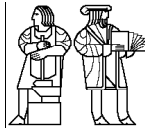


6.170 Lecture 14 Reasoning About Code



John Guttag
MIT EECS

Plan

A few minutes on left-over material

Quick look at informal reasoning

Reasoning using weakest preconditions

- Assignment
- Composition
- Flow of control

Reasoning about loops

- Partial correctness
- Termination

© Michael Ernst/John Guttag

Spring 2001

Slide 2

Reasoning About Code

Pre-assertion
code
Post-assertion

Total correctness

$P \{S\} Q$ -- if P is true before starting S , S will terminate cleanly in a state satisfying Q

Partial correctness

$P \{S\} Q$ -- if P is true of the state before starting S , if S terminates cleanly, the state will satisfy Q

© Michael Ernst/John Guttag

Spring 2001

Slide 3

Reasoning About Code

```
//assert x > 0  
x = x - 1;  
//assert x ≥ 0
```

```
x > 0 ⇒ (x - 1 ≥ 0)
```

Converted a conjecture about programs

Into a conjecture about integers

General plan

- Eliminate code a statement at a time
- Rely on knowledge of logic and types

© Michael Ernst/John Guttag

Spring 2001

Slide 4

Formalize Using Weakest Precondition

$wp(S, Q)$ is the weakest predicate s.t.

$wp(S, Q) \{S\} Q$ holds

To show

Pre-assertion $\{code\}$ Post-assertion

Show

Pre-assertion $\Rightarrow wp(code, Post-assertion)$

wp essentially turning a crank

Algorithmically converts assertions about programs to logic

Work backwards through program text

© Michael Ernst/John Guttag

Spring 2001

Slide 5

Weakest Precondition, Assignment

$wp(x = e, Q)$ is

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

E.g., $wp(x = x + 1, x > 0)$

$= (x + 1) > 0$

$= x \geq 0$

```
//assert x > 0  
x = x + 1;  
//assert x > 0
```

```
x > 0 ⇒ x ≥ 0  
true
```

© Michael Ernst/John Guttag

Spring 2001

Slide 6

Weakest Precondition, Composition

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

E.g., $\text{wp}(x = 0; y = x + 1, y > 0)$
 $= \text{wp}(y = x + 1, \text{wp}(x = 0, y > 0))$
 $= \text{wp}(y = x + 1, 0 > 0)$
 $= 0 > 0$
 $= \text{false}$

Whoops!
What's wrong here?

© Michael Ernst/John Guttag Spring 2001 Slide 7

Weakest Precondition, If

$$\text{wp}(\text{if } b \text{ S1 else S2, } Q) = b \Rightarrow \text{wp}(S1, Q) \ \& \ \neg b \Rightarrow \text{wp}(S2, Q)$$

Essentially case analysis

<pre>//assert P if b S1; else S2 //assert P'</pre>	<pre>P & b {S1} P' P & ¬b {S2} P'</pre>
------------------------------------------------------	-------------------------------------------------

© Michael Ernst/John Guttag Spring 2001 Slide 8

If, an Example

<pre>//assert true if x == 0 x = x + 1; else x = x/x; //assert x ≥ 0</pre>

$\text{true} \Rightarrow \text{wp}(\text{if } x == 0 \text{ } x = x + 1; \text{ else } x = x/x, x \geq 0)$
 $= x = 0 \Rightarrow (\text{wp}(x = x + 1, x \geq 0) \ \& \ x \neq 0 \Rightarrow \text{wp}(x = x/x, x \geq 0))$
 $= (x = 0 \Rightarrow x + 1 \geq 0) \ \& \ (x \neq 0 \Rightarrow x/x \geq 0)$
 $= 1 \geq 0 \ \& \ 1 \geq 0$
 $= \text{true}$

© Michael Ernst/John Guttag Spring 2001 Slide 9

Reasoning About Loops

Case analysis not obvious
A loop represents an unknown number of paths (cases)

Cannot enumerate all paths
What makes testing and reasoning hard
Along with recursion

<pre>//assert x ≥ 0 & y = 0 while (x != y) y = y + 1; //assert x = y</pre>

© Michael Ernst/John Guttag Spring 2001 Slide 10

Reasoning About Loops, Informal

<pre>//assert x ≥ 0 & y = 0 while (x != y) y = y + 1; //assert x = y</pre>

- 1) Pre-assertion guarantees that $x \geq y$
- 2) Every time through loop
y incremented by 1
x unchanged
Therefore y closer to x
- 3) Since there are only a finite number of integers between x and y, y will eventually equal x
- 4) Program exits loop as soon as $x = y$

© Michael Ernst/John Guttag Spring 2001 Slide 11

Reasoning About Loops, cont.

Just made an inductive argument

What are we inducting over?
Number of iterations

Computation induction
Show that conjecture holds if zero iterations
Assume that it holds after n iterations
Show that it holds after n+1 iterations

Two parts to induction hypothesis
Loop invariant – partial correctness
Preserved by each iteration
Decreasing function – termination
Reduced by each iteration

© Michael Ernst/John Guttag Spring 2001 Slide 12

6.170 Properties of Loop Invariant, I

$P \{ \text{while } b \ S; \} Q$

Find an invariant, I, such that

- 1) $P \Rightarrow I$
- 2) $I \ \& \ b \ \{S\} \ I$
- 3) $(I \ \& \ \neg b) \Rightarrow Q$

- 1) **Sufficient to know that if loop terminates, Q will hold**
- 2) **Finding the invariant the key to reasoning about loops**
- 3) **Inductive assertions**
 - 1) Complete method of proof
 - 2) If loop does satisfy pre/post conditions, there exists an invariant sufficient to show it

© Michael Ernst/John Guttag Spring 2001 Slide 13

6.170 Back to Example

```
//assert x ≥ 0 & y = 0
while (x != y)
  y = y + 1;
//assert x = y
```

So, what is a suitable invariant?

What makes the loop work?

$I = x \geq y$

- 1) $x \geq 0 \ \& \ y = 0 \Rightarrow I$
- 2) $I \ \& \ x \neq y \ \{y = y + 1;\} \ I$
- 3) $(I \ \& \ \neg(x \neq y)) \Rightarrow x = y$

© Michael Ernst/John Guttag Spring 2001 Slide 14

6.170 Still Have Only Shown Partial Correctness

Nothing I have done so far should convince you that the loop terminates

Decrementing function
Maps subset of program variables to well-ordered set

Well-ordered set
Ordered set
Every non-empty subset has least element

Which of following is well-ordered?
Natural numbers
Non-negative reals
Integers

© Michael Ernst/John Guttag Spring 2001 Slide 15

6.170 Decrementing Function

$P \{ \text{while } b \ S; \} Q$

D(X), where x is some subset of state, such that

- 1) $I \ \& \ b \ \{S\} \ D(X') < D(X)$
- 2) $(I \ \& \ D(X) = \text{minVal}) \Rightarrow \neg b$

© Michael Ernst/John Guttag Spring 2001 Slide 16

6.170 Proving Termination

```
//assert x ≥ 0 & y = 0
//Invariant: x ≥ y
//Decrements (x-y)
while (x != y)
  y = y + 1;
//assert x = y
```

- 1) $\text{assert } (x - y) = d0 \ \& \ (y \neq x)$
 $y = y + 1;$
 $\text{assert } (x - (y + 1)) < d0$
- 2) $(x \geq y \ \& \ x - y = 0) \Rightarrow (x = y)$


© Michael Ernst/John Guttag Spring 2001 Slide 17

6.170 What Happened to WP?

Did you notice that I stopped using wp when I got to loops?

Won't work for loops
Generating invariant and decrementing function not algorithmic

Given invariant & decrementing function
Can then just turn crank



© Michael Ernst/John Guttag Spring 2001 Slide 18



In Practice?

I don't routinely write

Loop invariants and decrementing functions

I do write them when I am unsure about a loop

When I have evidence that a loop is not working

Add invariant and decrementing function if missing
Write code to check them