CS 477: Hoare Logic

Sasa Misailovic

Based on previous slides by Elsa Gunter, which were based on earlier slides by Gul Agha, and Mahesh Viswanathan

University of Illinois at Urbana-Champaign

Simple Search: Contract

@pre: s (start) has a non-negative value

@pre: u (upper) has a value smaller than the array length

```
bool LinearSearch(int[] a, int s, int u, int e) {
   for (int i := s; i ≤ u; i := i + 1) {
       if (a[i] = e) return true;
   }
   return false;
}
```

@post: function returns true iff there exists an element in the list that has the value e

Simple Search

@pre $0 \leq s \wedge u < |a|$

```
bool LinearSearch(int[] a, int s, int u, int e) {
   for (int i := s; i ≤ u; i := i + 1) {
       if (a[i] = e) return true;
   }
   return false;
}
```

@post: function returns true iff there exists an element in the list that has the value e

Simple Search

@pre $0 \leq s \wedge u < |a|$

```
bool LinearSearch(int[] a, int s, int u, int e) {
   for (int i := s; i ≤ u; i := i + 1) {
       if (a[i] = e) return true;
   }
   return false;
}
```

@post $ret_{\texttt{LinearSearch}} \Leftrightarrow \exists i.\ s \leq i \leq u \wedge a[i] = e$

Axiomatic Semantics

- Also called Floyd-Hoare Logic
- Based on formal logic (first order predicate calculus)
- Axiomatic Semantics is a logical system built from *axioms* and *inference rules*
- Mainly suited to simple imperative programming languages

Axiomatic Semantics

- Used to formally prove a property (*post-condition*) of the *state* (the values of the program variables) after the execution of program, assuming another property (*pre-condition*) of the state holds before execution

Axiomatic Semantics

- Goal: Derive statements of form

  $$\{P\}\ C\ \{Q\}$$

  - P , Q logical statements about state,
    P precondition,
    Q postcondition,
    C program

- Example: $\{x > 1\}\ x := x + 1\ \{x > 2\}$

Axiomatic Semantics

- *Approach:* For each kind of language statement, give an axiom or inference rule stating how to derive assertions of form

  {P} C {Q}

  where C is a statement of that kind

- Compose axioms and inference rules to build proofs for complex programs

Axiomatic Semantics

- An expression {P} C {Q} is a *partial correctness* statement
- For *total correctness* must also prove that C terminates (i.e. doesn't run forever)
  - Written: [P] C [Q]
- Will only consider partial correctness here

Language

- We will give rules for simple imperative language

```
<command> ::=
      <variable> := <term>
    | <command>; … ;<command>
    | if <expression> then <command>
      else <command> fi
    | while <expression> do <command> od
```

- Could add more features, like for-loops

Substitution

- Notation: P[e/v] (sometimes P[v <- e])
- Meaning: Replace every v in P by e
- Avoid capture!
- Example:

$$(x + 2)\ [y\text{-}1/x] = ((y - 1) + 2)$$

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Example:

$$\frac{}{\{\quad ? \quad\}\ x := y\ \{x = 2\}}$$

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Example:

$$\frac{}{\{\boxed{\_} = 2\}\ x := y\ \{\boxed{x} = 2\}}$$

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Example:

$$\frac{}{\{\boxed{y} = 2\}\ x := y\ \{\boxed{x} = 2\}}$$

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Examples:

$$\frac{}{\{y = 2\}\ x := y\ \{x = 2\}}$$

$$\frac{}{\{y = 2\}\ x := 2\ \{y = x\}}$$

$$\frac{}{\{x + 1 = n + 1\}\ x := x + 1\ \ \{x = n + 1\}}$$

$$\frac{}{\{2 = 2\}\ x := 2\ \{x = 2\}}$$

The Assignment Rule – Your Turn

- What is a valid precondition of

$$x := x + y \ \{x + y = w - x\}?$$

$$\{ \quad ? \quad \}$$

$$x := x + y$$

$$\{x + y = w - x\}$$

The Assignment Rule – Your Turn

- What is a valid precondition of

$$x := x + y \ \{x + y = w - x\}?$$

$$\{(x + y) + y = w - (x + y)\}$$
$$x := x + y$$
$$\{x + y = w - x\}$$

Precondition Strengthening

$$\frac{P \rightarrow P' \qquad \{P'\}\ C\ \{Q\}}{\{P\}\ C\ \{Q\}}$$

- Meaning: If we can show that P implies P' (P → P') and we can show that {P'} C {Q}, then we know that {P} C {Q}

- P is *stronger* than P' means P → P'

Precondition Strengthening

- Examples:

$$\frac{x = 3 \rightarrow x < 7 \quad \{x < 7\}\ x := x + 3\ \{x < 10\}}{\{x = 3\}\ x := x + 3\ \{x < 10\}}$$

$$\frac{\text{True} \rightarrow 2 = 2 \quad \{2 = 2\}\ x := 2\ \{x = 2\}}{\{\text{True}\}\ x := 2\ \{x = 2\}}$$

$$\frac{x=n \rightarrow x+1=n+1 \quad \{x+1=n+1\}\ x:=x+1\ \{x=n+1\}}{\{x=n\}\ x:=x+1\ \{x=n+1\}}$$

Which Inferences Are Correct?

$$\frac{\{x > 0 \;\&\; x < 5\}\ x := x * x\ \{x < 25\}}{\{x = 3\}\ x := x * x\ \{x < 25\}}$$

$$\frac{\{x = 3\}\ x := x * x\ \{x < 25\}}{\{x > 0 \;\&\; x < 5\}\ x := x * x\ \{x < 25\}}$$

$$\frac{\{x * x < 25\}\ x := x * x\ \{x < 25\}}{\{x > 0 \;\&\; x < 5\}\ x := x * x\ \{x < 25\}}$$

Which Inferences Are Correct?

$$\frac{\{x > 0 \;\&\; x < 5\}\ x := x * x\ \{x < 25\}}{\{x = 3\}\ x := x * x\ \{x < 25\}}\ \checkmark$$

$$\cancel{\frac{\{x = 3\}\ x := x * x\ \{x < 25\}}{\{x > 0 \;\&\; x < 5\}\ x := x * x\ \{x < 25\}}}$$

$$\frac{\{x * x < 25\}\ x := x * x\ \{x < 25\}}{\{x > 0 \;\&\; x < 5\}\ x := x * x\ \{x < 25\}}\ \checkmark$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

- Example:

$$\frac{\{z = z\ \&\ z = z\}\ x := z\ \{x = z\ \&\ z = z\} \qquad \{x = z\ \&\ z = z\}\ y := z\ \{x = z\ \&\ y = z\}}{\{z = z\ \&\ z = z\}\ x := z;\ y := z\ \{x = z\ \&\ y = z\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

- Example:

$$\frac{\{z = z\ \&\ z = z\}\ x := z\ \{x = z\ \&\ z = z\} \qquad \{x = z\ \&\ z = z\}\ y := z\ \{x = z\ \&\ y = z\}}{\{z = z\ \&\ z = z\}\ x := z;\ y := z\ \{x = z\ \&\ y = z\}}$$

Postcondition Weakening

$$\frac{\{P\}\ C\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ C\ \{Q\}}$$

Example:

$$\frac{\{z = z\ \&\ z = z\}\ x := z;\ y := z\ \{x = z\ \&\ y = z\} \qquad \mathbf{(x = z\ \&\ y = z) \Rightarrow (x = y)}}{\{z = z\ \&\ z = z\}\ x := z;\ y := z\ \mathbf{\{x = y\}}}$$

Rule of Consequence

$$\frac{P \rightarrow P' \qquad \{P'\}\ C\ \{Q'\} \qquad Q' \rightarrow Q}{\{P\}\ C\ \{Q\}}$$

- Logically equivalent to the **combination of** Precondition Strengthening and Postcondition Weakening
- Uses $P \rightarrow P'$ and $Q' \rightarrow Q$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \quad \{P \text{ and } (\text{not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ \{Q\}}$$

- Example: Want

$$\{y=a\}$$

if x < 0 then y:= y-x else y:= y+x fi

$$\{y=a+|x|\}$$

Suffices to show:

(1) {y=a&x<0} y:=y-x {y=a+|x|} and

(4) {y=a&not(x<0)} y:=y+x {y=a+|x|}

$$\{y=a \ \& \ x<0\} \ \ y:=y-x \ \ \{y=a+|x|\}$$

$$\begin{array}{ll} (3) & (y=a \ \& \ x<0) \Rightarrow y-x=a+|x| \\ (2) & \{y-x=a+|x|\} \ \ y:=y-x \ \ \{y=a+|x|\} \\ \hline (1) & \{y=a \ \& \ x<0\} \ \ y:=y-x \ \ \{y=a+|x|\} \end{array}$$

(1) Reduces to (2) and (3) by ***Precondition Strengthening***

(2) Follows from ***assignment*** axiom

(3) Because from algebra: $x<0 \Rightarrow |x| = -x$

$\{y=a \ \& \ not(x<0)\}\ y:=y+x\ \{y=a+|x|\}$

$$
\begin{array}{ll}
(6) & (y=a \ \& \ not(x<0)) \rightarrow (y+x=a+|x|) \\
(5) & \{y+x=a+|x|\}\ \ y:=y+x\ \ \{y=a+|x|\} \\
\hline
(4) & \{y=a \ \& \ not(x<0)\}\ \ y:=y+x\ \ \{y=a+|x|\}
\end{array}
$$

(4) Reduces to (5) and (6) by **Precondition Strengthening**

(5) Follows from **assignment** axiom

(6) Because $not(x<0) \rightarrow |x| = x$

If Then Else

$$\frac{\begin{array}{ll} (1) & \{y=a \;\&\; x<0\}\ y:=y-x\ \{y=a+|x|\} \\ (4) & \{y=a \;\&\; not(x<0)\}\ y:=y+x\ \{y=a+|x|\} \end{array}}{\begin{array}{c} \{y=a\} \\ \text{if } x < 0 \text{ then } y:= y-x \text{ else } y:= y+x \\ \{y=a+|x|\} \end{array}}$$

By the **IfThenElse rule**

While

- We need a rule to be able to make assertions about **while** loops.
  - Inference rule because we can only draw conclusions if we know something about the body
  - Let's start with:

$$\frac{\{\ ?\ \}\ \ C\ \ \{\ ?\ \}}{\{\ ?\ \}\ \ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \ \textbf{od}\ \ \{\ P\ \}}$$

While

- The loop may never be executed, so if we want P to hold after, it had better hold before, so let's try:

$$\frac{\{\ ?\ \}\quad C\quad \{\ ?\ \}}{\{\ P\ \}\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ \{\ P\ \}}$$

While

- If all we know is P when we enter the **while** loop, then all we know when we enter the body is (P and B)
- If we need to know P when we finish the **while** loop, we had better know it when we finish the loop body:

$$\frac{\{\ P \text{ and } B\}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P\ \}}$$

While

- We can strengthen the previous rule because we also know that when the loop is finished, **not B** also holds
- Final **while** rule:

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

While

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

- P satisfying this rule is called a ***loop invariant*** because it must hold **before and after each iteration of the loop**

While

- **While** rule generally needs to be <u>used together</u> with precondition strengthening and postcondition weakening

- There is **NO** **algorithm for computing the correct P**; it requires intuition and an understanding of why the program works

Counting up to n

$$\frac{\{P \text{ and } B\}\ C\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ \{P \text{ and not } B\}}$$

- P satisfying this rule is called a ***loop invariant*** because it must hold **before and after the each iteration of the loop**

```
n := 10; x := 0;
while (x < n) {
    x := x + 1
}
```

$P \equiv x \leq n$

Want to show: x >= n

Sum of numbers 1 to n

```
x := 0
y := 0

while y < n {
    y := y + 1;
    x := x + y
}
```

$$P \equiv \quad x = 1 + 2 + \ldots + y \;\wedge\; y \leq n \;\wedge\; 0 \leq n$$

Want to show: x = 1 + ... + n

Fibonacci

```
x = 0; y = 1;
z = 1;

while (z < n) {
    y := x + y;
    x := y - x;
    z := z + 1
}
```

$$P \equiv \quad y = fib(z) \;\wedge\; x = fib(z-1) \;\wedge\; z \leq n \;\wedge\; 1 \leq n$$

Want to show: y = fib(n)

List Length

```
x = lst; y = 0

while (x ≠ nil) {
    x := tail x;
    y := y + 1
}
```

P ≡ y + len (x) = len (lst)

Want to show: y = len (lst)

All Rules on One Slide

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \qquad \{P \text{ and (not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ \{Q\}}$$

Precondition Strengthening

$$\frac{P \rightarrow P' \qquad \{P'\}\ C\ \{Q\}}{\{P\}\ C\ \{Q\}}$$

Postcondition Weakening

$$\frac{\{P\}\ C\ \{Q'\} \qquad Q' \rightarrow Q}{\{P\}\ C\ \{Q\}}$$

Rule of Consequence

$$\frac{P \rightarrow P' \qquad \{P'\}\ C\ \{Q'\} \qquad Q' \rightarrow Q}{\{P\}\ C\ \{Q\}}$$

While

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

Example (Use of Loop Invariant in Full Proof)

- Let us prove

  when x is non-negative and has some value a

```
fact := 1;
while x > 0 do {
   fact := fact * x;
   x := x - 1
} od
```

  variable fact has the value a! (math. factorial)

Example (Use of Loop Invariant in Full Proof)

- Let us prove

```
{x>= 0 and x = a}

fact := 1;
while x > 0 do {
   fact := fact * x;
   x := x - 1
} od

{fact = a!}
```

Example

- We need to find a condition P that is true both before and after the loop is executed, and such that

(P and not x > 0) => (fact = a!)

Example

- First attempt:

$$\mathbf{P} \equiv \mathbf{a! = fact * (x!)}$$

- Motivation:
  - What we want to compute: **a!**
  - What we have computed: **fact**

    which is the sequential product of **a** down through **(x + 1)**
  - What we still need to compute: **x!**

Example

Postcondition Weakening

$$\frac{\{P\}\ C\ \{Q'\} \quad Q' \rightarrow Q}{\{P\}\ C\ \{Q\}}$$

By post-condition weakening suffices to show

1. $\{x>=0 \wedge x = a\}$

```
fact := 1;
while x > 0 do (fact := fact * x; x := x -1) od
```

$\{a! = fact * (x!) \wedge \neg (x > 0)\}$

And

2. $a! = fact * (x!) \wedge \neg(x > 0) \Rightarrow fact = a!$

Problem!! (Dead End)

2. $a! = fact * (x!) \ \wedge \ \neg(x > 0) \ \Rightarrow \ fact = a!$

- Don't know this if $x < 0$!!
  - Need to know that $x = 0$ when loop terminates

- **Need a new loop invariant**
  - Try adding $x >= 0$
  - Then will have $x = 0$ when loop is done

Example

Second try, let us combine the two:

$$P \equiv a! = fact * (x!) \wedge x >= 0$$

We need to show:

1. $\{x>=0 \wedge x = a\}$

```
fact := 1;
```

$\{P\}$

```
while x > 0 do (fact := fact * x; x := x -1) od
```

$\{P \wedge \neg x > 0\}$

And

2. $P \wedge \neg x > 0 \Rightarrow fact = a!$

Example

```
{x>= 0 ∧ x = a} (*this was part 1 to prove*)
   fact := 1;
   while x > 0 do (fact := fact * x; x := x -1) od
{a! = fact * (x!) ∧ x >=0 ∧ ¬ (x>0)}
```

- For Part 1, by sequencing rule it suffices to show

3. $\{x>=0 \wedge x = a\}$

```
fact := 1
```

$\{a! = fact * (x!) \wedge x >=0 \}$

And

4. $\{a! = fact * (x!) \wedge x >=0\}$

```
while x > 0 do
   (fact := fact * x; x := x -1) od
```

$\{a! = fact * (x!) \wedge x >=0 \wedge \neg (x > 0)\}$

Example

- (Part 3 – Assignment) Suffices to show that

  $$a! = fact * (x!) \wedge x >= 0$$

  holds before the while loop is entered

- (Part 4 – While Loop) And that if

  $$(a! = fact * (x!)) \wedge x >= 0 \wedge x > 0$$

  holds before we execute the body of the loop, then

  $$(a! = fact * (x!)) \wedge x >= 0$$

  holds after we execute the body (part 4)

Example

3. {x>=0 and x = a}
   `fact := 1`
   {a! = fact * (x!) and x >=0 }

Precondition Strengthening

$$\frac{P \rightarrow P' \quad \{P'\}\ C\ \{Q\}}{\{P\}\ C\ \{Q\}}$$

(Part 3) By the assignment rule, we have

$$\{a! = \mathbf{1} * (x!) \wedge x >= 0\}$$

`fact := 1`

$$\{a! = \mathbf{fact} * (x!) \wedge x >= 0\}$$

Therefore, to show (3), by precondition strengthening, it suffices to show

$$(x >= 0 \text{ and } x = a) \rightarrow (a! = 1 * (x!) \wedge x >= 0)$$

It holds because $x = a \rightarrow x! = a!$.

- So, we have that $a! = fact * (x!) \wedge x >= 0$ **holds at the start of the while loop!**

Example

To prove (Part 4):

$\{a! = fact * (x!) \wedge x >= 0\}$

```
while x > 0 do
   (fact := fact * x; x := x -1)
od
```

$\{a! = fact * (x!) \wedge x >= 0 \wedge \neg(x > 0)\}$

we need to **show that** **$(a! = fact * (x!)) \wedge x >= 0$** is **a loop invariant**

- We will use **assignment** rule, **sequencing** rule and **precondition strengthening** rule

Example

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

- We look into the loop body:
  - `(fact := fact * x; x := x - 1)`
- By the sequencing rule, we need to show 2 things:
  - By the ***assignment rule***, show

$$\{(a! = fact * (x!)) \ \wedge\ x >= 0 \wedge \neg\, x > 0\}$$

$$fact = fact * x$$

$$\{Q\}$$

  - By the ***assignment rule***, show

$$\{Q\}$$

$$x := x - 1$$

$$\{(a! = fact * (\mathbf{x}!)) \wedge \mathbf{x} >= 0\}$$

Example

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

- We look into the loop body:
  - `(fact := fact * x; x := x - 1)`
- By the sequencing rule, we need to show 2 things:
  - By the ***assignment rule***, show

$$\{(a! = fact * (x!)) \wedge x >= 0 \wedge x > 0\}$$

$$fact = fact * x$$

$$\{Q\}$$

  - From the ***assignment rule***, we know:

$$\{(a! = fact * (\mathbf{(x-1)}!)) \wedge \mathbf{x - 1} >= 0\}$$

$$x := x - 1$$

$$\{(a! = fact * (\mathbf{x}!)) \wedge \mathbf{x} >= 0\}$$

Example

- We look into the loop body:
  - `(fact := fact * x; x := x - 1)`
- By the sequencing rule, we need to show 2 things:
  - By the ***assignment rule***, show

$$\{(a! = fact * (x!)) \wedge x >= 0 \wedge x > 0\}$$

$$fact = fact * x$$

$$\mathbf{\{(a! = fact * ((x-1)!)) \wedge x - 1 >= 0\}}$$

  - From the ***assignment rule***, we know:

$$\{(a! = fact * (\mathbf{(x-1)}!)) \wedge \mathbf{x - 1} >= 0\}$$

$$x := x - 1$$

$$\{(a! = fact * (\mathbf{x}!)) \wedge \mathbf{x} >= 0\}$$

Example

Precondition Strengthening

$$\frac{P \rightarrow P' \quad \{P'\}\ C\ \{Q\}}{\{P\}\ C\ \{Q\}}$$

- By the ***assignment rule***, we have that

$$\{(a! = \mathbf{(fact * x)} * ((x-1)!)) \wedge x - 1 >= 0\}$$

```
fact = fact * x
```

$$\{(a! = fact * ((x-1)!)) \wedge x - 1 >= 0\}$$

- By ***Precondition strengthening***, it suffices to show that

$$((a! = fact * (x!)) \wedge x >= 0 \wedge x > 0) \Rightarrow$$
$$((a! = (fact * x) * (\mathbf{(x-1)}!)) \wedge \mathbf{x - 1 >= 0})$$

From algebra we know that $fact * x * (x - 1)! = fact * x!$

and $(x > 0) \Rightarrow x - 1 >= 0$ since x is an integer, so

$$\{(\mathbf{a! = fact * (x!)}) \wedge x >= 0 \wedge x > 0\} \Rightarrow$$
$$\{(\mathbf{a! = (fact * x) * ((x-1)!)}) \wedge \mathbf{x - 1 >= 0}\}$$

Example

Second try, let us combine the two:

$$P \equiv a! = fact * (x!) \wedge x >= 0$$

We need to show:

1. $\{x>=0 \wedge x = a\}$

```
fact := 1;
```

$\{P\}$

```
while x > 0 do (fact := fact * x; x := x -1) od
```

$\{P \wedge \neg x > 0\}$

And

2. $P \wedge \neg x > 0 \Rightarrow fact = a!$

Example

- For Part 2, we need

$$(a! = fact * (x!) \wedge x >= 0 \wedge \neg (x > 0)) \Rightarrow (fact = a!)$$

Since we know $(x >= 0 \wedge \neg (x > 0)) \Rightarrow (x = 0)$ so

$$fact * (x!) = fact * (0!)$$

And since from algebra we know that 0! = 1,

$$fact * (0)! = fact * 1 = fact$$

- Therefore, we can prove:

$$(a! = fact * (x!) \wedge x >= 0 \wedge \neg (x > 0)) \Rightarrow (fact = a!)$$

Example

- We proved that (**a! = fact * (x!)**) and x >= 0 is the loop invariant
- We proved the sequence rule for the assignment and wile statements
- We applied postcondition weakening to prove the final predicate

This finishes the proof!

```
{x >= 0 and x = a}
  fact := 1;
  while x > 0 do (fact := fact * x; x := x - 1) od
{fact = a!}
```

Approaches in the Ideal World

- Strongest postcondition
  - Need to worry about existential quantifiers

    $sp(x:=e, Pre) = \exists x' .\ \ Pre[x'=x] \ \wedge\ x = (\ e[x'/x]\ )$
  - Captures the new and all previous values of x

Approaches in the Ideal World

- Weakest precondition (by Dijkstra)
  - No need to worry about existential quantifiers here:

$$wp(x:=e, Post) = Post[e/x]$$

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

  - Sequence and conditionals are handled also "nicely"
  - But loops pose problems!
  - Need to infer the invariant from the postcondition, and reason about total correctness (loop *variant*)

Verification Condition Generation

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \quad \{P \text{ and } (\text{not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ \{Q\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

While

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

- VC : Stmt x Predicate -> Predicate
- From the statement and postcondition infer precondition

- VC(x := e, Q) =
- VC(C1; C2 , Q) =
- VC(if b C1 else C2, Q) =

-

Verification Condition Generation

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \quad \{P \text{ and (not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ \{Q\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

While

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

- VC : Stmt x Predicate -> Predicate
- From the statement and postcondition infer precondition

- VC(x := e, Q) = Q [e/x]
- VC(C1; C2 , Q) = VC (C1, VC (C2, Q))
- VC(if b C1 else C2, Q) = $b \Rightarrow VC(C1, Q) \wedge \neg b \Rightarrow VC(C2, Q)$

-

Verification Condition Generation

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \quad \{P \text{ and (not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ \{Q\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

While

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

- VC : Stmt x Predicate -> Predicate
- From the statement and postcondition infer precondition

- VC(x := e, Q) = Q [e/x]
- VC(C1; C2 , Q) = VC (C1, VC (C2, Q))
- VC(if b C1 else C2, Q) = $b \Rightarrow VC(C1, Q) \wedge \neg b \Rightarrow VC(C2, Q)$

  (alternative:) $(b \wedge VC(C1, Q)) \vee (\neg b \wedge VC(C2, Q))$
-

Verification Condition Generation

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \quad \{P \text{ and (not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ \{Q\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

While

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

- VC : Stmt x Predicate -> Predicate
- From the statement and postcondition infer precondition

- VC(x := e, Q) = Q [e/x]
- VC(C1; C2 , Q) = VC (C1, VC (C2, Q))
- VC(if b C1 else C2, Q) = $b \Rightarrow VC(C1, Q) \wedge \neg b \Rightarrow VC(C2, Q)$

  (alternative:) $(b \wedge VC(C1, Q)) \vee (\neg b \wedge VC(C2, Q))$
- While loop becomes complicated… let us annotate it with the invariant first

Verification Condition Generation: While

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \quad \{P \text{ and (not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ \{Q\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

While

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

- While loop: let us annotate it with the invariant first

- We want to get a VC(while b C, Q)
  - Inv is initially true
  - The triple $\{Inv \wedge b\}\ C\ \{Inv\}$ is valid in every iteration
  - which means $Inv \wedge b \Rightarrow VC\ (C, Inv)$ for all variable assignments
  - $Inv \wedge \neg b \Rightarrow Q$

- Bring it together:

$$Inv \wedge (\forall x1 \ldots xn\ .\ (Inv \wedge b \Rightarrow VC\ (C, Inv)) \wedge (Inv \wedge \neg b \Rightarrow Q))$$

- Can simplify:
  (exercise)

Verification Condition Generation: While

$$\frac{\{\,P \text{ and } B\,\}\ C\ \{\,P\,\}}{\{\,P\,\}\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ \{\,P \text{ and not } B\,\}}$$

- While loop: let us annotate it with the invariant first

- We want to get a VC(while b C, Q)
  - Inv is initially true
  - The triple {Inv $\wedge$ b} C {Inv} is valid in every iteration
  - which means Inv $\wedge$ b $\Rightarrow$ VC (C, Inv) for all variable assignments
  - Inv $\wedge \neg$b $\Rightarrow$ Q

- Bring it together:

$$\text{Inv} \wedge (\forall x1 \ldots xn\ .\ (\text{Inv} \wedge b \Rightarrow \text{VC}\ (C, \text{Inv})) \wedge (\text{Inv} \wedge \neg b \Rightarrow Q))$$

- Can simplify:
  (exercise) $\text{Inv} \wedge (\forall x1 \ldots xn\ .\ \text{Inv} \Rightarrow ((b \Rightarrow \text{VC}\ (C, \text{Inv})) \wedge (\neg b \Rightarrow Q)))$

Example (Use of VC)

$Inv \equiv a! = fact * (x!) \wedge x \geq 0$

$Inv' \equiv a! = fact * (x!) \wedge x \geq 0 \wedge a \geq 0 \wedge fact \geq 0$

-

```
{x>= 0 and x = a}

fact := 1;
while_Inv x > 0 do {
   fact := fact * x;
   x := x - 1
} od

{fact = a!}
```

Design by Contract Methodology

- Answer to program decomposition problem:
  - How to compose smaller functions/modules and guarantee some end-to-end property?

- Bertrand Meyer: "must make sure that the contractor will perform the task as required. As in real life, this is only possible if the agreement is' spelled out precisely in a contract document. A contract document protects both sides:
  • It protects the client by specifying how much should be done: the client is entitled to receive a certain result.
  • It protects the contractor by specifying how little is acceptable: the contractor must not be liable for failing to carry out tasks outside of the specified scope. "

- http://se.inf.ethz.ch/~meyer/publications/old/dbc_chapter.pdf

Design by Contract: Toward Practical

- Our verification technology tells what is possible
- DbC aims to bring this methodology to the real-world software projects
- Extends the theory with specifications for classes (and esp. class inheritance) and exception handling

Example: Class Invariants

```
class TABLE {
    /// field and method declarations:
    /// put, item, delete, count, capacity, ...

    invariant o <= count <= capacity
}
```

Then the contract for every method becomes

{P and Invariant} TABLE.put/get {Q and Invariant}

Constructors should satisfy too: {P} TABLE.c {Inv}

Language Support

- Eiffel
- Java Modeling Language
- Spec#
- Scala / Stainless
- Dafny
- …

+ Lot of interest in the research community to make programming with automated verification more scalable and easy-to-use

NEXT… DEMO