Loop Invariants

We end this unit by considering a topic that was actually introduced in the text during the first week. This topic—loop invariants—addresses an issue of how one goes about proving the correctness of an algorithm under the conditions where our algorithm involves iteration. We begin by noting that the text shows three stages in proving correctness based on loop invariants:

1. Initialization
2. Maintenance
3. Termination

To explain and justify these steps, we will look a bit at the theory behind loop invariants. We will also consider a simple example around which we will center our discussion, based on the following theorem.

Theorem: For any integer $n \geq 0$ and integer $m > 0$, there exist q and r such that $n = q \cdot m + r$ and $0 \leq r < m$.

The way we will go about proving this theorem is by designing an algorithm to find q and r for any given n and m under the above conditions. We will then prove the algorithm is correct using loop invariants. This algorithm, by the way, is called the DIVISION algorithm and consists of the following:

Algorithm 1 Division

```
DIVISION(int n, int m)
  int q, r;
  q ← 0;
  r ← n;
  while r ≥ m do
    q ← q + 1;
    r ← r − m;
  end while;
  return q, r;
```

To prove this algorithm is correct (and thereby prove the theorem), we need to prove two things. First, we need to prove that the algorithm will terminate. Second, we need to prove that, when it terminates, it will terminate with the correct result. To accomplish the first (which corresponds to an important part of the Termination step in the text), we note that termination depends upon whether or not the while loop is an infinite loop. Notice that the initial value for r, which is the variable we test for termination, is n. Should $r < m$, we never enter the loop, and the algorithm terminates immediately. On the other hand, if $r \geq m$, we enter the loop. Now notice that the value of r is updated during each iteration of the loop with the statement $r \leftarrow r - m$. Recall that $m > 0$, so this means the value of r must reduce on each iteration, n, $n - m$, $n - 2m$, etc. Since r is strictly decreasing and $m > 0$, there must come a point where $r < m$, thus causing the loop to terminate. From this, we can conclude that for all values of n and m, the above algorithm must terminate.

For the second part of our proof, we will use the concept of a loop invariant. For this part, we need some additional background. First, for some terminology, notice that we are working with a while loop. In fact, what we are about to do works with any kind of loop, so we discuss the while loop without loss of generality. A while loop has the following general form:

```
while C do
   E;
end while;
```

In this template, we note that C corresponds to the logical conditions that must be satisfied for the loop to be entered. This is called the "guard" of the loop. The E part of the loop corresponds to the statements that are executed inside the loop. The loop is said to terminate once the guard is no longer true.

Definition: Let S be some statement. Then S is an *invariant* of the loop

```
while C do
   E;
end while;
```

when both S and C are true before any given iteration of the loop and S remains true after the iteration.

Theorem: Assume statement S is an invariant of the loop

```
while C do
   E;
end while;
```

Assume also that S is true on the first entry to the loop. Then

1. If the loop terminates, S is true after the last iteration, and
2. If the loop does not terminate, S remains true after every iteration.

Proof: By assumption, S is true upon the first entry into the loop. Assume there exists an iteration of the loop such that, following that iteration, S is no longer true (i.e., S is false). Let k indicate the iteration where this occurs. Therefore, for iterations $i = 0, 1, \ldots, k-1$, S remains true. Furthermore, the guard C must still be true after the $(k-1)^{\text{th}}$ iteration for the k^{th} iteration to occur. By the definition of a loop invariant, given above, S must still be true after the next iteration, which is a contradiction. Therefore, S remains true if the loop terminations and remains true for all iterations of the loop, even if the loop does not terminate. Thus our theorem is proved.

We are now ready to use this theorem to finish our correctness proof of the DIVISION algorithm. For this part, we need to determine a loop invariant. Here we will use the condition specification of the algorithm to serve as our invariant, namely

$$S = (n = q \cdot m + r) \wedge (r \geq 0).$$

First, we can show that S is true upon the initial entry into the loop. This corresponds to the Initialization step described in the text. In this case, we have that $r \geq 0$ since $r = n$ and $n \geq 0$. We can show the second part of S to be true by substition, namely,

$$\begin{aligned} n &= q \cdot m + r \\ &= 0 \cdot m + r \\ &= n. \end{aligned}$$

Notice that if the loop is not executed, these conditions remain true, so now we need to consider the situation where the loop is executed. This corresponds to the Maintenance step in the text, and when combined with our termination conditions above, gives us the Termination step as well. We start by assuming S is true prior to a given iteration. (Note that this process is very similar to an inductive proof). Examining the body, E, of the loop, we find that q and r both change such that $q' = q + 1$ and $r' = r - m$. After this occurs, we update S such that

$$\begin{aligned} n &= q' \cdot m + r' \\ &= (q+1) \cdot m + (r - m) \\ &= q \cdot m + m + r - m \\ &= q \cdot m + r. \end{aligned}$$

We see that the relative values do not change, so S remains unchanged. We also find that $r' \geq 0$ because, to enter the loop $r \geq m$ and all we did was subtract m from r. Otherwise, this iteration would not have occurred. Thus we see that S is indeed a loop invariant (it is unchanged after every iteration). Therefore, since the algorithm has also been shown to terminate and return values of q and r such that S holds, we know the algorithm is correct.