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

```java
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 $0 \leq s \land u < |a|$

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 \[ 0 \leq s \land u < |a| \]

```java
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 \[ \text{ret}_{\text{LinearSearch}} \iff \exists i. \ s \leq i \leq u \land a[i] = e \]
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 \( \{P\} \text{ C } \{Q\} \) 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

\[ \texttt{<command>} ::= \]
\[ \quad \texttt{<variable>} := \texttt{<term>} \]
\[ \mid \texttt{<command>}; \ldots ;\texttt{<command>} \]
\[ \mid \texttt{if} \texttt{<expression>} \texttt{then} \texttt{<command>} \texttt{else} \texttt{<command>} \texttt{fi} \]
\[ \mid \texttt{while} \texttt{<expression>} \texttt{do} \texttt{<command>} \texttt{od} \]

- 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:

$$(x + 2) [y-1/x] = ((y - 1) + 2)$$
The Assignment Rule

\[
\{\mathcal{P}[e/x]\} \ x := e \ \{\mathcal{P}\}
\]

Example:

\[
\{ \ ? \} \ x := y \ \{x = 2\}
\]
The Assignment Rule

\[
\{P[e/x]\} \ x := e \ {P}
\]

Example:

\[
\{\_ = 2 \} \ x := y \ \{x = 2\}
\]
The Assignment Rule

\[
\{ P [e/x] \} \ x := e \ \{ P \}
\]

Example:

\[
\{ y = 2 \} \ x := y \ \{ x = 2 \}
\]
The Assignment Rule

\[ \{ P[e/x] \} \ x := e \ \{ P \} \]

Examples:

\[ \{ y = 2 \} \ x := y \ \{ x = 2 \} \]

\[ \{ y = 2 \} \ x := 2 \ \{ y = x \} \]

\[ \{ x + 1 = n + 1 \} \ x := x + 1 \ \{ x = n + 1 \} \]

\[ \{ 2 = 2 \} \ x := 2 \ \{ x = 2 \} \]
The Assignment Rule – Your Turn

What is a valid precondition of

\[
x := x + y \{x + y = w - x\}
\]

\[
\{ \quad ? \quad \}
\]

\[
x := x + y
\]

\[
\{x + y = w - x\}
\]
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

\[ P \Rightarrow P' \quad \{ P' \} \subseteq \{ Q \} \]
\[ \{ P \} \subseteq \{ Q \} \]

- **Meaning:** If we can show that \( P \) implies \( P' \) \((P \Rightarrow P')\) and we can show that \( \{ P' \} \subseteq \{ Q \} \), then we know that \( \{ P \} \subseteq \{ Q \} \).

- **\( P \) is stronger** than \( P' \) means \( P \Rightarrow P' \).
Precondition Strengthening

- Examples:

  \[ x = 3 \Rightarrow x < 7 \{ x < 7 \} x := x + 3 \{ x < 10 \} \]
  \[ \{ x = 3 \} x := x + 3 \{ x < 10 \} \]

  True \Rightarrow 2 = 2 \{ 2 = 2 \} x := 2 \{ x = 2 \}
  \{ \text{True} \} x := 2 \{ x = 2 \}

  x = n \Rightarrow x + 1 = n + 1 \{ x + 1 = n + 1 \} x := x + 1 \{ x = n + 1 \}
  \{ x = n \} x := x + 1 \{ x = n + 1 \}
Which Inferences Are Correct?

\[
\begin{align*}
\{x > 0 \& x < 5\} & \quad x := x \times x \quad \{x < 25\} \\
\{x = 3\} & \quad x := x \times x \quad \{x < 25\} \\
\{x > 0 \& x < 5\} & \quad x := x \times x \quad \{x < 25\} \\
\{x \times x < 25\} & \quad x := x \times x \quad \{x < 25\} \\
\{x > 0 \& x < 5\} & \quad x := x \times x \quad \{x < 25\}
\end{align*}
\]
Which Inferences Are Correct?

\[
\begin{align*}
\{x > 0 \land x < 5\} & \quad x := x \times x \quad \{x < 25\} & & \checkmark \\
\{x = 3\} & \quad x := x \times x \quad \{x < 25\} \\
\{x > 0 \land x < 5\} & \quad x := x \times x \quad \{x < 25\} \\
\{x \times x < 25\} & \quad x := x \times x \quad \{x < 25\} & & \checkmark \\
\{x > 0 \land x < 5\} & \quad x := x \times x \quad \{x < 25\}
\end{align*}
\]
Sequencing

\[ \begin{align*}
\{P\} & \quad C_1 \quad \{Q\} \quad \{Q\} \quad C_2 \quad \{R\} \\
\{P\} & \quad C_1 ; \quad C_2 \quad \{R\}
\end{align*} \]

Example:

\[ \begin{align*}
\{z = z \land z = z\} & \quad x := z \quad \{x = z \land z = z\} \\
\{x = z \land z = z\} & \quad y := z \quad \{x = z \land y = z\} \\
\{z = z \land z = z\} & \quad x := z ; \quad y := z \quad \{x = z \land y = z\}
\end{align*} \]
Sequencing

\[
\begin{align*}
\{P\} & \quad C_1 \quad \{Q\} \\
\{Q\} & \quad C_2 \quad \{R\} \\
\{P\} & \quad C_1; \quad C_2 \quad \{R\}
\end{align*}
\]

**Example:**

\[
\begin{align*}
\{z = z \land z = z\} & \quad x := z \quad \{x = z \land z = z\} \\
\{x = z \land z = z\} & \quad y := z \quad \{x = z \land y = z\} \\
\{z = z \land z = z\} & \quad x := z; \quad y := z \quad \{x = z \land y = z\}
\end{align*}
\]
Postcondition Weakening

\[
\text{\{P\} C \{Q'\}} \quad Q' \Rightarrow Q \\
\text{\{P\} C \{Q\}}
\]

Example:

\[
\text{\{z = z & z = z\}} \quad x := z; \ y := z \quad \{x = z & y = z\} \\
\text{(x = z & y = z)} \Rightarrow (x = y) \\
\text{\{z = z & z = z\}} \quad x := z; \ y := z \quad \{x = y\}
\]
Rule of Consequence

\[ P \implies P' \quad \{P'\} \ C \ {\{Q'\}} \quad Q' \implies Q \]

- Logically equivalent to the combination of Precondition Strengthening and Postcondition Weakening
- Uses \( P \implies P' \) and \( Q' \implies Q \)
If Then Else

\[
\{P \text{ and } B\} \ C_1 \ \{Q\} \quad \{P \text{ and } \neg B\} \ C_2 \ \{Q\}
\]

\[
\{P\} \text{ if } B \text{ then } C_1 \text{ else } C_2 \text{ fi } \{Q\}
\]

- Example: Want

\[
\{y=a\}
\]

\[
\text{if } x < 0 \text{ then } y := y - x \text{ else } y := y + x \text{ fi}
\]

\[
\{y=a+|x|\}
\]

Suffices to show:

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

(4) \{y=a\&\neg(x<0)\} \ y := y + x \quad \{y=a+|x|\}
\{y=a \& x<0\} \ y:=y-x \ \{y=a+|x|\}

(3) \quad (y=a \& x<0) \Rightarrow y-x=a+|x|

(2) \quad \{y-x=a+|x|\} \ y:=y-x \ \{y=a+|x|\}

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

(1) Reduces to (2) and (3) by \textit{Precondition Strengthening}

(2) Follows from assignment axiom

(3) Because from algebra: \(x<0 \Rightarrow |x| = -x\)
\{y=a \land \neg(x<0)\} \ y:=y+x \ \{y=a+|x|\}

(6) \quad (y=a \land \neg(x<0)) \implies (y+x=a+|x|)

(5) \quad \{y+x=a+|x|\} \ y:=y+x \ \{y=a+|x|\}

(4) \quad \{y=a \land \neg(x<0)\} \ y:=y+x \ \{y=a+|x|\}

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

(5) Follows from **assignment** axiom

(6) Because \( \neg(x<0) \implies |x| = x \)
If Then Else

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

\{y=a\}

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:

```plaintext
{     ?     }     C    {      ?     }
{      ?      }
while B do C od {   P   }
```
The loop may never be executed, so if we want $P$ to hold after, it had better hold before, so let’s try:

$$\{ \ ? \ \} \ C \ \{ \ ? \ \}$$

$$\{ P \ \} \ while \ B \ do \ C \ od \ \{ P \ \}$$
While

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

$$\{ P \text{ and } B \} \ C \ \{ P \}$$

$$\{ P \} \ \textbf{while } B \ \textbf{do } C \ \textbf{od} \ \{ P \}$$
While

- We can strengthen the previous rule because we also know that when the loop is finished, \( \text{not } B \) also holds.

- Final *while* rule:

\[
\{ P \text{ and } B \} \ C \ { P \\
\{ P \} \text{ while } B \text{ do } C \text{ od} \ { P \text{ and not } B } \]


While

\[
\{ P \text{ and } B \} \ C \ \{ P \} \\
\{ P \} \text{ while } B \ \text{ do } \ C \ \text{ od} \ \{ P \text{ and not } B \}
\]

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 used together 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

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

P ≡ x ≤ n

Want to show: x ≥ n
Sum of numbers 1 to n

\[ x := 0 \]
\[ y := 0 \]

\textbf{while} \ y < n \ \{ \\
\quad y := y + 1; \\
\quad x := x + y \\
\}\]

Want to show: \[ x = 1 + 2 + \ldots + y \]
\[ \land \ y \leq n \]
\[ \land \ 0 \leq n \]

P \equiv \ x = 1 + 2 + \ldots + n \]
Fibonacci

\[ x = 0; \ y = 1; \]
\[ z = 1; \]

\[
\text{while} \ (z < n) \ {\}
\quad y := x + y;
\quad x := y - x;
\quad z := z + 1
\]

Want to show: \( y = \text{fib}(n) \)
List Length

\[
x = lst; \ y = 0
\]

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

\[P \equiv y + \text{len}(x) = \text{len}(lst)\]

Want to show: \(y = \text{len}(lst)\)
All Rules on One Slide

The Assignment Rule

\[
\{P \ [e/x]\} \ x := e \ {P}\n\]

Sequencing

\[
\{P\} \ C_1 \ {Q} \quad \{Q\} \ C_2 \ {R}\n\]
\[
\{P\} \ C_1; \ C_2 \ {R}\n\]

If Then Else

\[
\{P \text{ and } B\} \ C_1 \ {Q} \quad \{P \text{ and } \neg B\} \ C_2 \ {Q}\n\]
\[
\{P\} \text{ if } B \text{ then } C_1 \text{ else } C_2 \text{ fi } {Q}\n\]

Precondition Strengthening

\[
P \Rightarrow P' \quad \{P'\} \ C \ {Q}\n\]
\[
\{P\} \ C \ {Q}\n\]

Postcondition Weakening

\[
\{P\} \ C \ {Q'} \quad Q' \Rightarrow Q\n\]
\[
\{P\} \ C \ {Q}\n\]

Rule of Consequence

\[
P \Rightarrow P' \quad \{P'\} \ C \ {Q'} \quad Q' \Rightarrow Q\n\]
\[
\{P\} \ C \ {Q}\n\]

While

\[
\{P \text{ and } B\} \ C \ {P}\n\]
\[
\{P\} \text{ while } B \text{ do } C \text{ od } \{P \text{ and } \neg B\}\]
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 \geq 0 \text{ 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

$$(P \text{ and not } x > 0) \Rightarrow (\text{fact} = a!)$$
Example

- First attempt:
  \[ P \equiv a! = \text{fact} \times (x!) \]

- Motivation:
  - What we want to compute: \( a! \)
  - What we have computed: \( \text{fact} \)
    which is the sequential product of \( a \) down through \( (x + 1) \)
  - What we still need to compute: \( x! \)
Example

By post-condition weakening suffices to show

1. \(\{x \geq 0 \land x = a\}\)
   
   \[
   \begin{align*}
   \text{fact} & := 1; \\
   \text{while } x > 0 \text{ do (fact} & := \text{fact} \ast x; \ x := x - 1) \text{ od} \\
   \{a! = \text{fact} \ast (x!) \land \neg(x > 0)\}
   \end{align*}
   \]

And

2. \(a! = \text{fact} \ast (x!) \land \neg(x > 0) \Rightarrow \text{fact} = a!\)
Problem!! (Dead End)

2. \( a! = \text{fact} \times (x!) \land \neg(x > 0) \Rightarrow \text{fact} = a! \)

- Don’t know this if \( x < 0 \) !
  - Need to know that \( x = 0 \) when loop terminates

- Need a new loop invariant
  - Try adding \( x \geq 0 \)
  - Then will have \( x = 0 \) when loop is done
Example

Second try, let us combine the two:

\[ P \equiv a! = \text{fact} \times (x!) \land x \geq 0 \]

We need to show:

1. \( \{x \geq 0 \land x = a\} \)
   
   \[
   \begin{align*}
   \text{fact} &:= 1; \\
   \{P\} \\
   \text{while } x > 0 \text{ do } (\text{fact} := \text{fact} \times x; x := x - 1) \text{ od} \\
   \{P \land \neg x > 0\}
   \end{align*}
   \]

And

2. \( P \land \neg x > 0 \Rightarrow \text{fact} = a! \)

Example
Example

\{x \geq 0 \land x = a\} (*this was part 1 to prove*)

\[
\begin{align*}
&\text{fact} := 1; \\
&\text{while } x > 0 \text{ do (fact := fact } \times x; x := x - 1) \text{ od} \\
&\{a! = \text{fact } \times (x!) \land x \geq 0 \land \neg (x > 0)\}
\end{align*}
\]

For Part 1, by sequencing rule it suffices to show

3. \{x \geq 0 \land x = a\}
   \[
   \begin{align*}
   \text{fact} &:= 1 \\
   \{a! = \text{fact } \times (x!) \land x \geq 0\}
   \end{align*}
   \]

And

4. \{a! = \text{fact } \times (x!) \land x \geq 0\}
   \[
   \begin{align*}
   \text{while } x > 0 \text{ do} \\
   &\text{(fact := fact } \times x; x := x - 1) \text{ od} \\
   \{a! = \text{fact } \times (x!) \land x \geq 0 \land \neg (x > 0)\}
   \end{align*}
   \]
Example

- (Part 3 – Assignment) Suffices to show that
  \[ a! = \text{fact} \times (x!) \land x \geq 0 \]
  holds before the while loop is entered

- (Part 4 – While Loop) And that if
  \[ (a! = \text{fact} \times (x!)) \land x \geq 0 \land x > 0 \]
  holds before we execute the body of the loop, then
  \[ (a! = \text{fact} \times (x!)) \land x \geq 0 \]
  holds after we execute the body (part 4)
(Part 3) By the assignment rule, we have

\[
\{ a! = 1 \ast (x!) \land x \geq 0 \} \\
\text{fact} := 1 \\
\{ a! = \text{fact} \ast (x!) \land x \geq 0 \}
\]

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

\[(x \geq 0 \text{ and } x = a) \Rightarrow (a! = 1 \ast (x!) \land x \geq 0)\]

It holds because \(x = a \Rightarrow x! = a!\).

So, we have that \(a! = \text{fact} \ast (x!) \land x \geq 0\) holds at the start of the while loop!
Example

To prove (Part 4):

\[
\{ a! = \text{fact} \ast (x!) \land x \geq 0 \}
\]

while \( x > 0 \) do

\[
(\text{fact} := \text{fact} \ast x; x := x - 1)
\]

od

\[
\{ a! = \text{fact} \ast (x!) \land x \geq 0 \land \neg(x > 0) \}
\]

we need to show that \( (a! = \text{fact} \ast (x!)) \land x \geq 0 \)

is a loop invariant

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

- We look into the loop body:
  - \((\text{fact} := \text{fact} \times x; \; x := x - 1)\)

- By the sequencing rule, we need to show 2 things:
  - By the assignment rule, show
    \[
    \{(a! = \text{fact} \times (x!)) \land x \geq 0 \land \neg x > 0\}
    \]
    \[
    \text{fact} = \text{fact} \times x
    \]
    \[
    \{Q\}
    \]
  - By the assignment rule, show
    \[
    \{Q\}
    \]
    \[
    x := x - 1
    \]
    \[
    \{(a! = \text{fact} \times (x!)) \land x \geq 0\}
    \]
Example

- We look into the loop body:
  - \((\text{fact} := \text{fact} \times x; \ x := x - 1)\)

- By the sequencing rule, we need to show 2 things:
  - By the \textit{assignment rule}, show
    \[
    \{(a! = \text{fact} \times (x!)) \land x \geq 0 \land x > 0\}
    \]
    \[
    \text{fact} = \text{fact} \times x
    \]
    \[
    \{Q\}
    \]

- From the \textit{assignment rule}, we know:
  \[
  \{(a! = \text{fact} \times ((x-1)!)) \land x - 1 \geq 0\}
  \]
  \[
  x := x - 1
  \]
  \[
  \{(a! = \text{fact} \times (x!)) \land x \geq 0\}
  \]
Example

- We look into the loop body:
  - \((\text{fact} := \text{fact} \times x; \ x := x - 1)\)

- By the sequencing rule, we need to show 2 things:
  - By the \textit{assignment rule}, show:
    
    \[
    \{(a! = \text{fact} \times (x!)) \land x \geq 0 \land x > 0\}
    \]
    \[
    \text{fact} = \text{fact} \times x
    \]
    \[
    \{(a! = \text{fact} \times ((x-1)!)) \land x - 1 \geq 0\}
    \]

- From the \textit{assignment rule}, we know:
  
  \[
  \{(a! = \text{fact} \times ((x-1)!)) \land x - 1 \geq 0\}
  \]
  \[
  x := x - 1
  \]
  \[
  \{(a! = \text{fact} \times (x!)) \land x \geq 0\}
  \]
Example

- By the **assignment rule**, we have that
  \[
  \{(a! = (\text{fact} \times x) \times ((x-1)!)) \land x - 1 \geq 0\}
  \]
  \[
  \text{fact} = \text{fact} \times x
  \]
  \[
  \{(a! = \text{fact} \times ((x-1)!)) \land x - 1 \geq 0\}
  \]

- By **Precondition strengthening**, it suffices to show that
  \[
  ((a! = \text{fact} \times (x!)) \land x \geq 0 \land x > 0) \Rightarrow
  ((a! = (\text{fact} \times x) \times ((x-1)!)) \land x - 1 \geq 0)
  \]

From algebra we know that \( \text{fact} \times x \times (x - 1)! = \text{fact} \times x! \) and \( (x > 0) \Rightarrow x - 1 \geq 0 \) since \( x \) is an integer, so
\[
\{(a! = \text{fact} \times (x!)) \land x \geq 0 \land x > 0\} \Rightarrow
\{(a! = (\text{fact} \times x) \times ((x-1)!)) \land x - 1 \geq 0\}
Second try, let us combine the two: 

\[ P \equiv a! = \text{fact} \times (x!) \land x \geq 0 \]

We need to show:

1. \( \{ x \geq 0 \land x = a \} \)

\[
\begin{align*}
\text{fact} & := 1; \\
\{ P \} & \\
\text{while } x > 0 \text{ do } (\text{fact} := \text{fact} \times x; x := x - 1) \text{ od} \\
\{ P \land \neg x > 0 \}
\end{align*}
\]

And

2. \( P \land \neg x > 0 \Rightarrow \text{fact} = a! \)
Example

- For Part 2, we need

\((a! = \text{fact} \times (x!) \land x \geq 0 \land \neg (x > 0)) \Rightarrow (\text{fact} = a!))\)

Since we know \((x \geq 0 \land \neg (x > 0)) \Rightarrow (x = 0)\) so

\[
\text{fact} \times (x!) = \text{fact} \times (0!)
\]

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

\[
\text{fact} \times (0)! = \text{fact} \times 1 = \text{fact}
\]

- Therefore, we can prove:

\((a! = \text{fact} \times (x!) \land x \geq 0 \land \neg (x > 0)) \Rightarrow (\text{fact} = a!))\)
Example

- We proved that \((\text{a!} = \text{fact} \times (\text{x!}))\) and \(x \geq 0\) is the loop invariant
- We proved the sequence rule for the assignment and while statements
- We applied postcondition weakening to prove the final predicate

This finishes the proof!

\[
\{x \geq 0 \text{ and } x = a\}
\]
\[
\text{fact} := 1;
\]
\[
\text{while } x > 0 \text{ do (fact := fact} \times x; x := x - 1) \text{ od}
\]
\[
\{\text{fact} = a!\}
Approaches in the Ideal World

- Strongest postcondition
  - Need to worry about existential quantifiers
    \[ sp(x:=e, \text{Pre}) = \exists x'. \text{Pre}[x'=x] \land x = (e[x'/x]) \]
  - 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:

\[ \text{wp}(x:=e, \text{Post}) = \text{Post}[e/x] \]

- 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

- $VC : Stmt \times Predicate \rightarrow Predicate$
- From the statement and postcondition infer precondition

- $VC( x := e, Q ) =$
- $VC( C_1; C_2, Q ) =$
- $VC( if b\ C_1\ else\ C_2, Q ) =$
Verification Condition Generation

- 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 ⇒ VC( C1, Q ) ∧ ¬b ⇒ VC( C2, Q )
Verification Condition Generation

- **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) \land \neg b \Rightarrow VC(C2, Q)

  (alternative:) (b \land VC(C1, Q) ) \lor (\neg b \land VC(C2, Q))
Verification Condition Generation

- 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) \land \lnot b \Rightarrow VC(C2, Q)
  (alternative:) (b \land VC(C1, Q) ) \lor (\lnot b \land VC(C2, Q))
- While loop becomes complicated... let us annotate it with the invariant first
Verification Condition Generation: While

- 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 ∧ b\} C \{Inv\} is valid in every iteration
  - which means Inv ∧ b ⇒ VC (C, Inv) for all variable assignments
  - Inv ∧ ¬b ⇒ Q

- Bring it together:
  \[ \text{Inv} \land (\forall x_1 \ldots x_n . (\text{Inv} \land b \Rightarrow \text{VC} (C, \text{Inv})) \land (\text{Inv} \land \neg b \Rightarrow Q)) \]

- Can simplify:
  (exercise)
Verification Condition Generation: While

- 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 \( \{ \text{Inv} \land b \} \ C \ \{ \text{Inv} \} \) is valid in every iteration
  - which means \( \text{Inv} \land b \Rightarrow VC \ (C, \ \text{Inv}) \) for all variable assignments
  - \( \text{Inv} \land \neg b \Rightarrow Q \)

- Bring it together:
  \[
  \text{Inv} \land (\forall x_1 \ldots x_n \ . \ (\text{Inv} \land b \Rightarrow VC \ (C, \ \text{Inv})) \land (\text{Inv} \land \neg b \Rightarrow Q))
  \]

- Can simplify:
  \[
  (\text{exercise}) \quad \text{Inv} \land (\forall x_1 \ldots x_n \ . \ \text{Inv} \Rightarrow ((b \Rightarrow VC \ (C, \ \text{Inv})) \land (\neg b \Rightarrow Q)))
  \]
Example (Use of VC)

\[ \text{Inv} \equiv a! = \text{fact} \times (x!) \land x \geq 0 \]

\[ \text{Inv}' \equiv a! = \text{fact} \times (x!) \land x \geq 0 \land a \geq 0 \land \text{fact} \geq 0 \]

\{x \geq 0 \land x = a\}

\[
\begin{align*}
\text{fact} & := 1; \\
\text{while}_{\text{Inv}} \ x > 0 \ \text{do} \ {\} \\
& \quad \text{fact} := \text{fact} \times x; \\
& \quad x := x - 1 \\
\text{od}
\end{align*}
\]

\{\text{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](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