COMP2111 Week 6
Term 1, 2019
Hoare Logic IV

Summary

- Weakest precondition reasoning
- Handling termination
- Operational semantics
- Adding non-determinism
- Refinement calculus

Summary

- Weakest precondition reasoning
- Handling termination
- Operational semantics
- Adding non-determinism
- Refinement calculus

Finding a proof

Consider the following code:

Pow

$r := 1;$
$i := 0;$
while $i < m$ **do**
    $r := r * n;$
    $i := i + 1$
od

We would like to show $\{\varphi\}\,\textrm{Pow}\,\{r = n^m\}$.

- What should φ be? $m \geq 0 \wedge n > 0$
- What should the intermediate assertions be?

Finding a proof

Consider the following code:

```
Pow
r := 1;
i := 0;
while i < m do
    r := r * n;
    i := i + 1
od
```

We would like to show $\{\varphi\}\,\textsc{Pow}\,\{r = n^m\}$.

- What should φ be? $m \geq 0 \wedge n > 0$
- What should the intermediate assertions be?

Finding a proof

Consider the following code:

```
Pow
r := 1;
i := 0;
while i < m do
   r := r * n;
   i := i + 1
od
```

We would like to show $\{\varphi\}\,\text{Pow}\,\{r = n^m\}$.

- What should φ be? $m \geq 0 \wedge n > 0$
- What should the intermediate assertions be?

Determining a precondition

Here are some valid Hoare triples:

- $\{(x = 5) \land (y = 10)\}\, z := x/y\, \{z < 1\}$
- $\{(x < y) \land (y > 0)\}\, z := x/y\, \{z < 1\}$
- $\{(y \neq 0) \land (x/y < 1)\}\, z := x/y\, \{z < 1\}$

All are valid, but the third one is the most useful:

- it has the weakest precondition of the three
- it can be applied in the most scenarios (e.g. $x = 2 \land y = -1$)

Weakest precondition

Given a program P and a postcondition ψ the **weakest precondition of** P **with respect to** ψ, $wp(P, \psi)$, is a predicate φ such that

$$\{\varphi\}\, P\, \{\psi\} \quad \text{and} \quad \text{If } \{\varphi'\}\, P\, \{\psi\} \text{ then } \varphi' \to \varphi$$

We can compute wp based on the structure of P...

Weakest precondition

Given a program P and a postcondition ψ the **weakest precondition of** P **with respect to** ψ, $wp(P, \psi)$, is a predicate φ such that

$$\{\varphi\}\, P\, \{\psi\} \quad \text{and} \quad \text{If } \{\varphi'\}\, P\, \{\psi\} \text{ then } \varphi' \to \varphi$$

We can compute wp based on the structure of P...

Determining *wp*: Assignment

$$wp(x := e, \psi) = \psi[e/x]$$

Example

$$\{2 + y > 0\}\ x := 2\ \{x + y > 0\}$$

Determining *wp*: Assignment

$$wp(x := e, \psi) = \psi[e/x]$$

Example

$$\{2 + y > 0\}\, x := 2\, \{x + y > 0\}$$

Determining *wp*: Sequence

$$wp(P; S, \psi) = wp(P, wp(S, \psi))$$

Example

Let φ be the weakest precondition of:

$$\{\varphi\}\, x := x + 1; \;\; y := x + y \,\{y > 4\}$$

What should φ be? $x + y > 3$

- $wp(y := x + y, y > 4) \;=\; (x + y > 4)$
- $wp(x := x + 1, x + y > 4) \;=\; (x + 1 + y > 4) \;\equiv\; x + y > 3$

Determining *wp*: Sequence

$$wp(P; S, \psi) = wp(P, wp(S, \psi))$$

Example

Let φ be the weakest precondition of:

$$\{\varphi\}\, x := x + 1;\;\; y := x + y\, \{y > 4\}$$

What should φ be? $x + y > 3$

- $wp(y := x + y, y > 4) \;=\; (x + y > 4)$
- $wp(x := x + 1, x + y > 4) \;=\; (x + 1 + y > 4) \;\equiv\; x + y > 3$

Determining *wp*: Sequence

$$wp(P; S, \psi) = wp(P, wp(S, \psi))$$

Example

Let φ be the weakest precondition of:

$$\{\varphi\}\, x := x + 1;\;\; y := x + y\, \{y > 4\}$$

What should φ be? $x + y > 3$

- $wp(y := x + y, y > 4) \;=\; (x + y > 4)$
- $wp(x := x + 1, x + y > 4) \;=\; (x + 1 + y > 4) \;\equiv\; x + y > 3$

Determining wp: Sequence

$$wp(P; S, \psi) = wp(P, wp(S, \psi))$$

Example

Let φ be the weakest precondition of:

$$\{\varphi\}\, x := x + 1;\;\; y := x + y\, \{y > 4\}$$

What should φ be? $x + y > 3$

- $wp(y := x + y, y > 4) \;=\; (x + y > 4)$
- $wp(x := x + 1, x + y > 4) \;=\; (x + 1 + y > 4) \;\equiv\; x + y > 3$

Determining *wp*: Conditional

$$
\begin{aligned}
& wp(\textbf{if } b \textbf{ then } P \textbf{ else } Q \textbf{ fi}, \psi) \\
& \quad = (b \rightarrow wp(P, \psi)) \wedge (\neg b \rightarrow wp(Q, \psi)) \\
& \quad \equiv (b \wedge wp(P, \psi)) \vee (\neg b \wedge wp(Q, \psi))
\end{aligned}
$$

Example

$$
\begin{aligned}
& wp(\textbf{if } x > 0 \textbf{ then } z := y \textbf{ else } z := 0 - y \textbf{ fi},\ z > 5) \\
& \quad = ((x > 0) \rightarrow wp(z := y, z > 5)) \\
& \qquad \wedge\ ((x \leq 0) \rightarrow wp(z := 0 - y, z > 5)) \\
& \quad = ((x > 0) \rightarrow (y > 5))\ \wedge\ ((x \leq 0) \rightarrow (y < -5))
\end{aligned}
$$

Determining *wp*: Conditional

$$
\begin{aligned}
& wp(\textbf{if } b \textbf{ then } P \textbf{ else } Q \textbf{ fi}, \psi) \\
& \quad = (b \to wp(P,\psi)) \wedge (\neg b \to wp(Q,\psi)) \\
& \quad \equiv (b \wedge wp(P,\psi)) \vee (\neg b \wedge wp(Q,\psi))
\end{aligned}
$$

Example

$$
\begin{aligned}
& wp(\textbf{if } x > 0 \textbf{ then } z := y \textbf{ else } z := 0 - y \textbf{ fi},\ z > 5) \\
& \quad = ((x > 0) \to wp(z := y, z > 5)) \\
& \qquad \wedge\ ((x \leq 0) \to wp(z := 0 - y, z > 5)) \\
& \quad = ((x > 0) \to (y > 5))\ \wedge\ ((x \leq 0) \to (y < -5))
\end{aligned}
$$

Determining *wp*: Conditional

$$
\begin{aligned}
& wp(\textbf{if } b \textbf{ then } P \textbf{ else } Q \textbf{ fi}, \psi) \\
& \quad = (b \to wp(P, \psi)) \wedge (\neg b \to wp(Q, \psi)) \\
& \quad \equiv (b \wedge wp(P, \psi)) \vee (\neg b \wedge wp(Q, \psi))
\end{aligned}
$$

Example

$$
\begin{aligned}
& wp(\textbf{if } x > 0 \textbf{ then } z := y \textbf{ else } z := 0 - y \textbf{ fi},\ z > 5) \\
& \quad = ((x > 0) \to wp(z := y, z > 5)) \\
& \qquad \wedge\ ((x \leq 0) \to wp(z := 0 - y, z > 5)) \\
& \quad = ((x > 0) \to (y > 5))\ \wedge\ ((x \leq 0) \to (y < -5))
\end{aligned}
$$

Determining *wp*: Conditional

$$
\begin{aligned}
& wp(\textbf{if } b \textbf{ then } P \textbf{ else } Q \textbf{ fi}, \psi) \\
& \qquad = (b \rightarrow wp(P, \psi)) \wedge (\neg b \rightarrow wp(Q, \psi)) \\
& \qquad \equiv (b \wedge wp(P, \psi)) \vee (\neg b \wedge wp(Q, \psi))
\end{aligned}
$$

Example

$$
\begin{aligned}
& wp(\textbf{if } x > 0 \textbf{ then } z := y \textbf{ else } z := 0 - y \textbf{ fi},\ z > 5) \\
& \quad = ((x > 0) \rightarrow wp(z := y, z > 5)) \\
& \qquad\qquad \wedge\ ((x \leq 0) \rightarrow wp(z := 0 - y, z > 5)) \\
& \quad = ((x > 0) \rightarrow (y > 5))\ \wedge\ ((x \leq 0) \rightarrow (y < -5))
\end{aligned}
$$

Determining *wp*: Conditional

$$
\begin{aligned}
&wp(\textbf{if } b \textbf{ then } P \textbf{ else } Q \textbf{ fi}, \psi) \\
&\quad = (b \rightarrow wp(P, \psi)) \wedge (\neg b \rightarrow wp(Q, \psi)) \\
&\quad \equiv (b \wedge wp(P, \psi)) \vee (\neg b \wedge wp(Q, \psi))
\end{aligned}
$$

Example

$$
\begin{aligned}
&wp(\textbf{if } x > 0 \textbf{ then } z := y \textbf{ else } z := 0 - y \textbf{ fi},\ z > 5) \\
&= ((x > 0) \rightarrow wp(z := y, z > 5)) \\
&\qquad \wedge\ ((x \leq 0) \rightarrow wp(z := 0 - y, z > 5)) \\
&= ((x > 0) \rightarrow (y > 5))\ \wedge\ ((x \leq 0) \rightarrow (y < -5))
\end{aligned}
$$

Determining *wp*: Loops

$$wp(\textbf{while } b \textbf{ do } P \textbf{ od}, \psi) = ?$$

Loops are problematic:

- *wp* calculates a triple for a *single* program statement block.
- Loops consist of a block executed *repeatedly*
- Weakest precondition for 1 loop may be different from weakest precondition for 100 loops...

Handling loops

$$\{\varphi\}\ \textbf{while}\ b\ \textbf{do}\ P\ \textbf{od}\ \{\psi\}$$

Instead: Find a **loop invariant** I such that

- $\varphi \to I$ (establish)
- $\{I \wedge b\}\ P\ \{I\}$ (maintain)
- $I \wedge \neg b \to \psi$ (conclude)

NB

Finding (good) loop invariants is generally hard!
$\Rightarrow$ *Active area of research*

Handling loops

$$\{\varphi\}\ \textbf{while}\ b\ \textbf{do}\ P\ \textbf{od}\ \{\psi\}$$

Instead: Find a **loop invariant** I such that

- $\varphi \to I$ (establish)
- $\{I \wedge b\}\ P\ \{I\}$ (maintain)
- $I \wedge \neg b \to \psi$ (conclude)

NB

Finding (good) loop invariants is generally hard!
$\Rightarrow$ *Active area of research*

Handling loops

$$\{\varphi\}\ \textbf{while}\ b\ \textbf{do}\ P\ \textbf{od}\ \{\psi\}$$

Instead: Find a **loop invariant** I such that

- $\varphi \to I$ (establish)
- $\{I \wedge b\}\ P\ \{I\}$ (maintain)
- $I \wedge \neg b \to \psi$ (conclude)

NB

Finding (good) loop invariants is generally hard!
$\Rightarrow$ *Active area of research*

Handling loops

$$\{\varphi\}\ \textbf{while}\ b\ \textbf{do}\ P\ \textbf{od}\ \{\psi\}$$

Instead: Find a **loop invariant** I such that

- $\varphi \to I$ (establish)
- $\{I \wedge b\}\ P\ \{I\}$ (maintain)
- $I \wedge \neg b \to \psi$ (conclude)

NB

Finding (good) loop invariants is generally hard!
$\Rightarrow$ *Active area of research*

Handling loops

$$\{\varphi\}\ \textbf{while}\ b\ \textbf{do}\ P\ \textbf{od}\ \{\psi\}$$

Instead: Find a **loop invariant** I such that

- $\varphi \rightarrow I$ (establish)
- $\{I \wedge b\}\ P\ \{I\}$ (maintain)
- $I \wedge \neg b \rightarrow \psi$ (conclude)

NB

Finding (good) loop invariants is generally hard!
$\Rightarrow$ *Active area of research*

Back to the example

Pow

$\{\text{init}: (m \geq 0) \wedge (n > 0)\}$

$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$r := 1;$ $\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$i := 0;$

$\{\text{Inv}\}$

while $i < m$ **do** $\{\text{Inv} \wedge (i < m)\}$

$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$

$r := r * n;$ $\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$

$i := i + 1$ $\{\text{Inv}\}$

od $\{\text{Inv} \wedge (i \geq m)\}$

$\{r = n^m\}$

What would be a good invariant?

Inv: $r = n^i \ \wedge \ i \leq m \ \wedge \ \text{init}$

Back to the example

POW

$\{\text{init: } (m \geq 0) \land (n > 0)\}$

$\{(1 = n^0) \land (0 \leq m) \land \text{init}\}$

$r := 1;$

$\{(r = n^0) \land (0 \leq m) \land \text{init}\}$

$i := 0;$

$\{\text{Inv}\}$

while $i < m$ **do**

$\{\text{Inv} \land (i < m)\}$

$\{(r * n = n^{i+1}) \land (i + 1 \leq m) \land \text{init}\}$

$r := r * n;$

$\{(r = n^{i+1}) \land (i + 1 \leq m) \land \text{init}\}$

$i := i + 1$

$\{\text{Inv}\}$

od

$\{\text{Inv} \land (i \geq m)\}$

$\{r = n^m\}$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \land i \leq m \land \text{init}$$

Back to the example

Pow

$\{\text{init: } (m \geq 0) \wedge (n > 0)\}$

$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$r := 1;$ $\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$i := 0;$

$\{\text{Inv}\}$

while $i < m$ **do** $\{\text{Inv} \wedge (i < m)\}$

$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$

$\quad r := r * n;$ $\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$

$\quad i := i + 1$ $\{\text{Inv}\}$

od $\{\text{Inv} \wedge (i \geq m)\}$

$\{r = n^m\}$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \ \wedge \ i \leq m \ \wedge \ \text{init}$$

Back to the example

Pow

$\{\text{init: } (m \geq 0) \wedge (n > 0)\}$

$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$r := 1;$ $\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$i := 0;$

$\{\text{Inv}\}$

while $i < m$ **do** $\{\text{Inv} \wedge (i < m)\}$

$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$

  $r := r * n;$ $\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$

  $i := i + 1$ $\{\text{Inv}\}$

od $\{\text{Inv} \wedge (i \geq m)\}$

$\{r = n^m\}$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \ \wedge \ i \leq m \ \wedge \ \text{init}$$

Back to the example

Pow

$$\{\text{init: } (m \geq 0) \land (n > 0)\}$$
$$\{(1 = n^0) \land (0 \leq m) \land \text{init}\}$$
$$\{(r = n^0) \land (0 \leq m) \land \text{init}\}$$

$r := 1;$
$i := 0;$

$$\{\text{Inv}\}$$

while $i < m$ **do**

$$\{\text{Inv} \land (i < m)\}$$
$$\{(r * n = n^{i+1}) \land (i + 1 \leq m) \land \text{init}\}$$
$$\{(r = n^{i+1}) \land (i + 1 \leq m) \land \text{init}\}$$

  $r := r * n;$
  $i := i + 1$

$$\{\text{Inv}\}$$

od

$$\{\text{Inv} \land (i \geq m)\}$$
$$\{r = n^m\}$$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \ \land \ i \leq m \ \land \ \text{init}$$

Back to the example

POW

$\{\text{init}: (m \geq 0) \wedge (n > 0)\}$

$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$r := 1;$ $\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$i := 0;$

$\{\text{Inv}\}$

while $i < m$ **do** $\{\text{Inv} \wedge (i < m)\}$

$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$

$r := r * n;$ $\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$

$i := i + 1$ $\{\text{Inv}\}$

od $\{\text{Inv} \wedge (i \geq m)\}$

$\{r = n^m\}$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \ \wedge \ i \leq m \ \wedge \ \text{init}$$

Back to the example

POW	
	$\{\text{init: } (m \geq 0) \wedge (n > 0)\}$
	$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$r := 1;$	$\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$i := 0;$	
	$\{\text{Inv}\}$
while $i < m$ **do**	$\{\text{Inv} \wedge (i < m)\}$
	$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$\quad r := r * n;$	$\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$\quad i := i + 1$	$\{\text{Inv}\}$
od	$\{\text{Inv} \wedge (i \geq m)\}$
	$\{r = n^m\}$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \ \wedge \ i \leq m \ \wedge \ \text{init}$$

Back to the example

Pow

	$\{\text{init}: (m \geq 0) \wedge (n > 0)\}$
	$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$r := 1;$	$\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$i := 0;$	
	$\{\text{Inv}\}$
while $i < m$ **do**	$\{\text{Inv} \wedge (i < m)\}$
	$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$\quad r := r * n;$	$\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$\quad i := i + 1$	$\{\text{Inv}\}$
od	$\{\text{Inv} \wedge (i \geq m)\}$
	$\{r = n^m\}$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \ \wedge \ i \leq m \ \wedge \ \text{init}$$

Back to the example

Pow	
	$\{\text{init: } (m \geq 0) \wedge (n > 0)\}$
	$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$r := 1;$	$\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$i := 0;$	
	$\{\text{Inv}\}$
while $i < m$ **do**	$\{\text{Inv} \wedge (i < m)\}$
	$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$\quad r := r * n;$	$\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$\quad i := i + 1$	$\{\text{Inv}\}$
od	$\{\text{Inv} \wedge (i \geq m)\}$
	$\{r = n^m\}$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \ \wedge \ i \leq m \ \wedge \ \text{init}$$

Back to the example

Pow	
	$\{\text{init}: (m \geq 0) \wedge (n > 0)\}$
	$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$r := 1;$	$\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$i := 0;$	
	$\{\text{Inv}\}$
while $i < m$ **do**	$\{\text{Inv} \wedge (i < m)\}$
	$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$r := r * n;$	$\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$i := i + 1$	$\{\text{Inv}\}$
od	$\{\text{Inv} \wedge (i \geq m)\}$
	$\{r = n^m\}$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \ \wedge \ i \leq m \ \wedge \ \text{init}$$

Back to the example

Pow	
	$\{\text{init}: (m \geq 0) \wedge (n > 0)\}$
	$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$r := 1;$	$\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$i := 0;$	
	$\{\text{Inv}\}$
while $i < m$ **do**	$\{\text{Inv} \wedge (i < m)\}$
	$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$r := r * n;$	$\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}\}$
$i := i + 1$	$\{\text{Inv}\}$
od	$\{\text{Inv} \wedge (i \geq m)\}$
	$\{r = n^m\}$

What would be a good invariant?

$$\text{Inv:} \quad r = n^i \ \wedge \ i \leq m \ \wedge \ \text{init}$$

Proof obligations

$$\begin{aligned} \text{init:} \quad & (m \geq 0) \land (n > 0) \\ \text{Inv:} \quad & (r = n^i) \land (i \leq m) \land \text{init} \end{aligned}$$

- $\text{init} \quad \rightarrow \quad (1 = n^0) \land (0 \leq m) \land \text{init}$
- $\text{Inv} \land (i < m) \quad \rightarrow \quad (r * n = n^{i+1}) \land (i + 1 \leq m) \land \text{init}$
- $\text{Inv} \land (i \geq m) \quad \rightarrow \quad r = n^m$

Proof obligations

$$\begin{aligned} \text{init:} \quad & (m \geq 0) \wedge (n > 0) \\ \text{Inv:} \quad & (r = n^i) \wedge (i \leq m) \wedge \text{init} \end{aligned}$$

- $\text{init} \quad \rightarrow \quad (1 = n^0) \wedge (0 \leq m) \wedge \text{init}$
- $\text{Inv} \wedge (i < m) \quad \rightarrow \quad (r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}$
- $\text{Inv} \wedge (i \geq m) \quad \rightarrow \quad r = n^m$

Proof obligations

$$\begin{array}{rl} \text{init:} & (m \geq 0) \wedge (n > 0) \\ \text{Inv:} & (r = n^i) \wedge (i \leq m) \wedge \text{init} \end{array}$$

- $\text{init} \quad \rightarrow \quad (1 = n^0) \wedge (0 \leq m) \wedge \text{init}$
- $\text{Inv} \wedge (i < m) \quad \rightarrow \quad (r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init}$
- $\text{Inv} \wedge (i \geq m) \quad \rightarrow \quad r = n^m$

Summary

- Weakest precondition reasoning
- Handling termination
- Operational semantics
- Adding non-determinism
- Refinement calculus

Termination

Hoare triples for partial correctness:

$$\{\varphi\}\, P\, \{\psi\}$$

Asserts ψ holds *if* P terminates.

What if we wanted to make the stronger statement ψ holds *and* P terminates?

Hoare triples for total correctness:

$$[\varphi]\, P\, [\psi]$$

Asserts:
If φ holds at a starting state, and P is executed;
then P will terminate and ψ will hold in the resulting state.

Termination

Hoare triples for partial correctness:

$$\{\varphi\}\, P\, \{\psi\}$$

Asserts ψ holds *if* P terminates.

What if we wanted to make the stronger statement ψ holds *and* P terminates?

Hoare triples for total correctness:

$$[\varphi]\, P\, [\psi]$$

Asserts:
If φ holds at a starting state, and P is executed;
then P will terminate and ψ will hold in the resulting state.

Warning

Termination is hard!

- Algorithmic limitations (e.g. Halting problem)
- Mathematical limitations

Example

```
COLLATZ
while n > 1 do
    if n%2 = 0
    then
        n := n/2
    else
        n := 3 * n + 1
    fi
od
```

Warning

Termination is hard!

- Algorithmic limitations (e.g. Halting problem)
- Mathematical limitations

Example

COLLATZ

while $n > 1$ **do**
  if $n \% 2 = 0$
  then
    $n := n/2$
  else
    $n := 3 * n + 1$
  fi
od

Total correctness

How can we show:

$$[(m \geq 0) \wedge (n > 0)]\ \mathrm{Pow}\ [r = n^m]?$$

Use **Hoare Logic for total correctness**:

- (ass), (seq), (cond), and (cons) rules all the same
- Modified (loop) rule

Total correctness

How can we show:

$$[(m \geq 0) \wedge (n > 0)]\, \text{Pow}\, [r = n^m]?$$

Use **Hoare Logic for total correctness**:

- (ass), (seq), (cond), and (cons) rules all the same
- Modified (loop) rule

Rules for total correctness

$$\frac{}{[\varphi[e/x]]\; x := e\; [\varphi]} \quad \text{(ass)}$$

$$\frac{[\varphi]\; P\; [\psi] \qquad [\psi]\; Q\; [\rho]}{[\varphi]\; P; Q\; [\rho]} \quad \text{(seq)}$$

$$\frac{[\varphi \wedge g]\; P\; [\psi] \qquad [\varphi \wedge \neg g]\; Q\; [\psi]}{[\varphi]\; \textbf{if}\; g\; \textbf{then}\; P\; \textbf{else}\; Q\; \textbf{fi}\; [\psi]} \quad \text{(if)}$$

$$\frac{\varphi' \rightarrow \varphi \qquad [\varphi]\; P\; [\psi] \qquad \psi \rightarrow \psi'}{[\varphi']\; P\; [\psi']} \quad \text{(cons)}$$

Terminating while loops

$$\{\varphi\}\ \textbf{while}\ b\ \textbf{do}\ P\ \textbf{od}\ \{\psi\}$$

Partial correctness:
Find an invariant I such that:

- $\varphi \to I$ (establish)
- $\{I \wedge b\}\ P\ \{I\}$ (maintain)
- $(I \wedge \neg b) \to \psi$ (conclude)

Show termination:
Find a **variant** v such that:

- $(I \wedge b) \to v > 0$ (positivity)
- $[I \wedge b \wedge v = N]\ P\ [v < N]$ (progress)

Terminating while loops

$$[\varphi]\ \textbf{while}\ b\ \textbf{do}\ P\ \textbf{od}\ [\psi]$$

Partial correctness:
Find an invariant I such that:

- $\varphi \to I$ (establish)
- $[I \wedge b]\ P\ [I]$ (maintain)
- $(I \wedge \neg b) \to \psi$ (conclude)

Show termination:
Find a **variant** v such that:

- $(I \wedge b) \to v > 0$ (positivity)
- $[I \wedge b \wedge v = N]\ P\ [v < N]$ (progress)

Loop rule for total correctness

$$\frac{[\varphi \wedge g \wedge (v = N)]\ P\ [\varphi \wedge (v < N)] \qquad (\varphi \wedge g) \rightarrow (v > 0)}{[\varphi]\ \textbf{while}\ g\ \textbf{do}\ P\ \textbf{od}\ [\varphi \wedge \neg g]} \quad \text{(loop)}$$

Termination for Pow

Pow

$$\{\text{init}: (m \geq 0) \wedge (n > 0)\}$$
$$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$$

$r := 1;$ $\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$i := 0;$

$$\{\text{Inv}\}$$

while $i < m$ **do** $\{\text{Inv} \wedge (i < m) \wedge (v = N)\}$

$$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$$

$\quad r := r * n;$ $\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$

$\quad i := i + 1$ $\{\text{Inv} \wedge (v < N)\}$

od $\{\text{Inv} \wedge (i \geq m)\}$

$$\{r = n^m\}$$

What is a suitable variant? $v := (m - i)$

Termination for Pow

Pow	
	$\{\text{init}: (m \geq 0) \wedge (n > 0)\}$
	$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$r := 1;$	$\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$i := 0;$	
	$\{\text{Inv}\}$
while $i < m$ **do**	$\{\text{Inv} \wedge (i < m) \wedge (v = N)\}$
	$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$
$r := r * n;$	$\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$
$i := i + 1$	$\{\text{Inv} \wedge (v < N)\}$
od	$\{\text{Inv} \wedge (i \geq m)\}$
	$\{r = n^m\}$

What is a suitable variant? $v := (m - i)$

Termination for Pow

Pow	
	$\{\text{init: } (m \geq 0) \wedge (n > 0)\}$
	$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$r := 1;$	$\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$i := 0;$	
	$\{\text{Inv}\}$
while $i < m$ **do**	$\{\text{Inv} \wedge (i < m) \wedge (v = N)\}$
	$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$
$\quad r := r * n;$	$\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$
$\quad i := i + 1$	$\{\text{Inv} \wedge (v < N)\}$
od	$\{\text{Inv} \wedge (i \geq m)\}$
	$\{r = n^m\}$

What is a suitable variant? $v := (m - i)$

Termination for Pow

Pow	
	$\{\text{init}: (m \geq 0) \wedge (n > 0)\}$
	$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$r := 1;$	$\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$i := 0;$	
	$\{\text{Inv}\}$
while $i < m$ **do**	$\{\text{Inv} \wedge (i < m) \wedge (v = N)\}$
	$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$
$\quad r := r * n;$	$\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$
$\quad i := i + 1$	$\{\text{Inv} \wedge (v < N)\}$
od	$\{\text{Inv} \wedge (i \geq m)\}$
	$\{r = n^m\}$

What is a suitable variant? $v := (m - i)$

Termination for Pow

Pow	
	$\{\text{init: } (m \geq 0) \wedge (n > 0)\}$
	$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$r := 1;$	$\{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$
$i := 0;$	
	$\{\text{Inv}\}$
while $i < m$ **do**	$\{\text{Inv} \wedge (i < m) \wedge (v = N)\}$
	$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$
$\quad r := r * n;$	$\{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$
$\quad i := i + 1$	$\{\text{Inv} \wedge (v < N)\}$
od	$\{\text{Inv} \wedge (i \geq m)\}$
	$\{r = n^m\}$

What is a suitable variant? $v := (m - i)$

Termination for Pow

Pow

$$\{\text{init: } (m \geq 0) \wedge (n > 0)\}$$
$$\{(1 = n^0) \wedge (0 \leq m) \wedge \text{init}\}$$

$r := 1;$ $\quad \{(r = n^0) \wedge (0 \leq m) \wedge \text{init}\}$

$i := 0;$

$$\{\text{Inv}\}$$

while $i < m$ **do** $\quad \{\text{Inv} \wedge (i < m) \wedge (v = N)\}$

$$\{(r * n = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$$

$\quad r := r * n;$ $\quad \{(r = n^{i+1}) \wedge (i + 1 \leq m) \wedge \text{init} \wedge (v = N)\}$

$\quad i := i + 1$ $\quad \{\text{Inv} \wedge (v < N)\}$

od $\quad \{\text{Inv} \wedge (i \geq m)\}$

$$\{r = n^m\}$$

What is a suitable variant? $v := (m - i)$

Additional proof obligations

$$
\begin{aligned}
\text{init:} \quad & (m \geq 0) \wedge (n > 0) \\
\text{Inv:} \quad & (r = n^i) \wedge (i \leq m) \wedge \text{init} \\
v: \quad & m - i
\end{aligned}
$$

- $\text{Inv} \wedge (i < m) \quad \rightarrow \quad (v > 0)$
- $[v = N]\; i := i + 1\; [v < N]$

Summary

- Weakest precondition reasoning
- Handling termination
- Operational semantics
- Adding non-determinism
- Refinement calculus

Operational semantics

We gave Hoare Logic a **denotational semantics**:

- Programs given an abstract mathematical *denotation* (relation on $\textsc{Env}$)
- Validity of Hoare triples defined in terms of this denotation (inclusion of relational images)

Operational semantics is an alternative approach:

- Define/construct a **reduction relation** between programs, (start) states, and (end) states
- Validity defined in terms of the reduction relation

More formally

As before let $\textsc{Programs}$ be the set of valid $\mathcal{L}$ programs, and $\textsc{Env}$ be the set of states/environments (functions that map variables to numeric values).

The **Operational semantics of Hoare logic** involves defining a relation $\Downarrow \subseteq \textsc{Programs} \times \textsc{Env} \times \textsc{Env}$ recursively (on the structure of a program).

Intuitively $(P, \eta, \eta') \in \Downarrow$, written $[P, \eta] \Downarrow \eta'$, means that the program P reduces to the state η' when executed from state η.

Rules for constructing $\Downarrow$

$$\frac{[\![e]\!]^\eta = n}{[x := e, \eta] \Downarrow \eta[x \mapsto n]}$$

$$\frac{[P, \eta] \Downarrow \eta' \qquad [Q, \eta'] \Downarrow \eta''}{[P; Q, \eta] \Downarrow \eta''}$$

$$\frac{[\![b]\!]^\eta = \texttt{true} \qquad [P, \eta] \Downarrow \eta'}{[\textbf{if } b \textbf{ then } P \textbf{ else } Q \textbf{ fi}, \eta] \Downarrow \eta'}$$

$$\frac{[\![b]\!]^\eta = \texttt{false} \qquad [Q, \eta] \Downarrow \eta'}{[\textbf{if } b \textbf{ then } P \textbf{ else } Q \textbf{ fi}, \eta] \Downarrow \eta'}$$

$$\frac{[\![b]\!]^\eta = \texttt{true} \qquad [P, \eta] \Downarrow \eta' \qquad [\textbf{while } b \textbf{ do } P \textbf{ od}, \eta'] \Downarrow \eta''}{[\textbf{while } b \textbf{ do } P \textbf{ od}, \eta] \Downarrow \eta''}$$

$$\frac{[\![b]\!]^\eta = \texttt{false}}{[\textbf{while } b \textbf{ do } P \textbf{ od}, \eta] \Downarrow \eta}$$

Validity

Under Operational semantics, we say $\{\varphi\}\, P\, \{\psi\}$ is valid, written

$$\models_{\mathsf{os}} \{\varphi\}\, P\, \{\psi\}\,,$$

if

$$\forall \eta, \eta' \in \textsc{Env}.\ \big((\eta \in \langle\varphi\rangle) \wedge ([P, \eta] \Downarrow \eta')\big) \quad \rightarrow \quad \eta' \in \langle\psi\rangle.$$

Theorem

$\models_{os} \{\varphi\}\, P\, \{\psi\}$ *if and only if* $\models \{\varphi\}\, P\, \{\psi\}$

Validity

Under Operational semantics, we say $\{\varphi\}\, P\, \{\psi\}$ is valid, written

$$\models_{\mathsf{os}} \{\varphi\}\, P\, \{\psi\},$$

if

$$\forall \eta, \eta' \in \textsc{Env}.\ \big((\eta \in \langle\varphi\rangle) \land ([P, \eta] \Downarrow \eta')\big) \quad \rightarrow \quad \eta' \in \langle\psi\rangle.$$

Theorem

$$\models_{\mathit{os}} \{\varphi\}\, P\, \{\psi\} \quad \textit{if and only if} \quad \models \{\varphi\}\, P\, \{\psi\}$$

Summary

- Weakest precondition reasoning
- Handling termination
- Operational semantics
- Adding non-determinism
- Refinement calculus

Non-determinism

Non-determinism involves the computational model branching into one of several directions.

- Behaviour is unspecified: any branch can happen (decision made at run-time)
- Purely theoretical concept
- "Dual" of parallelism (one of many branches vs all of many branches); not quantum either

Non-determinism

Why add non-determinism?

- More general than deterministic behaviour
- In many computation models non-determinism represents "magic" behaviour:
  - Always choosing the "best" branch, leading to faster computation (e.g. P vs NP)
  - Error/exception handling
- Useful for abstraction (abstracted code is easier to reason about)
- Mathematically easier to deal with

Non-determinism

Why add non-determinism?

- More general than deterministic behaviour
- In many computation models non-determinism represents "magic" behaviour:
  - Always choosing the "best" branch, leading to faster computation (e.g. P vs NP)
  - Error/exception handling
- Useful for abstraction (abstracted code is easier to reason about)
- Mathematically easier to deal with

Non-determinism

Why add non-determinism?

- More general than deterministic behaviour
- In many computation models non-determinism represents "magic" behaviour:
  - Always choosing the "best" branch, leading to faster computation (e.g. P vs NP)
  - Error/exception handling
- Useful for abstraction (abstracted code is easier to reason about)
- Mathematically easier to deal with

Non-determinism

Why add non-determinism?

- More general than deterministic behaviour
- In many computation models non-determinism represents "magic" behaviour:
  - Always choosing the "best" branch, leading to faster computation (e.g. P vs NP)
  - Error/exception handling
- Useful for abstraction (abstracted code is easier to reason about)
- Mathematically easier to deal with

Non-determinism

Why add non-determinism?

- More general than deterministic behaviour
- In many computation models non-determinism represents "magic" behaviour:
  - Always choosing the "best" branch, leading to faster computation (e.g. P vs NP)
  - Error/exception handling
- Useful for abstraction (abstracted code is easier to reason about)
- Mathematically easier to deal with

$\mathcal{L}^+$: a simple language with non-determinism

We relax the Conditional and Loop commands in $\mathcal{L}$ to give us non-deterministic behaviour.

The programs of $\mathcal{L}^+$ are defined as:

Assign: $x := e$, where x is a variable and e is an expression

Predicate: φ, where φ is a predicate

Sequence: $P; Q$, where P and Q are programs

Choice: $P + Q$, where P and Q are programs; intuitively, make a non-deterministic choice between P and Q

Loop: P^*, where P is a program; intuitively, loopfor a non-deterministic number of iterations

$$P :: (x := e) \mid \varphi \mid P_1; P_2 \mid P_1 + P_2 \mid P_1^*$$

$\mathcal{L}^+$: a simple language with non-determinism

We relax the Conditional and Loop commands in $\mathcal{L}$ to give us non-deterministic behaviour.

The programs of $\mathcal{L}^+$ are defined as:

Assign: $x := e$, where x is a variable and e is an expression

Predicate: φ, where φ is a predicate

Sequence: $P; Q$, where P and Q are programs

Choice: $P + Q$, where P and Q are programs; intuitively, make a non-deterministic choice between P and Q

Loop: P^*, where P is a program; intuitively, loopfor a non-deterministic number of iterations

$$P :: (x := e) \mid \varphi \mid P_1; P_2 \mid P_1 + P_2 \mid P_1^*$$

$\mathcal{L}^+$: a simple language with non-determinism

We relax the Conditional and Loop commands in $\mathcal{L}$ to give us non-deterministic behaviour.

The programs of $\mathcal{L}^+$ are defined as:

Assign: $x := e$, where x is a variable and e is an expression

Predicate: φ, where φ is a predicate

Sequence: $P; Q$, where P and Q are programs

Choice: $P + Q$, where P and Q are programs; intuitively, make a non-deterministic choice between P and Q

Loop: P^*, where P is a program; intuitively, loopfor a non-deterministic number of iterations

$$P :: (x := e) \mid \varphi \mid P_1; P_2 \mid P_1 + P_2 \mid P_1^*$$

$\mathcal{L}^+$: a simple language with non-determinism

We relax the Conditional and Loop commands in $\mathcal{L}$ to give us non-deterministic behaviour.

The programs of $\mathcal{L}^+$ are defined as:

Assign: $x := e$, where x is a variable and e is an expression

Predicate: φ, where φ is a predicate

Sequence: $P; Q$, where P and Q are programs

Choice: $P + Q$, where P and Q are programs; intuitively, make a non-deterministic choice between P and Q

Loop: P^*, where P is a program; intuitively, loopfor a non-deterministic number of iterations

$$P :: (x := e) \mid \varphi \mid P_1; P_2 \mid P_1 + P_2 \mid P_1^*$$

$\mathcal{L}^+$: a simple language with non-determinism

$$P :: (x := e) \mid \varphi \mid P_1; P_2 \mid P_1 + P_2 \mid P_1^*$$

NB

$\mathcal{L}$ can be defined in $\mathcal{L}^+$ by defining:

- **if** b **then** P **else** Q **fi** $= (b; P) + (\neg b; Q)$
- **while** b **do** P **od** $= (b; P)^*; \neg b$

Example

Example

A program in $\mathcal{L}^+$ that non-deterministically checks if $(x \vee y) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee z)$ is satisfiable:

```
SAT
-----------------------------------
(x := 0) + (x := 1);
(y := 0) + (y := 1);
(z := 0) + (z := 1);
if((x = 1) ∨ (y = 1)) ∧
      ((x = 0) ∨ (z = 0)) ∧
      ((y = 0) ∨ (z = 1))
then
   r := 1
else
   r := 0
fi
```

Example

Example

A program in $\mathcal{L}^+$ that non-deterministically checks if $(x \vee y) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee z)$ is satisfiable:

SAT

$(x := 0) + (x := 1);$
$(y := 0) + (y := 1);$
$(z := 0) + (z := 1);$
if $((x = 1) \vee (y = 1)) \wedge$
$\quad ((x = 0) \vee (z = 0)) \wedge$
$\quad ((y = 0) \vee (z = 1))$
then
$\quad r := 1$
else
$\quad r := 0$
fi

Example

Example

A program in $\mathcal{L}^+$ that non-deterministically checks if $(x \vee y) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee z)$ is satisfiable:

SAT
$(x := 0) + (x := 1);$ $(y := 0) + (y := 1);$ $(z := 0) + (z := 1);$ **if**$((x = 1) \vee (y = 1)) \wedge$ $((x = 0) \vee (z = 0)) \wedge$ $((y = 0) \vee (z = 1))$ **then** $r := 1$ **else** $r := 0$ **fi**

Proof rules

Hoare logic rules are cleaner:

$$\frac{\{\varphi\}\, P\, \{\psi\} \qquad \{\varphi\}\, Q\, \{\psi\}}{\{\varphi\}\, P + Q\, \{\psi\}} \quad \text{(choice)}$$

$$\frac{\{\varphi\}\, P\, \{\varphi\}}{\{\varphi\}\, P^*\, \{\varphi\}} \quad \text{(loop)}$$

Semantics

Denotational semantics are cleaner:

- $[\![P + Q]\!] = [\![P]\!] \cup [\![Q]\!]$
- $[\![P^*]\!] = [\![P]\!]^*$

Operational semantics are cleaner:

$$\frac{[P, \eta] \Downarrow \eta'}{[P + Q, \eta] \Downarrow \eta'} \qquad \frac{[Q, \eta] \Downarrow \eta'}{[P + Q, \eta] \Downarrow \eta'}$$

$$\frac{}{[P^*, \eta] \Downarrow \eta} \qquad \frac{[P, \eta] \Downarrow \eta' \qquad [P^*, \eta'] \Downarrow \eta''}{[P*, \eta] \Downarrow \eta''}$$

Semantics

Denotational semantics are cleaner:

- $[\![P + Q]\!] = [\![P]\!] \cup [\![Q]\!]$
- $[\![P^*]\!] = [\![P]\!]^*$

Operational semantics are cleaner:

$$\frac{[P, \eta] \Downarrow \eta'}{[P + Q, \eta] \Downarrow \eta'} \qquad \frac{[Q, \eta] \Downarrow \eta'}{[P + Q, \eta] \Downarrow \eta'}$$

$$\frac{}{[P^*, \eta] \Downarrow \eta} \qquad \frac{[P, \eta] \Downarrow \eta' \qquad [P^*, \eta'] \Downarrow \eta''}{[P*, \eta] \Downarrow \eta''}$$

Summary

- Weakest precondition reasoning
- Handling termination
- Operational semantics
- Adding non-determinism
- Refinement calculus

Looking forward (beyond this course)

A program P **refines** a program Q (equivalently, Q is an **abstraction** of P), written $P \sqsupseteq Q$, if

$$[\![P]\!] \subseteq [\![Q]\!]$$

The goal of **refinement calculus** is to start from a very abstract specification, P_0, and to *calculate* refinements

$$P_0 \sqsubseteq P_1 \sqsubseteq P_2 \sqsubseteq \cdots$$

until something resembling code (e.g. $\mathcal{L}^+$) is reached.

Refinement calculus

Built around the same semantics: programs are relations.

$$\varphi \rightsquigarrow \psi$$

represents the most abstract program that takes states satisfying φ to states satisfying ψ: namely, $\langle\varphi\rangle \times \langle\psi\rangle$.

Rules introduce the language constructs:

- $(\varphi[e/x] \rightsquigarrow \varphi) \sqsubseteq x := e$ (assign)
- $(\varphi \rightsquigarrow \varphi \wedge g) \sqsubseteq g$ (guard)
- $(\varphi \rightsquigarrow \psi) \sqsubseteq (\varphi \rightsquigarrow \psi); (\psi \rightsquigarrow \rho)$ (seq)
- $(\varphi \rightsquigarrow \psi) \sqsubseteq (\varphi \rightsquigarrow \psi) + (\varphi \rightsquigarrow \psi)$ (choice)
- $(\varphi \rightsquigarrow \varphi) \sqsubseteq (\varphi \rightsquigarrow \varphi)^*$ (star)
- $(\varphi \rightsquigarrow \psi) \sqsubseteq (\varphi' \rightsquigarrow \psi')$ if $\varphi \rightarrow \varphi'$ and $\psi' \rightarrow \psi$ (cons)