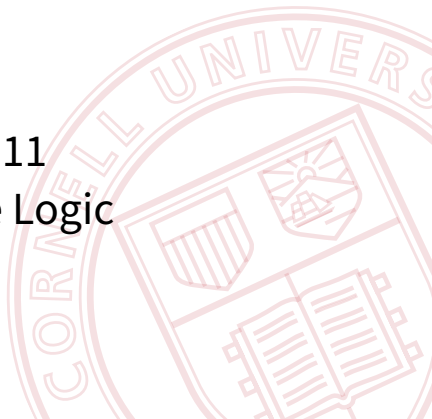


CS 4110

Programming Languages & Logics

Lecture 11
More Hoare Logic



Overview

Last time

- Hoare Logic

Today

- “Decorated” programs
- Weakest Preconditions

Review: Hoare Logic

$$\frac{}{\vdash \{P\} \mathbf{skip} \{P\}} \text{ SKIP}$$

$$\frac{}{\vdash \{P[a/x]\} x := a \{P\}} \text{ ASSIGN}$$

$$\frac{\vdash \{P\} c_1 \{R\} \quad \vdash \{R\} c_2 \{Q\}}{\vdash \{P\} c_1; c_2 \{Q\}} \text{ SEQ}$$

$$\frac{\vdash \{P \wedge b\} c_1 \{Q\} \quad \vdash \{P \wedge \neg b\} c_2 \{Q\}}{\vdash \{P\} \mathbf{if } b \mathbf{ then } c_1 \mathbf{ else } c_2 \{Q\}} \text{ IF}$$

$$\frac{\vdash \{P \wedge b\} c \{P\}}{\vdash \{P\} \mathbf{while } b \mathbf{ do } c \{P \wedge \neg b\}} \text{ WHILE}$$

$$\frac{\models P \Rightarrow P' \quad \vdash \{P'\} c \{Q'\} \quad \models Q' \Rightarrow Q}{\vdash \{P\} c \{Q\}} \text{ CONSEQUENCE}$$

Decorated Programs

Observation: Once we've identified loop invariants and uses of consequence, the structure of a Hoare logic is determined!

Notation: Can write proofs by “decorating” programs with:

- A precondition ($\{P\}$)
- A postcondition ($\{Q\}$)
- Invariants ($\{I\}$ **while** b **do** c)
- Uses of consequence $\{R\} \Rightarrow \{S\}$
- Assertions between sequences $c_1; \{T\}c_2$

A decorated program describes a valid Hoare logic proof if the rest of the proof tree's structure is implied. (Caveats: Invariants are constrained, etc.)

Example: Decorated Factorial

```
{x = n ∧ n > 0}
```

```
y := 1;
```

```
while x > 0 do {
```

```
    y := y * x;
```

```
    x := x - 1
```

```
}
```

```
{y = n!}
```

Example: Decorated Factorial

```
{x = n ∧ n > 0} ⇒  
{1 = 1 ∧ x = n ∧ n > 0}  
y := 1;  
{y = 1 ∧ x = n ∧ n > 0} ⇒  
{y * x! = n! ∧ x ≥ 0}  
while x > 0 do {  
    {y * x! = n! ∧ x > 0 ∧ x ≥ 0} ⇒  
    {y * x * (x - 1)! = n! ∧ (x - 1) ≥ 0}  
    y := y * x;  
    {y * (x - 1)! = n! ∧ (x - 1) ≥ 0}  
    x := x - 1  
    {y * x! = n! ∧ x ≥ 0}  
}  
{y * x! = n! ∧ (x ≥ 0) ∧ ¬(x > 0)} ⇒  
{y = n!}
```

Informal Rules for Decoration

Check whether a decorated program represents a valid proof using **local consistency** checks.

Informal Rules for Decoration

Check whether a decorated program represents a valid proof using **local consistency** checks.

For **skip**, the precondition and postcondition should be the same:

$$\begin{array}{c} \{P\} \\ \mathbf{skip} \\ \{P\} \end{array}$$

Informal Rules for Decoration

For sequences, $\{P\} c_1 \{R\}$ and $\{R\} c_2 \{Q\}$ must be (recursively) locally consistent:

$\{P\}$

$c_1;$

$\{R\}$

c_2

$\{Q\}$

Informal Rules for Decoration

Assignment should use the substitution from the rule:

$$\{P[a/x]\}$$
$$x := a$$
$$\{P\}$$

Informal Rules for Decoration

An **if** is locally consistent when both branches are locally consistent after adding the branch condition to each:

```
{P}
if b then
  {P ∧ b}
  C1
  {Q}
else
  {P ∧ ¬b}
  C2
  {Q}
{Q}
```

Informal Rules for Decoration

Decorate a **while** with the loop invariant:

```
{P}  
while b do  
  {P ∧ b}  
  c  
  {P}  
{P ∧ ¬b}
```

Informal Rules for Decoration

To capture the CONSEQUENCE rule, you can always write a (valid) implication:

$$\{P\} \Rightarrow \{Q\}$$

Example

```
{
```

```
  while ( $0 < y$ ) do (
```

```
     $x := x + 1;$ 
```

```
     $y := y - 1$ 
```

```
  )
```

```
{
```

Example

$\{x = m \wedge y = n \wedge 0 \leq n\}$

while ($0 < y$) **do** (

$x := x + 1;$

$y := y - 1$

)

$\{x = m + n\}$

Example

$$\{x = m \wedge y = n \wedge 0 \leq n\} \Rightarrow$$
$$\{I\}$$

while ($0 < y$) **do** (

$$\{I \wedge 0 < y\} \Rightarrow$$
$$\{I[y - 1/y][x + 1/x]\}$$
$$x := x + 1;$$
$$\{I[y - 1/y]\}$$
$$y := y - 1$$
$$\{I\}$$

)

$$\{I \wedge 0 \not< y\} \Rightarrow$$
$$\{x = m + n\}$$

Where I is $(x = m + n - y) \wedge 0 \leq y$.

Example

```
{ }
```

```
while (x  $\neq$  0) do (  
  x := x - 1  
)
```

```
{ }
```

Example

{true}

while ($x \neq 0$) **do** (

$x := x - 1$

)

{ $x = 0$ }

Example

```
{  
    }  
y := 1  
while (0 < x) do (  
    x := x - 1;  
    y := y * 2  
)  
{  
    }
```

Example

$\{x = n \wedge 0 \leq n\}$

$y := 1$

while $(0 < x)$ **do** (

$x := x - 1;$

$y := y * 2$

)

$\{y = 2^n\}$