Loops and invariants

CSE 331
University of Washington

Michael Ernst
Reasoning about loops

A loop represents an unknown number of paths
Case analysis is problematic
Recursion presents the same problem as loops
Cannot enumerate all paths
This is what makes testing and reasoning hard

Things to prove about a loop:
1. It computes the correct value
2. It terminates (no infinite loop)
Reasoning about loops: values and termination

// assert x ≥ 0 & y = 0
while (x != y) {
    y = y + 1;
}
// assert x = y

Does “x = y” hold after this loop?
Does this loop terminate?
1) Pre-assertion guarantees that x ≥ y
2) Every time through loop
   x ≥ y holds at the test – and if body is entered, x > y
   y is incremented by 1
   x is unchanged
   Therefore, y is closer to x (but x ≥ y still holds)
3) Since there are only a finite number of integers between x and y, y will eventually equal x
4) Execution exits the loop as soon as x = y (but x ≥ y still holds)
Understanding loops by induction

We just made an inductive argument
  Inducting over the number of iterations
Computation induction
  Show that conjecture holds if zero iterations
  Show that it holds after \( n+1 \) iterations
    (assuming that it holds after \( n \) iterations)
Two things to prove
  1. Some property is preserved (known as “partial correctness”), if the code terminates
     Loop invariant is preserved by each iteration, if the iteration completes
  2. The loop completes (known as “termination”)
     The “decrementing function” is reduced by each iteration and cannot be reduced forever
How to choose a loop invariant, LI

{P}
while (b) S;
{Q}

Find an invariant, LI, such that
1. \( P \Rightarrow LI \)  // true initially
2. \( \{ LI \land b \} S \{ LI \} \)  // true if the loop executes once
3. \( (LI \land \neg b) \Rightarrow Q \)  // establishes the postcondition

Finding the invariant is the key to reasoning about loops

Fact: For any loop and pre/post conditions it satisfies, there exists a loop invariant sufficient to prove them
(Equivalently: inductive assertions is a “complete method of proof”)
(We will not yet prove loop termination; it is sufficient to know that if loop terminates, Q will hold.)
Loop invariant for the example

// assert x ≥ 0 & y = 0
while (x != y) {
    y = y + 1;
}
// assert x = y

A suitable invariant:

\[ LI = x \geq y \]

1. \[ x \geq 0 \ & \ y = 0 \Rightarrow LI \]  // true initially
2. \{ LI \ & \ x \neq y \} y = y+1; \{ LI \}  // true if the loop executes once
3. \{ LI \ & \ \neg(x \neq y)\} \Rightarrow x = y  // establishes the postcondition
Total correctness = partial correctness + termination
We have not established that the loop terminates
Suppose that the loop always reduces some variable’s value. Does the loop terminate if the variable is a
  – Natural number?
  – Integer?
  – Non-negative real number?
  – Boolean?
  – ArrayList?
The loop terminates if:
  • each variable value maps to a natural number, and
  • the number decreases on each iteration
Well-ordered sets

We don’t *have* to use the natural numbers
The loop terminates if the variable values are, or can be mapped to, (a subset of) a well-ordered set
  – Ordered set
    – Every non-empty subset has a least element
The natural numbers are the best choice
Decrementing function

Decrementing function $D(X)$
Maps state (program variables) to a natural number

Consider: while (b) $S$;
We seek $D(X)$, where $X$ is the state, such that

1. An execution of the loop reduces the function’s value:
   $$\{ \text{LI} \land b \} \ S \ \{ D(X_{\text{post}}) < D(X_{\text{pre}}) \}$$

2. If the function’s value is minimal, the loop terminates:
   $$(\text{LI} \land D(X) = 0) \ \Rightarrow \ \neg b$$
Proving termination

// assert x ≥ 0 & y = 0
// Loop invariant: x ≥ y
// Loop decrements: (x-y)
while (x != y) {
    y = y + 1;
}
// assert x = y

Is this a good decrementing function?
1. Does the loop reduce the decrementing function’s value?
   // assert (y ≠ x); let d_{pre} = (x-y)
   y = y + 1;
   // assert (x_{post} - y_{post}) < d_{pre}
2. If the function has minimum value, does the loop exit?
   (x ≥ y & x - y = 0) ⇒ (x = y)
Choosing loop invariants

For non-looping code, the wp (weakest precondition) function enables proofs

For loops, you have to guess:
- The loop invariant
- The decrementing function

Then, use reasoning techniques to prove the goal property

If the proof doesn't work:
- Maybe you chose a bad invariant or decrementing function
  Choose another and try again
- Maybe the loop is incorrect
  Fix the code

Automatically choosing loop invariants is a research topic
When to use code proofs for loops

Most of your loops need no proofs

```java
for (String name : friends) {
   ... 
}
```

When you are unsure about a loop,
write loop invariants and decrementing functions

If a loop is not working:
  Add invariant and decrementing function if missing
  Write code to check them
  Understand why the code doesn't work
  Reason to ensure that no similar bugs remain
Example: Factorial

\{ n \geq 0 \land t = n \} \\
\textbf{r} = 1; \\
\textbf{while} \ (n \neq 0) \ { \\
\ \\
\ \\
\ \\
\textbf{r} = \textbf{r} \ast n; \\
\textbf{n} = \textbf{n} - 1; \\
\} \\
\{ \textbf{r} = t! \} \\

\textbf{n} \geq 0 \land t = n \land r = 1 \\
\textbf{r} = t! / n! \land t \geq n \geq 0 \\
\textbf{r} = t! / (n-1)! \land t \geq n > 0 \\
\textbf{r} = t! / n! \land t \geq n \geq 0
Example: Quotient and remainder

\[ r := x; \quad x = x + y \times 0 \]
\[ q := 0; \quad x = r + y \times 0 \]
\[ \text{while} \ (y \leq r) \ \{ \]
\[ \quad r := r - y; \]
\[ \quad q := 1 + q; \]
\[ \}\]
\[ \{ x = r + y \times q \text{ and } y > r \} \]
Example: Greatest common divisor

\{ x_1 > 0 \land x_2 > 0 \}
y_1 := x_1;
y_2 := x_2;
while (y_1 \neq y_2) do
    if y_1 > y_2
        then y_1 := y_1 - y_2
    else y_2 := y_2 - y_1
\{ y_1 = \gcd(x_1, x_2) \}

Recall: if \( y_1, y_2 \) are both positive integers, then:
- If \( y_1 > y_2 \) then \( \gcd(y_1, y_2) = \gcd(y_1 - y_2, y_2) \)
- If \( y_2 > y_1 \) then \( \gcd(y_1, y_2) = \gcd(y_1, y_2 - y_1) \)
- If \( y_1 - y_2 \) then \( \gcd(y_1, y_2) = y_1 = y_2 \)
Invariants help you write code as well as prove it correct

Dutch National Flag problem: Given an array containing balls of three colors, arrange them with like colors together and in the right order

- **Precondition:**
- **Postcondition:**
- **Loop invariant:**

Mixed colors: red, white, blue

<table>
<thead>
<tr>
<th>Red</th>
<th>White</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>White</td>
<td>Blue</td>
</tr>
<tr>
<td>Red</td>
<td>White</td>
<td>Mixed</td>
</tr>
<tr>
<td>Red</td>
<td>White</td>
<td>Blue</td>
</tr>
</tbody>
</table>