Motivation for Reasoning

• Want a way to determine correctness without running the code

• Most important part of the correctness techniques
  – tools, inspection, testing

• You need a way to do this in interviews
  – key reason why coding interviews are done without computers

• This is not easy
Our Approach

• We will learn a set of formal tools for proving correctness
  – (later, this will also allow us to generate the code)

• Most professionals can do reasoning like this in their head
  – most do an informal version of what we will see
  – eventually, it will be the same for you

• Formal version has key advantages
  – teachable
  – mechanical (no intuition or creativity required)
  – necessary for hard problems
    • we turn to formal tools when problems get too hard
Formal Reasoning

- Invented by Robert Floyd and Sir Anthony Hoare
  - Floyd won the Turing award in 1978
  - Hoare won the Turing award in 1980

Robert Floyd
picture from Wikipedia

Tony Hoare
Terminology of Floyd 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 Triples

- 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

• Floyd 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 \wedge (and) \vee (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

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

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

\[ \{\{ x \neq 0 \}\} \ y = x^2; \{\{ y > 0 \}\} \]

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

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

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

\[
\{ \{ z \neq 1 \} \} \ y = z*z ; \ \{ \{ y \neq z \} \}
\]

Invalid
- counterexample: \( z = 0 \)
Checking Validity

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

• Soon: mechanical process 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
Floyd Logic Facts

• Suppose \{P\} S \{Q\} is valid.

• If \(P1\) is stronger than \(P\), then \{\(P1\)\} S \{\(Q\)\} is valid.

• If \(Q1\) is weaker than \(Q\), then \{\(P\)\} S \{\(Q1\)\} is valid.

• Example:
  – Suppose \(P\) is \(x \geq 0\) and \(P1\) is \(x > 0\)
  – Suppose \(Q\) is \(y > 0\) and \(Q1\) 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
Floyd 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
Floyd Logic Facts

• When is \{P\} ; \{Q\} is valid?
  – with no code in between

• Valid if any state satisfying P also satisfies Q
• I.e., if P is **stronger** than Q
Forward & Backward Reasoning
Example of Forward Reasoning

Work forward from the precondition

\[ \{\{ w > 0 \}\} \]
\[ x = 17; \]
\[ \{\{ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\} \} \]
\[ y = 42; \]
\[ \{\{ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\} \} \]
\[ z = w + x + y; \]
\[ \{\{ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\} \} \]
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 \}

\texttt{x = 17 ;}

\{ w > 0 \text{ and } x = 17 \}

\texttt{y = 42 ;}

\{ w > 0 \text{ and } x = 17 \text{ and } y = 42 \}

\texttt{z = w + x + y ;}

\{ \text{ } \}

\texttt{z = w + x + y ;}

\{ \text{ } \}
Example of Forward Reasoning

Work forward from the precondition

{{ w > 0 }}

\[ x = 17; \]

{{ w > 0 and x = 17 }}

\[ y = 42; \]

{{ w > 0 and x = 17 and y = 42 }}

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

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

Work forward from the precondition

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

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

Work backward from the desired postcondition

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

Work backward from the desired postcondition

\{ \text{________________________} \}

\textbf{x} = 17;

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

\textbf{y} = 42;

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

\textbf{z} = w + x + y;

\{ \{ z < 0 \} \}
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...
Correctness by Forward Reasoning

Use forward reasoning to determine if this code is correct:

```c
{{ w > 0 }}
  x = 17;
  y = 42;
  z = w + x + y;
{{ z > 50 }}
```
Example of Forward Reasoning

{{ \ w > 0 \}}
\ x = 17;
{{ \ w > 0 \ and \ x=17 \}}
\ y = 42;
{{ \ w > 0 \ and \ x=17 \ and \ y=42 \}}
\ z = w + x + y;
{{ \ w > 0 \ and \ x=17 \ and \ y=42 \ and \ z = w + 59 \}}
{{ \ z > 50 \}}

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:

\[
\{\{ w < -60 \}\}\]

\[
x = 17;
\]

\[
y = 42;
\]

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

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

Use backward reasoning to determine if this code is correct:

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

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:

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

\[
\begin{array}{c}
\{\{ P \}\}\,
\end{array}
\]

\[
\begin{array}{c}
\text{S1} \\
\{\{ P1 \}\}
\end{array}
\]

\[
\begin{array}{c}
\text{Valid provided P1 implies Q1} \\
\{\{ Q1 \}\}
\end{array}
\]

\[
\begin{array}{c}
\text{S2} \\
\{\{ Q \}\}
\end{array}
\]\n
CSE 331 Fall 2022 35
Combining Forward & Backward

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