Burt Rosenberg

Loop Invariants

OUT: 12 OCTOBER, 1992

Loop invariants make designing and debugging code easier. We want to prove that the outcome of a while-loop in our program is correct. The method of loop invariants has us set up four mathematical statements:

1. the *Precondition*, the assumptions you start with,
2. the *Test*, controls loop termination,
3. the *Loop Invariant*,
4. and the *Postcondition*, the situation you want to end with.

The idea is to have each of these statements true at the moment you pass by their "assertions".

Arrange the Precondition, Test, Loop Condition and Postcondition in your programs, homeworks or exams as follows:

```
...
{Assert Precondition}
while Test do
begin
   S1 ; S2 ; ... ; Sn ;
   {Assert if Test then Loop Invariant}
end
{Assert Postcondition}
...
```

By *Assert X* we mean that if one were to stop the program at that moment and evaluate the statement X, if would be true. This evaluation happens in the context of certain truths, called *hypotheses.* The trick is to go from hypotheses to assertions which then become the new hypotheses. Employing the notation in Table 1, we have the following summary:

If the R, I and O are the Precondition, Loop Invariant and Postcondition, respectively, of a while-loop, S represents the body of the while-loop, and n is the number of passes through the while loop, the sequence of assertions and statements is,

$$\begin{array}{ll} R\,! \,;\, (S \,;\, I\,!)^{n-1} \,;\, S \,;\, O\,! & \textit{if}\, n \geq 1 \\ R\,! \,;\, O\,! & n = 0 \end{array}$$

$R\,!$	Assert Precondition true.
$I\,!$	Assert Loop Invariant true.
$O\,!$	Assert Postcondition true.
T	Test is true.
$\neg T$	Test is false.
S	Do while-loop body, statements S1 through Sn.

Table 1: Glossary of symbols.

The method of loop invariants proposes to show that each assertion in the above listing follows from it predecessor. By mathematical induction, then, one can show that the truth of the Postcondition, the thing you want, will follow from the truth of the Precondition, the thing you assume. Here is how it works:

1. You show that the Precondition is true when asserted. This involves assumptions on the correctness of the input data and the program performance up to this point. In symbols:

$$R\,! \tag{1}$$

2. You show that the Loop Invariant is true when asserted. This involves two cases.

   (a) If this is the first time through the loop, the assumptions are that the Precondition was true, Test as true before the while-loop body was run, and Test is again true after the while-loop body was run. You prove that under these assumptions Loop Invariant is still true. In symbols:

$$R\,!\ ;\ T\ ;\ S\ ;\ T\ \ \Rightarrow\ \ I\,! \tag{2}$$

   (b) Else this is not the first time through the loop. Assume that the Loop Invariant was true before this pass through the loop and that test was true before and after this pass of the loop,

$$I\,!\ ;\ T\ ;\ S\ ;\ T\ \ \Rightarrow\ \ I\,! \tag{3}$$

3. You show that the Postcondition is true when asserted. Again, this involves two cases.

   (a) The loop is never run. The Postcondition must follow from the Precondition and the Test being false:

   $$R\,! \;;\; \neg T \;\;\Rightarrow\;\; O\,! \tag{4}$$

   (b) The loop is run. You must show the Postcondition follows from the previous assertion of the Loop Invariant and that the Test has changed from true to false during this pass of the loop:

   $$I\,! \;;\; T \;;\; S \;;\; \neg T \;\;\Rightarrow\;\; O\,! \tag{5}$$

Now the Postcondition follows from the Precondition no matter how many times through the while-loop. The proof is by mathematical induction and is left as an exercise.

Let us take as an example the procedure `list-list` in Figure 1. We assume l equals nil signifies an empty list. Let us prove the program correct using loop invariants.

1. Assert Precondition. Either l is nil or it is not. If it is nil then we must show that Postcondition is true. The Postcondition is that all elements have been written but l is nil means the list is empty. So we have written all its elements. If l is not nil then we must show that Loop Invariant is true. Pointer l points to the first element of the list so there are no elements before l and nothing has been written. So the Loop Invariant is true. Therefore $R\,!$.

2. Assert Loop Invariant. If the while-loop is entered the first time, the Precondition tells us the the Loop Invariant is true when the loop is first entered. So we only have to prove formula 3 — formula 2 follows automatically. Assume $I\,!$ and T. Before the code S, pointer l points to an unwritten element and all elements before this in the list have been written. The code S writes this element and advances the pointer. After advancing, we assume T, so l points to another element in a list. Therefore, l now points to an unwritten element and all elements before this in the list have been written, i.e. $I\,! \;;\; T \;;\; S \;;\; T \;\Rightarrow\; I\,!$.

3. Assert Postcondition. Suppose the loop is never run, that is T is false when the Precondition is asserted. Then the Precondition reduces to

```
 procedure list-list( l : listPntr ) ;
 begin
   {Assert Precondition:}
   {  If (l<>nil) then (Loop Invariant) }
   {  else (Postcondition) }
   while l<>nil do
   begin
     writeln_string(l^.str) ;
     l := l^.next ;
     {If l<>nil then Assert Loop Invariant: }
     {   Pointer l points to an element of the list}
     {   which has not yet been written; but all the }
     {   elements in the list ahead of this one have }
     {   been written.}
   end ;
   {Assert Postcondition:}
   {   All elements of the list have been written.}
end ;
```

Figure 1: The procedure list-list

the Postcondition. So $R! \; ; \; \neg T \;\; \Rightarrow \;\; O!$. If the loop was run, then before the last time through the loop, the Loop Invariant was true. That is, l pointed to an unwritten element and all elements before this in the list have been written. The code S wrote this element and advanced to pointer to its present value, which is nil. So we have just listed the last element in the last and all previous elements have been already listed. Hence $I! \; ; \; T \; ; \; S \; ; \; \neg T \;\; \Rightarrow \;\; O!$.

Exercises

1. Use mathematical induction to proof that the method of loop invariants works. That is, show that from formulas 1–5 follow $R! \;\; \Rightarrow \;\; O!$ for any $n \geq 0$.

2. Prove the following program is correct using the method of loop invariants.

```
function max-list( l:listPntr ) : integer ;
var
   i : integer ;
begin
   if l=nil then {error} i := -1 ;
   else begin
     i := l^.number - 1;
     {Assert Precondition: }
     {   (l<>nil) and (i<max-list(l)) }
     while l<>nil do
     begin
       if l^.number > i then i:= l^.number;
       l := l^.next ;
       {if l<>nil then assert Loop Invar.}
       {  l points to a list element not yet }
       {  looked at, and i holds the largest }
       {  number on the list before l }
     end
     {Assert Postcond. i is the largest number }
     {  on the list. }
   end ;
   max-list := i ;
 end ;
```

3. Write and prove correct using the method of loop invariants a program which puts a list of integers in ascending order by repeatedly finding the largest integer in the input list, moving it to the head of the output list, deleting it from the input list.