CSE 331
Software Design & Implementation

Kevin Zatloukal
Spring 2020
Lecture 2 – Reasoning About Straight-Line Code
Hoare Logic
A Problem (from last time)

“Complete this method such that it returns the location of the largest value in the first \( n \) elements of the array \( \text{arr} \).”

```c
int maxLoc(int[] arr, int n) {
    ... 
    ...
}
```
A Solution?

```java
int maxLoc(int[] arr, int n) {
    int maxIndex = 0;
    int maxValue = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > maxValue) {
            maxIndex = i;
            maxValue = arr[i];
        }
    }
    return maxIndex;
}
```

Corner cases:
- What if there are ties?
- What if n is 0?

Error cases:
- What if arr.length < n?
- What if arr is null?

No way to tell!
How to Check Correctness

- Step 1: need a **specification** for the function
  - can’t argue correctness if we don’t know what it should do
  - surprisingly difficult to write!

- Step 2: determine whether the code meets the specification
  - apply **reasoning**
  - surprisingly easy with the tools we will learn
Our approach: formal reasoning

• **Hoare Logic**: classic approach to logical reasoning about code
  – named after its inventor, Sir Anthony Hoare
  – formal description of correctness

• In practice, reasoning is less formal
  – so it can be done at a *faster* pace

• Formal reasoning is still useful
  – slower but “turn the crank”
  – still used in practice for **hard** problems
    • in general, formalism comes out when the problems become difficult
Terminology of Hoare Logic

• The program state is the values of all the (relevant) variables

• An assertion is a true / false claim (proposition) about the state at a given point during execution (e.g., on line 39)

• An assertion holds for a program state if the claim is true when the variables have those values

• An assertion before the code is a precondition
  – these represent assumptions about when that code is used

• An assertion after the code is a postcondition
  – these represent what we want the code to accomplish
Hoare Logic

- A Hoare triple is two assertions and one piece of code:
  \[ \{ P \} \; S \; \{ Q \} \]
  - \(P\) the precondition
  - \(S\) the code
  - \(Q\) the postcondition

- A Hoare triple \(\{ P \} \; S \; \{ Q \}\) is called valid if:
  - in any state where \(P\) holds, executing \(S\) produces a state where \(Q\) holds
  - i.e., if \(P\) is true before \(S\), then \(Q\) must be true after it
  - otherwise the triple is called invalid
Notation

• Hoare logic writes assertions in {..}
  – since Java code also has {..}, I will use {{…}}
  – e.g., {{ w >= 1 }} x = 2 * w; {{ x >= 2 }}

• Assertions are math / logic not Java
  – you can use the usual math notation
    • (e.g., = instead of == for equals)
  – purpose is communication with other humans (not computers)
  – we will need and, or, not as well
    • can also write use \text{\&} (and) \text{\lor} (or) etc.

• The Java language also has assertions (assert statements)
  – throws an exception if the condition does not evaluate true
  – we will discuss these more later in the course
Example 1

Is the following Hoare triple valid or invalid?
- assume all variables are integers and there is no overflow

\[
\{ x \neq 0 \} \ y = x^x; \ \{ y > 0 \}\]
Example 1

Is the following Hoare triple valid or invalid?
- assume all variables are integers and there is no overflow

\[
\{ x \neq 0 \}\ y = x * x; \ {\{ y > 0 \}}
\]

Valid
- \( y \) could only be zero if \( x \) were zero (which it isn’t)
Example 2

Is the following Hoare triple valid or invalid?

- assume all variables are integers and there is no overflow

\[
\{z \neq 1\} \ y = z^2; \ \{y \neq z\}
\]
Example 2

Is the following Hoare triple valid or invalid?

– assume all variables are integers and there is no overflow

\[
\{\{ z \neq 1 \}\} \quad y = z \times z; \quad \{\{ y \neq z \}\}\n\]

Invalid

• counterexample: \( z = 0 \)
Checking Validity

• So far: decided if a Hoare triple is valid by ... hard thinking

• Soon: “turn the crank” methods for reasoning about
  – assignment statements
  – conditionals
  – [next lecture] loops
  – (all code can be understood in terms of those 3 elements)

• Can use those to check correctness in a “turn the crank” manner

• Next: a way to compare different assertions
  – useful, e.g., to compare possible preconditions
Weaker vs. Stronger Assertions

If $P_1$ implies $P_2$ (written $P_1 \Rightarrow P_2$), then:

- $P_1$ is stronger than $P_2$
- $P_2$ is weaker than $P_1$

Whenever $P_1$ holds, $P_2$ also holds

- So it is more (or at least as) “difficult” to satisfy $P_1$
  - the program states where $P_1$ holds are a subset of the program states where $P_2$ holds
- So $P_1$ puts more constraints on program states
- So it is a stronger set of requirements on the program state
  - $P_1$ gives you more information about the state than $P_2$
Examples

• $x = 17$ is stronger than $x > 0$

• $x$ is prime is neither stronger nor weaker than $x$ is odd

• $x$ is prime and $x > 2$ is stronger than $x$ is odd
Hoare Logic Facts

- Suppose \{P\} S \{Q\} is valid.

- If \(P_1\) is stronger than \(P\), then \{\(P_1\)\} S \{Q\} is valid.

- If \(Q_1\) is weaker than \(Q\), then \{\(P\)\} S \{\(Q_1\)\} is valid.

- Example:
  - Suppose \(P\) is \(x \geq 0\) and \(P_1\) is \(x > 0\)
  - Suppose \(Q\) is \(y > 0\) and \(Q_1\) is \(y \geq 0\)
  - Since \{\{ x \geq 0 \}\} \ y = x + 1 \{\{ y > 0 \}\}\ is valid, \{\{ x > 0 \}\} \ y = x + 1 \{\{ y \geq 0 \}\}\ is also valid
Hoare Logic Facts

• Suppose \{P\} S \{Q\} is valid.

• If \(P_1\) is stronger than \(P\), then \{\(P_1\)\} S \{\(Q\)\} is valid.

• If \(Q_1\) is weaker than \(Q\), then \{\(P\)\} S \{\(Q_1\)\} is valid.

• Key points:
  – always okay to strengthen a precondition
  – always okay to weaken a postcondition
Forward & Backward Reasoning
Example of Forward Reasoning

Work forward from the precondition

\{ \text{w > 0} \}\}
\text{x} = 17;
\{ \quad \}
\text{y} = 42;
\{ \quad \}
\text{z} = \text{w + x + y};
\{ \quad \}
Example of Forward Reasoning

Work forward from the precondition

{{ \ w > 0 \ }}
\ x = 17;
{{ \ w > 0 \ and \ x = 17 \ }}
\ y = 42;
{{ \ ____________________________ \ }}
\ z = w + x + y;
{{ \ ____________________________ \ }}
Example of Forward Reasoning

Work forward from the precondition

\{\{ w > 0 \}\} 
\hspace{1em} x = 17;
\{\{ w > 0 \text{ and } x = 17 \}\} 
\hspace{1em} y = 42;
\{\{ w > 0 \text{ and } x = 17 \text{ and } y = 42 \}\} 
\hspace{1em} z = w + x + y;
\{\{ \hspace{1em} \} \}
Example of Forward Reasoning

Work forward from the precondition

\{ \{ w > 0 \} \}
\quad x = 17;
\{ \{ w > 0 \text{ and } x = 17 \} \}
\quad y = 42;
\{ \{ w > 0 \text{ and } x = 17 \text{ and } y = 42 \} \}
\quad z = w + x + y;
\{ \{ w > 0 \text{ and } x = 17 \text{ and } y = 42 \text{ and } z = w + x + y \} \}
Example of Forward Reasoning

Work forward from the precondition

\[
\begin{align*}
\{ & w > 0 \} \\
& x = 17; \\
\{ & w > 0 \text{ and } x = 17 \} \\
& y = 42; \\
\{ & w > 0 \text{ and } x = 17 \text{ and } y = 42 \} \\
& z = w + x + y; \\
\{ & w > 0 \text{ and } x = 17 \text{ and } y = 42 \text{ and } z = w + 59 \}
\end{align*}
\]
Forward Reasoning

- Start with the **given** precondition
- Fill in the **strongest** postcondition

- For an assignment, \( x = y \)...
  - add the fact “\( x = y \)” to what is known
  - important **subtleties** here... (more on those later)

- Later: if statements and loops...
Example of Backward Reasoning

Work backward from the desired postcondition

\[ \{ \text{_______________________________} \} \]
\[ x = 17; \]
\[ \{ \text{_______________________________} \} \]
\[ y = 42; \]
\[ \{ \text{_______________________________} \} \]
\[ z = w + x + y; \]
\[ \{ z < 0 \} \]
Example of Backward Reasoning

Work backward from the desired postcondition

\{\{ \\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\} \}
\ x = 17;
\{\{ \\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\} \}
\ y = 42;
\{\{ w + x + y < 0 \} \}
\ z = w + x + y;
\{\{ z < 0 \} \}
Example of Backward Reasoning

Work backward from the desired postcondition

\[
\begin{align*}
\{ & \quad \text{______________________________} \} \\
\text{x} &= 17; \\
\{ & \quad \text{w + x + 42 < 0} \} \\
\text{y} &= 42; \\
\{ & \quad \text{w + x + y < 0} \} \\
\text{z} &= \text{w + x + y}; \\
\{ & \quad \text{z < 0} \}
\end{align*}
\]
Example of Backward Reasoning

Work backward from the desired postcondition

\[
\{\{ \ w + 17 + 42 < 0 \}\}\]

\[
x = 17;
\]

\[
\{\{ \ w + x + 42 < 0 \}\}\]

\[
y = 42;
\]

\[
\{\{ \ w + x + y < 0 \}\}\]

\[
z = w + x + y;
\]

\[
\{\{ \ z < 0 \}\}\]
Backward Reasoning

- Start with the **required** postcondition
- Fill in the **weakest** precondition

For an assignment, \( x = y \):
- just replace “\( x \)” with “\( y \)” in the postcondition
- if the condition using “\( y \)” holds beforehand, then the condition with “\( x \)” will afterward since \( x = y \) then

Later: if statements and loops...
Use forward reasoning to determine if this code is correct:

\[
\begin{align*}
\{ & \text{ w > 0 } \} \\
\text{ x = 17; } \\
\text{ y = 42; } \\
\text{ z = w + x + y; } \\
\{ & \text{ z > 50 } \}
\end{align*}
\]
Example of Forward Reasoning

\[
\begin{align*}
\{ w > 0 \} \\
\quad x &= 17; \\
\{ w > 0 \text{ and } x=17 \} \\
\quad y &= 42; \\
\{ w > 0 \text{ and } x=17 \text{ and } y=42 \} \\
\quad z &= w + x + y; \\
\{ w > 0 \text{ and } x=17 \text{ and } y=42 \text{ and } z = w + 59 \} \\
\{ z > 50 \}
\end{align*}
\]

Do the facts that are always true imply the facts we need? I.e., is the bottom statement weaker than the top one? (Recall that weakening the postcondition is always okay.)
Correctness by Backward Reasoning

Use backward reasoning to determine if this code is correct:

```plaintext
{{ w < -60 }}

x = 17;
y = 42;
z = w + x + y;

{{ z < 0 }}
```
Correctness by Backward Reasoning

Use backward reasoning to determine if this code is correct:

\[
\begin{align*}
&\{ w < -60 \} \\
&\{ w + 17 + 42 < 0 \} \iff \{ w < -59 \} \\
&\quad \quad \quad \quad x = 17; \\
&\{ w + x + 42 < 0 \} \\
&\quad \quad \quad \quad y = 42; \\
&\{ w + x + y < 0 \} \\
&\quad \quad \quad \quad z = w + x + y; \\
&\{ z < 0 \}
\end{align*}
\]

Do the facts that are always true imply the facts we need?

I.e., is the top statement stronger than the bottom one?

(Recall that strengthening the precondition is always okay.)
Combining Forward & Backward

It is okay to use both types of reasoning

• Reason forward from precondition
• Reason backward from postcondition

Will meet in the middle:

\[
\begin{align*}
\text{\{\{ P \}\}} \\
S1 \\
S2 \\
\text{\{\{ Q \}\}}
\end{align*}
\]
Combining Forward & Backward

It is okay to use both types of reasoning

- Reason forward from precondition
- Reason backward from postcondition

Will meet in the middle:

\[
\begin{align*}
&\{\{\ P \ \}\} \\
&\quad S1 \\
&\quad \{\{\ P1 \ \}\} \\
&\quad \{\{\ Q1 \ \}\} \\
&\quad S2 \\
&\{\{\ Q \ \}\} \\
\end{align*}
\]

Valid provided \( P1 \) implies \( Q1 \)
Combining Forward & Backward

Reasoning in either direction gives valid assertions
Just need to check adjacent assertions:
• top assertion must imply bottom one

\[
\begin{align*}
\{\{ P \}\} & \quad \{\{ P \}\} & \quad \{\{ P \}\} \\
S1 & \quad S1 & \quad \{\{ Q1 \}\}
\end{align*}
\]

\[
\begin{align*}
\{\{ P1 \}\} & \quad \{\{ P1 \}\} & \quad S1 \\
\{\{ Q1 \}\} & \quad \{\{ Q \}\}
\end{align*}
\]

\[
\begin{align*}
S2 & \quad S2 \\
\{\{ Q \}\} & \quad \{\{ Q \}\}
\end{align*}
\]
Subtleties in Forward Reasoning...

- Forward reasoning can fail if applied blindly...

\[
\begin{align*}
\{w = x + y; \} \\
\{\{ w = x + y \} \} \\
x = 4; \\
\{\{ w = x + y \text{ and } x = 4 \} \} \\
y = 3; \\
\{\{ w = x + y \text{ and } x = 4 \text{ and } y = 3 \} \}
\end{align*}
\]

This implies that \( w = 7 \), but that is not true!
- \( w \) equals whatever \( x + y \) was before they were changed
Fix 1

- Use **subscripts** to refer to old values of the variables
- Un-subscripted variables should always mean **current** value

```plaintext
w = x + y;

{{ w = x + y }}
x = 4;

{{ w = x₁ + y and x = 4 }}
y = 3;

{{ w = x₁ + y₁ and x = 4 and y = 3 }}
```
Fix 2 (better)

- Express prior values in terms of the current value

```
{{
    w = x + y;
    {{ w = x + y }}
    x = x + 4;
    {{ w = x₁ + y and x = x₁ + 4 }}
    \Rightarrow {{ w = x - 4 + y }}
}}
```

Now, \( x₁ = x - 4 \)

So \( w = x₁ + y \iff w = x - 4 + y \)

Note for updating variables, e.g., \( x = x + 4 \):
- Backward reasoning just substitutes new value (no change)
- Forward reasoning requires you to invert the “+” operation
Forward vs. Backward

• Forward reasoning:
  – Find strongest postcondition
  – Intuitive: “simulate” the code in your head
    • BUT you need to change facts to refer to prior values
  – Inefficient: Introduces many irrelevant facts
    • usually need to weaken as you go to keep things sane

• Backward reasoning
  – Find weakest precondition
  – Formally simpler
  – Efficient
  – (Initially) unintuitive
If Statements
If Statements

Forward reasoning

{{ P }}
if (cond)
  S1
else
  S2
{{ ? }}
If Statements

Forward reasoning

```plaintext
{{ P }}
if (cond)
  {{ P and cond }}
  S1
else
  {{ P and not cond }}
  S2
{{ ? }}
```
If Statements

Forward reasoning

```
{{ P }}
if (cond)
  {{ P and cond }}
  S1
  {{ P1 }}
else
  {{ P and not cond }}
  S2
  {{ P2 }}
{{ ? }}
```
If Statements

Forward reasoning

```
{{ P }}
if (cond)
  {{ P and cond }}
  S1
{{ P1 }}
else
  {{ P and not cond }}
  S2
{{ P2 }}
{{ P1 or P2 }}
```
If Statements

Backward reasoning

{{ ? }}
if (cond)
    S1
else
    S2
{{ Q }}
If Statements

Backward reasoning

```plaintext
{{ ? }}
if (cond)
   S1
   {{ Q }}
else
   S2
   {{ Q }}
   {{ Q }}
   {{ Q }}
```
If Statements

Backward reasoning

```plaintext
{ { Q } }
if (cond)
  { { Q1 } }
  S1
  { { Q } }
else
  { { Q2 } }
  S2
  { { Q } }
{ { Q } }
```
If Statements

Backward reasoning

\[
\begin{align*}
\{ \text{cond and Q1 or not cond and Q2} \}
\end{align*}
\]

if (cond)

\[
\begin{align*}
\{ \text{Q1} \}
\end{align*}
\]

S1

\[
\begin{align*}
\{ \text{Q} \}
\end{align*}
\]

else

\[
\begin{align*}
\{ \text{Q2} \}
\end{align*}
\]

S2

\[
\begin{align*}
\{ \text{Q} \}
\end{align*}
\]

\[
\begin{align*}
\{ \text{Q} \}
\end{align*}
\]
If-Statement Example

Forward reasoning

```c
{{
  if (x >= 0)
    y = x;
  else
    y = -x;

{{ ? }}
```
If-Statement Example

Forward reasoning

```java
{{
if (x >= 0)
  {{ x >= 0 }}
y = x;
else
  {{ x < 0 }}
y = -x;
{{ ? }}
}}
```
If-Statement Example

Forward reasoning

```plaintext
{{ }}
if (x >= 0)
  {{ x >= 0 }}
  y = x;
  {{ x >= 0 and y = x }}
else
  {{ x < 0 }}
  y = -x;
  {{ x < 0 and y = -x }}
{{ ? }}
```
If-Statement Example

Forward reasoning

\[
\begin{align*}
\{ & \} \\
& \text{if } (x \geq 0) \\
& \quad \{ \{ x \geq 0 \} \} \\
& \quad y = x; \\
& \quad \{ \{ x \geq 0 \text{ and } y = x \} \} \\
& \text{else} \\
& \quad \{ \{ x < 0 \} \} \\
& \quad y = -x; \\
& \quad \{ \{ x < 0 \text{ and } y = -x \} \} \\
& \{ \{ (x \geq 0 \text{ and } y = x) \text{ or } (x < 0 \text{ and } y = -x) \} \}
\end{align*}
\]
If-Statement Example

Forward reasoning

```plaintext
{{ }}
if (x >= 0)
    {{ x >= 0 }}
    y = x;
    {{ x >= 0 and y = x }}
else
    {{ x < 0 }}
    y = -x;
    {{ x < 0 and y = -x }}
{{ y = |x| }}
```
If-Statement Example

Forward reasoning

```plaintext
{{ }}
if (x >= 0)
  {{ x >= 0 }}
y = x;
  {{ x >= 0 and y = x }}
else
  {{ x < 0 }}
y = -x;
  {{ x < 0 and y = -x }}
{{ y = |x| }}
```

**Warning**: many write `{{ y >= 0 }}` here. That is true but it is *strictly* weaker. (It includes cases where y != x)
If-Statement Example

Forward reasoning

\[
\begin{align*}
\{} & \{} \\
\text{if } (x \geq 0) & \\
\{} & \{} x \geq 0 \} \\
y = x; & \\
\{} & \{} x \geq 0 \text{ and } y = x \} \\
\text{else} & \\
\{} & \{} x < 0 \} \\
y = -x; & \\
\{} & \{} x < 0 \text{ and } y = -x \} \\
\{} & \{} y = |x| \}\end{align*}
\]

Backward reasoning

\[
\begin{align*}
\{} & \{} \\
\text{if } (x \geq 0) & \\
\{} & \{} y = x; \} \\
\text{else} & \\
\{} & \{} y = -x; \} \\
\{} & \{} y = |x| \} \end{align*}
\]
If-Statement Example

Forward reasoning

```latex
{{ }}
if (x >= 0)
  {{ x >= 0 }}
y = x;
  {{ x >= 0 and y = x }}
else
  {{ x < 0 }}
y = -x;
  {{ x < 0 and y = -x }}
{{ y = |x| }}
```

Backward reasoning

```latex
{{ ? }}
if (x >= 0)
y = x;
  {{ y = |x| }}
else
  {{ y = |x| }}
y = -x;
  {{ y = |x| }}
{{ y = |x| }}
```
If-Statement Example

Forward reasoning

```
{ }  
if (x >= 0)  
{ { x >= 0 } }  
y = x;  
{ { x >= 0 and y = x } }  
else  
{ { x < 0 } }  
y = -x;  
{ { x < 0 and y = -x } }  
{ { y = |x| } }
```

Backward reasoning

```
{ ? }  
if (x >= 0)  
{ { x = |x| } }  
y = x;  
{ { y = |x| } }  
else  
{ { -x = |x| } }  
y = -x;  
{ { y = |x| } }  
{ { y = |x| } }
```
If-Statement Example

Forward reasoning

```
{{ }}
if (x >= 0)
    {{ x >= 0 }}
y = x;
    {{ x >= 0 and y = x }}
else
    {{ x < 0 }}
y = -x;
    {{ x < 0 and y = -x }}
{{ y = |x| }}
```

Backward reasoning

```
{{ ? }}
if (x >= 0)
    {{ x >= 0 }}
y = x;
    {{ y = |x| }}
else
    {{ x <= 0 }}
y = -x;
    {{ y = |x| }}
{{ y = |x| }}
```
If-Statement Example

Forward reasoning

\[
\begin{align*}
\{ & \} \\
\text{if } (x \geq 0) & \{ x \geq 0 \} \\
y & = x; \\
\{ & x \geq 0 \text{ and } y = x \} \\
\text{else} & \{ x < 0 \} \\
y & = -x; \\
\{ & x < 0 \text{ and } y = -x \} \\
\{ & y = |x| \}
\end{align*}
\]

Backward reasoning

\[
\begin{align*}
\{ & (x \geq 0 \text{ and } x \geq 0) \text{ or } \\
& (x < 0 \text{ and } x \leq 0) \} \\
\text{if } (x \geq 0) & \{ x \geq 0 \} \\
y & = x; \\
\{ & y = |x| \} \\
\text{else} & \{ x \leq 0 \} \\
y & = -x; \\
\{ & y = |x| \} \\
\{ & y = |x| \}
\end{align*}
\]
If-Statement Example

Forward reasoning

\[
\begin{align*}
\{ \} \\
\text{if } (x \geq 0) \\
\{ \{ x \geq 0 \} \} \\
y = x; \\
\{ \{ x \geq 0 \text{ and } y = x \} \} \\
\text{else} \\
\{ \{ x < 0 \} \} \\
y = -x; \\
\{ \{ x < 0 \text{ and } y = -x \} \} \\
\{ \{ y = |x| \} \}
\end{align*}
\]

Backward reasoning

\[
\begin{align*}
\{ \{ x \geq 0 \text{ or } x < 0 \} \} \\
\text{if } (x \geq 0) \\
\{ \{ x \geq 0 \} \} \\
y = x; \\
\{ \{ y = |x| \} \} \\
\text{else} \\
\{ \{ x \leq 0 \} \} \\
y = -x; \\
\{ \{ y = |x| \} \} \\
\{ \{ y = |x| \} \}
\end{align*}
\]
If-Statement Example

Forward reasoning

{\{ \}}
if (x >= 0)
{\{ x >= 0 \}}
y = x;
{\{ x >= 0 and y = x \}}
else
{\{ x < 0 \}}
y = -x;
{\{ x < 0 and y = -x \}}
{\{ y = |x| \}}

Backward reasoning

{\{ \}}
if (x >= 0)
{\{ x >= 0 \}}
y = x;
{\{ y = |x| \}}
else
{\{ x <= 0 \}}
y = -x;
{\{ y = |x| \}}
{\{ y = |x| \}}
Next time: Loops...