Outline: How to prove a program to be correct

- Precondition and postcondition
- Correctness of assignment statement
- Correctness of conditional statement
- Correctness of loop statement
- Loop invariant
- Example of Euclidean algorithm for GCD computation
- **Text book Sections 1.6 and 2.3**

Proofs of Program Correctness

- Secs 1.6 and 2.3 in Gersting book
- Program correctness?
  - correct = meets specification
  - is specification correct?
- Correctness can be checked by:
  - testing
  - proof
- Can testing guarantee correctness?

Example

- Consider the assignment statement
  - x = y + 10
- What must be true after the assignment?
- Suppose we want to know x = 14 after
  - What must be true before to ensure this?

Preconditions and Postconditions

- Separate assertions can specify what must be true <u>before</u> and <u>after</u> a program fragment is run.
- A *Hoare triple* gives before and after conditions for a program fragment:
  - written {Q} P {R}
  - Q is the <u>precondition</u>
  - P is the program fragment
  - R is the <u>postcondition</u>
- Means "if *Q* is true and *P* is executed, then *R* will be true"

Illustration of Hoare triple

Q(X) precondition

Y = P(X) program

R(X, Y) postcondition

Means

$(\forall X)\ Q(X) \rightarrow R(X, Y)$

$(\forall X)\ Q(X) \rightarrow R(X, P(X))$

- For a program to calculate square root of positive integers $(\forall x)(x>0 \rightarrow [P(x)]^2=x)$

Proving program correctness

- The program can be proved to be correct by successively proving preconditions and postconditions for each statement

{Q}
s_0
$\{R_1\}$
s_1
$\{R_2\}$
s_2
…
s_{n-1}
{R}

Example revisited

{y=4}
x=y+10
{x=14}

more concisely, {y=4} x=y+10 {x=14}

There is no single correct Hoare triple for a given program fragment

- It depends on what your goal is.
- Often easiest to check correctness backwards from the goal (from the end back)

Mechanical correctness checking

- We can make simple “proof rules” for checking correctness.
- One rule for each kind of program statement.
- For assignment, {Q} x=e {R} is correct if:
  - Q is the same as R except that everywhere x occurs it is replaced by e.

Example

- For assignment, {Q} x=e {R} is correct if:
  - Q is the same as R except that everywhere x occurs it is replaced by e.
- {y=4}
  x=y+10
  {x=14} ?? doesn't check
- {y+10=14}
  x=y+10
  {x=14} does check

Arithmetic Simplification

- We can put arithmetically equivalent assertions in sequence with no lines of code in between:
- {y=4}
  {y+10=14}
  x=y+10
  {x=14}

arithmetic simplification verifies the second from the first, so sequence is OK.

Example: Assignment Rule

{x = 2}
  y = x+2;
  y = 2*y;
{y = 8}
	Prove that the following computes x(x-1) correctly.
	  y = x-1;
	  y = x*y;

Correctness of Conditional Statements

- if x<0 then y=–x else y=x
- Suppose we want to know y>0 afterwards?
- { ?? }
  if x<0 then y=–x else y=x
  {y>0}

Correctness of Conditional Statements

- {Q} "if B then P_1 else P_2" {R} holds when
  - {Q and B} P_1 {R} and
  - {Q and ¬B} P_2 {R} both hold

- { ?? } ← ?? is $x \neq 0$

  if x<0 then y=–x else y=x

  {y>0}

- {?? ∧ x<0 } ← {–x>0} = {x<0}, so ?? can be empty

  y=–x

  {y>0}

  {?? ∧ x≥0} ← {x>0} = {x≠0 ∧ x≥0}, so ?? is $x \neq 0$

  y=x

  {y>0}

Example: Conditional Rule

Verify the correctness of the following program.

{x = 7}

if (x ≤ 0) y = x;

else y = 2*x;

{y = 14}

Example: Assignment & Conditional Rule

Verify the correctness of the following program.

```
{x = 11}
  y = x-1;
{y = 10}
  if (y ≤ 0) z = y-1;
  else z = y+3;
{z = 13}
```


Correctness of Looping Programs

- **while** B **do**
  S;
  end while
- Repeatedly perform statement S until B is false.
- How can we analyze this?
  - {Q}{R} ??
  - {Q}S{R}
  - {Q}S;S{R}
  - {Q}S;S;S{R}
  - …???

Loop Invariants

- **while** B **do**
  S;
  end while

- A *loop invariant* is an assertion that will be true before each execution of S.
  - The execution of S is preserving the invariant.
  - Show {Q ∧ B} S {Q}, where Q is the loop invariant

- The invariant together with ¬B should imply the conclusion you want verified on exit from the loop.

Loop Invariants Example

- *// Summing up* 0 + 1 + *…* + *n*–1
  while i ≠ n **do**
  j = j + i;
  i = i + 1;
  end while

- Invariant: j = sum of 0 … i–1

- At termination, we have
  (j = sum of 0 … i–1) and i = n
  So, j = sum of 0 …n–1, as desired.

- Precondition?
  - j = sum of 0 … i–1…loop invariant must hold on entry.

Loop Invariants Example, continued

- *// Summing up* 0 + 1 + … + *n*–1

  {j = sum of 0 … i–1}
  while i ≠ n **do**
  j = j + i;
  i = i + 1;
  end while
  {(j = sum of 0 … i–1) ∧ i = n}

- To prove this is correct, we must still show
  - {(j = sum of 0 … i–1) ∧ i ≠ n}
    j = j + i;
    i = i + 1;
    {j = sum of 0 … i–1}

Proof by induction

```
Sum(n)    // Calculate \sum(0…n-1)
i=1; j=0;
while (i≠n) do
  j = j+i;
  i = i+1;
end while
// j contains desired sum
```

Loop Correctness Rule

- If we have $\{Q \wedge B\}$ S $\{Q\}$
- We can derive

  $\{Q\}$

  while B **do** S

  $\{Q \wedge \neg B\}$
- Note: termination has not been proven.

Example — Euclidean Algorithm for GCD

GCD(non negative integer a, b)

// a ≥ b, not both a and b are zero

```
i=a
j=b
while j≠0 do
    r = i mod j
    i=j
    j=r
end while
```

{i = gcd(a,b)}

Example — Euclidean Algorithm for GCD

Find GCD(2420, 70)

Theorem Underlying Algorithm

- **Theorem**: GCD(i,j) = GCD(j, i mod j)
- **Proof**: See book page 114-115

Proof of Euclidean Algorithm

Loop invariant Q: gcd(i, j) = gcd(a, b)

Example: Loop Rule

- Function to return the value $x - y$ for $x, y \geq 0$

```
Difference(non-negative integers x, y)
i=0; j=x;
while (i≠y) do
   j = j-1;
   i = i+1;
end while
// j now has the value x-y
return j
```

Example: Loop Rule (Cont)

Steps:

1. Propose a loop invariant (*Q*) such that the loop invariant upon termination gives what you want

2. Show loop invariant (Q_0) holds upon first entry into loop

3. Prove loop invariant using induction (assume Q_k, prove Q_{k+1})