CSE 331
Software Design & Implementation

Spring 2025

Section 5 - Reasoning

Administrivia

- HW5 will be released later tonight and is due next Wednesday, 5/7, @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 <u>first</u> ≤ <u>last</u>
- 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 ≥ 1" and "y ≥ 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

Question …

Suppose we have the facts: $x = 3$, $y = 4$, $z > 5$ and we want to use proof by calculation to prove $x^2 + y^2 < z^2$

What are some fundamental problems with this example proof?

$$
\begin{aligned}
x^2 + y^2 &= 3^2 + y^2 && \text{since } x = 3 \\
&= 3^2 + 4^2 && \text{since } y = 4 \\
x^2 + y^2 + 9 &= 25 + 9 \\
&= 36 \\
&= 6^2 \\
&< z^2 && \text{since } z > 5
\end{aligned}
$$

Question …

Suppose we have the facts: $x = 3$, $y = 4$, $z > 5$ and we want to use proof by calculation to prove $x^2 + y^2 < z^2$

What are some fundamental problems with this example proof?

$$
\begin{aligned}
x^2 + y^2 &= 3^2 + y^2 && \text{since } x = 3 \\
&= 3^2 + 4^2 && \text{since } y = 4 \\
x^2 + y^2 + 9 &= 25 + 9 \\
&= 36 \\
&= 6^2 \\
&< z^2 && \text{since } z > 5
\end{aligned}
$$

- work done on both sides
- not strictly using the previous statement.

Question …

Corrected version:

$$
\begin{aligned}
x^2 + y^2 &= 3^2 + y^2 && \text{since } x = 3\\
&= 3^2 + 4^2 && \text{since } y = 4\\
&= 25\\
&= 5^2\\
&< z^2 && \text{since } z > 5
\end{aligned}
$$

- Here we strictly use the value derived right above, and don't apply logic to both sides.

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

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

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

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

Task 1: Twice things up

(b) Using the facts that $L = 1{::}2{::}\text{nil}$ and $R = 2{::}4{::}\text{nil}$, 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.

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

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

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 $\mathrm{n} : \mathbb{Z}$

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

  - 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:** a = True and a = False or n >= 0 and 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*,
$f(n) >= n$
holds in general

Task 3: Swapaholic

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

How many cases do we have?

What are the cases?

Task 3: Swapaholic

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

$$\begin{aligned}
&\text{swap}(a :: L) \\
&\qquad = \text{swap}(a :: \text{nil}) && \text{Def of } L \\
&\qquad = a :: \text{nil} && \text{Def of swap} \\
&\qquad \neq \text{nil}
\end{aligned}$$

$$= b :: a :: \text{swap}(R) \quad \text{Def of swap}$$

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

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(S) 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(x :: L) holds
5) **Conclusion** - “We have shown that P(S) holds for any list”

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 sum(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.

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

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

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

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

1) What's the claim?

Prove that sum(twice(S) = 2sum(S) holds for any list S by structural induction.

1) **Introduction** - define P(S) to be what we are trying to prove

2) What is the base case?

Prove that sum(twice(*S*) = 2sum(*S*) holds for any list *S* by structural induction.

2) Base Case - show P(nil) holds

2) Let's solve it!

Prove that sum(twice(S) = 2sum(S) holds for any list S by structural induction.

2) Base Case - show P(nil) holds

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

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

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

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

3) What's the Inductive Hypothesis

Prove that sum(twice(*S*) = 2sum(*S*) holds for any list *S* by structural induction.

3) Inductive Hypothesis - assume P(L) is true for an arbitrary list

4) What is our inductive step showing?

Prove that sum(twice(S) = 2sum(S) holds for any list S by structural induction.

4) Inductive Step - show P(x :: L) holds

4) Let's solve it!

Prove that sum(twice(S) = 2sum(S) holds for any list S by structural induction.

4) Inductive Step - show P(x :: L) holds

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

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

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

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

Conclusion

Prove that sum(twice(*S*) = 2sum(*S*) holds for any list *S* by structural induction.

4) Inductive Step - show P(x :: L) holds

Extra practice

We have an extra practice problem 5 on the section worksheet!

– link to extra examples:
https://courses.cs.washington.edu/courses/cse331/25sp/topics/#5-reasoning