Lecture 16 — Proving loop correctness

- For many years, computer scientists have studied w *prove* programs correct (as opposed to testing for bugs most important concept in this area is that of an *inva* We will study the notion of a *loop invariant*, which i to prove the correctness of loops.
- Topics we will cover are:
  - Hoare logic
  - Loop invariants
  - Termination conditions

From lecture 1: What you will le
this semester

- How to implement programming languages
  - Writing lexical analyzers and parsers
  - Translating programs to machine language
  - Implementing run-time systems
- How to write programs in a functional programming lan
- How to formally define languages (including the defi
  of type rules and of program execution)
- Key differences between statically-typed languages (e
  Java) and dynamically-typed languages (Python, JavaS
- Plus a few other things...

Invariants

- An *invariant* is a relationship among the variable program that is always known to hold at a given point program.
- Example: If L is a doubly-linked list, for each node nd able from L, if nd.next is not null, then nd = nd.next
  - Note that this invariant holds almost everywhere program, except possibly in the functions that add or r nodes.

Invariants (cont.)

- Invariants are *absolutely essential* in understanding
program works. When you have a bug in a program an
at the values of the variables and say, "Hmm, that v
shouldn't have that value at this point," you're sayin
the program has failed to maintain an invariant tha
assumed it would.

- One type of invariant is a *loop invariant*. This is a re
ship among the variables in a loop that should always h
the beginning and end of each iteration of the loop (t
not necessary within the loop body).

- Loop invariants can be used to formally prove the corre
of a program that uses loops.

Hoare triples

- Program correctness is usually formalized using a k
judgment called a *Hoare triple* (after C.A.R. Hoare):

$$P\{S\}Q$$

where P and Q are assertions involving the variables
program, and S is the program ("S" for "statement").

- This means: If P is true about the program variables
if S is executed, then Q will be true when it finishes.

- Examples:
  - $x > 0$ { $x = x - 1$ } $x \geq 0$
  - $x = x0 \wedge y = y0 \wedge x > 0 \wedge y > x$ { y = y-x } $gcd(x, y) = gcd(x$

Proving loops: Partial correctne

- Suppose we want to prove a Hoare triple of the form:

$$P \{ \text{ while } (b) \ \{S\} \ \} \ Q$$

- A *loop invariant* for this loop is a condition I on the pr
variables (like P and Q) that is always true at the beg
and end of every iteration of S.

- To prove the above Hoare triple:
  - Prove I is an invariant: $b \wedge I \{ S \} I$
  - Prove I is true at the start: $P \wedge b \supset I$
  - Prove Q is true after the loop: $\neg b \wedge I \supset Q$

Proving loops: Termination

- The Hoare triple only proves *partial correctness*: Q
  if the loop terminates.

- To prove that a loop terminates, define a function T
  gram variables $\rightarrow$ integers. Then prove:

1. For all values of the program variables x, y, ..., $T(x, y$
   0.
2. If x_0, y_0, ... are the values of the program variables
   start of S and x, y, ... are their values after execu
   once, then $T(x, y, \ldots) < T(x_0, y_0, \ldots)$

- Regardless of what T is, if these two conditions hol
  loop must terminate eventually.

Loop proving example 1

$x = n \wedge y = 1$ {

```
while (x!=0) {y = y*x;  x = x-1;}
```

} $y = n!$

x_0, y_0 = values of x and y at start of iteration

- **Invariant I:** $y = (x+1) \cdot \ldots \cdot n$
- **I is an invariant:** $y_0 = (x_0+1) \cdot \ldots \cdot n \Rightarrow y_0 \cdot x_0 = x_0 \cdot \ldots \cdot n$, i.e. $y = (x+1) \cdot \ldots \cdot n$
- **I holds at the start:** $x = n \Rightarrow (x+1) \cdot \ldots \cdot n = 1 = y$
- **Q holds at the end:** $y = (x+1) \cdot \ldots \cdot n \wedge !(x != 0) \Rightarrow y = 1 \cdot 2 \cdot \ldots \cdot n$
- $T(x, y, n) = x$
- $T(x, y, n) \geq 0$: loop terminates when $x = 0$
- $T(x, y, n) < T(x_0, y_0, n)$: obvious

Loop proving example 2

$a = lis \wedge b = 0$ {

```
    while (a != []) { b = b + hd(a); a = tl(a
```

} $b = \Sigma lis$

- ***Invariant I:*** $b = \Sigma lis - \Sigma a$
- ***I is an invariant:*** $b_0 = \Sigma lis - \Sigma a_0 \wedge b = b_0 + hd\ a_0 \wedge a = tl\ a_0$
  $\Rightarrow b = \Sigma lis - \Sigma a$
- ***I holds at the start:*** $b = 0 = \Sigma lis - \Sigma lis$
- ***Q holds at the end:*** $a = [\,] \Rightarrow \Sigma lis - \Sigma a = \Sigma lis$
- $T(a, b, lis) = |a|$
- $T(a, b, lis) \geq 0$: length of a list always ≥ 0
- $T(a, b, lis) < T(a_0, b_0, lis)$: size of a decreases in every iteration

Loop proving example 3

$a > 0 \land b > 0 \land a = x \land b = y$

```
{ while (a != b) if  (a > b) a = a - b;
                        else b = b - a; }
```
$a = gcd(x_0,$

- **Invariant I:** $gcd(a,b) = gcd(x,y)$
- ***I* is an invariant:** $n > m \Rightarrow gcd(n,m) = gcd(n-m, m)$
- ***I* holds at the start:** obvious, since $a = x$ and $b = y$
- ***Q* holds at the end:** $a = b \Rightarrow a = gcd(a,b)$
- $T(a, b, x, y) = a + b$
- $T(a, b, x, y) \geq 0$: a, b start positive, and always subtract smaller from larger
- $T(a, b, x, y) < T(a_0, b_0, x, y)$: Either a or b is decreased, and the other unchanged, at every iteration.

Loop proving example 4

$x = 0 \wedge y = 0$ {

```
    while (y < n) { y = y + 1; x:= x + y; }
```

} $x = 1 + \cdots + n$

- **Invariant I:** $x = \sum_{i=1}^{y} i$
- **I is an invariant:** $x_0 = \sum_{i=1}^{y_0} i \wedge y = y_0 - 1 \wedge x = x_0 + y \Rightarrow x = \sum_{i=1}^{y} i$
- **I holds at the start:** Σ over empty set $= 0$
- **Q holds at the end:** $y = n \Rightarrow x = \sum_{i=1}^{n} i$
- $T(x, y, n) = n - y$
- $T(x, y, n) \geq 0$: Loop ends when $y = n$
- $T(x, y, n) < T(x_0, y_0, n)$: obvious

Loop proving example 5

$x = 0 \wedge y = 1 \wedge z = 1 \wedge n \geq 1$ {

```
while (z != n) { y = x + y; x = y - x; z =
```

} $y = fib(n)$

- **Invariant I:** $y = fib\ z \wedge x = fib\ (z-1)$
- ***I*** **is an invariant:** $y = fib\ (z_0) + fib\ (z_0 - 1) = fib\ (z_0 + 1) = fib\ (z)$
  $x = y - x_0 = fib\ (z_0) + fib\ (z_0 - 1) - fib\ (z_0 - 1) = fib\ (z_0) = fib\ (z-1)$
- ***I*** **holds at the start:** $fib\ 1 = 1,\ fib\ 0 = 0$
- ***Q*** **holds at the end:** Immediate
- $T(x, y, z, n) = n - z$
- $T(x, y, z, n) \geq 0$: Loop terminates when $n = z$
- $T(x, y, z, n) < T(x_0, y_0, z_0, n)$: Obvious

Loop proving example 6

$x = lst \wedge y = 0$ {

```
    while (x != []) { x = tl x; y = y + 1; }
```

} $y = length(lst)$

- **Invariant I:** $y = |lst| - |x|$
- **I is an invariant:** $y_0 = |lst| - |x_0| \Rightarrow \underbrace{y_0 + 1}_{y} = |lst| - \underbrace{|tl(x_0)|}_{|x|}$
- **I holds at the start:** $0 = |lst| - |lst|$
- **Q holds at the end:** $x = [\,] \Rightarrow |lst| - |x| = |lst|$
- $T(x, y, lst) = |x|$
- $T(x, y, lst) \geqslant 0$: Length of list always ≥ 0
- $T(x, y, lst) < T(x_0, y_0, lst)$: Obvious

Loop proving example 7

$x = lst \wedge y = [\,]$ {

```
    while (x != []) { y = hd x :: y; x = tl x;
```

} $y = reverse(lst)$

- **Invariant I:** reverse(y) @ x = lst
- **I is an invariant:** reverse(y_0) @ (hd x_0 :: tl x_0) = reverse(hd x_0 :: y_0) @ (tl x_0)
- **I holds at the start:** y = [] ⇒ reverse y = [] ⇒ reverse y @ x = x = lst
- **Q holds at the end:** x = [] ⇒ rev(y) @ x = rev(y) ∧ rev(y) = lst ⇒ y = rev(lst)
- $T(x, y, lst) = |x|$
- $T(x, y, lst) > 0$: as above
- $T(x, y, lst) < T(x_0, y_0, lst)$: obvious

Hoare logic

- **C.A.R. Hoare presented a logic — axioms and ru inference, similar to SOS rules — for proving Hoare tri**

(Assignment) $P[e/x] \; \{ \; x = e \; \} \; P$

(While) $\dfrac{P \; \{ \text{ while } (b) \; S \; \} \; Q}{I \wedge b \; \{ \; S \; \} \; I}$

(if $P \wedge b \supset I$ and $P \wedge \neg b \supset$

(Sequence) $\dfrac{P \; \{ \; S_1; \; S_2 \; \} \; Q}{P \; \{ \; S_1 \; \} \; R \quad R \; \{ \; S_2 \; \} \; Q}$

(If) $\dfrac{P \; \{ \text{ if } (b) \; S_1 \text{ else } S_2 \; \}}{P \wedge b \; \{ \; S_1 \; \} \; Q \quad P \wedge \neg b \; \{ \; S_2 \; \} \; Q}$

(Consequence) $\dfrac{P \; \{ \; S \; \} \; Q}{P' \; \{ \; S \; \} \; Q'}$

(if $P \supset P'$ and $Q' \supset Q$)

Example of a proof in Hoare's lo

(If)	true { `if (x<0) y = -x; else y = x;` } $y = \|x\|$	
(Consequence)	$x < 0$ { `y = -x` } $y = \|x\|$	$(x < 0 \supset -x = \|x\|)$
(Assignment)	$-x = \|x\|$ { `y = -x` } $y = \|x\|$	
(Consequence)	$x \not< 0$ { `y = x` } $y = \|x\|$	$(x \not< 0 \supset x = \|x\|)$
(Assignment)	$x = \|x\|$ { `y = x` } $y = \|x\|$	

Wrap-up

- Today we discussed:
  - Loop invariants
  - Partial correctness
  - Proving termination
  - Hoare logic
- We discussed them because:
  - They can help you understand how to prove programs correct.
- In Thursday's class, we will:
  - Discuss the history of programming languages
- What to do now:
  - HW8