CS31 (Algorithms), Spring 2020 : Lecture 0 Supplement

Topic: Correctness of Programs with loops: Invariants

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors. Please discuss in Piazza/email errors to deeparnab@dartmouth.edu

---

In the lecture notes, I indicated we will not be focusing too much on proving correctness of algorithms. That ***does not***, in any way, minimize the importance of proving this. Rather, for most algorithms we design, correctness will direct our thought processes. This doesn't imply a formal proof, but it hopefully will give an idea of why it is true. At times, when this is not clear, we will indeed prove this. The rest of the times, to obtain a formal proof, one takes two broad approaches. If the algorithm is ***recursive***, then one uses *induction* to prove correctness. Here I will refer to these notes[1] to see how this is done.

All algorithms, however, are not necessarily written in a recursive fashion. For instance, the following algorithm (from the lecture notes), is a for-loop for finding the maximum. Now, most of you, can probably "see" why this finds the maximum index. But how does one formally prove it?

```
procedure MAX($A[1:n]$):
    ▷ Returns $j$ where $A[j] \geq A[i]$ for all $1 \leq i \leq n$.
    rmax ← 1.  ▷ Initialize running-max to 1
    for $2 \leq j \leq n$ do:
        if $A[j] > A[\text{rmax}]$ then:  ▷ If $A[j]$ bigger than running-max, swap.
            rmax ← $j$
    return rmax.
```

The main idea here is to introduce **loop invariants**. These are *assertions* which we will make for every loop such that after the loop is over, some other useful assertion is true. For the above algorithm, here I am writing it below with loop invariants.

```
procedure MAX($A[1:n]$):
    ▷ Returns $j$ where $A[j] \geq A[i]$ for all $1 \leq i \leq n$.
    rmax ← 1.  ▷ Initialize running-max to 1
    for $2 \leq j \leq n$ do:
        ▷ Pre: $A[\text{rmax}]$ is the maximum element in $A[1:j-1]$
        if $A[j] > A[\text{rmax}]$ then:  ▷ If $A[j]$ bigger than running-max, swap.
            rmax ← $j$
        ▷ Post: $A[\text{rmax}]$ is the maximum element in $A[1:j]$
    return rmax.
```

The *Pre* invariant asserts that as soon as one enters the jth loop (the for-loop with index j), $A[\text{rmax}]$ stores a maximum element of $A[1:j-1]$. The *Post* invariant asserts that after running the jth loop, $A[\text{rmax}]$ stores a maximum element of $A[1:j]$. We begin with a crucial observation which is common to *Pre* and *Post* conditions for any for-loop correctness.

Observation 1. For any j, the *Post* condition for loop j is the *Pre* condition for loop $(j+1)$.

[1] https://www.cs.dartmouth.edu/~deepc/Courses/W20/lecs/lec8.pdf

We now make a few claims to prove the algorithm's correctness.

Claim 1. If the *Pre* and *Post* invariants hold for all $2 \le j \le n$, then on termination $A[\mathsf{rmax}]$ is a maximum element of $A[1:n]$.

Proof. Indeed, at the end of the last for-loop, $j = n$, and thus $A[\mathsf{rmax}]$ will contain the maximum element of $A[1:n]$. And after the for-loop, rmax is not modified. □

So, we are now left with proving the *Pre* and *Post* conditions hold for all $2 \le j \le n$. This is done via (no surprises) induction.

Claim 2 (Base Case.)**.** At the beginning of the first for-loop (the case $j = 2$), *Pre* condition holds.

Proof. At the beginning of the first for-loop, $\mathsf{rmax} = 1$ (this is set in Line 3). We therefore need to show $A[1]$ is a maximum element of $A[1:1]$. Indeed, there is only one element in $A[1:1]$. Nothing to show. □

Claim 3 (Inductive Case.)**.** Suppose the *Pre* condition is true *before* loop j, for any $2 \le j \le n$. Then the *Post* condition is true *after* loop j.

Proof. Let $r_1 = \mathsf{rmax}$ be the value of rmax at the beginning of loop j. Since *Pre* condition is true, we have $A[r_1] \ge A[i]$ for all $1 \le i \le j - 1$. There are now only two cases possible:

Case 1. $A[r_1] \ge A[j]$. In this case, $A[r_1] \ge A[i]$ for all $1 \le i \le j$ as well. That is, $A[r_1]$ is a maximum element of $A[1:j]$. Indeed, in this case rmax doesn't change. Thus, *Post* is true after loop j.

Case 2. $A[j] > A[r_1]$. In this case, $A[j] > A[r_1] \ge A[i]$ for all $1 \le i \le j - 1$. That is, $A[j]$ is the maximum (note: this is actually at this point the unique maximum...but that is beside the point) of the array $A[1:j]$. Indeed, in this case rmax is changed to j. And thus, even in this case *Post* is true after loop j. □

The above three claims gives us the following theorem which formally asserts the correctness of MAX.

Theorem 1. The algorithm MAX returns index of a maximum element for any array $A[1:n]$.

Remark:

- *As you can see, writing the formal proof can be a bit tedious. Important, but a bit tedious.*
- *Coming up with a loop invariant takes some practice (and, indeed, due to our decision of not focusing on correctness, you'll not get enough practice). But hopefully the above example gave a flavor.*

Let us see another example: this time with a *while* loop. The algorithm below can be obviously written as a for-loop, but I am writing as a while loop to show flavors of correctness proofs. Consider the following program which takes power of a number. I am including the invariants as well.

```
 1: procedure Pow($a, b$):  ▷ Assumes $b$ is a non-negative integer
 2:     ▷ Returns $a^b$.
 3:     res ← 1.  ▷ Initialize result 1 (which is $a^0$)
 4:     $j$ ← 0.
 5:     while $j < b$ do:
 6:         ▷ Pre: res = $a^j$.
 7:         res ← res × $a$.
 8:         $j$ ← $j + 1$.
 9:         ▷ Post: res = $a^j$.
10:     ▷ Assert: $j = b$
11:     return res.
```

For a while loop, there is an extra assertion right *after* the while loop. It states that $j = b$ there. This also needs a proof.

Claim 4. After the end of the while loop, $j = b$.

Proof. j is initialized to 0 and is always incremented by 1 (if ever). Therefore j is always an integer. By the condition of the while loop, it exits when $j \geq b$. Let j' be the value of j in the last while loop. Then $j' < b$ (for the while loop executed), and since j' *and* b are integers, $j' \leq b - 1$. And $j = j' + 1$ (Line 8). That is, $j \leq b$. But $J \geq b$. Thus, $j = b$. □

Remark: *Do you see why the above claim is* ***not*** *true if b were not an integer. How would you modify the claim? Also, what happens if b is negative?*

Thus, if we can show that *Pre* and *Post* condition hold in every while loop, then we are done. I will leave this as an exercise. ✍

Exercise: *Prove the* Pre *and* Post *conditions. Actually write them down for the fullest practice.*

Theorem 2. The Algorithm *Pow*(a, b) returns a^b when b is a non-negative integer.