# *Finding Invariants*

## *Part 1: Adding Parameters by Replacing Constants by Variables*

## *CS 536: Science of Programming, Fall 2022*

### *A. Why*

- It is easier to write good programs and check them for defects than to write bad programs and then debug them.
- The hardest part of programming is finding good loop invariants.
- There are heuristics for finding them but no algorithms that work in all cases.

### *B. Objectives*

At the end of this activity assignment you should

- Be able to how to generate possible invariants using "replace a constant by a variable" or more generally "add a parameter".

### *C. Problems*

1. What are the constants in the postcondition $x = max( b[0], b[1], ..., b[n-1] )$? Using the technique "replace a constant by a variable," list the possible invariants for this postcondition. Also, what would the loop tests be? (Assume $n-1$ is a constant.) Hint: Think of $max( b[0], b[1], ..., b[n-1] )$ as standing for a function call on $b$ and two indexes.

2. Repeat, on the postcondition $x = n!$, where $n!$ is short for a function call $product(1, n)$.

3. Repeat, on the postcondition $\forall i \,.\, 0 \le i < n \rightarrow b[i] = 3$.

4. Repeat, on the postcondition $\forall i \,.\, \forall j \,.\, 0 \le i < m \wedge m \le j < n \rightarrow b[i] < b[j]$, which says that every value in $b[0 \dots m-1]$ is $<$ every value in $b[m \dots n-1]$.

 

## *Solution to Practice 19 (Finding Invariants; Examples)*

1. Certainly 0 is a constant; if we replace it by a variable i, we get

   {***inv*** x = max(b[i], ..., b[n-1]) ∧ 0 ≤ i ≤ n-1} ***while*** i ≠ 0 ***do*** …

   As a constant, n-1 seems better than just n or 1 by themselves:

   {***inv*** x = max(b[0], ..., b[j]) ∧ 0 ≤ j ≤ n-1} ***while*** j ≠ n-1 ***do*** …

   If you want to treat just n as a constant and replace it by a variable j, we get

   {***inv*** x = max(b[0], ..., b[j-1]) ∧ 1 ≤ j ≤ n} ***while*** j ≠ n ***do*** …

   Similarly, if you want replace just the 1 in n-1 by with j, we get

   {***inv*** x = max(b[0], ..., b[n-j]) ∧ 1 ≤ j ≤ n} ***while*** j ≠ 1 ***do*** …

2. We can replace n by a variable and get

   ***inv*** x = i! ∧ 1 ≤ i ≤ n} ***while*** i ≠ n ***do*** …

   We can replace 1 and get

   {***inv*** x = j*(j+1)*…*n ∧ 1 ≤ j ≤ n} ***while*** j ≠ 1 ***do*** …

3. For ∀ i . 0 ≤ i < n → b[i] = 3 as the postcondition, we can replace 0 or n or 3.

   Replace 0 by k:

   {***inv*** 0 ≤ k ≤ n-1 ∧ ∀ i . k ≤ i < n → b[i] = 3} ***while*** k ≠ 0 ***do*** …

   Replace n by k

   {***inv*** 0 ≤ k ≤ n ∧ ∀ i . 0 ≤ i < k → b[i] = 3} ***while*** k ≠ n ***do*** …

   Replace 3 by k (this doesn't look useful*)*

   {***inv*** ∀ i . 0 ≤ i < n → b[i] = k} ***while*** k ≠ 3 ***do*** …

4. For ∀ i . ∀ j .0 ≤ i < m ∧ m ≤ j < n → b[i] < b[j], we have constants 0, n, the two occurrences of m.

   Replace 0 by k:

   {***inv*** 0 ≤ k < m ∧ ∀ i . ∀ j . k ≤ i < m ∧ m ≤ j < n → b[i] < b[j]}
   ***while*** k ≠ 0

   Replace left m by k:

   {***inv*** 0 ≤ k < m ∧ ∀ i . ∀ j . 0 ≤ i < k ∧ m ≤ j < n → b[i] < b[j]}
   ***while*** k ≠ m

   Replace right m by k:

   {***inv*** m ≤ k ≤ n ∧ ∀ i . ∀ j . 0 ≤ i < m ∧ k ≤ j < n → b[i] < b[j]}
   ***while*** k ≠ m

   Replace n by k:

   {***inv*** m ≤ k ≤ n ∧ ∀ i . ∀ j . 0 ≤ i < m ∧ m ≤ j < k → b[i] < b[j]}
   ***while*** k ≠ n

   You could argue that the ranges for k could be 0 ≤ k < n, 0 ≤ k < n, 0 ≤ k ≤ n, and 0 ≤ k ≤ n for the four cases above; it depends on knowing more about the context of the problem.

 