Loop Invariant Extra Exercises

Jim Glenn

January 21, 2020

Problem 1 Prove with a loop invariant that the the following function correctly sums the elements in the array passed to it.

```
def sum(A, n):
    i = 0
    sum = 0
    while i < n:
        sum = sum + A[i]
        i = i + 1
    return sum
```

Preconditions:

- A is an array of real numbers (indexed starting from 0),
- n is the size of the array A (so n is an integer such that $n \geq 0$)

Postcondition: $sum = \sum_{k=0}^{n-1} A[k]$

Solution We use the following invariant:

(a) $sum = \sum_{k=0}^{i-1} A[k]$ (that is, sum is the sum of the first i elements of A)

(b) i is an integer such that $0 \leq i \leq n$

Basis/Initialization:

(a) sum and i are initialized to 0 and with $i = 0$, $\sum_{k=0}^{i-1} A[k] = 0$ because the sum is empty, so $sum = \sum_{k=0}^{i-1} A[k]$ since both are equal to 0.

(b) i is initially 0, which is an integer, and $0 \leq 0 = i$. Also, $i = 0 \leq n$ by the precondition on n. Putting those together gives $0 \leq i \leq n$.

Induction: Suppose the invariant is true before one iteration of the loop and the guard $i < n$ is true.

(a) Since the invariant is true before the loop, we have $sum_{old} = \sum_{k=0}^{i_{old}-1} A[k]$. The first statement inside the loop sets $sum_{new} = sum_{old} + A[i_{old}] = \sum_{k=0}^{i_{old}-1} A[k] + A[i_{old}] = \sum_{k=0}^{i_{old}} A[k]$. The second statement sets $i_{new} = i_{old} + 1$. Substituting that into the result of the first statement gives $sum_{new} = \sum_{k=0}^{i_{new}-1} A[k]$, which is part (a) of the invariant with the new values of the variables.

(b) By the invariant, i_{old} is an integer such that $0 \leq i_{old} \leq n$. Since the guard is true, $i_{old} < n$, which is equivalent to $i_{old} \leq n-1$ since i and n are integers. So $0 \leq i_{old} \leq n-1$ and $0 \leq i_{old} < i_{old}+1 \leq n$. Since the second statement in the loop sets $i_{new} = i_{old}+1$, we can rewrite that as $0 \leq i_{new} \leq n$, which is part (b) of the invariant with the new values of the variables.

Termination: i increases each time through the loop, so eventually $i \geq n$ and the guard becomes false to terminate the loop.

Postcondition: The falsity of the guard when the loop terminates means that $i \geq n$. Part (b) of the invariant says $i \leq n$. The only way $i \geq n$ and $i \leq n$ can both be true is if $i = n$. Plug that into part (a) of the invariant yields $sum = \sum_{k=0}^{n-1} A[k]$, which is the required postcondition for the function.

Problem 2 Prove that insertion sort is correct, using the invariant from class.

```
InsertionSort(A, n)
  i = 1
  while i < n
    insert A[i] into correct location among A[0], ..., A[i-1]
    i = i + 1
```

The "insert..." line is either an inner loop or, as we will treat it, a call to a function `insert(A, i)` that, given an array with the first i elements in sorted order, modifies A so that its first $i+1$ elements are unchanged but are reordered to be sorted, and does not change any of the other elements in A (so `insert` has preconditions 1) A is a non-empty array of real numbers, 2) $0 \leq i < len(A)$, and 3) $A[0] \leq \cdots \leq A[i-1]$; and postconditions 1) $A[0] \leq \cdots \leq A[i]$, 2) those 1st $i+1$ elements are unchanged but may be reordered, and 3) the elements after them are unchanged). (Fun exercise: write `insert` and prove that it is correct using a loop invariant. This is how we tackle nested loops: treat the innermost loop as a separate function, prove that it does what it is supposed to do, and then work on the next outer loop.)

Preconditions:

- A is a non-empty array of real numbers
- n is the size of A (so n is an integer such that $n \geq 1$)

Postcondition: $A[0] \leq \cdots \leq A[n-1]$ and A has the same elements as before (but possibly in a different order).

Invariant:

(a) $A[0] \leq A[1] \leq \cdots \leq A[i-1]$

(b) $A[0], \ldots, A[i-1]$, are the original first i elements of A, possibly in a different order

(c) $A[i], \ldots, A[n]$ contain their original values

(d) i is an integer such that $1 \leq i \leq n$

Solution Basis:

(a) i is initialized to 1, and when $i = 1$ part (a) is vacuously true

(b) i is initialized to 1 and there are no assignments to A before the loop, so $A[0]$ has its original value, which is part (b) of the invariant when $i = 1$.

(c) There are no assignments to A before the loop, so $A[1], \ldots, A[n-1]$ all have their original values, which is part (c) of the invariant when $i = 1$.

(d) i is initialized to 1, which is an integer. The precondition on n yields $n \geq 1$. So $i = 1 \leq n$. Relaxing $1 = i$ to $1 \leq i$ and combining with $i \leq n$ yields $1 \leq i \leq n$.

Induction: Suppose the invariant is true before an iteration of the loop and that the guard $i < n$ is also true.

(a) By part (a) of the invariant, $A[0]_{old} \leq \cdots \leq A[i_{old} - 1]_{old}$, and by part (d) and the guard, $0 \leq i_{old} < n$. Those are the preconditions for the call `insert(A, i)`, so after that call we have $A[0]_{new} \leq \cdots \leq A[i_{old}]_{new}$ by the postconditions of `insert`. By the last statement in the loop, we have $i_{new} = i_{old} + 1$, so we can rewrite the previous results as $A[0]_{new} \leq \cdots \leq A[i_{new} - 1]_{new}$, which is part (a) of the invariant using the new values of the variables.

(b) By part (b) of the invariant, $A[0]_{old}, \ldots, A[i_{old}-1]_{old}$ are the original first i_{old} values from A in some order. From part (c), $A[i_{old}]_{old}$ is its original value. Now the postcondition of `insert` says those elements are reordered but not changed, so the new values are the original first $i_{old} + 1 = i_{new}$ values from A in some order, which is part (b) of the invariant with the new values of the variables.

(c) By part (c) of the invariant, $A[i_{old}+1], \ldots, A[n]$ are their original values. There are no assignments to them in the loop, and the postcondition of `insert` guarantees that they are not changed. So they still hold their original values. Rewriting using $i_{new} = i_{old}+1$ yields $A[i_{new}], \ldots, A[n]$ have their original values, which is part (c) of the invariant using the new values of the variables.

(d) $0 \leq i_{old} \leq n$ by part (d) of the invariant. $i_{old} < n$ since the guard is true and hence $i_{old} \leq n-1$ since i and n are integers (by the part (d) of the invariant and the precondition on n respectively). Therefore, $0 \leq i_{old} < i_{old}+1 \leq n$. Since $i_{new} = i_{old}+1$, we can rewrite that as $0 \leq i_{new} \leq n$, which is part (d) of the invariant with the new values of the variables.

Termination: i increases each time through the loop, so eventually we have $i \geq n$, which breaks the loop.

Postconditions: When the loop terminates, we have $i \geq n$ from the guard being false, and $i \leq n$ from part (d) of the invariant, so $i = n$ at termination. Substituting n for i into parts (a) and (b) of the invariant then yields $A[0] \leq \cdots \leq A[n-1]$ and $A[0], \ldots, A[n-1]$ are their original values, reordered, which are the postconditions for sorting A.

Now that we've done some work with loop invariants including conditions on the loop counters, let's cut that tedious part out. If there is a variable x that is initialized to $x = c_1$ before the loop, the guard on the loop is $x < c_2$ for some $c_2 >= c_1$, c_1 and c_2 are integers, and the only assignment to x inside the loop is $x = x + 1$ (which is not in any other inner loop or conditional), then we may conclude that x is always an integer such that $c_1 \leq x \leq c_2$, and that the loop terminates when $x = c_2$. (Corollary: under the same conditions, but with guard $P \wedge x < c_2$, then the conditions on x still hold, but at the termination of the loop all we know is $\neg P \vee x = c_2$).