Note on Loop Invariants

Dr Christian Konrad

1 Why Loop Invariants

In this section, we will illustrate the purpose of loop invariants with an example. Consider the following algorithm:

Algorithm 1 Computing the sum of the elements of an array

```
Require: Array $A$ of length $n$
1: $S \leftarrow A[0]$
2: for $i \leftarrow 1 \dots n-1$ do
3:     $S \leftarrow S + A[i]$
4: end for
5: return $S$
```

This algorithm computes the sum of the elements of A. Let us see how this is done: S is initialized with $A[0]$. Then, we iterate through the array from positions $i = 1$ to $i = n - 1$ and add the current element $A[i]$ to S. This clearly computes the sum of all elements in A.

A loop invariant allows us to obtain a better understanding for why the algorithm does what it is supposed to do. Concerning the previous algorithm, the following loop invariant seems to capture the essence of the algorithm:

At the beginning of iteration i, the property $S = \sum_{j=0}^{i-1} A[j]$ holds

Observe that this property tells us much more about the algorithm than just the fact that it computes the sum. It tells us that at the beginning of iteration i, S is the sum of the prefix $A[0, \dots, i-1]$. After the last iteration $i = n - 1$, which coincides with the state of S before the nth iteration (which is never executed), the loop invariant tells us that $S = \sum_{j=0}^{n-1} A[j]$, i.e., S indeed is the sum of all elements of A.

In general, there are two tasks involved when analyzing algorithms with loop invariants. First, we need to come up with a suitable loop invariant that captures the behavior of the algorithm, and second, we need to prove that the loop invariant actually holds.

Identifying a suitable loop invariant is not always easy and requires a good understanding of the algorithm and some experience. Proving that a loop invariant is correct, however, can be done most of the time by following a recipe. This works as follows (and is similar to a proof by induction):

1. *Initialization:* First, show that the loop invariant holds at the beginning of the first iteration of the loop.

2. *Maintenance:* Suppose that the loop invariant holds at the beginning of iteration i. Prove that it also holds at the beginning of iteration $i + 1$.

3. *Termination:* Argue what the loop invariant implies when the last iteration of the loop terminates.

Let us see how this is done here. First, we need to prove that the loop invariant holds at the beginning of the first iteration, i.e., when $i = 1$ (Initialization). The loop invariant tells us that $s = \sum_{j=0}^{0} A[j] = A[0]$. This is indeed the case since S was initialized with $A[0]$ in the line before the loop. The loop invariant thus holds for $i = 1$.

Suppose next that the loop invariant holds at the beginning of some iteration i of the loop, that is, $S = \sum_{j=0}^{i-1} A[j]$. Let us write S_i to be the value of S at the beginning of iteration i. Then we need to show that $S_{i+1} = \sum_{j=0}^{i} A[j]$. We see that in iteration i, the instruction $S \leftarrow S + A[i]$ is executed. Hence, we have

$$S_{i+1} = S_i + A[i] = \left(\sum_{j=0}^{i-1} A[i] \right) + A[i] = \sum_{j=0}^{i} A[i] \ ,$$

which is what we needed to prove. The maintenance property thus holds.

We have already discussed the termination property: After the last iteration of the loop, we have that S indeed is the sum of all elements of A.

2 A More Interesting Example

Consider now the following algorithm:

Algorithm 2 Computing the maximum of the elements of an array

```
Require: Array $A$ of length $n$
1: $M \leftarrow A[0]$
2: for $i \leftarrow 1 \ldots n-1$ do
3:     if $M < A[i]$ then
4:         $M \leftarrow A[i]$
5:     end if
6: end for
7: return $M$
```

The algorithm computes the maximum of the elements in A. The following invariant seems to capture the behavior of the algorithm well:

At the beginning of iteration i, $M = \max\{A[j] \ : \ 0 \leq j \leq i-1\}$

We will use the previous three steps to show that the algorithm indeed computes the maximum of A. To this end, denote by M_i the value of M at the beginning of iteration i.

1. *Initialization* ($i = 1$): Observe that M is initialized as $A[0]$. The loop invariant claims for $i = 1$ that $M_1 = \max\{A[j] \ : \ 0 \leq j \leq 0\} = \max\{A[0]\} = A[0]$. The loop invariant hence holds for $i = 1$, since M is initialized with $A[0]$.

2. *Maintenance*: Assume that the loop invariant holds in the beginning of iteration i, i.e., $M_i = \max\{A[j] \ : \ 0 \leq j \leq i-1\}$. We need to show that $M_{i+1} = \max\{A[j] \ : \ 0 \leq j \leq i\}$. Observe that the body of the loop consists of an IF operation. We thus need to distinguish two cases: when the IF evaluates to true and when the IF evaluates to false.

   Suppose first that the IF evaluates to false. Then $M \geq A[i]$ holds and M is not updated. In this case we thus have $M_{i+1} = M_i$. Recall that $M_i = \max\{A[j] \ : \ 0 \leq j \leq i-1\}$. We thus need to show that in this case we have $\max\{A[j] \ : \ 0 \leq j \leq i-1\} = \max\{A[j] \ : \ 0 \leq j \leq i\}$. This is of course true since the fact that the IF evaluates to false implies

$M_i \geq A[i]$. Hence $\max\{A[j] \,:\, 0 \leq j \leq i-1\} \geq A[i]$ which in turn implies $\max\{A[j] \,:\, 0 \leq j \leq i-1\} = \max\{A[j] \,:\, 0 \leq j \leq i\}$.

Next, we need to see what happens if the IF evaluates to true. Then $M < A[i]$ and M is updated to $A[i]$. Observe that in this case $M_{i+1} = A[i]$. Observe that $M < A[i]$ means that $\max\{A[j] \,:\, 0 \leq j \leq i-1\} < A[i]$ and hence $\max\{A[j] \,:\, 0 \leq j \leq i\} = A[i]$. Since $M_{i+1} = A[i]$, the loop invariant thus holds.

3. *Termination*: We have that after the last iteration (or before the nth iteration that is never executed) $M = \max\{A[j] \,:\, 0 \leq j \leq n-1\}$. M is thus the maximum of the elements in A.

This example is slightly more complicated than the example of the last section since the body of the loop contains an IF operation. We therefore needed to conduct a case distinction. We could have avoided this case distinction by observing that the IF operation together with the next line is equivalent to the operation $M \leftarrow \max\{M, A[i]\}$. Indeed, this argument was presented in one of the lectures.

3 An Even More Interesting Loop Invariant

Consider the following algorithm:

Algorithm 3 Computing the two largest elements

```
Require: Array $A$ of length $n$
 1: if $A[0] \leq A[1]$ then
 2:   $x \leftarrow A[0]$, $y \leftarrow A[1]$
 3: else
 4:   $x \leftarrow A[1]$, $y \leftarrow A[0]$
 5: end if
 6: for $i \leftarrow 2 \ldots n-1$ do
 7:   if $y < A[i]$ then
 8:     $x \leftarrow y$
 9:     $y \leftarrow A[i]$
10:   else if $x < A[i]$ then
11:     $x \leftarrow A[i]$
12:   end if
13: end for
14: return $(x, y)$
```

The algorithm computes the two largest elements of the input array. As a simplification, we assume that the array A consists of distinct integers, i.e., $A[i] \neq A[j]$, for every i, j such that $i \neq j$. We will prove the following invariant:

$$\text{At the beginning of iteration } i,\ y = \max\{A[j] \,:\, j \leq i-1\} \text{ and}$$
$$x = \max\left(\{A[j] \,:\, j \leq i-1\} \setminus \{y\}\right).$$

In words, at the beginning of iteration i, y equals the maximum of the subarray $A[0,\ldots,i-1]$ and x equals the second largest element of the subarray $A[0,\ldots,i-1]$ (observe that the simplification that the integers in A are distinct helps in stating this formally in our loop invariant). Let us prove the invariant. Denote by x_i and y_i the values of x and y at the beginning of iteration i.

1. *Initialization* ($i = 2$): We need to show that y_2 is the largest element of $A[0, \dots, 1]$ and x_2 is the second largest of $A[0, \dots, 1]$. This is obvious from the code: If $A[0] \leq A[1]$ then $x_2 = A[0]$ and $y_2 = A[1]$, and the invariant holds, and if $A[0] > A[1]$ then $x_2 = A[1]$ and $y_2 = A[0]$ and the invariant also holds.

2. *Maintenance*: Suppose that the invariant holds at the beginning of iteration i, i.e., x_i is the largest element of $A[0, \dots, i-1]$ and y_i is the second largest element of $A[0, \dots, i-1]$. We need to distinguish three cases:

    - Suppose that $y_i < A[i]$. In this case, $y_{i+1} = A[i]$ and $x_{i+1} = y_i$. Since $A[i] > y_i$ and y_i is the maximum of $A[0, \dots, i-1]$, we clearly have that y_{i+1} is the maximum of $A[0, \dots, i]$. Next, again since y_i is the maximum of $A[0, \dots, i-1]$ and $A[i]$ is larger than any element in $A[0, \dots, i-1]$, x_{i+1} is clearly the second largest element in $A[0, \dots, i]$.
    - Next, suppose that $y_i \geq A[i]$ and $x_i < A[i]$ (i.e., $x_i < A[i] \leq y_i$ holds). In this case, $y_{i+1} = y_i$ and $x_{i+1} = A[i]$. Since $y_i \geq A[i]$ and by the fact that y_i is the maximum in $A[0, \dots, i-1]$, we have that y_{i+1} is clearly also the maximum in $A[0, \dots, i]$. Similarly, since x_i is the second largest element in $A[0, \dots, i-1]$ and $x_i < A[i] \leq y_i$, x_{i+1} (i.e., $A[i]$) is the second largest element in $A[0, \dots, i]$.
    - Last, suppose that $A[i] < x_i$. In this case nothing happens, i.e., $x_{i+1} = x_i$ and $y_{i+1} = y_i$. Since $A[i]$ is smaller than the largest two elements in $A[0, \dots, i-1]$, the invariant stays true.

3. *Termination*: The invariant stated for iteration n (which is never executed) states that x and y are the largest two elements of the array.

4 Exercises

Here are some exercises if you like to practice loop invariants some more. I would not expect perfect formal proofs, but the key parts of the proofs should be given. If you would like to have some feedback, please come to my office and I will have a look at your work.

4.1 Example 1

Algorithm 4 Computing $A[0] - A[n-1]$

```
Require: Array $A$ of length $n$ ($n \geq 2$)
1: $S \leftarrow A[0] - A[1]$
2: for $i \leftarrow 1 \dots n-2$ do
3:     $S \leftarrow S + A[i] - A[i+1]$
4: end for
5: return $S$
```

The algorithm computes the value $A[0] - A[n-1]$. Prove this via the following loop invariant:

At the beginning of iteration i, $S = A[0] - A[i]$ holds.

4.2 Example 2

Prove the following loop invariant:

At the beginning of iteration i, $S = 2^i - 1$ holds.

Algorithm 5 Computing $2^{n+1} - 1$

```
Require: integer $n$
1: $S = 1$
2: for $i \leftarrow 1 \dots n$ do
3:     $S \leftarrow S + 2^i$
4: end for
5: return $S$
```

4.3 Red and Blue Marbles

Suppose that we have a jar that contains n marbles. Each marble is either blue of red. We run the following algorithm:

Algorithm 6 Game with marbles

```
Require: A jar that contains $n \geq 1$ marbles
1: for $i = 1 \dots n - 1$ do
2:     take any arbitrary two marbles out of the jar
3:     if the two marbles have the same color then
4:         put a new red marble back into the jar
5:     else
6:         put a new blue marble back into the jar
7:     end if
8: end for
```

At the end of the algorithm, the jar contains a single marble. Proof the following:

- If initially the jar contains an odd number of blue marbles, then at the end of the algorithm, a single blue marble is left.
- If initially the jar contains an even number of blue marbles (we consider here that 0 is an even number), then at the end of the algorithm, a single red marble is left.

Prove this via the following loop invariant:

The parity of the number of blue marbles in iteration i is the same as before iteration 1.