Iteration and Loop Invariants

Murali Sitaraman (Clemson)
Bruce W. Weide (Ohio State)

RESOLVE/Reusable Software Research Group
http://www.cs.clemson.edu/group/resolve
http://cse.osu.edu/rsrg

We gratefully acknowledge NSF grants CCF-0811748, CCF-1161916, CCF-1162331, CNS-0745846, DUE-0942542, DUE-1022191, DUE-1022941, and DUE-1022191.
Iteration and Loop Invariants

• Developer must sometimes write a justification of correctness of code
  – Assertion that the developer claims to be true and that will be checked to be true by the verifier (for soundness)
  – Proving and using such an assertion is easier for a mechanical verifier than inventing it
  – Writing it requires education and experience

• A loop invariant is one kind of justification
What Is a Loop Invariant?

• For a typical while loop, an assertion that is true at the beginning and at the end of each iteration, including the first and the last

• More generally: an invariant is a property that is true every time execution reaches a certain point—in the case of a loop invariant, the loop condition test
Programmers Supply Invariants

• In general, abstract loop invariants cannot be inferred automatically
• So every modern verification system (see Klebanov, et al., FM 2011) demands programmer supply loop invariants
• Automated systems check the validity of the invariants and use them in verification
Identifying a Suitable Invariant

• Logical Approach: Suppose Q is the assertion to be proved after the loop. Suppose B is the loop condition. Find an invariant Inv so that:
  \( \text{not } B \land \text{Inv } \Rightarrow Q \)

• Tracing Approach: Trace the code for multiple iterations on some input and note the values of variables at the beginning of each iteration. Find a relationship among the values that holds for all iterations.
Logical Approach

• Pick an assertion such that the assertion and the negation of loop condition together imply the assertion to be confirmed after the loop (i.e., not B and Inv ⇒ Q)

• Make sure the assertion is indeed an invariant (see checking the invariant, parts I and II)!
Example

• Click on Item Components to bring up Finder
• Select Programs in the Finder
• Select the activity on reasoning with Stack objects
• The above activity includes the code on the next slide
Example

**Operation** Flip(updates S: Stack);

**ensures** S = Reverse(#S);

**Procedure**

Var Temp: Stack;

Var Next_Entry: Entry;

While ( 1 <= Depth(S) )

**maintaining** ??? -- loop invariant goes here

**decreasing** |S|;

do

Pop(Next_Entry, S);

Push(Next_Entry, Temp);

end;

Temp :=: S;

end Flip;
Identifying a Suitable Invariant

• Need to confirm (or prove) the *ensures* clause of Flip at the end of the code:
  \[ S = \text{Reverse}(#S) \]

• Need to confirm the following before the swap statement (\[ S :=: \text{Temp}; \])
  \[ \text{Temp} = \text{Reverse}(#S) \]

• Identify an invariant, Inv:
  \[ S = \text{Empty\_String and Inv } \Rightarrow \]
  \[ \text{Temp} = \text{Reverse}(#S) \]
Tracing Approach: Observing the Loop in Action

<table>
<thead>
<tr>
<th>After Iteration Number...</th>
<th>$S =$</th>
<th>$Temp =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$&lt; 10, 20, 30, 40 &gt;$</td>
<td>$&lt; &gt;$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Observing the Loop in Action

<table>
<thead>
<tr>
<th>After Iteration Number…</th>
<th>$S =$</th>
<th>$Temp =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt; 10, 20, 30, 40 &gt;</td>
<td>&lt; &gt;</td>
</tr>
<tr>
<td>1</td>
<td>&lt; 20, 30, 40 &gt;</td>
<td>&lt; 10 &gt;</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Observing the Loop in Action

<table>
<thead>
<tr>
<th>After Iteration Number...</th>
<th>$S =$</th>
<th>$Temp =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$&lt; 10, 20, 30, 40 &gt;$</td>
<td>$&lt; &gt;$</td>
</tr>
<tr>
<td>1</td>
<td>$&lt; 20, 30, 40 &gt;$</td>
<td>$&lt; 10 &gt;$</td>
</tr>
<tr>
<td>2</td>
<td>$&lt; 30, 40 &gt;$</td>
<td>$&lt; 20, 10 &gt;$</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Observing the Loop in Action

<table>
<thead>
<tr>
<th>After Iteration Number...</th>
<th>$S =$</th>
<th>$Temp =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$&lt; 10, 20, 30, 40 &gt;$</td>
<td>$&lt; &gt;$</td>
</tr>
<tr>
<td>1</td>
<td>$&lt; 20, 30, 40 &gt;$</td>
<td>$&lt; 10 &gt;$</td>
</tr>
<tr>
<td>2</td>
<td>$&lt; 30, 40 &gt;$</td>
<td>$&lt; 20, 10 &gt;$</td>
</tr>
<tr>
<td>3</td>
<td>$&lt; 40 &gt;$</td>
<td>$&lt; 30, 20, 10 &gt;$</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Observing the Loop in Action

<table>
<thead>
<tr>
<th>After Iteration Number...</th>
<th>$S =$</th>
<th>$Temp =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$&lt; 10, 20, 30, 40 &gt;$</td>
<td>$&lt; &gt;$</td>
</tr>
<tr>
<td>1</td>
<td>$&lt; 20, 30, 40 &gt;$</td>
<td>$&lt; 10 &gt;$</td>
</tr>
<tr>
<td>2</td>
<td>$&lt; 30, 40 &gt;$</td>
<td>$&lt; 20, 10 &gt;$</td>
</tr>
<tr>
<td>3</td>
<td>$&lt; 40 &gt;$</td>
<td>$&lt; 30, 20, 10 &gt;$</td>
</tr>
<tr>
<td>4</td>
<td>$&lt; &gt;$</td>
<td>$&lt; 40, 30, 20, 10 &gt;$</td>
</tr>
</tbody>
</table>
What Doesn’t Change?

• Claim: the concatenation of the reverse of $\text{Temp}$, with $S$, is always the same: it is equal to the value of $S$ before the first iteration

• Loop invariant (goal driven):
  \[
  \text{Reverse}(S) \circ \text{Temp} = \text{Reverse}(\#S)
  \]

• Alternative loop invariant (fact driven or tracing based):
  \[
  \#S = \text{Reverse}(\text{Temp}) \circ S
  \]
Progress Metric for Termination

• Progress metric is a natural number (an ordinal, in general) that is decreasing

• Programmer specifies a progress metric for loop termination and the verifier checks its validity and uses it in its proof

• Suitable progress metric for example: decreasing $|S|$

• Unsuitable one: decreasing $|Temp|$
Activity:
Key Points Preview

• Ideas are language-independent
• Programmer supplies an adequate invariant and progress metric for termination to justify loop correctness
• Verifier checks the validity of the above (otherwise it won’t be sound)
• Verifier uses them to complete its proof of code correctness
Invariant_Activity_2;

-- This is an in-class activity for learning loop invariants
uses Integer_Template, String_Theory;
requires 5 <= max_int;

Facility Queue_Fac is Queue_Template(Integer, 5)
realized by Circular_Array_Realiz;

Operation Inject_at_Front(alters E: Integer; updates Q: Queue);
requires 1 + |Q| <= 5;
ensures Q = <E> o #Q;

Procedure
Var T: Queue;
Enqueue(E, T);
while ( 1 <= Length(Q) )
   maintaining true;
   decreasing |Q|;
do
   Dequeue(E, Q);
   Enqueue(E, T);
end;
Q ::= T;
end Inject_at_Front;

end Invariant_Activity_2;

This research is funded in part by NSF grants DUE-0633506, DMS-0701187, CCF-0811748, DUE-1022941, and CCF-1161916.