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

Overall plan:
- 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

A Hoare triple: \{ P \} \texttt{code} \{ Q \}

- P and Q are logical statements (about program values)
- \texttt{code} is Java code

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

“\{ P \} \texttt{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”
wp("x=e;", Q) = Q with x replaced by e

Weakest = most general
Strongest = most specific
Method calls

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

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

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

If it has side effects: an assignment to every var in modifies
Use the method specification to determine the new value

// precondition: ??  z+1 = 22
incrementZ();     // spec:  z_post = z_pre + 1
// postcondition: z = 22
Composition (statement sequences; blocks)

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

Work from back to front
Precondition = wp("s1; s2;", Q) = wp("s1;", wp("s2;", Q))

Example:
  // precondition: ??
  x  =  0;
  y  =  x+1;
  // postcondition: y > 0
If statement example

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

// postcondition: x ≥ 9
If statements

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

Do case analysis:

wp("if (b) S1 else S2", Q) = ( b \Rightarrow wp("s1", Q) \\
\& \neg b \Rightarrow wp("s2", Q) )

= ( b \& wp("s1", Q) \\
\lor \neg b \& wp("s2", Q) )

(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
= wp("if (x<5) {x = x*x;} else {x = x+1}", x ≥ 9)
= (x < 5 ∧ wp("x=x*x", x ≥ 9)) ∨ (x ≥ 5 ∧ wp("x=x+1", x ≥ 9))
= (x < 5 ∧ x*x ≥ 9) ∨ (x ≥ 5 ∧ x+1 ≥ 9)
= (x ≤ -3) ∨ (x ≥ 3 ∧ x < 5) ∨ (x ≥ 8)