Maggie Johnson
CS109

Proving Properties of Loops

Key Topics:

* More on Induction
* Validation vs. Verification
* Loop Invariants
* Induction to Verify Loops
* Example Proofs

• Induction Challenger

While waiting for class to start, check out the following proof. Is it valid? If not, how would you fix it?

P(n): In any convex n-gon, the longest side is shorter than the sum of the lengths of the other sides.

base case: P(3), a triangle. It is a geometric fact that every side of a triangle is shorter than the sum of the other two.
inductive hypothesis: Assume P(n) is true, show P(n+1), i.e., that P(n) is true for an (n+1)-gon.

Proof:

Take any convex (n+1)-gon P. Label the sides 1, 2, ..., n+1 in clockwise order, with side 1 being the longest. Slice off sides 2 and 3, replacing them by a single side 2', and thus obtain a convex n-gon P'. The inductive hypothesis allows us to surmise that side 1 is shorter than the sum of the other sides of P'. If we now replace the length of side 2' by the sum of the lengths of sides 2 and 3, the sum of the lengths of the sides other than 1 gets even greater.

By the principle of mathematical induction, P(n) is true for all convex polygons.

• Validation vs. Verification

You **validate** that a program that you have written is correct by testing it with a wide range of relevant inputs. You **verify** a program is correct by using proof techniques to show that given any input variables that satisfy certain properties, the output variables (after execution of the program) satisfy other certain properties. This is an important distinction - both are necessary to be completely sure of a correct program.

In 1967, Robert Floyd published a paper that started a whole area of research on program verification. Several researchers since that time have developed techniques for proving program correctness (Dijkstra, Gries). These researchers contend that "it is not only the programmer's task to produce a correct program, but also to demonstrate its correctness in a convincing manner."

As with most techniques that are still in the process of development, methods for proving program correctness are somewhat awkward and unwieldy. Many people believe that real-world programs are just too complex to prove correct. Nonetheless, it is an important area of research and it also illustrates a valid computer-science application of proof techniques.

One point should be made before we proceed. There is an important distinction between "proofs of existence of correct behavior" (called **type-1 correctness**) and "proofs of the non-existence of incorrect behavior" (called **type-2 correctness**). We are dealing only with type-1 correctness proofs. In fact, type-2 correctness proofs are generally considered to be impossible, leaving testing as the only means for dealing with this requirement.

We will deal only with proving the correctness of loops. The key to proving a property about a loop is to select a *loop invariant*, which is a statement that is true each time we pass by a particular point in a loop.

>> **Program State:** The program state at a particular point during program execution is the value of each data object in the program at that point.
>> **Assertion:** a statement describing a property or characteristic of one or more states of a program during execution.
>> **Loop Invariant**: an assertion that describes the program state at the end of the Kth iteration of a loop, for any positive integer K. The loop invariant describes how the data values defined and used in the loop body are related to each other and to the number K (being the number of repetitions that the loop body has executed). Basically, we are describing the program state at the end of a *typical* loop iteration. If we can prove that a loop invariant is true each time we pass it while executing a loop, we have proven the correctness of the loop.

Examples of Loop Invariants:

1) /* program to sum the integers from 1 through n and store result in sum */
```
sum = 0;
for (i = 1; i <= n; i ++)
        sum +=  i;
        /* Loop invariant: On Kth iteration,   sum = K ( K+1 ) / 2;  i = K */
```

2) /* search array GradesList for the first grade > 90. Store subscript of that element in FirstA; if no grade above 90 is found, store 0 in FirstA */
```
FirstA = 0;
i = 1;
while ((i <= lastentry) && (FirstA == 0)) {
        If (GradesList[i] > 90 )
                FirstA = i;
        i++;
        /* Loop invariant: On Kth iteration, the first K elements of GradesList have been tested,
         and i = K+1.  If the Kth element is > 90, then FirstA = K */
}
```

3) /* a program to do multiplication; m & n are user-defined positive integers; loop calculates m*n */
```
Answer = 0;
Count = n;
while (Count  != 0) {
        Answer +=  m;
        Count --;
        /* Loop invariant: on the Kth iteration, Answer = K *  m; Count = n - K */
}
```

4) What is the loop invariant of the following loop?

```
sum = 0;
for (i = 1; i <= n; i ++) {
        temp = i % 2;
        if (temp != 0)
                sum +=  i;
```

Induction to prove loop correctness:

Once we know what the loop invariant of a program is, we can prove the correctness of the loop with respect to the loop invariant using induction. The process is:

i) Prove that the loop invariant is true after the first iteration (or if the loop does not iterate at all, prove the loop invariant is true after 0 iterations).
ii) Assume the loop invariant is true after K iterations, and then prove that it must also be true after K+1 iterations.

Example 1:

Consider the following code segment:

```
sum = 0;
for (i = 1; i <= n; i ++)
        sum +=  i;
        /* Loop invariant: On Kth iteration,   sum = K ( K+1 ) / 2;  i = K */
```

Prove the correctness of this loop:

i) base case: After the first iteration of the loop, sum = 1. When we pass the loop invariant, K = 1, so 1 (1+1) /2 = 1.
ii) Assume the induction hypothesis: We assume that after the Kth iteration, sum = K(K+1)/2, and show that after K+1 iterations, sum = (K+1)[(K+1) + 1] / 2.

PROOF:
On the K+1 iteration, sum = sum + i and i = K+1. We know the value of sum from the inductive hypothesis: K(K+1)/2. So on the K+1 iteration, sum = K(K+1)/2 + (K+1)

$$K(K+1)/2 + (K+1) = [K(K+1) + 2(K+1)] / 2$$
$$= (K^2 + 3K + 2) / 2$$
$$= [(K+1)(K+2)] / 2$$
$$= (K+1)[(K+1)+1] / 2$$

The K+1 iteration has been verified when the K iteration is correct, and therefore the loop is correct.

Example 2:

Consider the following code segment:

```
scanf("%d %d", &limit, &x);
answer = 0;
count = limit;
while (count  !=  0) {
        answer += x;
        count --;
        /* Loop invariant: on the Kth iteration, answer = K *  x; count = K - 1 */
}
```

Prove the correctness of this loop:

i) base case: Since it is possible to not enter the loop at all, the basis of induction is 0. Therefore, K = 0: answer = 0 * X = 0.
ii) Assume the induction hypothesis: We assume that after the Kth iteration, answer = K * X, and show that after K+1 iterations, answer = (K+1) * X.

PROOF:
On the K+1 iteration, answer = answer + X. The value of answer from the inductive hypothesis is K * X:

$$\begin{aligned} \text{answer} &= \text{answer} + X \\ &= (K * X) + X \\ &= (K * X + X) \\ &= (K + 1) * X. \end{aligned}$$

The K+1 iteration has been verified when the K iteration is correct, and therefore the loop is correct.

`Do you find any flaws in the preceding proof?`

An important consideration with while loops, unlike for loops, is the possibility that the condition on the while loop may never become false, i.e., the loop may never terminate. So, with while loops, we must also include a proof of loop termination.

`How would you` *`prove`* `that a while loop terminates?`

One way to prove loop termination is to identify an expression E involving the variables of the guard condition of the loop, such that:

1) The value of E decreases by at least 1 each time around the loop
2) The loop condition is false if E is as low as some specified constant.

`What would be such an expression E for the previous proof?`

Example 3:

The following code segment implements factorial: 1 * 2 * 3 * ... * n = n!

```
n = GetInteger();
i = 2;
fact = 1;
while (i <= n) {
        fact *= i;
        i ++;
}
printf("%d", fact);
```

```
What would be an appropriate expression E for this loop?
What is a good loop invariant for this loop?
Prove the correctness of this loop:
```

One important observation is that the inductive step of program correctness proofs amounts to just walking through a typical iteration of the loop. You might ask what does this really prove? A loop invariant is a concise description of what a loop does (and therefore provides useful documentation). If you can prove that the actual execution of the loop produces results that match the loop invariant, then your loop is doing what it's supposed to do (according to the loop invariant). A programmer usually doesn't have time to do inductive proofs on loops, but he/she intuitively knows why a loop works. Whether they know it or not, programmers use inductive reasoning when writing loops.

Bibliography

For more information on program correctness, refer to section 3.5 in Rosen.

O.J. Dahl, E. W. Dijkstra, C.A.R. Hoare, *Structured Programming,* London: 1972.

D. Gries, *The Science of Programming*, New York: Springer Verlag, 1981.

R. W. Floyd, "Assigning meanings to programs," *Proc. Symp. Appl. Math.,* Amer. Math. Society 19 (1967), 19-32.

Z. Manna and R. Waldinger, *The Logical Basis for Programming*, Reading, MA: Addision Wesley, 1990.

• Answer to Induction Challenger

The assertion is true, but side 1 may not be the longest side of P'; it could be that the new side 2' is longest. Thus, P(n) may not apply.

One way to fix this is to modify the original assertion: In any convex n-gon, "any" side is shorter than the sum of the lengths of the other sides.