

CS 313, Loop Invariant Examples

- Example 1

Our goal is to prove the correctness of the ArrayFind algorithm¹:

Algorithm arrayFind(x,A):

Input: An element x and an n -element array, A .

Output: The index i such that $x = A[i]$ or -1 if no element of A is equal to x .

```
i = 0
while i < n do
  if x == A[i] then
    return i
  else
    i = i + 1
return -1
```

This example proof is a little more verbose than necessary, but we will include extra detail to make it as clear as possible.

We will prove that, if the loop terminates (without returning early), then x is not in A .² The loop condition, γ , is: “ $i < n$ ”. We define the loop invariant, α , as: “ $0 \leq i \leq n$ and x is not stored in $A[0..(i-1)]$ ”, where $A[0..(i-1)]$ refers to the values stored in array A from indices 0 through $i-1$, inclusive. We now use this loop invariant to prove the correctness of the loop by showing that it satisfies all three properties:

- (i) Initialization: From the assignment statement before the start of the loop, $i = 0$. Clearly, $0 \leq 0 \leq n$. Because $i = 0$, $A[0..i-1]$ is an array of zero elements and the second half of the invariant is trivially true.
- (ii) Maintenance: Let i refer to the value of variable i at the beginning of the loop and i' to its value at the end of the loop. From the execution of the loop body, $i' = i + 1$. Since $0 \leq i$ (from α) and $i < n$ (from γ), $0 \leq i + 1 \leq n$, so $0 \leq i' \leq n$. Furthermore, from α we know that x is not contained in $A[0..(i-1)]$. From the loop body, if the loop continues then x is not in $A[i]$ either. Therefore, x is not contained in $A[0..i]$. From the definition of i' , we can conclude that x is not contained in $A[0..(i'-1)]$ and the invariant remains true with the new value of i .
- (iii) Termination: Since $0 \leq i \leq n$ (from α) and $i \not< n$ (from $\neg\gamma$), $i = n$. Substituting $i = n$ back into the loop invariant, we can conclude that x is not stored in $A[0..(n-1)]$, which is the entire array.

- Example 2

Question: Use a loop invariant to prove the correctness of the following algorithm for finding the maximum element on an array:

Algorithm arrayMax(A, n)

Input array A of n integers

¹This example is taken from Chapter 1 of the data structures textbook by Goodrich and Tamassia, which used to be the standard textbook for CIS 313.

²If the algorithm *does* return early, then from the preceding if statement we know that it must have found x and is returning the correct index. Therefore, whenever the algorithm returns early it returns the correct result.

Output maximum element A

```
currentMax = A[0]
i = 0
while i < n-1 do
    if A[i+1] > currentMax
        currentMax = A[i+1]
    i = i + 1
```

Answer: We use the loop invariant: “ $0 \leq i \leq n - 1$ and currentMax is the maximum value stored in $A[0..i]$ ”, where $A[0..i]$ refers to the values stored in array A from indices 0 through i , inclusive. We now use this loop invariant to prove the correctness of the loop by showing that it satisfies all three properties:

- (i) From the initialization statements, $i = 0$ and $\text{currentMax} = A[0]$. Because $i = 0$, $A[0..i] = A[0]$, so the maximum value stored in $A[0..i]$ is simply $A[0]$. From the precondition, currentMax already equals this value, satisfying the loop invariant. Also, since $i = 0$, clearly $0 \leq i \leq n - 1$.
- (ii) Let the primed variables i' and $\text{currentMax}'$ refer to the values of i and currentMax at the end of the loop, and the unprimed variables i and currentMax to their values at the beginning of the loop. From the execution of the loop body, $i' = i + 1$. Since $0 \leq i$ (from α) and $i < n - 1$ (from γ), $0 \leq i + 1 \leq n$, so $0 \leq i' \leq n$. Furthermore,

$$\begin{aligned} \text{currentMax}' &= \max(\text{currentMax}, A[i + 1]) && \text{Execution of the loop} \\ &= \max(\max(A[0..i]), A[i + 1]) && \text{Follows from } \alpha \\ &= \max(A[0..i + 1]) && \text{max is associative} \\ &= \max(A[0..i']). && \text{Substitution} \end{aligned}$$

Since the new values $\text{currentMax}'$ and i' satisfy the loop invariant, α remains true at the end of the loop.

- (iii) Since $0 \leq i \leq n - 1$ (from α) and $i \neq n - 1$ (from $\neg\gamma$), $i = n - 1$. Therefore, currentMax is the maximum value stored in $A[0..n - 1]$, which is the maximum value in A .