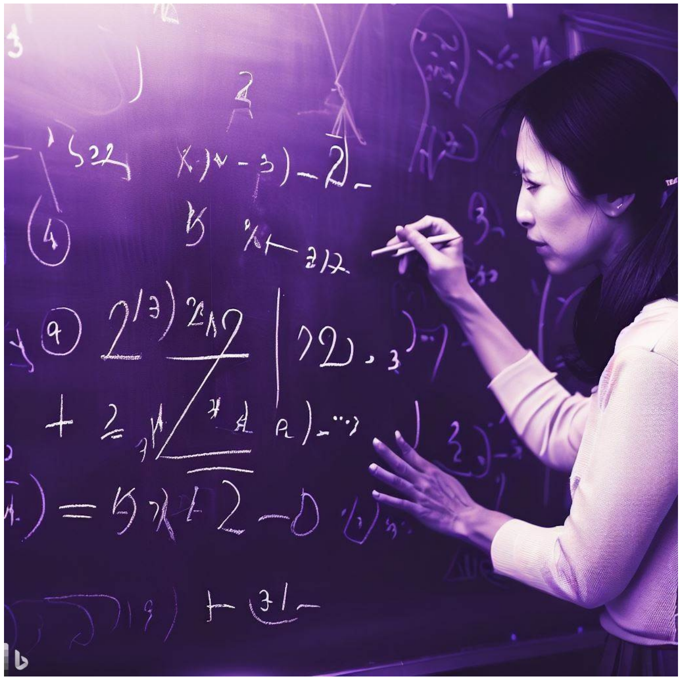


CSE 331

Basics of Reasoning

Kevin Zatloukal

Administrivia

- Section tomorrow on HW3
  - assignment released tomorrow night
- May be considerably more work than HW1–2
  - going from ~5% of grade up to ~8%

    (these percentages are still tentative)
- Start early!
  - consider one problem per day

HW2 Problem 5

- Given code uses tuple indexing

(a)

```
/** @param t consisting of a boolean and a non-negative integer */
const u = (t: [boolean, number]): number => {
  if (t[1] === 0) {
    return 1;
  } else if (t[1] === 1) {
    return 2;
  } else {
    return 3 + u(t[0], t[1]-1);
  }
};
```

- **Understand** why we don't allow this

Review: Understanding Inductive Data Types

type List := nil | cons(hd: $\mathbb{Z}$, tl: List)

- In Math, this is **data**

  cons(1, cons(2, nil))

- In TypeScript, we represent it by this **data**

  ```
  {kind: "cons", hd: 1, tl: {kind: "cons", hd: 2, tl: "nil"}}
  ```

  - but we can create it with this **code**

  ```
  cons(1, cons(2, nil))
  ```

Formalizing Specifications

Correctness Levels

Level	Description	Testing	Tools	Reasoning
-1	small # of inputs	exhaustive		
0	straight from spec	heuristics	type checking	code reviews
1	no mutation	"	libraries	calculation induction
2	local variable mutation	"	"	Floyd logic
3	array / object mutation	"	"	rep invariants

"straight from spec" requires us to have a formal spec!

Formalizing a Specification

- Sometimes the instructions are written in English
  - English is often imprecise or ambiguous
- First step is to “formalize” the specification:
  - translate it into math with a precise meaning
- How do we tell if the specification is wrong?
  - specifications can contain bugs
  - we can only **test** our definition on some examples

    (formal) reasoning can only be used *after* we have a formal spec
- Usually best to start by looking at some examples

Definition of Sum of Values in a List

- **Sum of a List: "add up all the values in the list"**
- **Look at some examples…**

L	sum(L)
nil	0
cons(3, nil)	3
cons(2, cons(3, nil))	2+3
cons(1, cons(2, cons(3, nil)))	1+2+3
…	…

Definition of Sum of Values in a List

- Look at some examples…

L	sum(L)
nil	0
cons(3, nil)	3
cons(2, cons(3, nil))	2+3
cons(1, cons(2, cons(3, nil)))	1+2+3
…	…

- Mathematical definition

func sum(nil) :=

sum(cons(x, S)) := for any $x \in \mathbb{Z}$ and any $S \in$ List

Sum of Values in a List

- **Mathematical definition of sum**

func sum(nil)	:=	0	
sum(cons(x, S))	:=	x + sum(S)	for any x ∈ ℤ and any S ∈ List

- **Translation to TypeScript**

```
const sum = (L: List): number => {
  if (L === nil) {
    return 0;
  } else {
    return L.hd + sum(L.tl);
  }
};
```

Level 0

Definition of Reversal of a List

- Reversal of a List: “same values but in reverse order”
- Look at some examples…

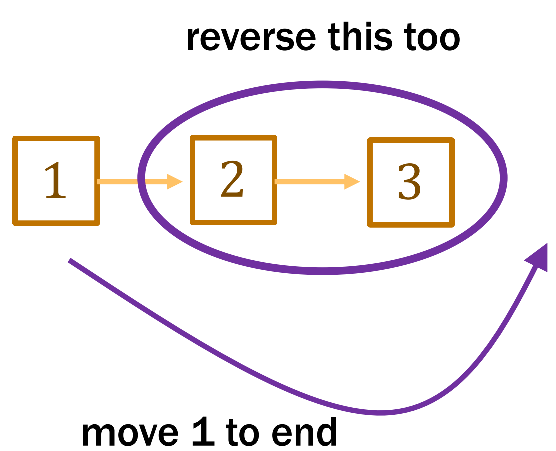
L	rev(L)
nil	nil
cons(3, nil)	cons(3, nil)
cons(2, cons(3, nil))	cons(3, cons(2, nil))
cons(1, cons(2, cons(3, nil)))	cons(3, cons(2, cons(1, nil)))
…	…

Definition of Reversal of a List

- **Look at some examples…**

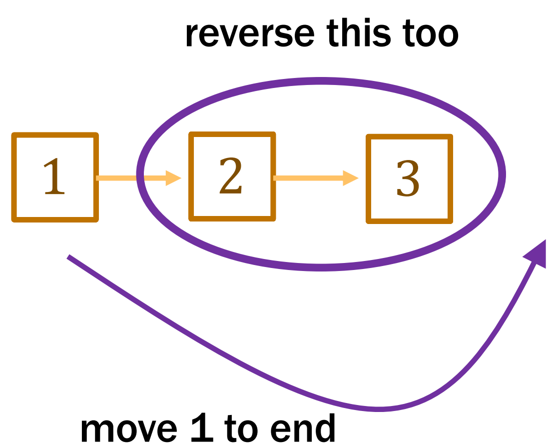
L	rev(L)
nil	nil
cons(3, nil)	cons(3, nil)
cons(2, cons(3, nil))	cons(3, cons(2, nil))
cons(1, cons(2, cons(3, nil)))	cons(3, cons(2, cons(1, nil)))

- **Draw a picture?**



Reversing A Lists

- Draw a picture?



- Mathematical definition of rev

func rev(nil) :=
rev(cons(x, S)) := for any $x \in \mathbb{Z}$ and any $S \in$ List

Reversing A Lists

- Mathematical definition of rev

$$
\begin{aligned}
&\textbf{func}\ \mathrm{rev}(\mathrm{nil}) &&:= \mathrm{nil} \\
&\phantom{\textbf{func}}\ \mathrm{rev}(\mathrm{cons}(x, S)) &&:= \mathrm{concat}(\mathrm{rev}(S), \mathrm{cons}(x, \mathrm{nil})) \quad \text{for any } x \in \mathbb{Z} \text{ and} \\
& && \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{any } S \in \mathrm{List}
\end{aligned}
$$

- Other definitions are possible, but this is **simplest**

- No help from reasoning tools until later
  - only have testing and thinking about what the English means

- Always make definitions as simple as possible

Reasoning

Correctness Levels

Level	Description	Testing	Tools	Reasoning	
-1	small # of inputs	exhaustive			
0	straight from spec	heuristics	type checking	code reviews	HW2
1	no mutation	“	libraries	calculation induction	HW3 HW4
2	local variable mutation	“	“	Floyd logic	HW6
3	array / object mutation	“	“	rep invariants	HW7

Facts

- Basic inputs to reasoning are "facts"
  - things we know to be true about the variables
  - typically, "=" or "≤"

```
// n must be a natural number
const f = (n: number): number => {
  const m = 2*n;
  return (m + 1) * (m - 1);
};
```

find facts by reading along path from top to return statement

- At the return statement, we know these facts:
  - $n \in \mathbb{N}$ (or $n \in \mathbb{Z}$ and $n \geq 0$)
  - $m = 2n$

Facts

- Basic inputs to reasoning are “facts”
  - things we know to be true about the variables
  - typically, “=” or “≤”

```
// n must be a natural number
const f = (n: number): number => {
  const m = 2*n;
  return (m + 1) * (m - 1);
};
```

- No need to include the fact that n is a number ($n \in \mathbb{R}$)
  - that is true, but the type checker takes care of that
  - no need to repeat reasoning done by the type checker

Implications

- We can use the facts we know to prove more facts
  - if we can prove R using facts P and Q,
    we say that R “follows from” or “is implied by” P and Q
  - proving this fact is proving an “implication”

- Proving implications is necessary for checking correctness

Checking Correctness

- **Specifications include two kinds of facts**
  - promised facts about the inputs (P and Q)
  - required facts about the outputs (R)
- **Checking correctness is just proving implications**
  - proving facts about the **return** values
  - we need to use reasoning to do that

Implications

- We can use the facts we know to prove more facts
  - if we can prove R using facts P and Q,
    we say that R "follows from" or "is implied by" P and Q

- Proving implications is the **core skill** of reasoning
  - other techniques output implications for us to prove

- The techniques we will learn are
  - proof by calculation
  - proof by cases
  - structural induction

gives us two implications,
each usually proven by calculation

Proof by Calculation

- Proves an implication
  - fact to be shown is an equation or inequality

- Uses known facts and definitions
  - latter includes, e.g., the fact that $\text{len}(\text{nil}) = 0$

Example Proof by Calculation

- Given $x = y$ and $z \leq 10$, prove that $x + z \leq y + 10$
  - show the third fact follows from the first two

- Start from the left side of the inequality to be proved

$$x + z$$

Example Proof by Calculation

- Given $x = y$ and $z \leq 10$, prove that $x + z \leq y + 10$
  - show the third fact follows from the first two

- Start from the left side of the inequality to be proved

$x + z$	$= y + z$	since $x = y$
	$\leq y + 10$	since $z \leq 10$

  - "calculation block", includes explanations in right column

    proof by calculation means using a calculation block

Calculation Blocks

- Chain of “=” shows first = last

$$\begin{aligned} a &= b \\ &= c \\ &= d \end{aligned}$$

  - proves that $a = d$
  - all 4 of these are the same number

Calculation Blocks

- **Chain of "=" and "≤" shows <u>first</u> ≤ <u>last</u>**

$$
\begin{aligned}
x + z &= y + z && \textbf{since } x = y \\
&\le y + 10 && \textbf{since } z \le 10 \\
&= y + 3 + 7 && \\
&\le w + 7 && \textbf{since } y + 3 \le w
\end{aligned}
$$

  - **each number is equal or strictly larger that previous**

    last number is strictly larger than the first number

  - **analogous for "≥"**

Using Calculation to Prove Correctness

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

- Known facts "$x \geq 1$" and "$y \geq 1$"

- Correct if the return value is a positive integer

$$x + y$$

Using Calculation to Prove Correctness

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

- Known facts "$x \geq 1$" and "$y \geq 1$"

- Correct if the return value is a positive integer

$x + y$	$\geq x + 1$	since $y \geq 1$
	$= 1 + 1$	since $x \geq 1$
	$= 2$	
	≥ 1	

  – calculation shows that $x + y \geq 1$

Using Calculation to Prove Correctness

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

- Known facts "$x \in \mathbb{Z}$" and "$y \in \mathbb{Z}$"

- Correct if the return value is a positive <u>integer</u>
  - we know that "x + y" is an integer
  - should be second nature from Java programming
  - unless there is *division* involved, we will skip this

Using Calculation to Prove Correctness

```
// Inputs x and y are integers with x > 8 and y > -9
// Returns a positive integer.
const f = (x: number, y, number): number => {
  return x + y;
};
```

- Known facts "$x \geq 9$" and "$y \geq -8$"

- Correct if the return value is a positive integer

$$x + y$$

Using Calculation to Prove Correctness

```
// Inputs x and y are integers with x > 8 and y > -9
// Returns a positive integer.
const f = (x: number, y, number): number => {
  return x + y;
};
```

- Known facts "$x \geq 9$" and "$y \geq -8$"

- Correct if the return value is a positive integer

$x + y$	$\geq x + -8$	since $y \geq -8$
	$\geq 9 - 8$	since $x \geq 9$
	$= 1$	

Using Calculation to Prove Correctness

```
// Inputs x and y are integers with x > 3 and y > 4
// Returns an integer that is 10 or larger.
const f = (x: number, y, number): number => {
  return x + y;
};
```

- Known facts "$x \geq 4$" and "$y \geq 5$"

- Correct if the return value is 10 or larger

$$x + y$$

Using Calculation to Prove Correctness

```
// Inputs x and y are integers with x > 3 and y > 4
// Returns an integer that is 10 or larger.
const f = (x: number, y, number): number => {
  return x + y;
};
```

- Known facts “$x \geq 4$” and “$y \geq 5$”

- Correct if the return value is 10 or larger

$x + y$	$\geq x + 5$	since $y \geq 5$
	$\geq 4 + 5$	since $x \geq 4$
	$= 9$	

proof doesn't work because the **code is wrong**!

Using Calculation to Prove Correctness

```
// Inputs x and y are integers with x > 8 and y > -9
// Returns a positive integer.
const f = (x: number, y, number): number => {
  return x + y;
};
```

- Known facts "$x > 8$" and "$y > -9$"

- Correct if the return value is a positive integer

$$\begin{aligned} x + y &> x + -9 && \text{since } y > -9 \\ &> 8 - 9 && \text{since } x > 8 \\ &= -1 \end{aligned}$$

proof doesn't work because the **proof is wrong**

warning: avoid using ">" (or "<") *multiple* times in a calculation block

Using Definitions in Calculations

- Most useful with function calls
  - cite the definition of the function to get the return value

- For example

$$
\begin{array}{llll}
\textbf{func}\ \mathrm{sum(nil)} & := 0 & & \\
\phantom{\textbf{func}\ }\mathrm{sum(cons(x, L))} & := \mathrm{x} + \mathrm{sum(L)} & & \text{for any } \mathrm{x} \in \mathbb{Z} \\
& & & \text{and any } \mathrm{L} \in \mathrm{List}
\end{array}
$$

- Can cite facts such as
  - $\mathrm{sum(nil)} = 0$
  - $\mathrm{sum(cons(a, cons(b, nil)))} = \mathrm{a} + \mathrm{sum(cons(b, nil))}$

second case of definition with $\mathrm{x} = \mathrm{a}$ and $\mathrm{L} = \mathrm{cons(b, nil)}$

Using Definitions in Calculations

func sum(nil)	$:=\ 0$		
$\mathrm{sum}(\mathrm{cons}(x, L))$	$:=\ x + \mathrm{sum}(L)$	for any $x \in \mathbb{Z}$ and any $L \in \mathrm{List}$	

- **Know** "$a \geq 0$", "$b \geq 0$", **and** "$L = \mathrm{cons}(a, \mathrm{cons}(b, \mathrm{nil}))$"
- **Prove the** "$\mathrm{sum}(L)$" **is non-negative**

$\mathrm{sum}(L)$

Using Definitions in Calculations

func sum(nil)	$:=\ 0$		
$\mathrm{sum}(\mathrm{cons}(x, L))$	$:=\ x + \mathrm{sum}(L)$	for any $x \in \mathbb{Z}$ and any $L \in \mathrm{List}$	

- **Know** "$a \geq 0$", "$b \geq 0$", **and** "$L = \mathrm{cons}(a, \mathrm{cons}(b, \mathrm{nil}))$"
- **Prove the** "$\mathrm{sum}(L)$" **is non-negative**

$\mathrm{sum}(L)$	$= \mathrm{sum}(\mathrm{cons}(a, \mathrm{cons}(b, \mathrm{nil})))$	**since** $L = \mathrm{cons}(a, \mathrm{cons}(b, \mathrm{nil}))$
	$= a + \mathrm{sum}(\mathrm{cons}(b, \mathrm{nil}))$	**def of** sum
	$= a + b + \mathrm{sum}(\mathrm{nil})$	**def of** sum
	$= a + b$	**def of** sum
	$\geq 0 + b$	**since** $a \geq 0$
	≥ 0	**since** $b \geq 0$