Iterative Algorithms:

We prove partial correctness for iterative algorithms by finding a loop invariant and proving that loop invariant using induction on the number of iterations. The proof of termination for Iterative algorithms involves associating a decreasing sequence of natural numbers to the iteration number. We can then conclude the termination from the following theorem.

Theorem 1. *Every decreasing sequence of natural numbers is finite.*

Exercise. *Prove the theorem using well-ordering principle.*

Example 1. *Algorithm to compute x^y.*

```
1: function POW(x, y)
2:     prod = 1
3:     p = 0
4:     while p < y do
5:         prod = prod × x
6:         p = p + 1
7:     end while
8:     return prod
9: end function
```

Precondition: $x \in \mathbb{R}$ and $y \in \mathbb{N}$.

Postcondition: Returns x^y. Moreover, we assume $0^0 = 1$.

Ignoring the first few assignments in the algorithm, the majority part of this algorithm implements a loop. Hence, the correctness of the algorithm is significantly dependent on the correctness of the loop. Let's see what this loop is doing.

iter num	0	1	2	3	4	$\cdots$
p	0	1	2	3	4	$\cdots$
prod	1	x^1	x^2	x^3	x^4	$\cdots$

Table 1: Values of interest as the function of iteration number.

The table suggests that we can prove "Pow" returns x^y, if we can prove that, for $x \in \mathbb{R}$ and $y \in \mathbb{N}$, starting from $prod = 1$ (loop precondition) the loop terminates with $prod = x^y$ (loop postcondition).

Loop invariant: Many statements can be used as loop invariants. For example, $prod_i \geq 0$, $y \geq 0$, $p_i = i$, etc. where v_i denotes the value of v at the end of iteration i. But we have to pick the one that help us prove loop's postcondition ($prod = x^y$). The values in Table 1 suggest that $prod = x^p$ can be a good candidate. Because by the end of loop, $p = y$ and hence $prod = x^y$. Notice, the last conclusion has an implicit assumption, i.e., $p \leq y$. Therefore, the complete loop invariant should be stated as: $p \leq y$ and $prod = x^p$. Now, let us prove this invariant.

Proof. We prove the loop invariant by induction on the iteration number. In what follows, the subscripts denote the iteration number of the loop.

Base Case: Before the loop starts, $p_0 = 0 \leq y \in \mathbb{N}$ and $prod_0 = 1 = x^0 = x^{p_0}$, so loop invariant holds.

Induction step: Suppose $k \geq 0$ and loop invariant holds for k^{th} iteration of loop, i.e., $p_k \leq y$ and $prod_k = x^{p_k}$ (HI). We will prove loop invariant holds after $k + 1^{th}$ iteration. Consider two cases.

Case 1: There is no iteration number $k + 1$. Then loop invariant for iteration $k + 1$ is equivalent to loop invariant for iteration k. Since $prod_{k+1} = prod_k$ and $p_{k+1} = p_k$ then loop invariant holds by IH.

Case 2: There is an iteration number $k+1$. Then, $p_k < y$ because the loop condition was true and from algorithm, $prod_{k+1} = prod_k \times x$ and $p_{k+1} = p_k + 1$. Hence, $p_{k+1} = p_k + 1 \leq y$, and

$$\begin{aligned} prod_{k+1} &= prod_k \times x \\ &= x^{p_k} \times x \qquad \text{(by IH)} \\ &= x^{p_k+1} \\ &= x^{p_{k+1}} \end{aligned}$$

Hence, loop invariant holds at the end of iteration $k+1$. Therefore, by induction, loop invariant holds after each iteration. □

When loop terminates, $p \geq y$ (from the loop test) and $p \leq y$ and $prod = x^p$ (from loop invariant) so $p = y$ and $prod = x^p = x^y$. This proves loop postcondition. Now, because the function returns *prod* after the loop, the function returns x^y which proves the partially correctness.

Termination: As discussed earlier, we have to find a quantity that is a decreasing function of iteration number. We have three major variables p, *prod* and y. Table 1 suggests that p and *prod* are increasing functions of iteration number while y remains constant. But as p gets larger, closer to y, $d = y - p$ becomes smaller. By looop inveraince, $d_k = y - p_k \geq 0$. Moreover y and p are natural numbers and hence $d \in \mathbb{N}$. If iteration $k+1$ is performed,

$$d_{k+1} = y - p_{k+1} = y - (p_k + 1) = y - p_k - 1 = d_k - 1 < d_k$$

Hence, d is a decreasing sequence of natural numbers and by Theorem 1 it is finite. In other words, the loop terminates and hence "Pow" terminates.

Example 2 (MergeSort)**.** *In previous lecture, we accepted, without proof, that assuming $B[s..m]$ is sorted and $B[m+1..f]$ is also sorted, the following code merges these two sublists into $A[s..f]$ where A is sorted in a non-decreasing order.*

```
1:  c = s
2:  d = m + 1
3:  for i = s, ··· , f do
4:      if d > f or (c ≤ m and B[c] < B[d]) then
5:          A[i] = B[c]
6:          c = c + 1
7:      else                                   ▷ d ≤ f and (c > m or B[c] ≥ B[d])
8:          A[i] = B[d]
9:          d = d + 1
10:     end if
11: end for
```

Let's prove this now.

The first two lines are just initializations. So we try to prove the correctnesss of the for loop.

Precondition: $B[s..m]$ is sorted in non-decreasing order. $B[m+1..f]$ is sorted in non-decreasing order. Moreover, $s \leq f$.

Postcondition: $A[s..f]$ contains element from $B[s..m]$ and $B[m+1..f]$ that are sorted in no-decreasing order. Moreover $s \leq f$.

Loop invariant: Looking at the body of the for loop and our postcondition, we can define the loop invariant as: $A[s..i-1]$ contains elements from $B[s..c-1]$ and $B[m+1..d-1]$, sorted in non-decreasing order where $i = c + d - m - 1$. Moreover, the elements in $A[s..i-1]$ are less than or equal to elements in both $B[d..f]$ and $B[c..m]$.

Partial correctness: We prove partial correctness by induction. In what follows, the subscripts denote the iteration number of the loop.

Proof. Base Case: Before the loop starts, $c_0 = s$, $d_0 = m + 1$ and $i_0 = s$. Hence $A[s..i_0 - 1]$, $B[s..c_0 - 1$ and $B[m+1..d_0-1$ are empty which means that loop invariant is vacuously true. Moreover $i_0 = s = s+m+1-m-1 = c_0 + d_0 - m - 1$.

Induction step: Suppose $k \geq 0$ and loop invariant holds for k^{th} iteration of loop, i.e., $A[s..i_k - 1]$ contains elements from $B[s..c_k - 1$ and $B[m + 1..d_k - 1]$ sorted in non-decreasing order. Moreover the elements of $A[s..i_k - 1]$ are no greater than elements in both $B[d_k..f]$ and $B[c_k..m]$ (HI). We will prove loop invariant holds after $k+1^{th}$ iteration. Consider two cases.

Case 1: There is no iteration number $k + 1$. Then loop invariant for iteration $k + 1$ is equivalent to loop invariant for iteration k. Since $i_{k+1} = i_k$, $c_{k+1} = c_k$ and $d_{k+1} = d_k$ then loop invariant holds by IH.

Case 2: There is an iteration number $k+1$. Then $i_{k+1} = i_k + 1$ and hence $i_{k+1} - 1 = i_k$. Now, two things can happen:

sub-case 1: $B[c_k] < B[d_k]$. Then $A[i_k] = B[c_k]$, $c_{k+1} = c_k + 1$ and we have

$$\begin{aligned} & B[s..c_k - 1] && \subseteq A[s..i_k - 1] && \text{(By IH)} \\ \Rightarrow\ & B[s..c_k] && \subseteq A[s..i_k - 1] \cup B[c_k] \\ & && = A[s..i_k - 1] \cup A[i_k] \\ & && = A[s..i_k] \\ \Rightarrow\ & B[s..c_{k+1} - 1] && \subseteq A[s..i_{k+1} - 1] \end{aligned}$$

We can also conclude that $d_{k+1} = d_k$ and hence

$$i_{k+1} = i_k + 1 \underbrace{=}_{\text{by IH}} c_k + d_k - m - 1 + 1 = c_{k+1} + d_k - m - 1 = c_{k+1} + d_{k+1} - m - 1$$

We can also show that

$$\begin{aligned} B[m + 1..d_k - 1] &\subseteq A[s..i_k - 1] && \text{(By IH)} \\ &\subseteq A[s..i_k - 1] \cup A[i_k] \\ &= A[s..i_{k+1} - 1] \end{aligned}$$

Moreover, by IH, $A[s..i_k - 1] \leq B[c_k]$. Since B is sorted in non-decreasing order then $B[c_k] \leq B[c_k + 1..m] = B[c_{k+1}..m]$. Hence by choice of $A[i_k]$, $A[s..i_{k+1} - 1] = A[s..i_k] \leq B[c_k] \leq B[c_{k+1}..m]$. Moreover, $A[s..i_{k+1} - 1] \leq B[c_k] \leq B[d_k] \leq B[d_k..f]$.

sub-case 2: $B[d_k] < B[c_k]$. Then $A[i_k] = B[d_k]$, $d_{k+1} = d_k + 1$ and $c_{k+1} = c_k$. The rest of the proof that concludes loop invariant also holds in this case is similar to sub-case 1 with c_k is replaced with d_k and vice versa (do it as an exercise). Therefore, by induction loop invariant holds at each iteration. □

Upon termination $i_k = f + 1$ which can only happen if $c_k = m + 1$ and $d_k = f + 1$. Therefore, by loop invariant $A[s..i_k - 1] = A[s..f]$ contains elements of $B[s..m]$ and $B[m + 1..f]$ sorted in non-decreasing order which proves the partial correctness.

Termination: Because i_k increase at each iteration, then $f - i_k$ is a decreasing sequence of iteration number and hence we can conlude that loop terminates in finite steps.

Remark. *If a loop consists of multiple loops, we can follow these steps:*

1. Consecutive loops (not nested): *Simply prove separate loop invariants one after the other, where proof of second one can rely on first one being true at the end of the first loop.*

2. Nested loops: *Work inside-out. Prove termination and loop invariant for inside loop first, for arbitrary value of the outside loop variables, and use that to prove termination and invariant for outside loop.*

Remark. *Remember that a counting loop (for $x = a, \cdots, b$) is equivalent to*

$x = a$
while $x \leq b$ ***do***
    $\cdots$
    $x = x + 1$
end while

This affects how we write invariants and prove termination. In particular, value of x at end will be $b + 1$ (not b).

In practice, loop invariant is part of the code **design**, i.e., loop invariant is used to help us write the loop. Now, let's look at how to use loop invariants to "design" correct algorithms.

Example 3 (Iterative Binary Search)**.** We start with a sorted list A and a value x which is comparable with $A[1..length(A)]$ (precondition). Upon termination the index $1 \leq p \leq length(A)$ is selected such that $A[1..p-1] < x \leq A[p..n]$ where $n = length(A)$ (postcondition).

Within loop, we want to maintain a search range $[s..f]$ such that $A[1..s-1] < x \leq A[f+1..n]$. Comparing this condition with our postconditions, it seems that this condition can be our loop invariant. Now, let's start writing our loop.

To satisfy the precondition, we can start: $s = 1, f = n$. It only make sense to continue our search as long as range $[s..f]$ is not empty, i.e., $s \leq f$. So

while $s \leq f$ **do**
    $\cdots$
end while

To terminate the loop, we have to update s or f such that $s > f$ at one point. Moreover, the name of algorithm suggests that the body should make a binary decision. We choose to compare middel element with x:

$m = (s + f)/2$ ▷ integer division
if $A[m] < x$ **then**
    $s = m + 1$
else
    $f = m - 1$
end if

The condition of the if statement guarantees that loop invariant holds. Moreover, our update rule guarantees that $s \leq f$ at each iteration which we can add to the loop invariant. The aforementioned logic cuts the search range almost in half in each iteration which can be used to ensure that termination is also guaranteed. Now, we can choose $p = s = f + 1$ upon termination of loop to guarantee that postcondition of algorithm holds. Putting it altogether, we have:

```
1: function IterBSearch(A, x)
2:     s = 1
3:     f = length(A)
4:     while s ≤ f do
5:         m = (s + f)/2                ▷ integer division
6:         if A[m] < x then
7:             s = m + 1
8:         else
9:             f = m − 1
10:        end if
11:    end while
12:    p = s return p
13: end function
```