Hoare Logic, continued

Reasoning About Loops

Announcements

- HW0 due today
  - Commit to SVN then Submit in the HW Server
- HW1 will be up after class
  - Check Homeworks for announcement. Then Update src in Eclipse
- QUIZ 1 today. You have 10 minutes

Backward Reasoning:

Rule for Assignment

{ wp("x=expression",Q) }

\[
x = expression;
{ Q }
\]

Rule: the weakest precondition

wp("x=expression",Q) is Q with all

occurrences of x in Q replaced by

expression

Backward Reasoning:

Rule for Sequence

// find weakest precondition for sequence S1;S2 and Q

{ wp(S1,wp(S2,Q)) }

S1; // statement

{ wp(S2,Q) }

S2; // another statement

{ Q }

Simple Example

\[
\{ x + 1 + y > 1 \} \text{ equiv. to } \{ x + y > 0 \}
\]

x = x + 1;

\[
\{ x + y > 1 \}
\]

y = x + y;

\[
\{ y > 1 \}
\]

Rule for If-then-else

// wp: ?? \((b && \text{wp}(S1,Q)) || (\neg b && \text{wp}(S2,Q))\)

if (b) {
  S1;
}

else {
  S2;
}

\[
\{ Q \}
\]

Fall 15 CSCI 2600, A Milanova
Proving Correctness

Goal: Prove that \( \{ P \} \) code \( \{ Q \} \) is a valid triple

- Backward reasoning
  - derive \( \{ P' \} \)
  - code
  - derive \( \{ Q' \} \)
Then show \( P' \Rightarrow P \)

- Forward reasoning:
  - \( \{ P \} \) code
  - \( \{ Q \} \)
Then show \( Q' \Rightarrow Q \)

What Happens When There Is a Loop

Precondition: \( x \geq 0 \);

\[
\begin{align*}
i &= x; & (x \geq 0 \land i = x) \\
z &= 0; & (x \geq 0 \land i = x \land z = 0) \\
\text{while } (i \neq 0) & \\
\{ z = z+1; & \\
i &= i-1; &
\}
\]

Does the postcondition hold? Yes.
The key is to guess the loop invariant.

Outline

- Reasoning about loops
- Total correctness = partial correctness + termination
- Loop invariants
- Computation induction (to prove partial correctness)
- Decrementing function (to prove termination)

Reasoning About Loops

- Reasoning about loops is difficult
  - Unknown number of iterations and unknown number of paths. Recursion poses similar issues
  - We cannot enumerate all paths
  - Key is to guess a loop invariant
- Two things to prove about loops
  1. It computes correct values (partial correctness)
  2. It terminates (does not go into infinite loop)

Example: Partial Correctness + Termination

Precondition: \( x \geq 0 \);

\[
\begin{align*}
i &= x; \\
z &= 0; \\
\text{while } (i \neq 0) & \\
\{ z = z+1; \\
i &= i-1; \\
\}
\]

Postcondition: \( x = z \);

I. If the loop terminated does \( x = z \) hold?
II. Does loop terminate?

Reasoning About Loops by Induction

- \( i+z = x \) is a loop invariant (a fact that holds true before and after each loop iteration)
- Even though \( i \) and \( z \) change, \( i+z=x \) stays true
- We just made an inductive argument over the number of iterations of the loop

\[
\begin{align*}
\text{1. Loop terminates. The precondition } & x \geq 0 \text{ guarantees } i \geq 0 \text{ before loop. At every iteration } i \text{ decreases by 1, thus it eventually reaches 0.}
\end{align*}
\]
Reasoning About Loops by Induction

1. Partial correctness
   - Establish and prove loop invariant using computation induction
   - Loop exit condition and loop invariant must imply the desired postcondition
     - \( i = 0 \) (loop exit condition) and \( i + z = x \) imply \( z = x \)

2. Termination
   - (Roughly) Establish "decrementing function" \( D \). \( D \geq 0 \) before loop, each iteration decrements \( D \), loop invariant and \( D \) at 0 imply loop exit condition

We will discuss Partial Correctness first

Another Example: Partial Correctness

\[ \sum_{j=0}^{i-1} arr[j] \] stands for \( arr[0]+arr[1]+\ldots+arr[i-1] \)
Loop invariant: \( i \leq len \&\& \sum = \sum_{j=0}^{i-1} arr[j] \)
1) \( i \leq len \&\& \sum = \sum_{j=0}^{i-1} arr[j] \) holds before loop
2) Assume \( \sum = \sum_{j=0}^{i-1} arr[j] \) holds after \( k \)th iteration
   \[ \sum_{new} = \sum + arr[i] \]
   \[ i_{new} = i+1 \]
   Thus \( \sum = \sum_{j=0}^{i-1} arr[j] \) holds after \( (k+1) \)st iteration
   \( i \leq len \) also holds (had we had \( i = len \) after \( k \)th iteration, there wouldn’t have been a \( (k+1) \)st iteration!)

Another Example: Termination

Precondition: \( len \geq 0 \&\& arr.length = len \)
\[\text{int } \sum = 0.0;\]
\[\text{int } i = 0;\]
\[\text{while (}\langle i < len \rangle \{ \]
\[\text{sum = sum + arr[i];}\]
\[i = i+1;\]
\[\}\}\]
4) We now argue termination (informally).
   At each iteration \( i \) increases by one, while \( len \) stays the same. Thus, eventually \( i \) reaches \( len \).
   \( i = len \) implies loop exit condition.
   More on termination later!

Another Example: Partial Correctness + Termination

Let us prove that \( \sum \) is correct:
Precondition: \( len \geq 0 \&\& arr.length = len \)
\[\text{int } \sum = 0;\]
\[\text{int } i = 0;\]
\[\text{while (}\langle i < len \rangle \{ \]
\[\text{sum = sum + arr[i];}\]
\[i = i+1;\]
\[\}\}\]
Postcondition: \( \sum = arr[0]+\ldots+arr[arr.length-1] \)

Another Example: Partial Correctness

\[ \sum_{j=0}^{i-1} arr[j] \] stands for \( arr[0]+arr[1]+\ldots+arr[i-1] \)
3) \( i < len \) which is \( i > len \) plus loop invariant
   \[ i \leq len \&\& \sum = \sum_{j=0}^{i-1} arr[j] \]
   imply
   \[ \sum = \sum_{j=0}^{i-1} arr[j] , \text{ which with precondition} \]
   \[ \text{arr.length} = len \text{ gives us desired postcondition} \]
   \[ \sum = arr[0]+\ldots+arr[arr.length-1] \]
   We still must argue termination!

Partial Correctness, More Formally

\{ P \} while (b) S \{ Q \}
We need to "guess" a loop invariant \textit{Inv} such that
1) \( P \Rightarrow \text{Inv} \) // \textit{Inv} holds before loop. Base case
2) \{ b \&\& \textit{Inv} \} S \{ \textit{Inv} \} // Assuming \textit{Inv} held after \( k \)th iteration and execution took a \( (k+1) \)st iteration, then \textit{Inv} holds after \( (k+1) \)st iteration
3) \( \neg b \&\& \textit{Inv} \) \Rightarrow Q // The exit condition and loop invariant imply desired postcondition
Choosing Loop Invariant

What is a suitable loop invariant?

Precondition: x >= 0;

\[
i = x;
\]
\[
z = 0;
\]
while (i != 0) {
\[
z = z+1;
\]
\[
i = i-1;
\]
}

Postcondition: x=z

Fall 15 CSCI 2600, A Milanova

Choosing Loop Invariant

What is a suitable loop invariant?

Precondition: x >= 0 && y = 0

while (x != y) {
\[
y = y+1;
\]
}

Postcondition: x=y

Fall 15 CSCI 2600, A Milanova

Choosing Loop Invariant

What is a suitable loop invariant?

Precondition: len ≥ 0 && arr.length = len

int sum = 0;
int i = 0;
while (i < len) {
    sum = sum + arr[i];
    i = i+1;
}

Postcondition: sum = arr[0]+...+arr[arr.length-1]

Fall 15 CSCI 2600, A Milanova

Choosing Loop Invariant

What is a suitable loop invariant?

Precondition: n >= 0

i = 0;
while (i < n) {
\[
i = i+1;
\]
\[
r = r*i;
\]
}

Postcondition: r = n!

Fall 15 CSCI 2600, A Milanova

A “tedious” Invariant

Inv: sum = a[0]+a[1]

Precondition: len ≥ 0 && a.length = len
int sum = 0;
int i = 0;
while (i < len) {
    invariant sum = a[0]+...+a[i-1] && i<=len
    sum = sum + a[i];
    i = i+1;
}
Postcondition: sum = a[0]+...+a[a.length-1]

Fall 15 CSCI 2600, A Milanova

A “tedious” Invariant

Inv: sum = a[0]+a[1]+a[2]

Precondition: len ≥ 0 && a.length = len
int sum = 0;
int i = 0;
while (i < len) {
    invariant sum = a[0]+...+a[i-1] && i<=len
    sum = sum + a[i];
    i = i+1;
}
Postcondition: sum = a[0]+...+a[a.length-1]
As the induction variable $i$ moves through the array, $sum$ contains the correct partial result. At the end of the array, $sum$ contains the correct total. Similar invariants for min, max, avg, sorting algorithms, etc. These invariants are easy to guess.

An “tedious” Invariant

Inv: \(i+z = x\)

A “tedious” Invariant

Inv: \(i+z = x\)

An “elegant” Invariant

What is a suitable loop invariant?

Precondition: $x \geq 0$;
\[ i = x; \]
\[ z = 0; \]
\[ \text{while} \ (i \neq 0) \ { \}
\[ \quad z = z+1; \]
\[ \quad i = i-1; \]
\[ \} \]
Postcondition: $x=z$

Choosing Decrementing Function

What is a suitable decrementing function?

Precondition: $x \geq 0$;
\[ i = x; \]
\[ z = 0; \]
\[ \text{while} \ (i \neq 0) \ { \}
\[ \quad z = z+1; \]
\[ \quad i = i-1; \]
\[ \} \]
Postcondition: $x=z$

Choosing Decrementing Function

What is a suitable decrementing function?

Precondition: $x \geq 0 \& y = 0$
\[ \text{while} \ (x \neq y) \ { \}
\[ \quad y = y+1; \]
\[ \} \]
Postcondition: $x=y$
Choosing Decrementing Function

What is a suitable decrementing function?

Precondition: len ≥ 0

\[ D = \text{len} - i \]

A arr.length = len

double sum = 0.0;

int i = 0;

while (i < len) {
    sum = sum + arr[i];
    i = i+1;
}

Postcondition: sum = arr[0]+...+arr[arr.length-1]

Choosing Decrementing Function

What is a suitable decrementing function?

Precondition: n >= 0

i = 0;

r = 1;

while (i < n) {
    i = i+1;
    r = r*i;
}

Postcondition: r = n!

Reasoning About Loops is Difficult

For non-looping code, weakest precondition enables proof

For loops, we have to guess

- The loop invariant
- The decrementing function

Then use the proof techniques we discussed

- If the proof doesn’t work
  - Maybe you chose wrong invariant/function: Fix
  - Maybe the loop is incorrect!

Why Study Formal Proofs

- Helps us write correct programs!
- Most of the time, our loops don’t need proofs
  for i in seq: print i

- Sometimes we have to write more complex loops
  - Establish precondition and postcondition
  - Establish a loop invariant
  - Reason about correctness and termination

Reasoning About Loops, Recap

Total correctness = partial correctness + termination

1. Partial correctness
   - “Guess” and prove loop invariant
   - Loop exit condition and loop invariant must imply the postcondition
   - This gives us: “If the loop terminated then the postcondition holds”. But does the loop terminate?

2. Termination
   - (Roughly) “Guess” decrementing function D. Each iteration decrements D, D at its minimum value (usually 0) must imply loop exit condition
Example: Another Factorial

Precondition: n >= 0
r = 1;
n = t;
while (n != 0) {
    r = r*n;
    n = n-1;
}
Postcondition: r = t!

Example: Integer Division x/y

Precondition: x >= 0 && y >= 0
r = x;
q = 0;
while (y <= r) {
    r = r-y;
    q = q+1;
}
Postcondition: x = y*q + r && r < y

Example: Greatest Common Divisor

Precondition: x1 > 0 && x2 > 0
y1 = x1;
y2 = x2;
while (y1 != y2) {
    if (y1 > y2) {
        y1 = y1-y2
    } else {
        y2 = y2-y1;
    }
}
Postcondition: y1 = gcd(x1,x2)