest pre-condition"

Weakest Pre-condition (wp)

- "wp" is a meta function:

  wp : Stmt X Pred → Pred

$wp(S, Q) = \{s$
$\forall s.(S, s \rightarrow s' / > s' \models Q)$

- wp(S,Q) gives the weakest (largest) pre-cond such that executing S in any state in any state in this pre-cond results in states in Q.
  - Partial correctness → termination assumed
  - Total correctness → termination demanded

Weakest pre-condition

- Let W = wp(S,Q)

s S s' stands for (S,s) → s'

- Two properties of W
  - Reachability: from any s|=W, if s S s' then s' |= Q
  - Maximality: s S s' and s' |= Q implies s|=W

Defining wp

- In terms of our abstract model:

$$wp(S,Q) \;=\; \{\; s \;\mid \text{forall } s' .\; s \; S \; s' \text{ implies } s' \models Q \;\}$$

- Abstract characterization:

$$\boxed{\{\; P \;\}\; S \;\{\; Q \;\} \;=\; P \Rightarrow wp(S,Q)}$$

- Nice, but this is not a constructive definition (does not tell us how to actually construct "W")

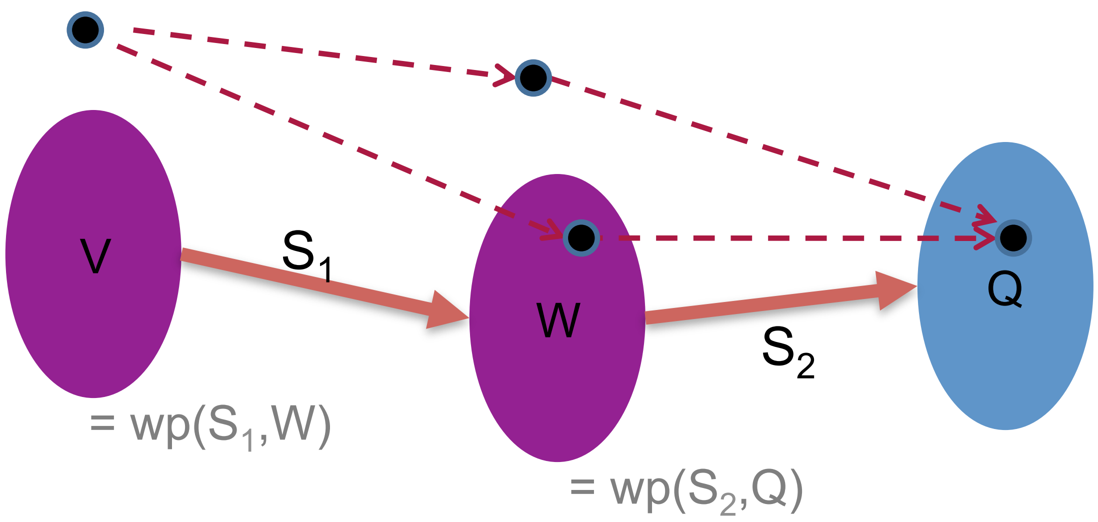
Some examples

- All these pre-conditions are the weakest:
- { y=10 }   x:=10   { y=x }
- { Q }   skip   { Q } wp (skip, Q) = Q
- { Q[e/x] }   x:=e   { Q } wp (x:=e , Q) = Q [x/e]

$$\boxed{\begin{array}{lcl} \text{wp skip } Q & = & Q \\ \text{wp } (x{:=}e)\ Q & = & Q[e/x] \end{array}}$$

wp of SEQ

$$wp\ ((S_1\ ;\ S_2)\ ,Q) \quad = \quad wp(S_1\ ,\ (wp(S_2,Q)))$$

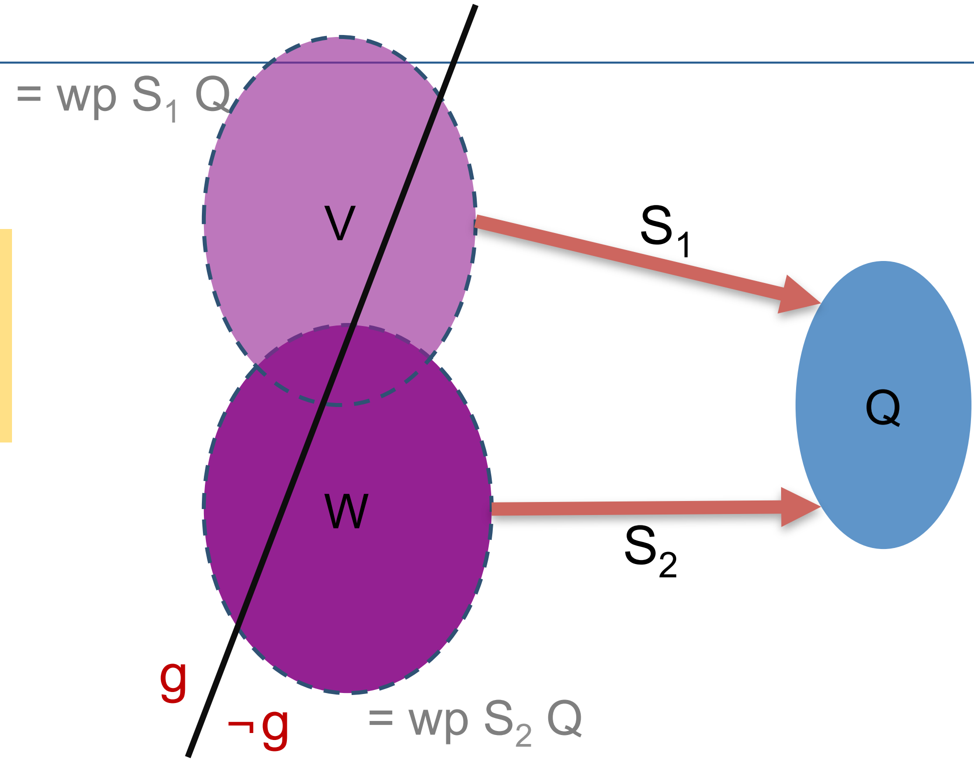


wp of IF

$$wp(\,(\underline{if}\;\; g\;\; \underline{then}\;\; S_1\; \underline{else}\; S_2),Q) \;=\; g \wedge wp(S_1,Q) \vee \neg g \wedge wp(S_2,Q)$$

Other formulation :

$(g \Rightarrow wp(S_1, Q))$
$\wedge$
$(\neg g \Rightarrow wp(S_2, Q))$



Proof: homework ☺

How does a proof proceed now ?

- $\{ x \neq y \}$ tmp:= x ; x:=y ; y:=tmp $\{ x \neq y \}$

1. Calculate:

   $$W = wp(\ (tmp := x\ ;\ x := y\ ;\ y := tmp)\ ,\ x \neq y\ )$$

2. Then prove: $x \neq y \Rightarrow W$

- We <u>calculate</u> the intermediate assertions, rather than figuring them out by hand!

$\{y \neq TEMP\}[TEMP/x] = \{y \neq x\}$

$TEMP := x;$

$\{x \neq TEMP\}[x/y] = \{y \neq TEMP\}$

$x := y;$

$\{x \neq y\}[y/TEMP] = \{x \neq TEMP\}$

$y := TEMP;$

$\{x \neq y\}$

THE WP TO OBTAIN $\{x \neq y\}$

IS $\{y \neq TEMP\}[TEMP/x] = \{y \neq x\}$

Proof via wp

- Wp calculation is *fully syntax driven*. *(But no while yet!)*
  - No human intelligence needed.
  - Can be automated.
- Works, as long as we can calculate "wp" → not always possible.
- Recall this abstract def:

$$\{\ P\ \}\ \ S\ \ \{\ Q\ \} \quad = \quad P \Rightarrow \underbrace{wp(S,Q)}_{W}$$

It follows: if $P \Rightarrow W$ <u>not valid</u>, then so does the original spec.

EX: WHAT IS THE WP TO EXECUTE THIS PROGRAM AND OBTAIN $x=100$?

```
x := 90 - y;
IF (x=0) THEN {
    IF (y>10) THEN
        x := y - x;
    ELSE x := 10 - x
}
x := x + y;
y := 10 + y;
```

90-50=0 FALSE

$\{(x=0 \wedge y=50) \vee (x\neq 0 \wedge x+y=100)\}[x/90-y] = \{(90-y=0 \wedge y=50) \vee (90-y\neq 0 \wedge 90=100)\}$

x := 90 - y;

$\{(x=0 \wedge y>10 \wedge 2y-x=100 \vee y\leq 10 \wedge y-x=90) \vee (x\neq 0 \wedge x+y=100)\}$

IF (x=0) THEN {

$\{y>10 \wedge 2y-x=100 \vee y\leq 10 \wedge y-x=90\}$

IF (y>10) THEN

$\{x+y=100\}[x/y-x] = \{2y-x=100\}$

x := y - x;

$\{x+y=100\}[x/10-x] = \{y-x=90\}$

ELSE x := 10 - x

$\{x+y=100\}$

}

$\{x=100\}[x/x+y] = \{x+y=100\}$

x := x + y;

$\{x=100\}[y/10+y] = \{x=100\}$

y := 10 + y;

$\{x=100\}$

$wp(\delta, x=100) = \text{FALSE}$

Example

```
bool find(a,n,x) {

    int i = 0 ;
    bool found = false ;

    while (¬found /\ i<n)  {
                                        found   =  (∃k : 0≤k<i :  a[k]=x)
        found :=  a[i]=x ;
        i++
                                        found   =  (∃k : 0≤k<i :  a[k]=x)

    }
                                        found   =  (∃k : 0≤k<n :  a[k]=x)
    return found ;
}
```

Example

$\boxed{0 \le i}$

$\{\ \underbrace{\neg found \ \land\ \ldots\ \land\ (found = (\exists k : 0 \le k < i : a[k]=x))}\ \}$

$\Downarrow$

found := a[i]=x ;

$\boxed{(a[i]=x) = (\exists k : 0 \le k < i+1 : a[k]=x)}$

i:=i+1

$\boxed{found = (\exists k : 0 \le k < i+1 : a[k]=x)}$

$\{\ found = (\exists k : 0 \le k < i : a[k]=x)\ \}$

wp (x:=e) Q = Q[e/x]

Reasoning about loops

How to prove this ?

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

- Calculate wp first ?
  - We don't have to
  - But wp has nice property → wp completely captures the statement:

$$\{ P \} \ S \ \{ Q \} \quad = \quad P \Rightarrow wp(S,Q)$$

wp of a loop ….

- Recall :
  - wp(S,Q) = { s | forall s' . s S s' implies s' |=Q }
  - { P } S { Q } = $P \Rightarrow$ wp(S,Q)
- But none of these definitions are actually useful to <u>construct</u> the weakest pre-condition.
- In the case of a loop, a constructive definition is not obvious. → pending.

How to prove this ?

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

- Plan-B: try to come up with an inference rule:

$$\frac{\begin{array}{l}\textit{condition about } g \\ \textit{condition about } S\end{array}}{\{\ P\ \}\ \underline{\text{while}}\ g\ \underline{\text{do}}\ S\ \{\ Q\ \}}$$

- The rule only need to be “sufficient”.

Idea

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

Still need to capture this.

- Try to come up with a predicate I that holds after each iteration :

$$
\begin{array}{llll}
iter_1 : & // g // ; S & \{ I \} & \\
iter_2 : & // g // ; S & \{ I \} & \\
\ldots & & & \\
iter_n : & // g // ; S & \{ I \} & \text{// last iteration!} \\
exit : & // \neg g // & &
\end{array}
$$

- $I \wedge \neg g$ holds as the loop exit!

So, to get postcond Q, sufficient to prove:

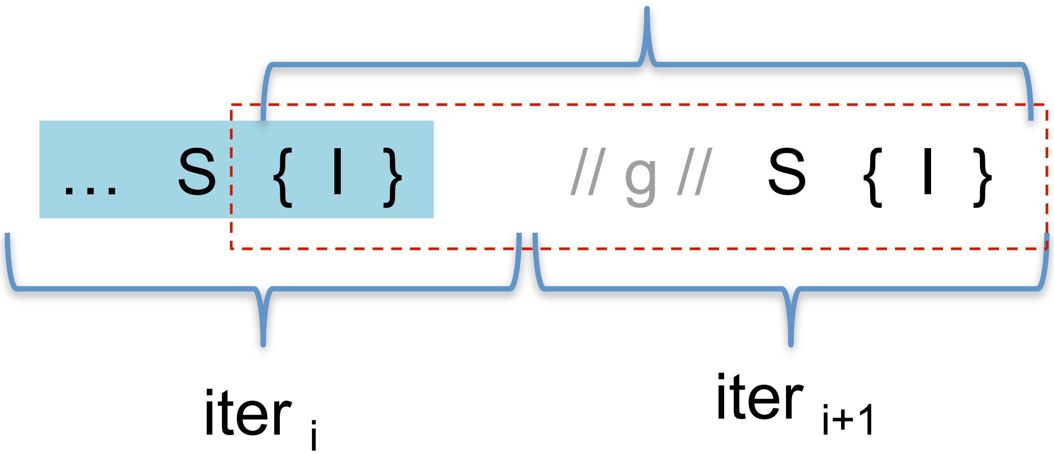
$$I \wedge \neg g \Rightarrow Q$$

Idea

- <u>while</u> g <u>do</u> S

- I is to holds after each iteration

Sufficient to prove: { I /\ g } S { I }



Except for the first iteration !

Idea

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

- For the first iteration :

Additionally we need : $P \Rightarrow I$

Recall the condition: $\{ I \land g \}\ S\ \{ I \}$

$$\{ P \} \quad \{ I \} \quad \underbrace{//\ g\ //\ \ S\ \ \{ I \}}_{Iter_1}$$

We know this from the given pre-cond

To Summarize

FIND AN INVARIANT I

- Capture this in an inference rule:

$$\dfrac{P \Rightarrow I \qquad \{\, g \wedge I \,\}\ S\ \{\, I \,\} \qquad I \wedge \neg g \Rightarrow Q}{\{\, P \,\}\ \underline{\text{while}}\ g\ \underline{\text{do}}\ S\ \{\, Q \,\}}$$

$P \Rightarrow I$ // setting up I

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

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

- This rule is only good for partial correctness though.
- I satisfying the second premise above is called <u>invariant</u>.

EX $\{x=1\}$ WHILE $(x<10)$ DO $x := x+1$ $\{x=10\}$ INVARIANT $x \le 10$? YES

GIVE ME I S.T. $\{P\}$ WHILE g DO δ $\{Q\}$

1. $P \supset I$
2. $\neg g \wedge I \supset Q$
3. $\{g \wedge I\}\ \delta\ \{I\}$

$I = x \le 10$!

1. $x=1 \supset x \le 10$? ✓
2. $x \ge 10 \wedge x \le 10 \supset x = 10$? ✓
3. $\{x < 10 \wedge x \le 10\}\ x := x+1\ \{x \le 10\}$

$\{x < 10 \wedge x \le 10\} \supset wp(x := x+1, \{x \le 10\})$?

$\{x \le 10\}[x/x+1] = \{x+1 \le 10\}$
$x := x+1$
$\{x \le 10\}$

$x < 10 \wedge x \le 10 \supset x+1 \le 10$ ✓

EX. $\{i=1\}$ WHILE $(i<64)$ DO $i := i \cdot 2$ $\{i=64\}$ INVARIANT $i \le 64$? NO!

1 $i=1 \supset i \le 64$ ✓
2. $i \ge 64 \wedge i \le 64 \supset i = 64$ ✓
3. $\{i < 64 \wedge i \le 64\}\ i := i \cdot 2\ \{i \le 64\}$

$P \supset I$
$\neg g \wedge I \supset Q$
$\{g \wedge I\}\ \delta\ \{I\}$

$wp(i := i \cdot 2, \{i \le 64\})$?

$\{i \le 64\}[i/i \cdot 2] = \{i \le 32\}$
$i := i \cdot 2$
$\{i \le 64\}$

$i < 64 \wedge i \le 64 \supset i \le 32$ ✗

EX. $\{i=1\}$ WHILE $(i<64)$ DO $i:=i\cdot 2$ $\{i=64\}$ INVARIANT $i=2^n \wedge i<128$? YES

$I = \exists n.\ i=2^n \wedge i<128$

1. $i=1 \supset \exists n.\ i=2^n \wedge i<128$ ✓
2. $i\geq 64 \wedge \exists n.\ i=2^n \wedge i<128 \supset i=64$ ✓
3. $\{i<64 \wedge \exists n.\ i=2^n \wedge i<128\}\ i:=i\cdot 2\ \{\exists n.\ i=2^n \wedge i<128\}$

$wp(i:=i\cdot 2, \{\exists n.\ i=2^n \wedge i<128\})$?

$\{\exists n.\ i=2^n \wedge i<128\}[i/i\cdot 2] = \{\exists n.\ i=2^{n-1} \wedge i<64\}$

$i:=i\cdot 2$

$\{\exists n.\ i=2^n \wedge i<128\}$

$i<64 \wedge \exists n.\ i=2^n \wedge i<128 \supset \exists n.\ i=2^{n-1} \wedge i<64$ ✓

Examples

- Prove:

  { i=0 } <u>while</u> i<n <u>do</u> i++ { i=n }

- Prove:

  { i=0 /\ s=0 }

  while i<n do { s = s +a[i] ; i++ }

  { s = SUM(a[0..n)) }

Note

- Recall :

  wp ((<u>while</u> g <u>do</u> S),Q) =
  { s | forall s'. s (<u>while</u> g <u>do</u> S) s' implies s' |= Q }

- Theoretically, we can still construct this set if the state space is <u>finite</u>. The construction is exactly as the def. above says.
- You need a way to tell when the loop does not terminate:
  - Maintain a history H of states after each iteration.
  - Non-termination if the state t after i-th iteration is in H from the previous iteration.
- Though then you can just as well 'execute' the program to verify it (testing), for which you don't need Hoare logic.

Tackling while termination: invariant and variant

To prove

$\{P\}$ **while** B **do** S **end** $\{Q\}$

find invariant J and well-founded variant function vf such that:

- invariant holds initially: $P \Rightarrow J$
- invariant is maintained: $\{J \wedge B\}\ S\ \{J\}$
- invariant is sufficient: $J \wedge \neg B \Rightarrow Q$
- variant function is bounded:

  $J \wedge B \Rightarrow 0 \leq vf$
- variant function decreases:

  $\{J \wedge B \wedge vf=VF\}\ S\ \{vf<VF\}$

Proving termination

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

- Idea: come up with an integer expression m, satisfying :

    1. At the start of every iteration $m \geq 0$

    2. Each iteration decreases m

- These imply that the loop will terminates.

Capturing the termination conditions

- At the start of every iteration $m \geq 0$:
  - $g \Rightarrow m \geq 0$
  - If you have an invariant: $I \wedge g \Rightarrow m \geq 0$
- Each iteration decreases m :

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

To Summarize

- $P \Rightarrow I$ // 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
  $I \land g \Rightarrow m \geq 0$ // m bounded below

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

- Since we also have this pre-cond strengthening rule:

  $P \Rightarrow I\ \ ,\ \{\ I\ \}$ while g do S $\{\ Q\ \}$

  $\{\ P\ \}$ while g do S $\{\ Q\ \}$

Lec notes often refer to this rule

-

$$\{\ g \wedge I\ \}\quad S\quad \{\ I\ \} \qquad \text{// invariance}$$
$$I \wedge \neg g \Rightarrow Q \qquad \text{// exit cond}$$
$$\{\ I \wedge g\ \}\quad C{:=}m;\ S\quad \{\ m<C\ \} \qquad \text{// m decreasing}$$
$$I \wedge g \Rightarrow m \geq 0 \qquad \text{// m bounded below}$$
$$\overline{\{\ I\ \}\ \underline{\text{while}}\ g\ \underline{\text{do}}\ S\ \{\ Q\ \}}$$