CSE 331
Software Design & Implementation

Winter 2025

Section 5 - Reasoning

Administrivia

- HW5 will be released later tonight and is due next Wednesday, 2/12, @11pm!
- Remember to check the autograder / linter output when you submit!

Proof By Calculation - Review

- Proving implications is the **core step** of reasoning
- Uses known facts and definitions (ex: len(nil) = 0)
  - Written in our math notation!

- Start from the left side of the inequality to be proved
- Chain of "=" shows first = last
- Chain of "=" and "≤" shows first ≤ last
- Directly cite the definition of a function

Proof By Calculation - Example

```
// Inputs x and y are positive integers
// Returns a positive integer.
const f = (x: bigint, y, bigint): bigint => {
  return x * y;
};
```

- Known facts "$x \geq 1$" and "$y \geq 1$"
- Correct if the return value is a positive integer

$$
\begin{aligned}
x * y &\geq x * 1 \quad \text{since } y \geq 1 \\
&\geq 1 * 1 \quad \text{since } x \geq 1 \\
&\geq 1
\end{aligned}
$$

- Calculation shows that $x * y \geq 1$

Proof By Calculation - Citing Functions

sum(nil) := 0

sum(x :: L) := x + sum(L)

- Know "a ≥ 0", "b ≥ 0", and "L = a :: b :: nil"

- Prove the "sum(L)" is non-negative

sum(L)	= sum(a :: b :: nil)	since L = a :: b :: nil
	= a + sum(b :: nil)	def of sum
	= a + b + sum(nil)	def of sum
	= a + b def of sum	
	≥ 0 + b since a ≥ 0	
	≥ 0 since b ≥ 0	

Task 1: Twice things up

You see the following snippet in some TypeScript code. It uses cons and nil, which are TypeScript implementations of "cons" and "nil", and also `equal`, which is a TypeScript implementation of "=" on lists.

```
if (equal(L, cons(1, cons(2, nil)))) {
  const R = cons(2, cons(4, nil));  // = twice(L)
  return cons(0, R);                // = twice(cons(0, L))
}
```

The comments show the definition of what *should* be returned (the specification), but the code is *not* a direct translation of those. Below, we will use reasoning to prove that the code is correct.

(a) Using the fact that $L = 1{::}2{::}\text{nil}$, prove by calculation that $\text{twice}(L) = R$, where R is the constant list defined in the code. I.e., prove that

$$\text{twice}(L) = 2{::}4{::}\text{nil}$$

twice : List ·

twice(nil) :=

twice(a :: L) :=

$\text{twice}(L)$

$= \text{twice}(1{::}2{::}\text{nil})$	Def of L
$= 2{::}\text{twice}(2{::}\text{nil})$	Def of twice
$= 2{::}4{::}\text{twice}(\text{nil})$	Def of twice
$= 2{::}4{::}\text{nil}$	Def of twice

Task 1: Twice things up

(b) Using the facts that L = 1::2::nil and R = 2::4::nil, prove by calculation that the code above returns the correct value, i.e., prove that

$$\text{twice}(0 :: L) = 0 :: R$$

Feel free to cite part (a) in your calculation.

$$\text{twice} : \text{List} \to \text{List}$$

$$\begin{aligned} \text{twice}(\text{nil}) &:= \text{nil} \\ \text{twice}(\text{a} :: \text{L}) &:= \text{2a} :: \text{twice}(\text{L}) \end{aligned}$$

Task 2: It's Raining Len

You see the following snippet in some TypeScript code. It uses `twice_evens`, which is a TypeScript implementation of twice-evens from the previous problem, as well as `len` from before.

```
return 2 + len(twice_evens(L));   // = len(twice-evens(cons(3, cons(4, L))))
```

The comment shows the definition of what should be returned (the specification), but the code is not a direct translation of that. Below, we will use reasoning to prove that the code is correct.

Task 2: It's Raining Len

(a) Let a and b be any integers. Prove by calculation that

$$\text{len}(\text{twice-evens}(a :: b :: L)) = 2 + \text{len}(\text{twice-evens}(L))$$

$$\text{twice-evens} : \text{List} \to \text{List}$$

$$\begin{aligned} \text{twice-evens}(\text{nil}) &:= \text{nil} \\ \text{twice-evens}(a :: L) &:= 2a :: \text{twice-odds}(L) \end{aligned}$$

$$\text{twice-odds} : \text{List} \to \text{List}$$

$$\begin{aligned} \text{twice-odds}(\text{nil}) &:= \text{nil} \\ \text{twice-odds}(a :: L) &:= a :: \text{twice-evens}(L) \end{aligned}$$

$$\text{len} : \text{List} \to \mathbb{Z}$$

$$\begin{aligned} \text{len}(\text{nil}) &:= 0 \\ \text{len}(a :: L) &:= 1 + \text{len}(L) \end{aligned}$$

Task 2: It's Raining Len

(b) Explain why the direct proof from part (a) shows that the code is correct according to the specification (written in the comment).

Defining Function By Cases – Review

- Sometimes we want to define functions by cases
  - **Ex:** define $f(\mathrm{n})$ where $\mathrm{n} : \mathbb{Z}$

$$\begin{aligned} \textbf{func } \mathrm{f}(\mathrm{n}) &:= 2\mathrm{n} + 1 && \textbf{if } \mathrm{n} \geq 0 \\ \mathrm{f}(\mathrm{n}) &:= 0 && \textbf{if } \mathrm{n} < 0 \end{aligned}$$

  - To use the definition $f(\mathrm{m})$, we need to know if $\mathrm{m} > 0$ or not
  - This new code structure requires a new proof structure

Proof By Cases – Review

- Split a proof into cases:
  - **Ex:** $\texttt{a = True}$ and $\texttt{a = False}$ or $\texttt{n >= 0}$ and $\texttt{n < 0}$
  - These cases needs to be *exhaustive*

- **Ex:** **func** $f(n) := 2n + 1$ **if** $n \geq 0$
  $f(n) := 0$ **if** $n < 0$

  Prove that $f(n) \geq n$ **for any** $n : \mathbb{Z}$

Case $n \geq 0$:

$f(n) = 2n + 1$ def of f (since $n \geq 0$)
$> n$ since $n \geq 0$

Case $n < 0$:

$f(n) = 0$ def of f (since $n < 0$)
$\geq n$ since $n < 0$

Since these 2 cases are *exhaustive*,
$\texttt{f(n) >= n}$
holds in general

Task 3: Swapaholic

Prove by cases that $\text{swap}(a :: L) \neq \text{nil}$ for any integer $a : \mathbb{Z}$ and list L.

$$\text{swap} : \text{List} \to \text{List}$$

$$\begin{aligned}
\text{swap}(\text{nil}) &:= \text{nil} \\
\text{swap}(a :: nil) &:= a :: \text{nil} \\
\text{swap}(a :: b :: L) &:= b :: a :: \text{swap}(L)
\end{aligned}$$

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 know 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(cons(x, L))** for any x : *Z*, L : List
    - Direct proof
    - Use know facts and definitions and Inductive Hypothesis
- Assuming we know P(S), if we prove P(cons(x, L)), we then prove recursively that P(S) holds for any List

Structural Induction - 331 Format

The following is the structural induction format we recommend for using in your homework (the staff solution also follows this format)

1) **Introduction** - define P(L) to be what we are trying to prove
2) **Base Case** - show P(nil) holds
3) **Inductive Hypothesis** - assume P(L) is true for an arbitrary list
4) **Inductive Step** - show P(cons(a, L)) holds
5) **Conclusion** - “We have shown that P(L) holds for any list”

Note: You do not have to follow this format but your solution MUST include all the information above

Task 4: Here Comes The Sum

You see following snippet in some TypeScript code:

```
const s = sum(L);
...
return 2 * s;  // = sum(twice(L))
```

This code claims to calculate the answer $\text{sum}(\text{twice}(L))$, but it actually returns $2\,\text{sum}(L)$. Prove this code is correct by showing that $\text{sum}(\text{twice}(S)) = 2\,\text{sum}(S)$ holds for any list S by structural induction.

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

$$\begin{aligned} \text{sum}(\text{nil}) &:= 0 \\ \text{sum}(a :: L) &:= a + \text{sum}(L) \end{aligned}$$

$$\text{twice} : \text{List} \to \text{List}$$

$$\begin{aligned} \text{twice}(\text{nil}) &:= \text{nil} \\ \text{twice}(\text{a} :: \text{L}) &:= \text{2a} :: \text{twice}(\text{L}) \end{aligned}$$

Task 5: Can You Sum A Few Bars?

Prove that

$$\mathsf{sum}(\mathsf{twice\text{-}evens}(L)) + \mathsf{sum}(\mathsf{twice\text{-}odds}(L)) = 3\,\mathsf{sum}(L)$$

holds for any list S by structural induction.

$$\mathsf{twice\text{-}evens} : \mathsf{List} \to \mathsf{List}$$

$$\begin{aligned} \mathsf{twice\text{-}evens}(\mathsf{nil}) &:= \mathsf{nil} \\ \mathsf{twice\text{-}evens}(a :: L) &:= 2a :: \mathsf{twice\text{-}odds}(L) \end{aligned}$$

$$\mathsf{twice\text{-}odds} : \mathsf{List} \to \mathsf{List}$$

$$\begin{aligned} \mathsf{twice\text{-}odds}(\mathsf{nil}) &:= \mathsf{nil} \\ \mathsf{twice\text{-}odds}(a :: L) &:= a :: \mathsf{twice\text{-}evens}(L) \end{aligned}$$

$$\mathsf{sum} : \mathsf{List} \to \mathbb{Z}$$

$$\begin{aligned} \mathsf{sum}(\mathsf{nil}) &:= 0 \\ \mathsf{sum}(a :: L) &:= a + \mathsf{sum}(L) \end{aligned}$$