
CSE 331

Software Design & Implementation

Autumn 2024
Section 5 - Reasoning

Administrivia

- HW 5 written released tonight, due Saturday 11/02 at 11pm
 - HW 5 coding released Thursday, due Monday 11/04 at 11pm
 - If using a late day for HW4 coding, get it in by tonight at 11pm
- EDiquette: please! categorize your posts with the correct homework + make it public/private appropriately

Proof By Calculation - Review

- Proving implications is the **core step** of reasoning
- Uses known facts and definitions (ex: $\text{len}(\text{nil}) = 0$)
 - Written in our math notation!
- Start from the left side of the inequality to be proved
- Chain of “=” shows $\text{first} = \text{last}$
- Chain of “=” and “ \leq ” shows first \leq 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

$$\begin{aligned}x * y &\geq x * 1 \text{ since } y \geq 1 \\ &\geq 1 + 1 \text{ since } x \geq 1 \\ &= 2 \\ &\geq 1\end{aligned}$$

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

Proof By Calculation - Citing Functions

$$\text{sum}(\text{nil}) := 0$$

$$\text{sum}(x :: L) := x + \text{sum}(L)$$

- Know “ $a \geq 0$ ”, “ $b \geq 0$ ”, and “ $L = a :: b :: \text{nil}$ ”
- Prove the “ $\text{sum}(L)$ ” is non-negative

$$\begin{aligned} \text{sum}(L) &= \text{sum}(a :: b :: \text{nil}) && \text{since } L = a :: b :: \text{nil} \\ &= a + \text{sum}(b :: \text{nil}) && \text{def of sum} \\ &= a + b + \text{sum}(\text{nil}) && \text{def of sum} \\ &= a + b && \text{def of sum} \\ &\geq 0 + b && \text{since } a \geq 0 \\ &\geq 0 && \text{since } b \geq 0 \end{aligned}$$

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

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

<code>twice : List</code>	<code>twice(L)</code>	
<code>twice(nil) :=</code>	<code>= twice(1::2::nil)</code>	Def of L
<code>twice(a :: L) :=</code>	<code>= 2::twice(2::nil)</code>	Def of <code>twice</code>
	<code>= 2::4::twice(nil)</code>	Def of <code>twice</code>
	<code>= 2::4::nil</code>	Def of <code>twice</code>

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.

```
twice : List → List
```

```
twice(nil) := nil
```

```
twice(a :: L) := 2a :: twice(L)
```

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

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

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

$\text{twice-evens}(\text{nil}) \quad := \quad \text{nil}$

$\text{twice-evens}(a :: L) \quad := \quad 2a :: \text{twice-odds}(L)$

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

$\text{twice-odds}(\text{nil}) \quad := \quad \text{nil}$

$\text{twice-odds}(a :: L) \quad := \quad a :: \text{twice-evens}(L)$

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 $f(n)$ where $n : \mathbb{Z}$

$$\begin{array}{ll} \text{func } f(n) := 2n + 1 & \text{if } n \geq 0 \\ f(n) := 0 & \text{if } n < 0 \end{array}$$

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

Proof By Cases – Review

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

- **Ex:** $\text{func } f(n) := 2n + 1 \quad \text{if } n \geq 0$
 $f(n) := 0 \quad \text{if } n < 0$
Prove that $f(n) \geq n$ for any $n : \mathbb{Z}$

Case $n \geq 0$:

$$\begin{array}{ll} f(n) = 2n + 1 & \text{def of } f \text{ (since } n \geq 0) \\ > n & \text{since } n \geq 0 \end{array}$$

Case $n < 0$:

$$\begin{array}{ll} f(n) = 0 & \text{def of } f \text{ (since } n < 0) \\ \geq n & \text{since } n < 0 \end{array}$$


Since these 2 cases are *exhaustive*,
 $f(n) \geq n$ 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 .

```
swap : List → List
swap(nil)      := nil
swap(a :: nil) := a :: nil
swap(a :: b :: L) := b :: a :: swap(L)
```

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 $x : Z, L : \text{List}$
 - 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 $P(\text{nil})$ holds
- 3) **Inductive Hypothesis** - assume $P(L)$ is true for an arbitrary list
- 4) **Inductive Step** - show $P(\text{cons}(a, L))$ 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.

$$\text{sum} : \text{List} \rightarrow \mathbb{Z}$$
$$\text{sum}(\text{nil}) \quad := \quad 0$$
$$\text{sum}(a :: L) \quad := \quad a + \text{sum}(L)$$
$$\text{twice} : \text{List} \rightarrow \text{List}$$
$$\text{twice}(\text{nil}) \quad := \quad \text{nil}$$
$$\text{twice}(a :: L) \quad := \quad 2a :: \text{twice}(L)$$

Task 5: Can You Sum A Few Bars?

Prove that

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

holds for any list S by structural induction.