

# Rules for reasoning about code

CSE 331

University of Washington

Michael Ernst

# Review:

## Forward vs. backward reasoning

**Forward reasoning** is more **intuitive** for most people

Helps you understand what will happen (simulates the code)

Introduces facts that may be irrelevant to the goal

Set of current facts may get large

Takes longer to realize that the task is hopeless

**Backward reasoning** is usually more **helpful**

Helps you understand what should happen

Given a specific goal, indicates how to achieve it

Given an error, gives a test case that exposes it

# Reasoning about code statements

Goal: Convert **assertions about programs** into **logic**

Mechanical process; just follow rules:

- Rule for each type of statement

- Rule for combining/eliminating statements

There is a (forward and backward) rule for each statement in the programming language

- Loops have no rule: you have to *guess a loop invariant*

# Hoare triples: A notation for properties about code



Sir Anthony Hoare

A Hoare triple:  $\{ P \} \mathbf{code} \{ Q \}$

P and Q are logical statements (about program values)

**code** is Java code

“ $\{ P \} \mathbf{code} \{ Q \}$ ” means “if P is true and you execute **code**, then Q is true afterward”

“ $\{ P \} \mathbf{code} \{ Q \}$ ” is a logical formula like “ $x + y = z$ ”

Examples:

“ $1 + 2 = 3$ ” is true

“ $2 + 2 = 5$ ” is false

“ $\{ x > 0 \} x++ \{ x > 1 \}$ ” is true

“ $\{ x < 0 \} x++ \{ x < 0 \}$ ” is false

“ $\{ x > 0 \} x++ \{ x > -5 \}$ ” is true

Is this notation good for forward or for backward reasoning?

# Backward reasoning: Assignment

```
// precondition: ??
```

```
x = e;
```

```
// postcondition: Q
```

Precondition = Q with all (free) occurrences of x replaced by e

Examples:

```
// assert: ??
```

```
y = x + 1;
```

```
// assert y > 0
```

Precondition =  $(x+1) > 0$

```
// assert: ??
```

```
z = z + 1;
```

```
// assert z > 0
```

Precondition =  $(z+1) > 0$

Notation: wp for “*weakest* precondition”

$\text{wp}(\text{“x=e;”}, Q) = Q$  with x replaced by e

Weakest = most general

Strongest = most specific

# Aside: weaker and stronger

Strength of an assertion corresponds to logical implication

- “x is stronger than y” corresponds to “ $x \Rightarrow y$ ”
- “x is stronger than y” means x guarantees more than y
- “x is stronger than y” means fewer worlds satisfy x

Suppose that all of the following are true:

- $a \Rightarrow b$      $b \Rightarrow c$      $x \Rightarrow y$      $y \Rightarrow z$
- $\{ b \}$  **mycode**  $\{ y \}$

Then which of these are true?

- $\{ a \}$  **mycode**  $\{ y \}$
- $\{ c \}$  **mycode**  $\{ y \}$
- $\{ b \}$  **mycode**  $\{ x \}$
- $\{ b \}$  **mycode**  $\{ z \}$

# Method calls

```
// precondition: ??  
x = foo () ;  
// postcondition: Q
```

If the method has no side effects: just like ordinary assignment

<pre>// precondition: ?? <b>x = Math.sqrt (y) ;</b> // postcondition: x = 3</pre>	<pre>// precondition: ?? <b>x = Math.abs (y) ;</b> // postcondition: x = 22</pre>
<p>Precondition: (y = 9) and (x = anything)</p>	<p>Precondition: (y = 22 or y = -22)</p>

If it has side effects: an assignment to every var method may modify

Use the method specification to determine the new value

```
// precondition: ??    z+1 = 22  
incrementZ () ;    // spec:  $z_{\text{post}} = z_{\text{pre}} + 1$   
// postcondition: z = 22
```

# Composition (statement sequences; blocks)

```
// precondition: ??  
S1;      // some statement  
S2;      // another statement  
// postcondition: Q
```

Work from back to front

Precondition =  $\text{wp}(\text{"S1; S2;"}, Q) = \text{wp}(\text{"S1;"}, \text{wp}(\text{"S2;"}, Q))$

Example:

```
// precondition: ??  
x = 0;  
y = x+1;  
// postcondition: y > 0
```

Think of this as:

```
// precondition: ??  
x = 0;  
// postcondition: ??  
// precondition: ??  
y = x+1;  
// postcondition: y > 0
```

Same condition



# If statement example

```
// precondition: ??
```

```
if (x < 5) {
```

```
    x = x*x;
```

```
} else {
```

```
    x = x+1;
```

```
}
```

```
// postcondition:  $x \geq 9$ 
```

# If statements

```
// precondition: ??  
if (b) S1 else S2  
// postcondition: Q
```

Do case analysis:

$$\begin{aligned} & \text{wp}(\text{"if (b) s1 else s2"}, Q) \\ &= ( \quad b \Rightarrow \text{wp}(\text{"s1"}, Q) \\ & \quad \wedge \neg b \Rightarrow \text{wp}(\text{"s2"}, Q) \quad ) \\ &= ( \quad b \wedge \text{wp}(\text{"s1"}, Q) \\ & \quad \vee \neg b \wedge \text{wp}(\text{"s2"}, Q) \quad ) \end{aligned}$$

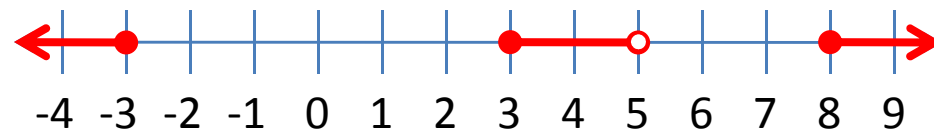
(Why is there no substitution in the condition?)

# If statement example redux

```
// precondition: ??  
if (x < 5) {  
    x = x*x;  
} else {  
    x = x+1;  
}  
// postcondition: x ≥ 9
```

Precondition

$$\begin{aligned} &= \text{wp}(\text{"if (x<5) \{x = x*x;\} else \{x = x+1;\}"}, x \geq 9) \\ &= (x < 5 \wedge \text{wp}(\text{"x=x*x"}, x \geq 9)) \vee (x \geq 5 \wedge \text{wp}(\text{"x=x+1"}, x \geq 9)) \\ &= (x < 5 \wedge x*x \geq 9) \vee (x \geq 5 \wedge x+1 \geq 9) \\ &= (x \leq -3) \vee (x \geq 3 \wedge x < 5) \vee (x \geq 8) \end{aligned}$$



# If statements review

## Forward reasoning

```
{P}
if B
  {P ∧ B}
  S1
  {Q1}
else
  {P ∧ !B}
  S2
  {Q2}
{Q1 ∨ Q2}
```

## Backward reasoning

```
{ (B ∧ wp(S1, Q)) ∨ (¬B ∧ wp(S2, Q)) }
if B
  {wp(S1, Q)}
  S1
  {Q}
else
  {wp(S2, Q)}
  S2
  {Q}
{Q}
```

# If statement with one branch empty

```
// precondition: ??  
if (x > y) {  
    tmp = x;  
    x = y;  
    y = x;  
}  
// postcondition:  $x < y$ 
```