CSE 331
Software Design & Implementation

Kevin Zatloukal
Spring 2020
Lecture 2 – Reasoning About Straight-Line Code
Outline

• Adminstrivia

• Recap (highlights only)

• Q & A

• Exercises

• More Examples (bit more complex)
Administrivia: HW

- HW0 may have been a struggle
  - will show you how to make this easy

- HW1 posted shortly
  - worksheet
  - practice applying these ideas
    - verifying correctness of short, non-loop code
  - due on Monday by 11pm
Administrivia: Section Splits

Sections

Each section will split into two sub-sections. For example, section AA on the calendar becomes AA-1 and AA-2 that both meet at 8:30am. The table below shows which students should go to which of the two subsections based on one of the digits in their UW Student Number.

See the Zoom page to find the link to the meeting for that section (e.g., "Section AA-1").

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Split</th>
<th>Value</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30</td>
<td>AA-1</td>
<td>last digit</td>
<td>odd</td>
<td>Yihang</td>
</tr>
<tr>
<td></td>
<td>AA-2</td>
<td>last digit</td>
<td>even</td>
<td>Chloe</td>
</tr>
<tr>
<td>9:30</td>
<td>AB-1</td>
<td>last digit</td>
<td>odd</td>
<td>Alexey</td>
</tr>
<tr>
<td></td>
<td>AB-2</td>
<td>last digit</td>
<td>even</td>
<td>Rachel</td>
</tr>
<tr>
<td>10:30</td>
<td>AC-1</td>
<td>last digit</td>
<td>odd</td>
<td>Andrew</td>
</tr>
<tr>
<td></td>
<td>AC-2</td>
<td>last digit</td>
<td>even</td>
<td>Manchen</td>
</tr>
<tr>
<td>11:30</td>
<td>AD-1</td>
<td>last digit</td>
<td>odd</td>
<td>Dmitriy</td>
</tr>
<tr>
<td></td>
<td>AD-2</td>
<td>last digit</td>
<td>even</td>
<td>Chanwut</td>
</tr>
<tr>
<td>12:30</td>
<td>AE-1</td>
<td>second digit</td>
<td>odd</td>
<td>Frank</td>
</tr>
<tr>
<td></td>
<td>AE-2</td>
<td>second digit</td>
<td>even</td>
<td>Jasmine</td>
</tr>
</tbody>
</table>

https://canvas.uw.edu/courses/1370605/pages/sections
Administrivia: Section

• Each section has 16-20 students
  – hopefully, you will get to know the other students

• Section plan: Q&A, review, worksheet
  – may want to print the worksheet beforehand (if you can)
  – worksheet is similar to HW1
Quick Recap (10 min)

Correctness Toolkit
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
  - code is correct iff triple is valid
Reasoning Forward & Backward

• Forward:
  – start with the given precondition
  – fill in the strongest postcondition

• Backward
  – start with the required postcondition
  – fill in the weakest precondition

• Finds the “best” assertion that makes the triple valid
Reasoning: Assignments

\[ x = \ldots \]

- **Forward**
  - add the fact “\( x = \ldots \)” to what is known
  - BUT you must *fix* any existing references to “\( x \)”

- **Backward**
  - just replace “\( x \)” with “\( \ldots \)” in the postcondition (substitution)
Reasoning: If Statements

Forward reasoning

\[
\begin{align*}
\{\ P \} \\
\text{if (cond)} \\
\{\ P \text{ and } \text{cond} \} \\
\text{S1} \\
\{\ P1 \} \\
\text{else} \\
\{\ P \text{ and not } \text{cond} \} \\
\text{S2} \\
\{\ P2 \} \\
\{\ P1 \text{ or } P2 \} \\
\end{align*}
\]

Backward reasoning

\[
\begin{align*}
\{\ \text{cond and } Q1 \text{ or} \\
\text{not } \text{cond and } Q2 \} \\
\text{if (cond)} \\
\{\ Q1 \} \\
\text{S1} \\
\{\ Q \} \\
\text{else} \\
\{\ Q2 \} \\
\text{S2} \\
\{\ Q \} \\
\{\ Q \} \\
\end{align*}
\]
Validity with Fwd & Back Reasoning

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

{ { P } }
S1
{ { P } }
S2
{ { Q } }  

{ { P } }
S1
{ { Q1 } }
S2
{ { Q } }  

{ { P } }
S1
{ { P1 } }
S2
{ { Q } }  

CSE 331 Spring 2020
Q & A
Dropping Irrelevant Facts

- Forward reasoning often adds many irrelevant facts
- Dropping them is *usually* okay

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

- Result is still a valid triple (ok to weaken postcondition)
- BUT no longer the *strongest* postcondition

- May get a final postcondition that doesn’t imply the given one
- In that case, put them back and try again...
Exercises
Hoare Triples

Valid or invalid?
   - (Assume all variables are integers without overflow)

- \{x \neq 0\} y = x*x; \{y > 0\} \quad \text{valid}
- \{z \neq 1\} y = z*z; \{y \neq z\} \quad \text{invalid}
- \{x \geq 0\} y = 2*x; \{y > x\} \quad \text{invalid}
- {} \text{if}(x > 7) \{y=4;\} \text{else} \{y=3;\} \{y < 5\} \quad \text{valid}
- {} x = y; z = x; \{y=z\} \quad \text{valid}
- \{x=7 \land y=5\}
  \quad \text{tmp}=x; x=\text{tmp}; y=x; \quad \text{invalid}
- \{y=7 \land x=5\}
Forward Reasoning

```c
{x >= 0}
    if (x != 0) {
        z = x;
    } else {
        z = x + 1;
    }

{x

_________}
```
Forward Reasoning

\[
\begin{align*}
\{& x \geq 0 \} \\
\text{if } (x \neq 0) & \{ \\
& \{ x > 0 \} \\
& z = x; \\
& \{ x > 0 \text{ and } z = x \} \\
\} \text{ else } & \{ \\
& \{ x = 0 \} \\
& z = x + 1; \\
& \{ x = 0 \text{ and } z = 1 \} \\
\}
\end{align*}
\]

\[
\{ (x > 0 \text{ and } z = x) \text{ or } (x = 0 \text{ and } z = 1) \} \Rightarrow \{ z > 0 \}
\] but strictly weaker
Backward Reasoning

{{ ________________________________ }}

if (x > 7) {
    y = x;
} else {
    y = 20;
}

{{ y > 5 }}
Backward Reasoning

\[
\{ (x > 7 \text{ and } x > 5) \text{ or } (x \leq 7) \} \iff \{ (x > 7) \text{ or } (x \leq 7) \}
\]

if \( x > 7 \) {
\[
\{ x > 5 \}
\]
\[ y = x; \]
\[
\{ y > 5 \}
\]
} else {
\[
\{ 20 > 5 \} \iff \{ \}
\]
\[ y = 20; \]
\[
\{ y > 5 \}
\]
}
\[ y > 5 \]
More Examples
Harder Example

Compute $x/2$ rounded toward minus infinity.

```java
{{
if (x >= 0)
    y = x/2;
else
    y = -((-x+1)/2);
{{ 2y = x or 2y = x - 1 }}
```}

Note that, in Java, $a/b$ rounds toward zero.
Harder Example

Compute \( x/2 \) rounded toward minus infinity.

\[
\begin{align*}
\text{if } (x \geq 0) & \quad \text{then} \\
\quad y &= x/2; \\
\text{else} & \\
\quad y &= -(\frac{-x+1}{2}); \\
\end{align*}
\]

\[
\begin{align*}
\text{2y = x or 2y = x - 1}
\end{align*}
\]
Harder Example

Compute $x/2$ rounded toward minus infinity.

```c
if (x >= 0)
    y = x/2;
else
    y = -((-x+1)/2);
```

CSE 331 Spring 2020
Harder Example

Compute \( x/2 \) rounded toward minus infinity.

```
{{ }}
if (x >= 0)
    {{ x >= 0 }}
y = x/2;
    {{ 2y = x or 2y = x - 1 }}
else
    {{ x < 0 }}
y = -((-x+1)/2);
    {{ 2y = x or 2y = x - 1 }}
{{ 2y = x or 2y = x - 1 }}
```
Harder Example

Compute \( x/2 \) rounded toward minus infinity.

\[
\begin{cases}
\text{if } (x \geq 0) & \text{since } x \geq 0, \text{“/” rounds down so this is valid} \\
\qquad \{ 2y = x \text{ or } 2y = x - 1 \} \\
\text{else} & \\
\qquad \{ x < 0 \} \\
\qquad y = -(\((-x+1)/2\)); \\
\qquad \{ 2y = x \text{ or } 2y = x - 1 \} \\
\{ 2y = x \text{ or } 2y = x - 1 \}
\end{cases}
\]
Harder Example

Compute $x/2$ rounded toward minus infinity.

```c
if (x >= 0)
{
    ...
} else
{
    x < 0
    y = (x+1)/2; // was $y = -((-x+1)/2)$;
    y = -y;
    2y = x or 2y = x - 1
}```
Harder Example

Compute \(x/2\) rounded toward minus infinity.

\[
\begin{align*}
\text{if (x } & \geq 0) \\
\text{...}
\text{else}
\text{x < 0} \\
y & = (\text{-x+1})/2; \\
\text{2y = -x or 2y = -x + 1} \\
y & = \text{-y}; \\
\text{2y = x or 2y = x - 1}
\end{align*}
\]
Harder Example

Compute $x/2$ rounded toward minus infinity.

```c
{{ }}
if (x >= 0)
    ...
else
    {{ x < 0 }}
    y = (-x+1)/2;
    {{ 2y = (-x + 1) - 1 or 2y = -x + 1 }}
    y = -y;
    {{ 2y = x or 2y = x - 1 }}
```
Harder Example

Compute x/2 rounded toward minus infinity.

```c
{{ }}
if (x >= 0)
    ...
else
    {{ x < 0 }}
y = (-x+1)/2;
{{ 2y = (-x + 1) - 1 or 2y = -x + 1 }}
y = -y;
{{ 2y = x or 2y = x - 1 }}
```

since -x > 0, "/" rounds down so this is valid
Useful Subscripts Example: swap

- Consider code for a swapping \( x \) and \( y \)

\[
\begin{align*}
\{ & \text{tmp} = x; \\
\{ & \text{tmp} = x \} \\
\{ & x = y; \\
\{ & \text{tmp} = x_0 \text{ and } x = y \} \\
\{ & y = \text{tmp}; \\
\{ & \text{tmp} = x_0 \text{ and } x = y_0 \text{ and } y = \text{tmp} \}
\end{align*}
\]

- Post condition implies \( x = y_0 \) and \( y = x_0 \)
- i.e., their final values are equal to the original values swapped