CSE 331
Software Design & Implementation

Autumn 2025

Section 4 – Floyd Logic

Administrivia

- HW 4 released tonight, due **Friday 10/24 @ 6pm**

Ed Posting Guidelines

When posting on Ed, please follow our Ed posting guidelines. Thank you!

Ed Title Format

Homeworks: [AHW#] Q#: Short Title Describing the question

- Ex: [AHW2] Q1.a: proof formatting

InClass Work: [ICW# and Version] Q#: Short Title Describing the question

- Ex: [ICW1a] Q1.a: Stronger vs Weaker Spec
- Ex: [ICW2c] Q1.a: proof formatting

Section: [SC#] Q#: Short Title Describing the question

- Ex: [SC2] Q1.a: proof formatting

Proof By Calculation – Review

- The goal of proof by calculation is to *show* that an assertion is true *given* facts that you already know

- You should ***start*** the proof with the left side of the assertion and ***end*** the proof with the right side of the assertion. Each symbol (=, >, <, etc.) connecting each line of the proof is that line's relationship to the **previous line on the proof**

- Only modify one side. Never do work on both sides. We can only work with what you have from the previous line, using definitions and facts.

Structural Induction – Review

- Let **P(S)** be the claim
- To Prove P(S) holds for any list S, we need to prove two implications: base case and inductive case
  - **Base Case:** prove P(nil)
    - Use any known facts and definitions
  - **Inductive Hypothesis:** assume **P(L)** is true for a L: List
    - Use this in the inductive step ONLY ↩️
  - **Inductive Step:** prove **P(x :: L)** for any x : *Z,* L : List
    - Direct proof
    - Use known facts and definitions and Inductive Hypothesis
- Assuming we know P(L), if we prove P(x :: L), we then prove recursively that P(S) holds for any List

Hoare Triples – Review

- A Hoare Triple has 2 assertions and some code

  {{ P }}

  S

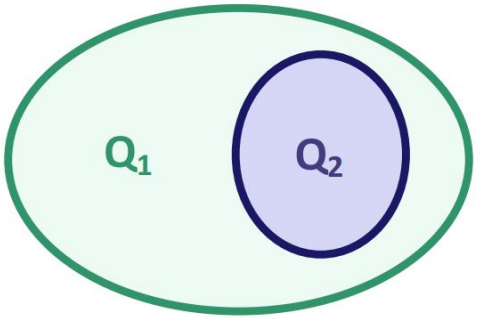
  {{ Q }}

  - P is a precondition, Q is the postcondition
  - S is the code

- Triple is "valid" if the code is correct:
  - S takes any state satisfying P into a state satisfying Q
    - Does not matter what the code does if P does not hold initially
    - We use Proof By Calculation to prove our Hoare Triples!

Stronger vs Weaker – Review

- Assertion is stronger iff it holds in a subset of states
  - **Stronger** assertion implies the **weaker** one:
    If Q_2 is true, Q_1 must also be true, $Q_2 \rightarrow Q_1$



- Different from strength in *specifications*

Question …

Which is the strongest assertion:

- $x > 3$
- $x \geq 3$
- $x > 3$ and $x \in \{2, 4, 6, 8, 10\}$
- $x > 3$ and $x \% 2 = 0$

Discuss with the person next to you

Question …

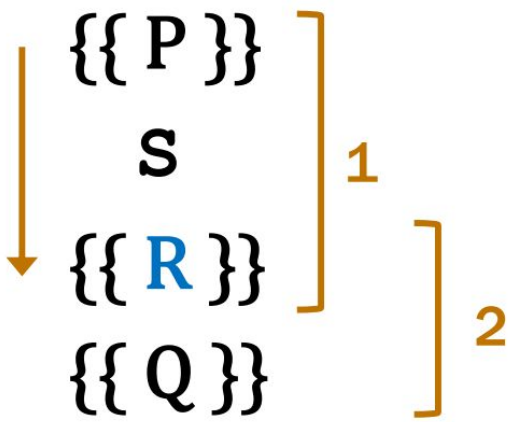
Which is the strongest assertion:

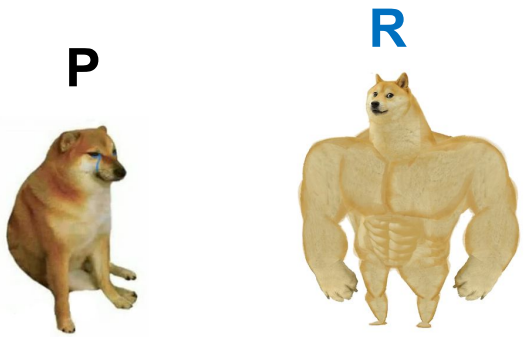
- $x > 3$
- $x \geq 3$
- **$x > 3$ and $x \in \{2, 4, 6, 8, 10\}$**
- $x > 3$ and $x \,\%\, 2 = 0$

Discuss with the person next to you

Forward Reasoning – Review

- Forwards reasoning fills in the postcondition
  - Gives strongest postcondition making the triple valid
- Apply forward reasoning to fill in R





  - Check second triple by proving that R implies Q

Forward Reasoning Error Example

```
{{ x > 1 }}
x = x + 1;
{{ x = x0 + 1 and x > 1 }}
y = 3 * x;
{{ x = x0 + 1 and y = 3 * x }}
z = y + 1;
{{ x = x0 + 1 and y = 3 * x and z = (3 * x) + 1 }}
```

Drops this assertion

What's wrong with these assertions?

Uses subscripts for an invertible operation

Simplifies assertions too early

Corrected Forward Reasoning Example

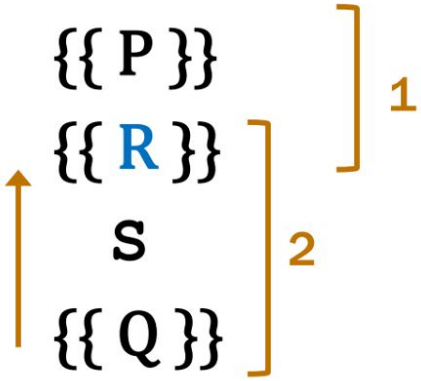
```
{{ x > 1 }}
x = x + 1;
{{ x - 1 > 1 }}
y = 3 * x;
{{ x - 1 > 1 and y = 3 * x }}
z = y + 1
{{ x - 1 > 1 and y = 3 * x and z = y + 1 }}
```

does not simplify assertions early

updates x for this operation rather than introducing subscripts

Backward Reasoning – Review

- Backwards reasoning fills in preconditions
  - **Just use substitution!**
  - Gives weakest precondition making the triple valid
- Apply backwards reasoning to fill in R



  - Check first triple by proving that P implies R

Forward & Backward General Rules

Forward Reasoning:

- After each line of code *update* variables in assertions based on how they they were changed by the line of code

Backward Reasoning:

- As you work your way up the code *directly* substitute how variables are modified in the code into your assertions

General:

- Do **not** drop or simplify assertions
- Do **not** use subscripts for invertible operations (addition and subtraction are *always* invertible)

Task 1 - Found Guilty of Reason

a) Use forward reasoning to fill in the missing assertions in the following code:

$\{\{ x \geqslant 5 \}\}$

```
y = x - 2;
```

{{ *x >= 5 and y = x - 2* }}

```
z = 3 * y;
```

{{ *x >= 5 and y = x - 2 and z = 3y* }}

```
z = z - 4;
```

{{ P : *x >= 5 and y = x - 2 and z + 4 = 3y* }}

$\{\{ Q : \ z \geqslant 0 \}\}$

Task 1 - Found Guilty of Reason

b) Show that the code is correct by proving by **calculation** that P implies Q.

We can see that Q holds since

$$
\begin{aligned}
z &= 3y - 4 && \text{since } z + 4 = 3y \\
&= 3(x - 2) - 4 && \text{since } y = x - 2 \\
&= 3x - 10 \\
&\geqslant 3 \cdot 5 - 10 && \text{since } x \geqslant 5 \\
&= 5 \\
&\geqslant 0
\end{aligned}
$$

Task 1 - Found Guilty of Reason

c) Use forward reasoning to fill in the missing assertions in the following code:

$\{\{\ y > 5 \text{ and } z > 2\ \}\}$

```
x = 4 * y - 3;
```

{{ *y > 5 and z > 2 and x = 4y - 3* }}

```
y = y - 5;
```

{{ *y + 5 > 5 and z > 2 and x = 4(y + 5) - 3* }}

```
z = z * y;
```

{{ P : *y + 5 > 5 and z/y > 2 and x = 4(y + 5) - 3* }}

$\{\{\ Q:\ x < 2z + 20\ \}\}$

Task 1 - Found Guilty of Reason

d) Show that the code is correct by proving by calculation that P implies Q.

We can see that Q holds since

$$
\begin{aligned}
x &= 4y + 17 && \text{since } x = 4(y+5) - 3 \\
&< 2z + 17 && \text{since } z > 2y \text{ because } y > 0 \\
&< 2z + 20
\end{aligned}
$$

Task 2 - Does a Duck Say “Back”?

a) Use backward reasoning to fill in the missing assertions in the following code:

Fill in each blank by applying the rules *exactly* as taught in lecture. Then, if you want, you can simplify the resulting assertion, but do not weaken it. Separate any simplified statement from the original by “↔”.

$\{\{ P : \ c \geqslant 0 \}\}$

$\{\{ Q : \ $ *2c + 1 >= c - 1* $\}\} \leftrightarrow \{\{ c \geqslant -2 \}\}$

```
b = 2*c;
```

$\{\{ $ *b + 1 >= c - 1* $\}\}$

```
c = c - 1;
```

$\{\{ $ *b + 1 >= c* $\}\}$

```
a = b + 1;
```

$\{\{ a \geqslant c \}\}$

Task 2 - Does a Duck Say “Back”?

b) Show that the code is correct by proving by calculation that P implies Q.

We can see that Q holds since $c \geqslant 0 \geqslant -2$.

Task 2 - Does a Duck Say “Back”?

c) Use backward reasoning to fill in the missing assertions in the following code:

$\{\{ P: \; x < w + 1 \text{ and } w > 0 \}\}$

$\{\{ Q: \;$ *2x - 8 < 4w* $\}\} \leftrightarrow \{\{ 2x < 4w + 8 \}\} \leftrightarrow \{\{ x < 2w + 4 \}\}$

```
y = 4 * w;
```

{{ *2x - 8 < y* }}

```
x = x * 2;
```

{{ *x - 8 < y* }}

```
z = x - 8;
```

$\{\{ z < y \}\}$

Task 2 - Does a Duck Say “Back”?

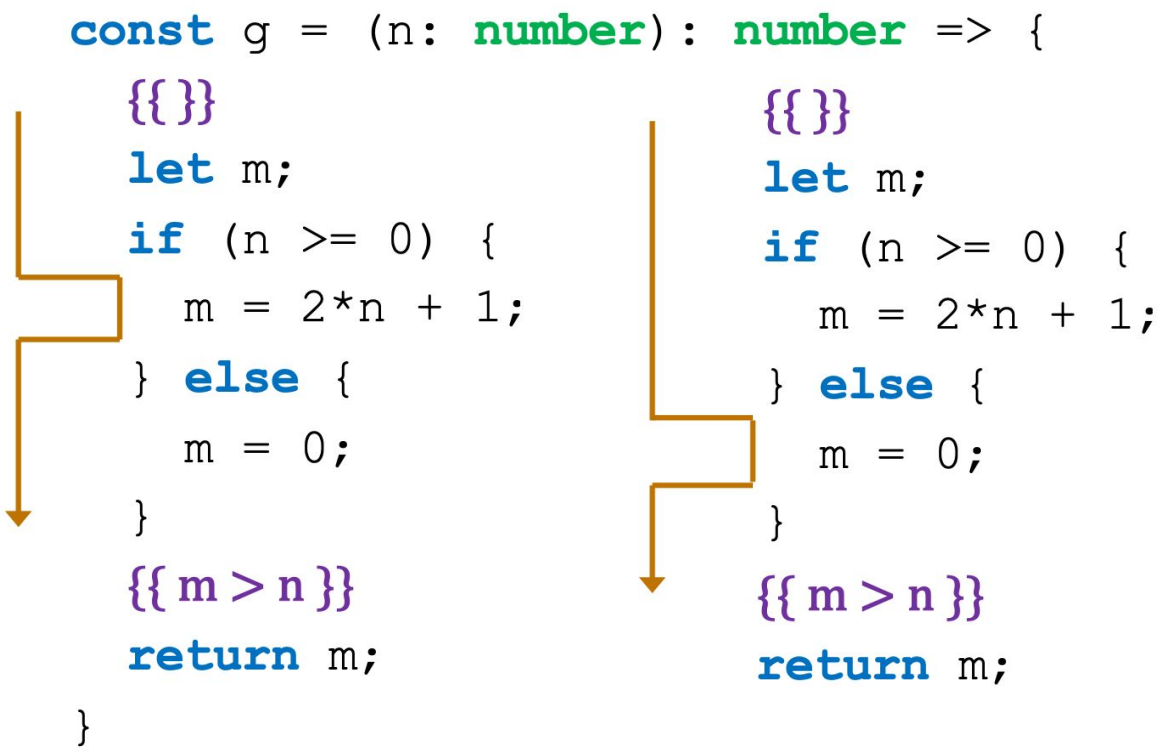
d) Show that the code is correct by proving by calculation that P implies Q.

We can see that Q holds since

$$\begin{aligned} x &< w + 1 \\ &< 2w + 1 \quad \text{since } w > 0 \\ &< 2w + 4 \end{aligned}$$

Conditionals – Review

- Reason through “then” and “else” branches independently and combine last assertion of both branches with an “or” at the end
- Prove that each implies post condition by cases
- **Note**: this is important for your homework!



Task 3 - Nothing To Be If-ed At

Use forward reasoning to fill in the assertions. Then, prove, by cases, that what we know at the end of the conditional implies the post condition. The final assertion of the if and else branches are labeled as P1 and P2 respectively, please this abbreviation in future assertions and your proofs to refer to the same set of facts.

{{ $s > 0$ and $k = s^2$ }}

```
if (s < 5) {
```

{{ 0 < s < 5 and k = s^2 }}

```
  j = k + s;
```

{{ 0 < s < 5 and k = s^2 and j = k + s }}

```
  j = j / 2;
```

{{ $P1$: 0 < s < 5 and k = s^2 and j0 = k + s and j = ⌊j0/2⌋ }}

```
} else {
```

{{ s >= 5 and k = s^2 }}

```
  j = k - s;
```

{{ s >= 5 and k = s^2 and j = k - s }}

```
  j = j + 2;
```

{{ $P2$: s >= 5 and k = s^2 and j - 2 = k - s }}

```
}
```

{{ P1 or P2 }}

{{ $j \leqslant k + s$ }}

Task 3 - Nothing To Be If-ed At

We'll prove by cases that the assertion just below the conditional implies the post condition $\{\{\ j \leqslant k + s\ \}\}$:

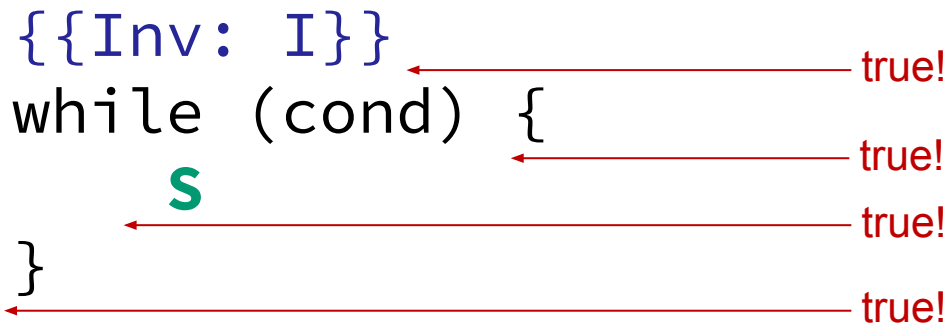
First, assuming $P1$:

$$
\begin{array}{ll}
j = \lfloor j_0 \ / \ 2 \rfloor & \text{Since } j = \lfloor j_0 \ / \ 2 \rfloor \\
\leqslant j_0/2 & \text{Def of floor} \\
= (k+s)/2 & \text{Since } j_0 = k + s \\
\leqslant k + s & \text{Since } s > 0 \text{ and } k = s^2 > 0
\end{array}
$$

Now, assuming $P2$:

$$
\begin{array}{ll}
j = k - s + 2 & \text{Since } j - 2 = k - s \\
\leqslant k + s & \text{Since } s \geqslant 5, \text{ we know } -s + 2 \leqslant s
\end{array}
$$

Loop Invariant – Review



- Loop invariant must be true **every time** at the top of the loop
  - The first time (before any iterations) and for the beginning of each iteration
- Also true every time at the bottom of the loop
  - Meaning it's true immediately after the loop exits
- During the body of the loop (during **S**), it isn't true

- Must use "`Inv`" notation to indicate that it's not a standard assertion

Question ….

Where is it allowed for a loop invariant not to hold?

- before the loop
- after the loop
- after entering the loop
- before exiting the loop
- during the code execution inside of the loop

Question ….

Where is it allowed for a loop invariant not to hold?

- before the loop
- after the loop
- after entering the loop
- before exiting the loop
- **during the code execution inside of the loop**

Task 4 - Everybody Loops

$\{\{\, x = x_0 \text{ and } x_0 \geqslant 0 \,\}\}$

```
int y = 0;
```

$\{\{\,$ Inv: $x_0 - 10y = x$ and $x \geqslant 0 \,\}\}$

```
while (x >= 10) {
    y = y + 1;
    x = x - 10;
}
```

$\{\{\, 10y \leqslant x_0 \text{ and } x_0 < 10(y+1) \text{ and } x = x_0 - 10y \,\}\}$

a) Prove that the invariant is true when we get to the top of the loop the first time.

Forward reasoning tells us that $x = x_0$, $x_0 \geqslant 0$, and $y = 0$. The first part of the invariant holds since

$$\begin{aligned} x_0 - 10y &= x - 10y && \text{since } x = x_0 \\ &= x && \text{since } y = 0 \end{aligned}$$

The second fact holds since $x = x_0 \geqslant 0$

Task 4 - Everybody Loops

b) Prove that, when we exit the loop, the postcondition holds.

When we exit the loop, we know that $x_0 - 10y = x$, $x \geqslant 0$, and $x < 10$. The first part of the postcondition holds since

$$\begin{aligned} 10y &= x_0 - x && \text{since } x_0 - 10y = x \\ &\leqslant x_0 && \text{since } x \geqslant 0 \end{aligned}$$

the second part of the postcondition holds since

$$\begin{aligned} x_0 &= x + 10y && \text{since } x_0 - 10y = x \\ &< 10 + 10y && \text{since } x < 10 \\ &= 10(y+1) \end{aligned}$$

and the third part is a restatement of the first fact from the invariant.

Task 4 - Everybody Loops

c) Prove that the invariant is preserved by the body of the loop. Use either forward or backward reasoning (your choice) to reduce the body to an implication and then check that it holds.

We can apply backward reasoning in the loop to get condition Q shown here:

$\{\{\, P : \; x_0 - 10y = x \text{ and } x \geqslant 0 \text{ and } x \geqslant 10 \,\}\}$

$\{\{\, Q : \; x_0 - 10y = x \text{ and } x \geqslant 10 \,\}\}$

$\{\{\, x_0 - 10(y+1) = x - 10 \text{ and } x - 10 \geqslant 0 \,\}\}$

```
y = y + 1n;
```

$\{\{\, x_0 - 10y = x - 10 \text{ and } x - 10 \geqslant 0 \,\}\}$

```
x = x - 10n;
```

$\{\{\, x_0 - 10y = x \text{ and } x \geqslant 0 \,\}\}$

Both parts of Q are explicitly provided in P, so no calculations are needed.