CSE 331
Software Design & Implementation

Spring 2025

Section 6 – Floyd Logic

Administrivia

- HW 6 released tonight, due Wednesday 5/14 at 11pm

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.

Hoare Triples – Review

- A Hoare Triple has 2 assertions and some code

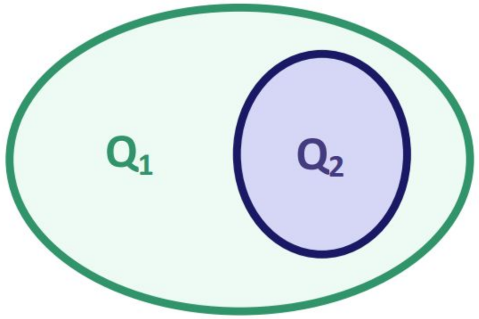
```
{{ P }}
  S
{{ Q }}
```

  - 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

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*:
  - A more **permissive** spec:
    - Stronger postcondition: guarantees more specific output
    - Weaker precondition: handles more allowable inputs

    compared to a weaker one (i.e. weaker assertion)
  - A more **restrictive** spec:
    - Weaker postcondition: guarantees more general output
    - Stronger precondition: requires more restrictive input

Question …

Which is the stronger assertion:

- $x > 3$
- $x \geq 3$

discuss with the person next to you

Question …

Which is the stronger assertion:

- $\mathbf{x > 3}$
- $x \geq 3$

Question …

Which is the stronger 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 …

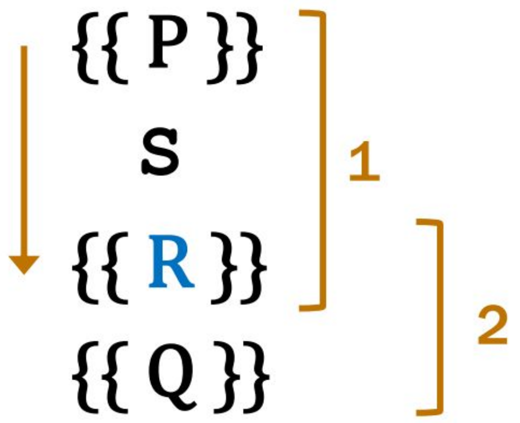
Which is the stronger assertion:

- $x > 3$
- $x \geq 3$
- **$x > 3$ 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

Question 1a: It's Forward Against Mine

Use forward reasoning to fill in the missing assertions.

{{ $x \geq 4$ }}
y = x - 2n;
{{ $x \geq 4$ and $y = x - 2$ }}
z = 2n * y;
{{ $x \geq 4$ and $y = x - 2$ and $z = 2y$ }}
z = z - 2n;
{{ $x \geq 4$ and $y = x - 2$ and $z + 2 = 2y$ }}
{{ $z \geq 0$ }}

We can see that the last assertion implies the postcondition $z \geq 0$ as follows:

$z = 2y - 2$	since $z + 2 = 2y$
$= 2(x-2) - 2$	since $y = x - 2$
$= 2x - 6$	
$\geq 2 * 4 - 6$	since $x \geq 4$
$= 2$	

Since $z \geq 2$, the assertion implies the postcondition

Question 1b

Use forward reasoning to fill in the missing assertions, then prove that the postcondition holds.

{{ $x \leq 4$ }}

y = x + 4;

{{ $x \leq 4$ and $y = x+4$ }}

x = x / 2;

{{ $x_0 \leq 4$ and $x = x_0 / 2$ and $y = x_0+4$ }}

y = y + 2 * x;

{{ $x_0 \leq 4$ and $x = x_0 / 2$ and $y - 2x = x_0+4$ }}

{{ $y < 14$ }}

Question 1b

Prove that the postcondition holds.

From our final assertion we have:

$x_0 \leq 4,\ x = x_0 / 2,\ y - 2x = x_0+4$

Which simplifies to:

$x_0 \leq 4,\ x = x_0 / 2,\ y = x_0+ 4 + 2x$

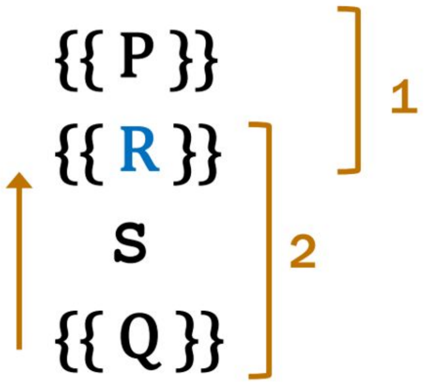
Using these facts, we can do:

$$
\begin{aligned}
y &= x_0+ 4 + 2x && \text{since } y-2x = x_0+4 \\
&= x_0 + 4 + 2(x_0/2) && \text{since } x = x_0/2 \\
&\leq 4 + 4 + 2(4/2) && \text{since } x_0 \leq 4 \\
&\leq 4 + 4 + 4 \\
&\leq 12
\end{aligned}
$$

Since **$y \leq 12$**, the assertion implies the postcondition: **$y < 14$**

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

- **Good example problems in section worksheet!**

Question 2a: Not for a Back of Trying

Use backward reasoning to fill in the missing assertions, then prove that the precondition implies what is required.

```
{{ x < w + 1 }}
{{        3x - 9 < 3w                          }}
y = 3n * w;
{{        3x - 9 < y                           }}
x = x * 3n;
{{        x - 9 < y                            }}
z = x - 9n;
{{ z < y }}
```

Question 2a: Not for a Back of Trying

From our final (top most) assertion, we have:

```
3x - 9 < 3w
```

Which simplifies to:

```
x < w + 3
```

Using that, we get:

```
We can see that x < w + 1 implies the precondition:


x < w + 1            from the pre condition
< (w + 1) + 2       adding 2 to w+1 is larger than w+1
= w + 3
```

Question 2b

Use backward reasoning to fill in the missing assertions, then prove that the precondition implies what is required.

```
{{ x > 1 }}
{{ ____x - 4 >= -3____________________ }}
y = x - 4n;                         this reduces to: y >= -3
{{ ________3y + 6 >= y________________ }}
z = 3n * y;
{{ ______z + 6 >= y___________________ }}
z = z + 6n;
{{ z ≥ y }}
```

Question 2b

From our final (top most) assertion, we have:

```
x - 4 >= -3
```

Which simplifies to:

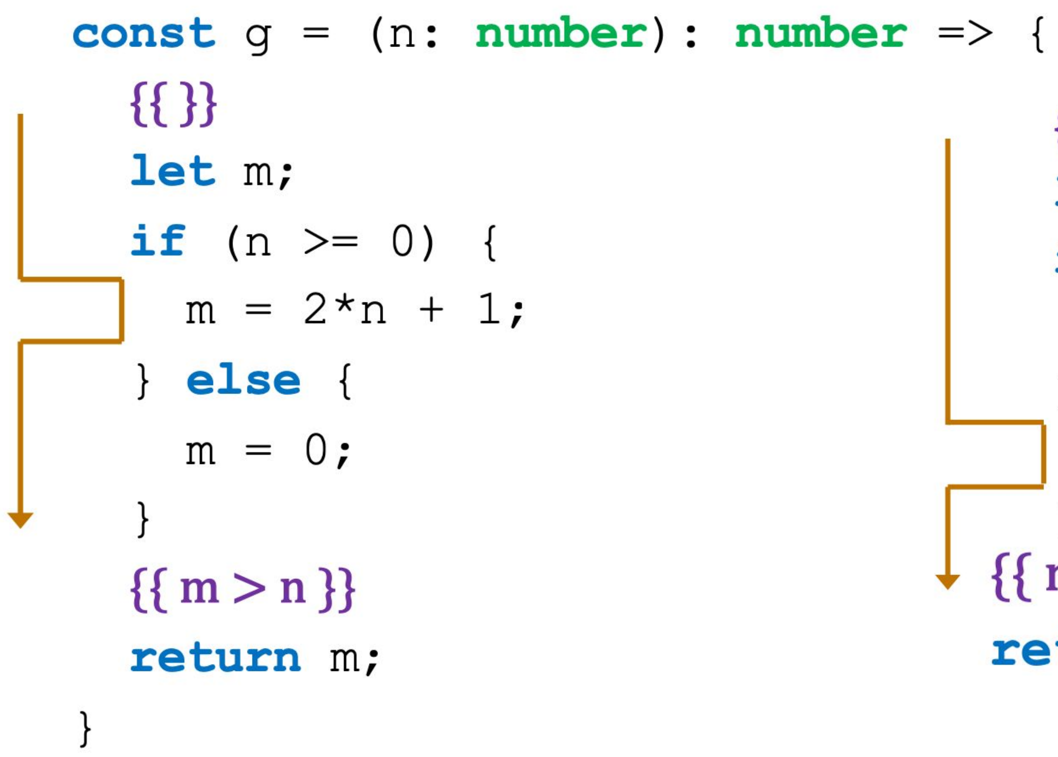
```
x >= 1
```

From that we get:

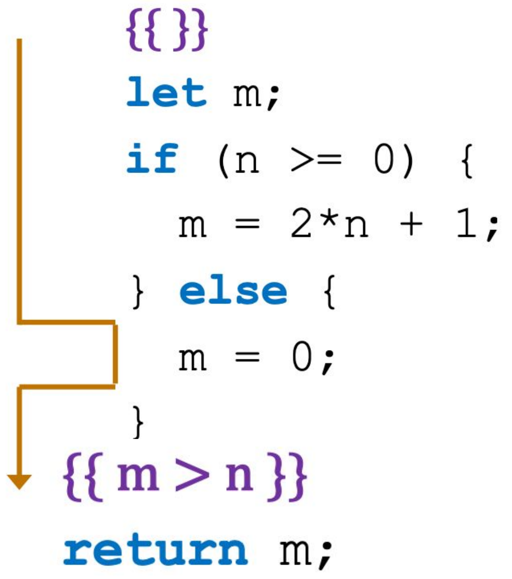
```
we can immediately see from the specification above
that x > 1 implies x >= 1
```

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



```
const g = (n: number): number => {
  {{ }}
  let m;
  if (n >= 0) {
    m = 2*n + 1;
  } else {
    m = 0;
  }
  {{ m > n }}
  return m;
}
```



```
{{ }}
let m;
if (n >= 0) {
  m = 2*n + 1;
} else {
  m = 0;
}
{{ m > n }}
return m;
```

Question 3a: Nothing to Be If-ed At

a) Use forward reasoning to fill in the assertions. Then, combine the branches to assert the invariant we know at the end of the conditional and complete an argument by cases that this invariant implies {{y > 2}}.

Assume that x and y are both integers.

```
{{ x ≥ 0 }}
if (x >= 6n) {
  {{ x >= 6 }}
  y = 2n * x - 10n;
  {{ x >= 6 and y = 2x - 10 }}
} else {
  {{ x >= 0 and x < 6 }}
  y = 20n - 3n * x;
  {{ x >= 0 and x < 6 and y = 20 - 3x }}
}
{{ (x >= 6 and y = 2x - 10) or (x >= 0 and x < 6 and y = 20 - 3x)}}
{{ y ≥ 2 }}
```

Question 3a: Nothing to Be If-ed At

We want to show that our assertions imply y >= 2.

What are the cases?

```
x ≥ 6 and y = 2x - 10

and

x ≥ 0 and x < 6 and y = 20 - 3x
```

Question 3a – Proof by cases

Case 1: `x ≥ 6 and y = 2x - 10`

We can show:

$y = 2x - 10$

$\quad \geq 2(6) - 10$ since $x \geq 6$

$\quad = 2$

Case 2: `x ≥ 0 and x < 6 and y = 20 - 3x`

First note that $x < 6$ which mean $-x > -6$ which means that $-3x > -18$

We can show:

$y = 20 - 3x$

$\quad > 20 - 18$ since $-3x > -18$

$\quad = 2$

Question 3b – "then" branch

Use forward reasoning to fill in the assertions. Then, combine the branches to assert the invariant we know at the end of the conditional and complete an argument by cases that this invariant implies {{s >= 1}}

```
{{ s ≠ t and t > 0 }}
  if (s > t) {
    {{ $t > 0$ and $s > t$ }}
    s = s / t;
    {{ $t > 0$ and $s_0 > t$ and $s = s_0 / t$ }}
  } else {
    s = t - s;
  }
{{ $t > 0$ and $s_0 > t$ and $s = s_0 / t$ }}
{{ s ≥ 1 }}
```

Question 3b – “else” branch

```
{{ s ≠ t and t > 0 }}
if (s > t) {
    s = s / t;
} else {
    {{ $t > 0$ and $s < t$ }}
    s = t - s;
    {{ $t > 0$ and $t - s < t$ }}
}
{{ (t > 0 and $s_0$ > t and s = $s_0$/t) or (t > 0 and t–s < t) }}
{{ s ≥ 1 }}
```

Question 3b

We want to show that our assertions imply s >= 1

What are the cases?

$t > 0$, $s_0 > t$, $s = s_0/t$

and

$t > 0$, $t - s < t$

Question 3b – Proof by cases

Case 1: $t > 0$ and $s_0 > t$ and $s = s_0 / t$

$s = s_0 / t$

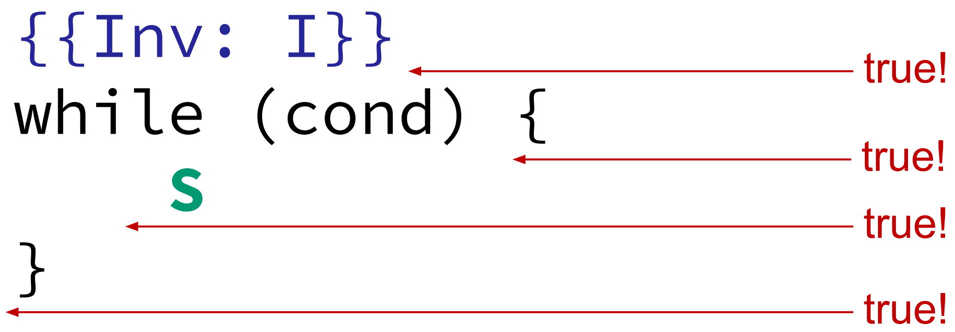
$> t/t$ since $s_0 > t$ *and* $t > 0$ (meaning $t \neq 0$)

$= 1$

Case 2: $t > 0$ and $t - s < t$

We can rearrange $t - s < t$ to get $0 < s$. Since s is an integer, this means $1 \leq s$, or equivalently $s \geq 1$.

Loop Invariant – Review



```
{{Inv: I}}
while (cond) {
    S
}
```

- 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**

Well-Known Facts About Lists

- Feel free to cite these in your proofs! They're easily proven by structural induction (and you don't have to do that again)

- Lemma 2: **concat(L, nil) = L for any list L**

- Lemma 3: **rev(rev(L)) = L for any list L**

- Lemma 4: **concat(concat(L, R), S)**
  = concat(L, concat(R, S)) for any lists L, R, S

See course website for proof of these statements

Question 4a: The Only Game in Down

The function “countdown” takes an integer argument “n” and returns a list containing the numbers n, …, 1. It can be defined recursively as follows:

countdown : N → List

countdown(0)	:=	nil
countdown(n + 1)	:=	(n + 1) :: countdown(n)

This function is defined recursively on a natural number so it fits the natural number template from lecture. In this problem, we will prove the following code correctly calculates countdown(n). The invariant for the loop is already provided.

```
let i: bigint = 0;
let L: List = nil;
{{Inv: L=countdown(i)}}
while (i !== n) {
    i = i + 1;
    L = cons(i, L);
}
{{ L = countdown(n) }}
```

Prove that the invariant is true at top of loop the first time.

At the top of the loop initially, we can see that:
L = *nil*   and   *i* = 0,   so we have:

L	= *nil*	
	= countdown(0)	def countdown
	= countdown(i)	since i = 0

Question 4b

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

After the loop we know that i = n and that L = countdown(i). Thus, we can see:

L = countdown(i)	Given by invariant
= countdown(n)	since i = n

https://tinyurl.com/sp331secBD6

Question 4c

(c) Prove that the invariant is preserved by the body of the loop. To do this, use backward reasoning to reason until the statement "`i = i + 1;`". Then complete the correctness check by verifying that the assertion you produced with backward reasoning implies the invariant.

```
let i: bigint = 0;
let L: List = nil
{{ Inv: L = countdown(i) }}
while (i !== n) {
    {{ L = countdown(i) and i ≠ n }}
    {{ (i + 1) :: L = countdown(i + 1) }}
    i = i + 1;
    {{ i :: L = countdown(i) }}
    L = cons(i, L);
    {{ L = countdown(i) }}
}
{{L = countdown(n)}}
```

To prove that `L=countdown(i)` implies `(i+1)::L = countdown(i+1)`, we can calculate:

(i + 1) :: L = (i + 1) :: countdown(i)	since L = countdown(i)
= countdown(i + 1)	def of countdown

Practice problem

Please practice on your own section problem 5. Good practice for homework!

THANK YOU :)

https://tinyurl.com/sp331secBD6