What is program correctness?

Program correctness: if precondition then (termination and post-condition)

So proving correctness means proving

precondition => (termination and post-condition)

//

Correctness: Adding Loops

When proving that an iterative program (with a loop) is correct, we prove partial correctness and termination separately:

(a) partial correctness: prove (precondition and termination) => post-condition

(b) termination: prove precondition => termination

For part (a), we need a loop invariant.

For part (b), we need a loop variant.

//

Steps in proving correctness of an iterative program:

1. Formulate a loop invariant (LI). \* Note: When formulating the loop invariant, sometimes it helps to trace an instance of the loop. The loop invariant should describe the purpose of each variable involved, and have some connection to the post-condition. \*

2. Use induction to prove the loop invariant.

3. Use LI to prove partial correctness: Assuming precondition holds, LI holds, and program terminates, show that when program terminates, post-condition holds.

4. Use loop variant to prove termination.

//

Our partial correctness proof proves that the program is correct, only under the assumption that the program terminates.

We must prove that this termination actually does occur.

To prove termination:

Find a loop variant v that uses the variables in the loop, such that v is:

(a) decreasing after every iteration

(b) invariant and loop guard together imply that v >= 0 (i.e. v is always a natural number at the beginning of each loop iteration)

If v decreases on each iteration yet cannot drop below 0, then at some point the loop must terminate.

=====================================================================

TEMPLATE FOR WRITING PROOF OF CORRECTNESS FOR ITERATIVE ALGORITHMS

=====================================================================

\* Replace all in [[...]] with information particular to the proof

STEP 1: STATE THE PRECONDITION AND POSTCONDITION

Precondition: [[....]]

Post-condition: [[....]]

STEP 2: PROOF OF LOOP INVARIANT (i.e. prove that loop invariant (L.I.) is correct)

Predicate (loop invariant): Let P(i) be the statement that at the beginning of every ith iteration of the loop, [[..loop invariant is true..]].

\* Note: When formulating the loop invariant, sometimes it helps to trace an instance of the loop. The loop invariant should describe the purpose of each variable involved, and have some connection to the post-condition. \*

Base case: [[..Prove P(0).. (i.e. prove that LI holds prior to the first iteration of the loop)]]

Induction Step: Let j be an arbitrary natural number such that j >= 0. Assume P(j) holds (i.e. at the beginning of the jth iteration of the loop, [[..loop invariant is true..]] and assume that there is a (j+1)th iteration.

[[..Prove that if P(j) is true then P(j+1) is also true..]]

///

OR The course notes way:

- no need to state a predicate

- prove that LI holds prior to first iteration

- for induction, we use it implicitly, and just assume Inv(x\_0) holds, where x is some variable involved in the invariant, and prove if so, then Inv(x\_1) holds

///

We can thus conclude that L.I. holds for all iterations of the loop.

STEP 3: PROOF OF PARTIAL CORRECTNESS

Partial correctness: (precondition and termination) => post-condition

\* Note: Since the loop terminates at the negation of the loop guard, what we need to prove can be written as:

(loop exit condition and LI) => post-condition \*

When the loop terminates, we know that [[..something about a variable in the LI..]]. So, by LI, when the loop terminates, [[..value = ..]], which satisfies the post-condition.

STEP 4: PROOF OF TERMINATION

A loop variant v for this algorithm is: [[...state variant...]].

We must prove that v is (1) decreasing with each iteration of the loop, and (2) always a natural number at the beginning of each loop iteration.

[[…give explanation of why (1) and (2) are true for this variant…]]

Since we have established that v is a decreasing, bounded variant for the loop, this loop terminates, and therefore the algorithm terminates.

STEP 5: STATE CONCLUSION

In conclusion, if the precondition holds (i.e. [[state precondition]]), the algorithm will terminate and when it does, the post-condition will hold (i.e. [[state post-condition]]).