Reading #3

Proving the Partial Correctness of a Simple Algorithm with a Loop

The objective of this document is to begin a study of how to prove the correctness of a simple `while` loop. This makes use of the same notions of a ***computational problem*** and the ***correctness of an algorithm*** that were introduced in Reading #2, and is continued in Reading #4.

An Example Problem and Algorithm

Recall the following computational problem, which was also introduced in Reading #2:

Maximal Element of Integer Array

Precondition: An integer array A, with positive length n, is given as input.

Postcondition: The largest element in the set

$$\{A[0], A[1], \ldots, A[n-1]\}$$

is returned as output.

Consider, as well, the algorithm shown in Figure 1 on page 2.

Loop Invariants

Definition 1. Consider an algorithm, including a `while` loop, for a given computational problem. An assertion is a ***loop invariant*** for this `while` loop if it is satisfied at the following times, whenever the algorithm is executed with the problem's precondition satisfied:

(a) At the ***beginning*** of every execution of the `while` loop.

(b) At the ***beginning*** of every execution of the body of the `while` loop.

(c) At the ***end*** of every execution of the body of the `while` loop.

(d) At the ***end*** of every execution of the `while` loop.

```
integer arrayMax (integer[] A) }
1. if (A.length == 1) {
2.   return A[0]
   } else {
3.   integer i := 0
4.   integer maxSoFar := A[0]
5.   while (i < A.length - 1) {
6.     i := i + 1
7.     if (maxSoFar < A[i]) {
8.       maxSoFar := A[i]
       }
     }
9.   return maxSoFar
   }
}
```

Figure 1: An Example Algorithm

Note: The computing literature seems to include references to at least *three* similar — but somewhat different — kinds of assertion called "loop invariants". The definition of a loop invariant, given above, is the one that will be used throughout these notes.

Establishing a Loop Invariant

Claim 2. *Consider the algorithm* `arrayMax` *for the "Maximal Element of Integer Array" problem, and the* `while` *loop included in this algorithm. The following is a loop invariant for this* `while` *loop.*

Loop Invariant:

1. `A` *is an input integer array with some length,* n*, such that* $n \geq 2$*.*
2. `i` *is an integer variable such that* $0 \leq \texttt{i} \leq n - 1$*.*
3. `maxSoFar` *is an integer variable such that*

$$\texttt{maxSoFar} = \max(\texttt{A}[0], \texttt{A}[1], \ldots, \texttt{A}[\texttt{i}]).$$

It will be useful to state and prove two other claims before proving Claim 2.

Claim 3. *Consider the algorithm* `arrayMax` *for the "Maximal Element of Integer Array" problem, the* `while` *loop included this algorithm, and the assertion called the "loop invariant" in Claim 2. If the algorithm is executed when the above problem's precondition is satisfied, and the loop is reached during this execution of the algorithm, then this assertion is satisfied when the loop is reached.*

Proof. Consider an execution of the above algorithm with its problem's precondition is satisfied — so that `A` is an integer array with some positive length n.
If $n = 1$ then the claim is trivial (and satisfied): This execution of the algorithm ends after the execution of steps 1 and 2, so that the `while` loop is never reached at all.
Suppose, instead, that $n \geq 2$. Since there is no step in this algorithm that changes the input array `A`, the first part of the assertion (called the "loop invariant" in Claim 2) is implied by the problem's precondition in this case.
Indeed, the `while` loop is reached, in this case, after the execution of steps 1, 3 and 4. One can see by inspection of the code that the `while` loop is never reached again, so it suffices to consider this first execution of the loop.
Since step 3 has been executed, and the value of `i` is not changed at step 4, `i` is an input such that $0 = \texttt{i} \leq 1 \leq n - 1$ when the `while` loop is reached, as needed to establish the second part of the assertion.
Since `maxSoFar` is set to be an integer variable with value $\texttt{A}[0]$ and $\texttt{i} = 0$,

$$\texttt{maxSoFar} = \texttt{A}[0] = \max(\texttt{A}[0], \texttt{A}[1], \ldots, \texttt{A}[i])$$

as well, when the `while` loop is reached, as needed to establish the third part of the assertion and complete the proof of the claim. □

Claim 4. *Consider the algorithm* `arrayMax` *for the "Maximal Element of Integer Array" problem, the* `while` *loop included this algorithm, and the assertion called the "loop invariant" in Claim 2. If this assertion is satisfied at the beginning of an execution of the body of this* `while` *loop then it also satisfied at the end of this execution of the body of the loop.*

Proof. Consider an execution of the body of the `while` loop, that is part of an execution of the `arrayMax` algorithm, such that the assertion called a "loop invariant" in Claim 2 is satisfied when this execution of the loop body begins.
Then, at the beginning of this execution of the loop body, `A` is an input integer array with some length, n, such that $n \geq 2$ — because this is stated as the first part of this assertion.
One can see by inspection of the code that it does not include any statements that change the input array `A`. It must therefore be true that `A` is an input array with some length n, such that $n \geq 2$ — and the first part of the assertion is satisfied — at the end of the execution of the loop body as well.
Since this is implied by the second part of the assertion it also follows that `i` is an integer variable such that $0 \leq \texttt{i} \leq n - 1$ at the beginning of this execution of the loop body. Furthermore, the loop test at line 5 must have been checked and passed immediately before this, so

$\texttt{i} < \texttt{A.length} - 1 = n - 1$ as well. Since `i` and n are both integers, it follows that

$$1 \leq \texttt{i} \leq n - 2 = \texttt{A.length} - 2$$

at the beginning of this execution of the loop body.
Since the value of `i` is increased by one at step 6 and is not changed by the `if-then` test at lines 7–8 it follows that `i` is an integer variable such that

$$2 \leq \texttt{i} \leq n - 1 = \texttt{A.length} - 1,$$

as required to satisfy the second part of the assertion, at the end of this execution of the loop body.
Since the third part of the assertion is satisfied at the beginning of this execution of the loop body, `maxSoFar` is an integer variable such that

$$\texttt{maxSoFar} = \max(\texttt{A}[0], \texttt{A}[1], \ldots, \texttt{A[i]})$$

at this point — and, as noted above, $1 \leq \texttt{i} \leq \texttt{A.length} - 2$.
Since the value of `i` is increased by one and `A` and `maxSoFar` are unchanged, `maxSoFar` is an integer variable such that

$$\texttt{maxSoFar} = \max(\texttt{A}[0], \texttt{A}[1], \ldots, \texttt{A}[\texttt{i} - 1])$$

and $2 \leq \texttt{i} \leq \texttt{A.length} - 1$ immediately after the execution of step 6.
Now, either $\texttt{maxSoFar} < \texttt{A[i]}$ or $\texttt{maxSoFar} \geq \texttt{A[i]}$.

- If $\texttt{maxSoFar} < \texttt{A[i]}$ then it follows that the test at line 7 succeeds and step 8 is executed. Since
$$\texttt{A[i]} > \texttt{maxSoFar} = \max(\texttt{A}[1], \texttt{A}[2], \ldots, \texttt{A}[\texttt{i} - 1])$$
before this, it follows that
$$\texttt{A[i]} = \texttt{maxSorFar} = \max(\texttt{A}[1], \texttt{A}[2], \ldots, \texttt{A[i]})$$
after the execution of step 8. Thus the third part of the assertion (and, therefore, the entire assertion) is satisfied at this execution of the body of the `while` loop in this case.
- On the other hand, if $\texttt{maxSoFar} \geq \texttt{A[i]}$ then the test at line 7 is not executed and this execution of the body of the `while` loop ends immediately after that. In this case
$$\begin{aligned} \texttt{maxSoFar} &= \max(\texttt{A}[1], \texttt{A}[2], \ldots \texttt{A}[\texttt{i} - 1]) \\ &= \max(\texttt{A}[1], \texttt{A}[2], \ldots, \texttt{A[i]}). \end{aligned}$$
Thus the third part of the assertion — and, therefore, the entire assertion — is satisfied at the end of this execution of the loop body as well, as needed to complete the proof of the claim. □

Proof of Claim 2. It is necessary, and sufficient, to show that if the algorithm is executed, when its problem's precondition is satisfied, then the assertion in the claim is satisfied

(a) at the ***beginning*** of every execution of the `while` loop,

(b) at the ***beginning*** of every execution of the body of the `while` loop,

(c) at the ***end*** of every execution of the body of the `while` loop, and

(d) at the ***end*** of every execution of the `while` loop.

One can see, by inspection of the structure of the code, that the `while` loop is executed at most once, so it suffices to consider the *first* execution of the `while` loop. It now follows by Claim 3 that the assertion is satisfied (under the conditions described above) at the beginning of every execution of the `while` loop.
Now consider the following.

Subclaim 5. *Consider the algorithm* `arrayMax` *for the "Maximal Element of Integer Array" problem, and the* `while` *loop included in this algorithm. Suppose that this algorithm is executed when the precondition for this problem is satisfied.*
If ℓ is a positive integer such that the body of the `while` *loop is executed at least ℓ times when the* `while` *loop is being executed, then the assertion called the "loop Invariant" in Claim 2 is satisfied at both the beginning and end of this ℓ^{th} execution of the loop body.*

Proof of Subclaim 5. This will be established by induction on ℓ. The standard form of mathematical induction will be used.
Basis ($\ell = 1$): Suppose that there is at least $\ell = 1$ execution of the body of the `while` loop during the execution of the `arrayMax` algorithm that is mentioned in the claim.
Then the loop is certainly reached during this execution of the algorithm, and it follows by Claim 3 that the assertion is satisfied when the loop is reached.
The only thing happening after this, before the first execution of the body of the `while` loop, is the execution of the loop test at line 5. This does not change the value of any input, variable, or global data, so it does not effect the assertion: Thus, the assertion is is still satisfied at the beginning of the first execution of the body of the `while` loop.
It now follows by Claim 4 that the assertion is also satisfied at the *end* of the first execution of the body of the `while` loop (that is part of the current execution of the loop), as needed to establish the claim when $\ell = 1$, and complete the basis.
Inductive Step: Let k be an integer such that $k \geq 1$. It is necessary and sufficient to use the following

> Inductive Hypothesis: Consider the algorithm `arrayMax` for the 'Maximal Element of Integer Array' problem, and the `while` loop included in this algorithm. Suppose that the precondition for this problem is satisfied at the beginning of an execution of this algorithm.

> If the body of the `while` loop is executed at least k times (during the sole execution of the loop) then the assertion called the "loop invariant" in Claim 2 is satisfied at both the beginning and the end of the k^{th} execution of the loop body.

to prove the following.

> Inductive Claim: Consider the algorithm `arrayMax` for the "Maximal Element of Integer Array" problem, and the `while` loop included in this algorithm. Suppose that the precondition for this problem is satisfied at the beginning of an execution of this algorithm.
>
> If the body of the `while` loop is executed at least $k + 1$ times (during the sole execution of the loop) then the assertion called the "loop invariant" in Claim 2 is satisfied at both the beginning and the end of the $k+1^{\text{st}}$ execution of the loop body.

With that noted, suppose that the loop body is executed at least $k + 1$ times (during the conditions described in the subclaim): The Inductive Claim is trivially true otherwise.
Then it is certainly true that the loop body has been executed at least k times under the conditions described in the subclaim, and it follows by the Inductive Hypothesis that the assertion is satisfied at both the beginning and the end of the k^{th} execution of the body of the `while` loop.
Since the loop test (at line 5) does not change any inputs, variables, or global data, it follows that the assertion is also satisfied at the beginning of the $k + 1^{\text{st}}$ execution of the body of the `while` loop.
It follows by Claim 4 that the assertion is also satisfied at the end of the $k + 1^{\text{st}}$ execution of the body of the `while` loop, as needed to establish the Inductive Claim — completing the inductive step, and the proof of the subclaim. □

Since every execution of the loop body is the k^{th} execution, for some positive integer k, it now follows that the assertion is satisfied at both the beginning and end of every execution of the body of the `while` loop.
Once again, one can see by inspection of the code that the loop is never reached at all (and the claim is trivial) if $\texttt{n} \leq 1$. It is therefore sufficient to consider the case that $\texttt{n} \geq 2$, as we try to establish that assertion also holds at the end of the execution of the `while` loop.
Since `i` initially has value 0, one can see by inspection of the code that the loop test at line 5 is initially passed, so that the body of the `while` loop is executed at least once, when $\texttt{n} \geq 2$.
In this case, if the end of the loop is eventually reached then this happens after the k^{th} execution of the loop body — when the assertion is satisfied — for some positive integer k. Once again, since the loop test does not change any inputs, variables, or global data, the assertion is still satisfied after the test, when the loop ends — as needed to establish the claim. □

The following can be used to simplify the above process:

Theorem 6 (Loop Theorem #1)**.** *Consider an algorithm, for a given computational problem, with a* `while` *loop*

```
while (t) {
  S
}
```

and an assertion $\mathcal{A}$. If

(a) an execution of the loop test `t` *has no side-effects — that is, does not change the value of any inputs, variables, or global data,*

(b) $\mathcal{A}$ is satisfied whenever the loop is reached, during an execution of the algorithm starting with the problem's precondition being satisfied, and

(c) if $\mathcal{A}$ is satisfied at the beginning of any execution of the loop body `S` *(when the problem's precondition was satisfied when execution of the algorithm started) then $\mathcal{A}$ is satisfied, once again, when this execution of the loop body ends,*

then $\mathcal{A}$ is a loop invariant for this `while` *loop.*

The ***proof*** of this is a generalization of the proof of Claim 2. A supplementary document, including a proof of this theorem, is also available.
This theorem makes it easier to prove Claim 2: All that is necessary, now, is to do the following:

(a) Note that the loop test, at step 5, does not change the value of any inputs, variables, or global data.

(b) State and prove Claim 3 — which establishes that the second condition, mentioned in the above "Loop Theorem", is satisfied.

(c) State and prove Claim 4 — which establishes that the third condition, mentioned in the above "Loop Theorem", is satisfied as well.

One can then apply the theorem and conclude that Claim 2, and the stated loop invariant, are both correct.

Partial Correctness

Definition 7. An algorithm for a given computational problem is ***partially correct*** if one, or the other, of the following properties is satisfied whenever this algorithm is executed when the problem's precondition is satisfied: Either

(a) this execution of the algorithm eventually ends with the problem's postcondition satisfied — and with no undocumented inputs or global data accessed, and no undocumented data being modified, or

(b) this execution of the algorithm never ends, at all.

As the proof of the following may suggest, *well chosen* and *complete* loop invariants are useful because they can be used to establish the partial correctness of algorithms:

Claim 8. *The* `arrayMax` *algorithm is partially correct, when considered as an algorithm for the "Maximal Element of Integer Array" problem.*

Proof. Consider an execution of this algorithm that begins when the precondition for the "Maximal Element of Integer Array" problem is satisfied. Then the input is an integer array `A` with some positive length n.
One can see by inspection of the code that this algorithm does not access any undocumented inputs or global data and no undocumented data is modified: The algorithm's signature can be used to confirm that there are no undocumented inputs, and an inspection of each instruction in the pseudocode confirms that none of them accesses undocumented global data or makes undocumented modifications to any data at all.
If $n = 1$ then this execution of algorithm ends after the execution of steps 1 and 2, with the value

$$\mathtt{A}[0] = \max(\mathtt{A}[0], \mathtt{A}[1], \ldots, \mathtt{A}[n-1]),$$

so that the problem's postcondition is satisfied — as needed to establish the claim in this case: Condition (a) in the definition of "partial correctness" is satisfied.
On the other hand, if $n \geq 2$ then steps 1, 3 and 4 are executed — and the `while` loop is reached and executed. Either the execution of the loop terminates, or it does not.
On the other hand, if $n \geq 2$ then steps 1, 3 and 4 are executed — and the `while` loop is reached and executed. Either the execution of the loop terminates, or it does not.

- If the execution of the loop terminates then it follows by Claim 2 (and the definition of a "loop invariant") that the loop invariant, given in the claim, is satisfied when the execution of the loop ends — so that `i` is an integer variable such that $0 \leq \mathtt{i} \leq n-1$ and `maxSoFar` is an integer variable such that

$$\mathtt{maxSoFar} = \max(\mathtt{A}[0], \mathtt{A}[1], \ldots, \mathtt{A[i]}).$$

- It must also be true the loop test (at step 5) was checked and failed, so that $\mathtt{i} \geq \mathtt{A.length} - 1 = n - 1$. Thus $\mathtt{i} = n - 1$ and

$$\mathtt{maxSoFar} = \max(\mathtt{A}[0], \mathtt{A}[1], \ldots, \mathtt{A}[n-1]).$$

- It follows, in this case, that the execution of the algorithm ends with the execution of step 9 — at which point the largest element in the set

$$\{\mathtt{A}[0], \mathtt{A}[1], \ldots, \mathtt{A}[n-1]\}$$

```
integer badArrayMax (integer[] A) {
1. if (A.length == 1) {
2.   return A[0]
   } else {
3.   integer i := 0
4.   integer maxSoFar := A[0]
5.   while (i < A.length - 1) {
6.     i := i
     }
7.   return maxSoFar
   }
}
```

Figure 2: An Incorrect Algorithm that is Partially Correct

is returned as output — satisfying the problem's postcondition, as needed to establish the claim in this case: Once again, condition (a) in the definition of "partial correctness" is satisfied.

- On the other hand, if the execution of the loop does not end then the execution of the algorithm certainly does not, either — as needed to establish partial correctness in this case, as well — because property (b) in the definition of "partial correctness" is satisfied — and to complete the proof. □

It follows from the definition of "correctness" of an algorithm, in the previous set of notes, and the definition of "partial correctness", given above, that if an algorithm is *correct* then it is also *partially correct*.
With that noted, consider the algorithm shown in Figure 2.

Exercises:

1. Modify the proof of Claim 2 in order to show that the loop invariant, given in that claim, is also a loop invariant for the `while` loop in the `badArrayMax` algorithm (when this is considered as an algorithm for the "Maximal Element of Integer Array" problem).

2. Modify the proof of Claim 8 in order to show that the `badArrayMax` algorithm is also partially correct.

3. Confirm that an execution of this algorithm does not halt when the problem's precondition is satisfied, but the input array `A` has length $n \geq 2$ — so that this algorithm is *not* correct.

The next reading will introduce a method to prove that an algorithm with a `while` loop is guaranteed to terminate whenever it is executed with its problem's precondition satisfied — as needed to complete a proof of correctness. A method to bound the ***running time*** of such an algorithm will also be introduced.