Lecture 16 — Proving loop correctness

- **For many years, computer scientists have studied ways to $prove$ programs correct (as opposed to testing for bugs). The most important concept in this area is that of an $invariant$. We will study the notion of a $loop\ invariant$, which is used to prove the correctness of loops.**

- **Topics we will cover are:**

  - **Hoare logic**
  - **Loop invariants**
  - **Termination conditions**

From lecture 1: What you will learn 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 language**

● **How to formally define languages (including the definitions of type rules and of program execution)**

● **Key differences between statically-typed languages (e.g. C, Java) and dynamically-typed languages (Python, JavaScript)**

● **Plus a few other things...**

Invariants

- **An *invariant* is a relationship among the variables in a program that is always known to hold at a given point in the program.**

- **Example: If `L` is a doubly-linked list, for each node `nd` reachable from `L`, if `nd.next` is not null, then `nd = nd.next.prev`.**

  - **Note that this invariant holds almost everywhere in the program, except possibly in the functions that add or remove nodes.**

Invariants (cont.)

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

- **One type of invariant is a** *loop invariant*. **This is a relationship among the variables in a loop that should always hold at the beginning and end of each iteration of the loop (though not necessary within the loop body).**

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

Hoare triples

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

$$P\{S\}Q$$

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

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

- **Examples:**

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

Proving loops: Partial correctness

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

$$P \; \{ \; \textbf{while } (b) \; \{S\} \; \} \; Q$$

- **A** *loop invariant* **for this loop is a condition** I **on the program variables (like** P **and** Q**) that is always true at the beginning 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 holds *if the loop terminates*.**

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

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

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

Loop proving example 1

$$x = n \wedge y = 1 \ \{$$

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

$$\} \ y = n!$$

- **Invariant I:**

- **I is an invariant:**

- **I holds at the start:**

- **Q holds at the end:**

- $T(x, y, n) \ =$

- $T(x, y, n) \ > \ 0$**:**

- $T(x, y, n) < \ T(x_0, y_0, n)$**:**

Loop proving example 2

$$a = lis \wedge b = 0 \{$$

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

$$\} \ b = \Sigma lis$$

- **Invariant I:**

- **I is an invariant:**

- **I holds at the start:**

- **Q holds at the end:**

- $T(a, b, lis) =$

- $T(a, b, lis) > 0$:

- $T(a, b, lis) < T(a_0, b_0, lis)$:

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(a0, b0)
```

- **Invariant I:**

- **I is an invariant:**

- **I holds at the start:**

- **Q holds at the end:**

- $T(a, b, x, y) =$

- $T(a, b, x, y) > 0$:

- $T(a, b, x, y) < T(a_0, b_0, x, y)$:

Loop proving example 4

$$x = 0 \,\wedge\, y = 0 \; \{$$

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

$$\} \; x = 1 + \cdots + n$$

- **Invariant I:**

- **I is an invariant:**

- **I holds at the start:**

- **Q holds at the end:**

- $T(x, y, n) \;=\;$

- $T(x, y, n) \;>\; 0$**:**

- $T(x, y, n) \;<\; T(x_0, y_0, n)$**:**

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 = z + 1; }
```

$$\} \ y = fib(n)$$

- **Invariant I:**

- **I is an invariant:**

- **I holds at the start:**

- **Q holds at the end:**

- $T(x, y, z, n) \ =$

- $T(x, y, z, n) \ > \ 0$**:**

- $T(x, y, z, n) < \ T(x_0, y_0, z_0, n)$**:**

Loop proving example 6

$$x = lst \ \wedge \ y = 0 \ \{$$
```
      while (x != []) { x = tl x; y = y + 1; }
```
$$\} \ y = length(lst)$$

- **Invariant I:**

- **I is an invariant:**

- **I holds at the start:**

- **Q holds at the end:**

- $T(x, y, lst) \ =$

- $T(x, y, lst) \ > \ 0$**:**

- $T(x, y, lst) < \ T(x_0, y_0, lst)$**:**

Loop proving example 7

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

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

$$\}\ y = reverse(lst)$$

- **Invariant I:**

- **I is an invariant:**

- **I holds at the start:**

- **Q holds at the end:**

- $T(x, y, lst) =$

- $T(x, y, lst) > 0$**:**

- $T(x, y, lst) < T(x_0, y_0, lst)$**:**

Hoare logic

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

(Assignment) $P[e/x] \{ x = e \} P$ (While) $P \{ \text{while } (b) \ S \} Q$
$$I \wedge b \{ S \} I$$
$$(\text{if } P \wedge b \supset I \text{ and } P \wedge \neg b \supset Q)$$

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

(Consequence) $P \{ S \} Q$
$$P' \{ S \} Q'$$
$$(\text{if } P \supset P' \text{ and } Q' \supset Q)$$

Example of a proof in Hoare's logic

(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**