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
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
  Some property is preserved (known as “partial correctness”),
    if the code terminates
    Loop invariant is preserved by each iteration, if the iteration completes
  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

It is sufficient to know that if loop terminates, Q will hold.

Finding the invariant is the key to reasoning about loops.

Inductive assertions is a “complete method of proof”:

If a loop satisfies pre/post conditions, then there exists an invariant sufficient to prove it
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. \( x \geq 0 \ & \ y = 0 \Rightarrow LI \)  \hspace{1cm} // true initially
2. \{ LI \ & \ x \neq y \} y = y + 1; \{ LI \}  \hspace{1cm} // true if the loop executes once
3. \( LI \ & \ \neg(x \neq y) \Rightarrow x = y \)  \hspace{1cm} // establishes the postcondition
Total correctness via well-ordered sets

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 the variable values are (a subset of) a well-ordered set

– Ordered set
– Every non-empty subset has least element
Decrementing function

Decrementing function $D(X)$
Maps state (program variables) to some well-ordered set
Tip: always use the natural numbers
This greatly simplifies reasoning about termination

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:
   `{ LI & b } S { D(X_{\text{post}}) < D(X_{\text{pre}}) }$
2. If the function’s value is minimal, the loop terminates:
   $(LI & D(X) = \text{minVal}) \Rightarrow \neg b$
// 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 straight-line code, the wp (weakest precondition) function gives us the appropriate property.

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) { ... }
```

Write loop invariants and decrementing functions when you are unsure about a loop

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

\[
\{ n \geq 0 \land t = n \} \\
r = 1; \\
\textbf{while} \ (n \neq 0) \ { \\
\quad r = r \times n; \\
\quad n = n - 1; \\
\} \\
\{ r = t! \}
\]
Example: Quotient and remainder

\[ r := x; \quad x = x + y \times 0 \]

\[ q := 0; \quad x = r + y \times 0 \]

while \( y \leq r \) {

\[ r := r - y; \quad x = r + y \times q \]

\[ q := 1 + q; \]

}\{ x = r + y \times q \text{ and } y > r \}
Example: Greatest common divisor

\{ x1>0 \land x2>0 \}

\begin{align*}
y1 &:= x1; \\
y2 &:= x2; \\
\text{while } \neg (y1=y2) \text{ do} \\
& \quad \text{if } y1>y2 \text{ then } y1 := y1 - y2 \\
& \quad \quad \text{else } y2 := y2 - y1 \text{ fi} \\
\text{od} \\
\{ y1 = \gcd(x1,x2) \}
\end{align*}

Recall: if \( y1, y2 \) are both positive integers, then:
\begin{itemize}
  \item If \( y1 > y2 \) then \( \gcd(y1,y2) = \gcd(y1-y2,y2) \)
  \item If \( y2 > y1 \) then \( \gcd(y1,y2) = \gcd(y1,y2-y1) \)
  \item If \( y1-y2 \) then \( \gcd(y1,y2) = y1 = y2 \)
\end{itemize}
Dutch National Flag

• Given an array containing balls of three colors, arrange them with like colors together and in the right order

• Precondition:

• Postcondition:

• Loop invariant: