CSE 331
Software Design & Implementation

Winter 2025

Section 6 – Floyd Logic

Administrivia

- HW 6 released tonight, due Wednesday 2/19 at 11pm

Hoare Triples – Review

- A Hoare Triple has 2 assertions and some code

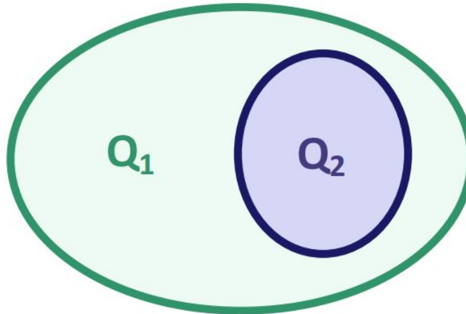
  {{ P }}
  S
  {{ Q }}

  - 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, $Q_2 \rightarrow Q_1$

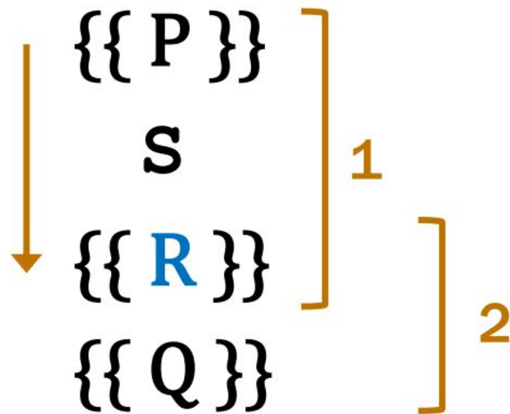


- Different from strength in *specifications*:
  - A stronger spec:
    - Stronger postcondition: guarantees more specific output
    - Weaker precondition: handles more allowable inputs

  compared to a weaker one

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;
```

$\{\{$ ______________________________ $\}\}$

```
z = 2n * y;
```

$\{\{$ ______________________________ $\}\}$

```
z = z - 2n;
```

$\{\{$ ______________________________ $\}\}$

$\{\{\, z \geq 0 \,\}\}$

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

Question 1b

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

$\{\{\, x \leqslant 4 \,\}\}$

```
y = x + 4n;
```

$\{\{\ \underline{\hspace{12cm}}\ \}\}$

```
x = x / 2n;
```

$\{\{\ \underline{\hspace{12cm}}\ \}\}$

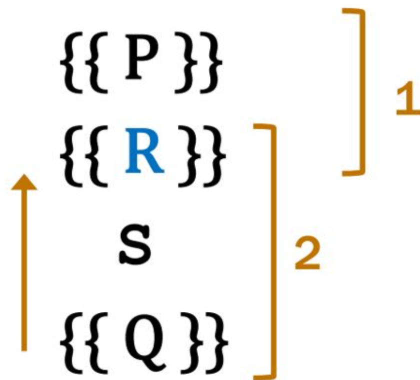
```
y = y + 2 * x;
```

$\{\{\ \underline{\hspace{12cm}}\ \}\}$

$\{\{\, y < 14 \,\}\}$

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.

$\{\{\ x < w + 1\ \}\}$

$\{\{$ ______________________________ $\}\}$

```
y = 3n * w;
```

$\{\{$ ______________________________ $\}\}$

```
x = x * 3n;
```

$\{\{$ ______________________________ $\}\}$

```
z = x - 9n;
```

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

Question 2b

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

$\{\{\, x > 1 \,\}\}$

$\{\{$ ______________________________ $\}\}$

```
y = x - 4n;
```

$\{\{$ ______________________________ $\}\}$

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

$\{\{$ ______________________________ $\}\}$

```
z = z + 6n;
```

$\{\{\, z \geq y \,\}\}$

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 $\{\{y \geqslant 2\}\}$.

Assume that x and y are both integers.

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

Question 3b – "then" branch

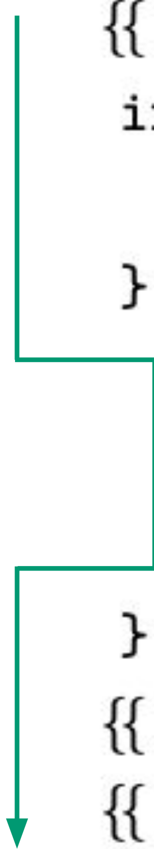
Use forward reasoning to fill in the assertions. Then, combine the branches to assert the invariant we know at the end of the conditional and complete an argument by cases that this invariant implies {{s >= 1}}

$\{\{\, s \neq t \text{ and } t > 0 \,\}\}$

```
if (s >= t) {
  {{ ______________________________ }}
  s = s / t;
  {{ ______________________________ }}
} else {
  s = t - s;
}
  {{ ______________________________ }}
```

$\{\{\, s \geq 1 \,\}\}$

Question 3b – "else" branch



$\{\{\ s \neq t \text{ and } t > 0\ \}\}$

```
if (s >= t) {
    s = s / t;
} else {
```

$\{\{$ ______________________________ $\}\}$

```
    s = t - s;
```

$\{\{$ ______________________________ $\}\}$

```
}
```

$\{\{$ ______________________________ $\}\}$

$\{\{\ s \geq 1\ \}\}$

Loop Invariant – Review

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

true!
true!
true!
true!

- Loop invariant must be true **<u>every time</u>** 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

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

Question 4a: The Only Game in Down

The function "countdown" takes an integer argument "n" and returns a list containing the numbers $n, \ldots, 1$. It can be defined recursively as follows:

$$\text{countdown} : \mathbb{N} \to \text{List}$$

$$\begin{aligned} \text{countdown}(0) &:= \text{nil} \\ \text{countdown}(n+1) &:= (n+1) :: \text{countdown}(n) \end{aligned}$$

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) }}
```

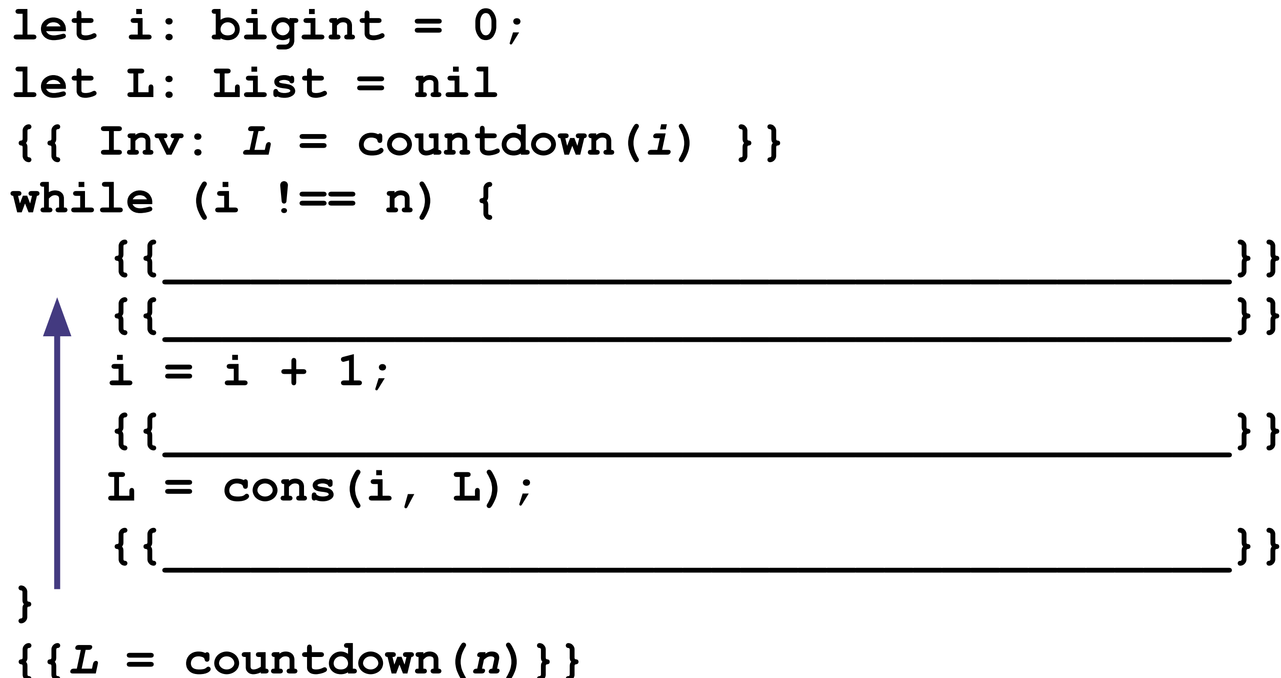
Prove that the invariant is true at top of loop the first time.

Question 4b

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

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.



Question 5: Chicken Noodle Loop

The function sum-abs calculates the sum of the absolute values of the numbers in a list. We can give it a formal definition as follows:

$$\text{sum-abs} : \text{List} \to \mathbb{Z}$$

$$\begin{aligned}
\text{sum-abs}(\text{nil}) &:= 0 \\
\text{sum-abs}(x :: L) &:= -x + \text{sum-abs}(L) \quad \text{if } x < 0 \\
\text{sum-abs}(x :: L) &:= x + \text{sum-abs}(L) \quad \text{if } x \geqslant 0
\end{aligned}$$

In this problem, we will prove that the following code correctly calculates sum-abs(L). The invariant for the loop is already provided. It references L_0, which is the initial value of L when the function starts.

```
let s: bigint = 0;
{{ Inv: s + sum-abs(L) = sum-abs(L_0) }}
while (L.kind !== ''nil'') {
   if (L.hd < 0n) {
      s = s + -L.hd;
   } else {
      s = s + L.hd;
   }
   L = L.tl;
}
{{ s = sum-abs(L_0) }}
```

Question 5a

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

```
let s: bigint = 0;
{{ Inv: s + sum-abs(L) = sum-abs(L_0) }}
while (L.kind !== ''nil'') {
    if (L.hd < 0n) {
        s = s + -L.hd;
    } else {
        s = s + L.hd;
    }
    L = L.tl;
}
{{ s = sum-abs(L_0) }}
```

$$\text{sum-abs} : \text{List} \to \mathbb{Z}$$

$$
\begin{aligned}
\text{sum-abs}(\text{nil}) &:= 0 & \\
\text{sum-abs}(x :: L) &:= -x + \text{sum-abs}(L) & \text{if } x < 0 \\
\text{sum-abs}(x :: L) &:= x + \text{sum-abs}(L) & \text{if } x \geqslant 0
\end{aligned}
$$

Question 5b

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

```
let s: bigint = 0;
{{ Inv: s + sum-abs(L) = sum-abs(L_0) }}
while (L.kind !== ''nil'') {
    if (L.hd < 0n) {
        s = s + -L.hd;
    } else {
        s = s + L.hd;
    }
    L = L.tl;
}
{{ s = sum-abs(L_0) }}
```

$\text{sum-abs} : \text{List} \to \mathbb{Z}$

$$
\begin{aligned}
\text{sum-abs}(\text{nil}) &:= 0 \\
\text{sum-abs}(x :: L) &:= -x + \text{sum-abs}(L) && \text{if } x < 0 \\
\text{sum-abs}(x :: L) &:= x + \text{sum-abs}(L) && \text{if } x \geqslant 0
\end{aligned}
$$

Question 5c

(c) Prove that the invariant is preserved by the body of the loop. To do this, use backward reasoning to reason through the last assignment statement "`L = L.tl;`". Then, use forward reasoning for each branch of the "`if`" statement (as in Problem 3). Finally, complete the correctness check by verifying that each of the assertions you produced with forward reasoning implies the assertion produced by backward reasoning immediately above the last assignment statement.

We have previously used the fact that, when $L \neq \text{nil}$, we know that $L = \text{cons}(x, R)$ for some $x : \mathbb{Z}$ and $R : \text{List}$. However, in the code, we know exactly what x and R are, namely, $x = L.\text{hd}$ and $R = L.\text{tl}$. Hence, when $L \neq \text{nil}$, we actually have $L = \text{cons}(L.\text{hd}, L.\text{tl})$. Feel free to use that in your proof.

Question 5c: Fill in & verify assertions

```
{{Inv: s + sum-abs(L) = sum-abs(L0)
While (L.kind !== “nil”){
   {{________________________________________________}}
   if (L.hd < 0n) {
      {{______________________________________________}}
      s = s + - L.hd;
      {{______________________________________________}}
   } else {
      {{______________________________________________}}
      s = s + L.hd
      {{______________________________________________}}
   }
   {{________________________________________________}}
     {{______________________________________________}}
   {{________________________________________________}}
   L = L.tl
   {{________________________________________________}}
}  {{  s + sum-abs(L) = sum-abs(L0)  }}
```