CMSC 250: Program Verification

1. The Big Idea

   We'd like to verify that an algorithm does what it should. What this essentially means is that before the algorithm we have a certain number of pre-conditions that are set, then the algorithm executes, and then we have some post-conditions which we require to be satisfied.

   Defn: We say that the algorithm is correct if, when various pre-conditions are set and the algorithm is executed, the post-conditions are met.

2. A Pseudo-Example.

   Suppose we wish to write an algorithm which takes a positive real n and finds another real m with $m^2 = n$. The basic structure would then look like this:

```
\\ Precondition: n is any positive real.
MYSTERIOUS ALGORITHM: Calculates m somehow.
\\ Postcondition: m^2 = n
```

   Here are some examples of correct and incorrect algorithms:

   Correct:

```
\\ Precondition: n is any positive real.
Assign m = sqrt(n)
\\ Postcondition: m^2 = n
```

   Correct:

```
\\ Precondition: n is any positive real.
Assign m = -sqrt(n)
\\ Postcondition: m^2 = n
```

   Inorrect (because the post-condition will not be met for any positive real):

```
\\ Precondition: n is a positive integer.
Assign m = n
\\ Postcondition: m^2 = n
```

3. Fact: This is Hard!

   It's not easy to verify whether a given algorithm is correct, especially when the algorithm is long and complicated. Testing can be performed whereby lots of input is provided and output is tested and this can provide some assurance but it is by no means a proof.

   Here we'll focus specifically on the correctness of loops.

4. Correct Loops

   We can make the above definition more specific when it comes to a loop.

   Defn: We say that a loop is correct (with regards to its pre- and post-conditions) if, when various pre-conditions are met and the loop terminates after a finite numbers of steps, the post-conditions are met.

5. Loop Invariants

Focusing on loops allows us to introduce the concept of a loop invariant. A loop invariant is a predicate (either `T` or `F`) whose status we can monitor through the loop and which can help us determine if a loop is correct.

Defn: A loop invariant is a predicate which satisfies the following:

- For each iteration, if it is true at the start of the iteration then it is true when the iteration has finished.

In and of itself this is not particularly useful but when we attach two other conditions:

- It is true before the first iteration of the loop.
- If the loop terminates after finitely many steps, then if the loop invariant is true then the post-conditions are satisfied.

Then the loop will be correct.

A loop invariant can be denoted `LI(n)` where `n` denotes the iteration count of the loop. With this notation `LI(0)` represents the loop invariant before the first iteration, `LI(n)` represents the loop invariant at the start of the `n^th` iteration, `LI(n+1)` represents the loop invariant at the end of the `n^th` iteration (or at the start of the `(n+1)^st` iteration), and so on. When the loop ends the loop invariant will be represented by `LI(N)` where `N` is the number of iterations which the loop took to complete.

The correctness of the loop as it relates to the loop invariant is then summarized in the:

Loop Invariant Theorem: Let a while loop with guard predicate `G` be given, along with pre- and post-condition predicates `PRE` and `POST`. Also let a predicate `LI(n)` be given. Then, if the following four conditions are met, the loop will be correct:

(a) **Basis Property:** If `PRE` is true then `LI(0)` is true (in other words that `LI` is true before the first iteration of the loop).

(b) **Inductive Property:** For all `k >= 0` if the guard `G` is true and if `LI(k)` is true, then `LI(k+1)` is true. Note that it can be helpful when doing this to introduce new variables to clarify when variables change within the loop. We'll see this in our examples.

(c) **Termination:** After a finite number of steps we have `~G`, meaning the loop terminates.

(d) **Correctness of Post-Condition:** If `LI(N)` then `POST`, where `N` is the least number of steps after which the loop terminates (that is, `G` is false).

Proof: Suppose `PRE` is true. By the Basis Property then `LI(0)` is true. The Basis Property coupled with the Inductive Propery then tells us that `LI(n)` is true for all `n >= 0`. By Termination we know the loop terminates and so `~G`. Let `N` be the last number of steps after which it terminates then we know that `LI(N)` is true. Then by the Correctness of Post-Condition we have `POST`.

$\mathcal{QED}$

In brief (this is a summary of the above) there are four things we have to prove:

(a) **Basis:** If `PRE` then `LI(0)`.

(b) **Inductive:** If `G` and `LI(k)` then `LI(k+1)`.

(c) **Termination:** Eventually `~G`.

(d) **Correctness of Post:** If `LI(N)` then `POST`, where `N` is the least number of steps after which `~G`.

6. **Example:** Finding x^p for any $x \in \mathbb{R}^*$ and any $p \in \mathbb{Z}^+$.

   We have a nonzero real number `x` and a positive integer `p` and we wish to calculate `x^n`. Here is the pseudocode:

```
\\ PRE: x is a nonzero real number
\\ PRE: p is a positive integer
\\ PRE: i = 1
\\ PRE: result=1
while i <= p
    result=result*x
    i=i+1
end
\\ POST: result = x^p
```

   We'll introduce the list invariant;

   `LI(n): n = i-1 and result = x^(i-1)`

   Now then let's observe the requirements of the theorem:

   Base: Check that `PRE` implies `LI(0)`.

   Let's assume `PRE`, so then we have: `x` a nonzero real number, `p` a positive integer and `i = 1`.

   We claim `LI(0)`, in other words, we claim that `0 = i-1 and result = x^0`

   Well `i = 1` and so `0 = 1-1` and `result = 1 = x^0` as desired.

   Induction: Check that for all `k >= 0` if the guard `G` is true and if `LI(k)` is true, then `LI(k+1)` is true.

   Suppose the guard is true (so `i <= n`) and `LI(k)` is true. To have `LI(k)` true means:

   `k = i-1 and result = x^(i-1)`

   At the end of the iteration we have the new value of `i`, which we'll denote `i'` and a new `result`, which we'll denote `result'`, and we need to check that `LI(k+1)` is true, meaning:

   `k+1 = i'-1 and result' = x^(i'-1)`

   First, observe that `i' = i+1` and so `k = i-1` $\Rightarrow$ `k+1 = i` $\Rightarrow$ `k+1 = i'-1`.

   Second, observe that `result = x^(i-1)` and `result' = result*x` tell us that and the end of the iteration we have `result' = x^(i-1)*x = x^i = x^(i'-1)`.

   Termination: Check that after a finite number of steps we have `~G`.

   The loop starts at `i = 1`, performs `i = i+1`, and terminates at `i = p+1` so the loop will terminate after `p` iterations.

   Post Check: Check that if `N` is the least number of steps after which `~G` and if `LI(N)`, then `POST`.

   Since the loop terminates after `p` iterations we have `N = p`. Suppose that `LI(N) = LI(n)` is true, which means:

   `N=i-1 and result = x^(i-1)`

   Then observe `result = x^(i-1) = x^N = x^p`, which is `POST`.

   QED

7. **Example:** Finding the maximum of a list.

   We have a list `A` for which we with to find the minimum. Here is the pseudocode:

```
\\ PRE: A = list of real numbers.
\\ PRE: len = length of A
\\ PRE: max = A[0]
\\ PRE: i = 1
while i < len
    if A[i] > max
        max = A[i]
    end
    i=i+1
end
\\ POST: max = maximum of A[0..len-1].
```

   We'll introduce the list invariant;

   `LI(n): n = i-1 and max=maximum value in A[0..i-1]`

   Now then let's observe the requirements of the theorem:

   Base: Check that `PRE` implies `LI(0)`.

   Let's assume `PRE`, so then we have: `A` a list of numbers, `max = A[0]` and `i = 1`.

   We claim `LI(0)`, in other words, we claim that `0 = i-1 and max=maximum value in A[0..i-1]`.

   Well `i = 1` and so `0 = 1-1` and `maximum value in A[0..0] = A[0] = max` as desired.

   Induction: Check that for all `k>=0` if the guard `G` is true and if `LI(k)` is true, then `LI(k+1)` is true.

   Suppose the guard is true (so `i < n`) and `LI(k)` is true. To have `LI(k)` true means:

   `k = i-1 and max=maximum value in A[0..i-1].`

   At the end of the iteration we have the new value of `i`, which we'll denote `i'` and a new `max`, which we'll denote `max'`, and we need to check that `LI(k+1)` is true, meaning:

   `k+1 = i'-1 and max' = maximum value in A[0..i'-1]`

   First, observe that `i' = i+1` and so `k = i-1` $\Rightarrow$ `k+1 = i` $\Rightarrow$ `k+1 = i'-1`.

   Second, since at the start of the loop `max = maximum value in A[0..i-1]`. The code examines `A[i]` and if `A[i] > max` then `max` is updated to this new value, otherwise it is left alone. Consequently at the end of the loop we have `max'=maximum value in A[0..i] = A[0..i'-1]` as desired.

   Termination: Check that after a finite number of steps we have `~G`.

   Since the loop starts at `i = 1`, performs `i = i+1`, and terminates at `i = len` the loop will terminate after `len-1` iterations.

   Post Check: Check that if `N` is the least number of steps after which `~G` and if `LI(N)`, then `POST`.

   Since the loop terminates after `len-1` iterations we have `N = len-1`. Suppose that `LI(N)` is true, which means:

   `N = i-1 and max = maximum value in A[0..i-1]`

   Then observe `max = maximum value in A[0..i-1] = A[0..N] = A[len-1]`, which is `POST`.

   QED

8. **Example:** Performing an insert sort on a list.

   We have a list `A` and we wish to perform an insert sort. We're focusing on the outer `while` loop.

```
\\ PRE: A = a list
\\ PRE: len = the length
\\ PRE: i = 1
while i < len
    nextval = A[i]
    j = i-1
    while j >= 0 and nextval < A[j]
        A[j+1] = A[j]
        j = j-1
    end
    A[j+1] = nextval
    i = i + 1
end
\\ POST: A is a sorted list
```

   We'll introduce the list invariant;

   `LI(n): n = i-1 and A[0..i-1] is sorted`

   Now then let's observe the requirements of the theorem:

   Base: Check that `PRE` implies `LI(0)`.

   Let's assume `PRE`, so then we have: `A` is a list and `i = 1`.

   We claim `LI(0)`, in other words, we claim that `0 = i-1 and A[0..i-1]` is ordered.

   Well `i = 1` and so `0 = 1-1` and `A[0..i-1] = A[0]` which is a single element and hence is ordered.

   Induction: Check that for all `k >= 0` if the guard `G` is true and if `LI(k)` is true, then `LI(k+1)` is true.

   Suppose the guard is true (so `i < n`) and `LI(k)` is true. To have `LI(k)` true means:

   `k = i-1 and A[0..i-1] is sorted`

   At the end of the iteration we have the new value of `i`, which we'll denote `i'`, and an updated list which we'll denote `A'`, and we need to check that `LI(k+1)` is true, meaning:

   `k+1 = i'-1 and A'[0..i'-1] is sorted`

   First, observe that `i' = i+1` and so `k = i-1` $\Rightarrow$ `k+1 = i` $\Rightarrow$ `k+1 = i'-1`.

   Second, at the start of the loop `A[0..i-1]` is sorted. The code assigns `nextval = A[i]`, this is the value it needs to insert into the correct position in `A[0..i]`. The code starts at index `j=i-1` and for each `j` if `nextval < A[j]` then `nextval` needs to be inserted somewhere before index `j` and so `A[j]` is moved to the right by assigning `A[j+1] = A[j]`. Eventually either `nextval >= A[j]` which means `nextval` should go to the right of index `j` or else `j == -1` which will happen if the code reaches the start of the list without finding a value smaller than `nextval`. Either way the inner while loop ends and `nextval` is inserted using `A[j+1] = nextval`. We now have, for our updated list `A'`, that `A'[0..i] = A'[0..i'-1] is sorted` as desired.

   Termination: Check that after a finite number of steps we have `~G`.

   Since the loop starts at `i = 1`, performs `i = i+1`, and terminates at `i = len` the loop will terminate after `len-1` iterations.

   Post Check: Check that if `N` is the least number of steps after which `~G` and if `LI(N)`, then `POST`.

   Since the loop terminates after `len-1` iterations we have `N = len-1`. Suppose that `LI(N)` is true.

   `N = i-1 and A[0..i-1] is sorted`

   Then observe `A[0..i-1] = A[0..N] = A[0..len-1] is sorted`, which is `POST`.

   QED