University of Nevada, Las Vegas Computer Science 477/677

Loop Invariants

How can you see that a program is correct? For some simple programs, that is obvious. For others, not so much.

One technique to prove correctness is the use of *loop invariants*. A loop invariant is for a particular loop. For simple loops, the loop invariant is something that you understand unconciously while you're writing the program. Here's an example.

```
int total = 0;
for(int i = 1; i < n; i++)
 total = total + i*i;
```

The loop invariant is the English sentence, "total is the sum of the squares of the first i integers." It is easier to discuss loop invariants if we stick to while loops. So we rewrite the code:

```
int total = 0;
int i = 0;
    // HERE
while(i < N)
 {
    // HERE
  i++;
    // But not HERE
  total = total + i*i;
    // HERE
 }
    // HERE
cout << total << endl;
```

The loop invariant is, the statement, "total is the sum of the squares of the first i integers." The invariant is always a 0/1 statement, that is, a statement which is either true or false. Here are the rules for a loop invariant:

1. The invariant is true before the first iteration.
2. If the invariant is true at the beginning of an iteration, it is true at the end of that iteration.

It follows that the invariant must be true after the last iteration of the loop.

The goal of this program is to write the sum of the squares of the first N integers. I'm sure you agree that the program is obviously correct, but let's prove it anyway.

Condition 1 is clearly true, since both i and total are zero. We now prove condition 2:

Suppose total $= 1^2 + 2^2 + \cdots + i^2$ at the beginning of an iteration. The value of i changes during the loop, let the new value be $i' = i + 1$, and let total' be the new value of total. We need only show that total' =

$1^2 + 2^2 + \cdots + (i+1)^2$, because that is the loop invariant after i and total have been changed. We see that total' = total + $(i+1)^2 = 1^2 + 2^2 + \cdots + i^2 + (i+1)^2$, and so the loop invariant holds. At the end of the last iteration, i = N, and the loop invariant holds, which implies that total is the sum of the squares of the first N integers.

That was so easy it was trivial. Let's look at a harder example.

```
float multiply(float x, int b)
 {
  assert(b >= 0);
  float rslt = 0;
  float y = x;
  int a = b;
  while(a > 0)
   {
    if(a%2) rslt = rslt + y;
    y = 2*y;
    a = a/2;
   }
  return rslt;
 }
```

The loop invariant is the equation: $x * b = rslt + y * a$

Before the first iteration, the loop invariant is true since y = x, a = b, and rslt = 0. Can you show that Condition 2 holds for this loop? If so, after the last iteration, $x * b = rslt$ since $a = 0$. This gives a proof that the function returns the product of its two parameters.

I claim that *every* loop in any practical program has a loop invariant, although it's usually not written down. Can you find the loop invariant for the loop in the function *mystery* in the second homework?

Recurrence and Complexity

The Generalized Master Theorem We now solve the recurrence

$$F(n) = \sum_{i=1}^{n} \alpha_i F(\beta_i n) + n^\gamma$$

where the α_i, β_i, and γ are constants. With the restriction that $\alpha_i > 0$ and $0 < \beta_i < 1$ for each i. The master theorem is a special case of the generalized master theorem, where $n = 1$, $\alpha = A$, $\beta = 1/B$ and $\gamma = C$.

The rules are as folows.

1. Find the real number d such that $\sum_{i=1}^{n} \alpha_i \beta_i^d = 1$. If the value of d is not obvious, it can be approximated by binary search, as follows.
   - (a) Estimate the value of d.
   - (b) Compute $\sum_{i=1}^{n} \alpha_i \beta_i^d$.
   - (c) If that total is less than 1, pick a smaller d. If it is greater than 1, pick a larger d.
   - (d) Repeat.

   Then $F(n) = \Theta\left(\begin{cases} n^\gamma \text{ if } d < \gamma \\ n^\gamma \log n \text{ if } d = \gamma \\ n^d \text{ if } d > \gamma \end{cases}\right)$

Examples.

1. $F(n) = F(3n/5) + F(4n/5) + n$

   Here $\alpha_1 = \alpha_2 = 1$, $\beta_1 = 3/5$, $\beta_2 = 4/5$, and $\gamma = 1$. We can compute $d = 2$, since $(3/5)^2 + (4/5)^2 = 1$.

   Thus $F(n) = \Theta(n^2)$ (The third case.)

2. $F(n) = F(3n/5) + F(4n/5) + n^2$ α_i and β_i are the same as in the previous example, and $d = 2$. But $\gamma = 2 = d$, thus $F(n) = \Theta(n^2 \log n)$ (The second case.)

3. This recurrence comes up in analysis of the median of medians algorithm, which we will learn this semester:

   $T(n) \leq F(n/5) + F(7n/10) + \Theta(n)$.

   Then $\alpha_1 = \alpha_2 = 1$, $\beta_1 = 1/5$, $\beta_2 = 7/10$, and $\gamma = 1$. d is some hideous irrational number, but we don't need to know it. Since $\alpha_1 \beta_1^\gamma + \alpha_2 \beta_2^\gamma = 9/10$ which is less than 1, we know that $d < \gamma$. Hence (The first case) $T(n) = \Theta(n)$.

4. $F(n) = F(2n/3) + 5F(n/3) + n$

   $(2/3)^2 + 5(1/3)^2 = 1$, thus $d = 2$, and $F(n) = \Theta(n^2)$ (Third case.)