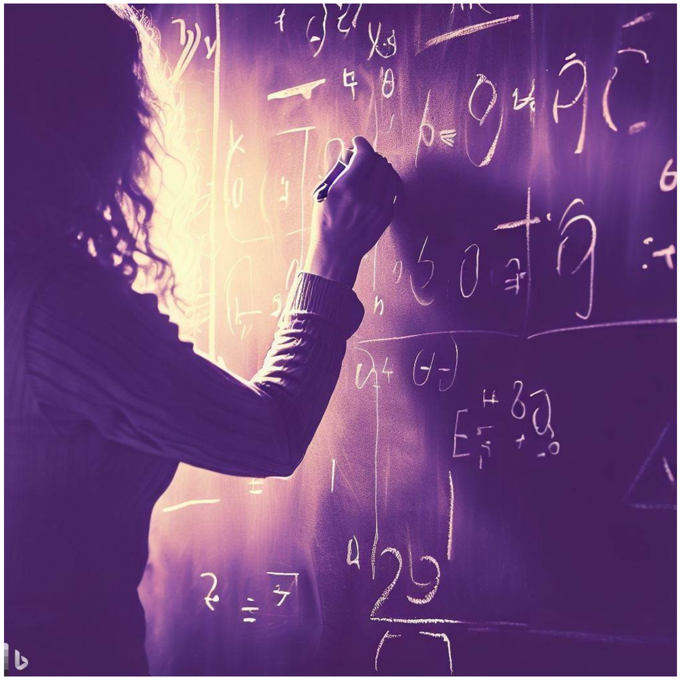


CSE 331

Structural Induction & Cases

Kevin Zatloukal

Direct Proof

- Our proofs so far have used fixed-length lists
  - e.g., $\text{len}(\text{twice}(\text{cons}(a, \text{cons}(b, \text{nil})))) = \text{len}(\text{cons}(a, \text{cons}(b, \text{nil})))$
  - problems in HW2 restrict to this case

- Would like to prove correctness on <u>any</u> list L

- Need more tools for this...
  - structural recursion *calculates* on inductive types
  - structural induction *proves* facts about inductive types
  - both tools are specific to inductive types

Structural Induction

Structural Induction

Let $\mathrm{P(S)}$ **be the claim** "$\mathrm{len(twice(S)) = len(S)}$"

To prove $\mathrm{P(S)}$ **holds for any list** S**, prove two implications**

Base Case: prove $\mathrm{P(nil)}$

- **use any known facts and definitions**

Inductive Step: prove $\mathrm{P(cons(x, L))}$ **for any** $\mathrm{x} : \mathbb{Z}$**,** $\mathrm{L : List}$

- **direct proof**
- **use any known facts and definitions plus <u>one more fact…</u>**

Structural Induction

To prove $\mathrm{P}(S)$ holds for any list S, prove two implications

Base Case: prove $\mathrm{P}(\mathrm{nil})$

- use any known facts and definitions

Inductive Hypothesis: assume P(L) is true

- use this in the inductive step, but not anywhere else

Inductive Step: prove $\mathrm{P}(\mathrm{cons}(\mathrm{x}, \mathrm{L}))$ for any $\mathrm{x} : \mathbb{Z}$, $\mathrm{L} : \mathrm{List}$

- direct proof
- use known facts and definitions and Inductive Hypothesis

Why This Works

With Structural Induction, we prove two facts

$\mathrm{P}(\mathrm{nil})$ $\quad$ $\mathrm{len}(\mathrm{twice}(\mathrm{nil})) = \mathrm{len}(\mathrm{nil})$

$\mathrm{P}(\mathrm{cons}(\mathrm{x}, \mathrm{L}))$ $\quad$ $\mathrm{len}(\mathrm{twice}(\mathrm{cons}(\mathrm{x}, \mathrm{L}))) = \mathrm{len}(\mathrm{cons}(\mathrm{x}, \mathrm{L}))$

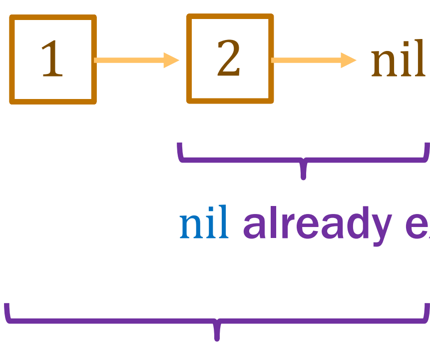
(**second assuming** $\mathrm{len}(\mathrm{twice}(\mathrm{L})) = \mathrm{len}(\mathrm{L})$)

Why is this enough to prove $\mathrm{P}(S)$ for any S : List

Why This Works

Build up an object using constructors:

nil	**first constructor**
cons(2, nil)	**second constructor**
cons(1, cons(2, nil))	**second constructor**



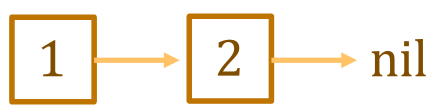
nil **already exists when building** cons(2, nil)

cons(2, nil) **already exists when building** cons(1, cons(2, nil))

Why This Works

Build up an object using constructors:

nil	**first constructor**
cons(2, nil)	**second constructor**
cons(1, cons(2, nil))	**second constructor**



nil **already exists when building** cons(2, nil)

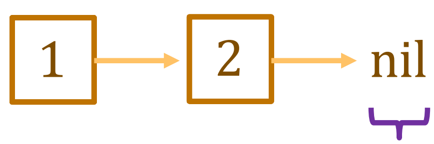
cons(2, nil) **already exists when building** cons(1, cons(2, nil))

Why This Works

Build up a proof the same way we built up the object

$P(\text{nil})$	$\text{len}(\text{twice}(\text{nil})) = \text{len}(\text{nil})$
$P(\text{cons}(x, L))$	$\text{len}(\text{twice}(\text{cons}(x, L))) = \text{len}(\text{cons}(x, L))$

(**second assuming** $\text{len}(\text{twice}(L)) = \text{len}(L)$)



$P(\text{nil})$

$P(\text{nil})$ **already proven when proving** $P(\text{cons}(2, \text{nil}))$

$P(\text{cons}(2, \text{nil}))$ **already proven when proving** $P(\text{cons}(1, \text{cons}(2, \text{nil})))$

Structural Induction in General

- General case: assume P holds for constructor *arguments*

type $\mathrm{T} := \mathrm{A} \mid \mathrm{B}(\mathrm{x} : \mathbb{Z}) \mid \mathrm{C}(\mathrm{y} : \mathbb{Z}, \mathrm{t} : \mathrm{T}) \mid \mathrm{D}(\mathrm{z} : \mathbb{Z}, \mathrm{u} : \mathrm{T}, \mathrm{v} : \mathrm{T})$

- To prove $\mathrm{P}(\mathrm{t})$ for any t, we need to prove:
  - $\mathrm{P}(\mathrm{A})$
  - $\mathrm{P}(\mathrm{B}(\mathrm{x}))$ for any $\mathrm{x} : \mathbb{Z}$
  - $\mathrm{P}(\mathrm{C}(\mathrm{y}, \mathrm{t}))$ for any $\mathrm{y} : \mathbb{Z}$ and $\mathrm{t} : \mathrm{T}$ assuming $\mathrm{P}(\mathrm{t})$ is true
  - $\mathrm{P}(\mathrm{D}(\mathrm{z}, \mathrm{u}, \mathrm{v}))$ for any $\mathrm{z} : \mathbb{Z}$ and $\mathrm{u}, \mathrm{v} : \mathrm{T}$ assuming $\mathrm{P}(\mathrm{u})$ and $\mathrm{P}(\mathrm{v})$

- Each inductive type has its own form of induction
  - special way to prove facts about all values of that type

Structural Induction in General

- General case: assume P holds for constructor *arguments*

type $\mathrm{T} := \mathrm{A} \mid \mathrm{B}(\mathrm{x} : \mathbb{Z}) \mid \mathrm{C}(\mathrm{y} : \mathbb{Z}, \mathrm{t} : \mathrm{T}) \mid \mathrm{D}(\mathrm{z} : \mathbb{Z}, \mathrm{u} : \mathrm{T}, \mathrm{v} : \mathrm{T})$

- To prove $\mathrm{P}(\mathrm{t})$ for any t, we need to prove:
  - $\mathrm{P}(\mathrm{A})$
  - $\mathrm{P}(\mathrm{B}(\mathrm{x}))$ for any $\mathrm{x} : \mathbb{Z}$
  - $\mathrm{P}(\mathrm{C}(\mathrm{y}, \mathrm{t}))$ for any $\mathrm{y} : \mathbb{Z}$ and $\mathrm{t} : \mathrm{T}$ — assuming $\mathrm{P}(\mathrm{t})$ is true
  - $\mathrm{P}(\mathrm{D}(\mathrm{z}, \mathrm{u}, \mathrm{v}))$ for any $\mathrm{z} : \mathbb{Z}$ and $\mathrm{u}, \mathrm{v} : \mathrm{T}$ — assuming $\mathrm{P}(\mathrm{u})$ and $\mathrm{P}(\mathrm{v})$

- These four facts are enough to prove $\mathrm{P}(\mathrm{t})$ for any t
  - for each constructor, have proof that it produces an object satisfying P

Example: Repeating List Elements

- **Consider the following function:**

func echo(nil) := nil
echo(cons(x, L)) := cons(x, cons(x, echo(L))) for any x : $\mathbb{Z}$, L : List

- **Produces a list where every element is repeated twice**

echo(cons(1, cons(2, nil)))
= cons(1, cons(1, echo(cons(2, nil)))) **def of** echo
= cons(1, cons(1, cons(2, cons(2, echo(nil))))) **def of** echo
= cons(1, cons(1, cons(2, cons(2, nil)))) **def of** echo

Example: Repeating List Elements

$$\textbf{func}\ \text{echo}(\text{nil}) := \text{nil}$$
$$\text{echo}(\text{cons}(x, L)) := \text{cons}(x, \text{cons}(x, \text{echo}(L))) \quad \text{for any } x : \mathbb{Z}, L : \text{List}$$

- **Suppose we have the following code:**

```
const m: number = len(S);      // S is some List
const R: List = echo(S);
...
return 2*m;  // = len(echo(S))
```

Level 1

- **spec says to return** $\text{len}(\text{echo}(S))$ **but code returns** $2\ \text{len}(S)$

- **Need to prove that** $\text{len}(\text{echo}(S)) = 2\ \text{len}(S)$

Example: Repeating List Elements

$$
\begin{aligned}
&\textbf{func}\ \text{echo}(\text{nil}) && := \text{nil} \\
&\quad \text{echo}(\text{cons}(x, L)) && := \text{cons}(x, \text{cons}(x, \text{echo}(L))) \quad \text{for any } x : \mathbb{Z},\ L : \text{List}
\end{aligned}
$$

- **Prove that** $\text{len}(\text{echo}(S)) = 2\ \text{len}(S)$ **for any** $S : \text{List}$

Base Case (nil):

Need to prove that $\text{len}(\text{echo}(\text{nil})) = 2\ \text{len}(\text{nil})$

$\text{len}(\text{echo}(\text{nil})) \quad =$

Example: Repeating List Elements

func echo(nil) := nil
echo(cons(x, L)) := cons(x, cons(x, echo(L))) for any x : $\mathbb{Z}$, L : List

- **Prove that** $\text{len}(\text{echo}(S)) = 2\ \text{len}(S)$ **for any** S : List

Base Case (nil):

len(echo(nil))	$= \text{len}(\text{nil})$	**def of** echo
	$= 0$	**def of** len
	$= 2 \cdot 0$	
	$= 2\ \text{len}(\text{nil})$	**def of** len

Example: Repeating List Elements

func echo(nil) := nil
echo(cons(x, L)) := cons(x, cons(x, echo(L))) for any $x : \mathbb{Z}$, L : List

- **Prove that** $\text{len}(\text{echo}(S)) = 2\ \text{len}(S)$ **for any** S : List

Inductive Step (cons(x, L)):

Need to prove that $\text{len}(\text{echo}(\text{cons}(x, L))) = 2\ \text{len}(\text{cons}(x, L))$

Get to assume claim holds for L, **i.e., that** $\text{len}(\text{echo}(L)) = 2\ \text{len}(L)$

Example: Repeating List Elements

func echo(nil) := nil
echo(cons(x, L)) := cons(x, cons(x, echo(L))) for any x : $\mathbb{Z}$, L : List

- **Prove that** $\text{len}(\text{echo}(S)) = 2\ \text{len}(S)$ **for any** S : List

Inductive Hypothesis: assume that $\text{len}(\text{echo}(L)) = 2\ \text{len}(L)$

Inductive Step (cons(x, L)):

len(echo(cons(x, L)))

= 2 len(cons(x, L))

Example: Repeating List Elements

$$\begin{aligned}
&\textbf{func}\ \mathrm{echo}(\mathrm{nil}) && := \mathrm{nil} \\
&\quad \mathrm{echo}(\mathrm{cons}(x, L)) && := \mathrm{cons}(x, \mathrm{cons}(x, \mathrm{echo}(L))) \quad \text{for any } x : \mathbb{Z}, L : \mathrm{List}
\end{aligned}$$

- **Prove that** $\mathrm{len}(\mathrm{echo}(S)) = 2\ \mathrm{len}(S)$ **for any** $S : \mathrm{List}$

Inductive Hypothesis: assume that $\mathrm{len}(\mathrm{echo}(L)) = 2\ \mathrm{len}(L)$

Inductive Step $(\mathrm{cons}(x, L))$**:**

$$\begin{aligned}
\mathrm{len}(\mathrm{echo}(\mathrm{cons}(x, L))) &= \mathrm{len}(\mathrm{cons}(x, \mathrm{cons}(x, \mathrm{echo}(L)))) && \textbf{def of } \mathrm{echo} \\
&= 1 + \mathrm{len}(\mathrm{cons}(x, \mathrm{echo}(L))) && \textbf{def of } \mathrm{len} \\
&= 2 + \mathrm{len}(\mathrm{echo}(L)) && \textbf{def of } \mathrm{len} \\
&= 2 + 2\ \mathrm{len}(L) && \textbf{Ind. Hyp.} \\
&= 2(1 + \mathrm{len}(L)) && \\
&= 2\ \mathrm{len}(\mathrm{cons}(x, L)) && \textbf{def of } \mathrm{len}
\end{aligned}$$

Example 2: Repeating List Elements

$$\textbf{func}\ \mathrm{echo}(\mathrm{nil}) := \mathrm{nil}$$
$$\mathrm{echo}(\mathrm{cons}(x, L)) := \mathrm{cons}(x, \mathrm{cons}(x, \mathrm{echo}(L))) \quad \text{for any } x : \mathbb{Z}, L : \mathrm{List}$$

- **Suppose we have the following code:**

```
const y: number = sum(S);      // S is some List
const R: List = echo(S);
...
return 2*y;  // = sum(echo(S))
```

Level 1

  - **spec says to return** $\mathrm{sum}(\mathrm{echo}(S))$ **but code returns** $2\ \mathrm{sum}(S)$

- **Need to prove that** $\mathrm{sum}(\mathrm{echo}(S)) = 2\ \mathrm{sum}(S)$

Example 2: Repeating List Elements

$$\begin{aligned} \textbf{func}\ \text{echo}(\text{nil}) &:= \text{nil} \\ \text{echo}(\text{cons}(x, L)) &:= \text{cons}(x, \text{cons}(x, \text{echo}(L))) \quad \text{for any } x : \mathbb{Z}, L : \text{List} \end{aligned}$$

- **Prove that** $\text{sum}(\text{echo}(S)) = 2\ \text{sum}(S)$ **for any** $S : \text{List}$

Base Case (nil):

$$\text{sum}(\text{echo}(\text{nil})) =$$

$$= 2\ \text{sum}(\text{nil})$$

Example 2: Repeating List Elements

func $\text{echo}(\text{nil}) := \text{nil}$
$\text{echo}(\text{cons}(x, L)) := \text{cons}(x, \text{cons}(x, \text{echo}(L)))$ for any $x : \mathbb{Z}$, $L : \text{List}$

- **Prove that** $\text{sum}(\text{echo}(S)) = 2\ \text{sum}(S)$ **for any** $S : \text{List}$

Base Case (nil):

$\text{sum}(\text{echo}(\text{nil}))$	$= \text{sum}(\text{nil})$	**def of** echo
	$= 0$	**def of** sum
	$= 2 \cdot 0$	
	$= 2\ \text{sum}(\text{nil})$	**def of** sum

Inductive Step (cons(x, L)):

Need to prove that $\text{sum}(\text{echo}(\text{cons}(x, L))) = 2\ \text{sum}(\text{cons}(x, L))$
Get to assume claim holds for L**, i.e., that** $\text{sum}(\text{echo}(L)) = 2\ \text{sum}(L)$

Example 2: Repeating List Elements

$$\textbf{func}\ \mathrm{echo}(\mathrm{nil}) := \mathrm{nil}$$
$$\mathrm{echo}(\mathrm{cons}(x, L)) := \mathrm{cons}(x, \mathrm{cons}(x, \mathrm{echo}(L))) \quad \text{for any } x : \mathbb{Z},\ L : \mathrm{List}$$

- **Prove that** $\mathrm{sum}(\mathrm{echo}(S)) = 2\ \mathrm{sum}(S)$ **for any** $S : \mathrm{List}$

Inductive Hypothesis: assume that $\mathrm{sum}(\mathrm{echo}(L)) = 2\ \mathrm{sum}(L)$

Inductive Step $(\mathrm{cons}(x, L))$:

$\mathrm{sum}(\mathrm{echo}(\mathrm{cons}(x, L))) =$

$= 2\ \mathrm{sum}(\mathrm{cons}(x, L))$

Example 2: Repeating List Elements

$$\textbf{func } \text{echo}(\text{nil}) \quad := \text{nil}$$
$$\text{echo}(\text{cons}(x, L)) \quad := \text{cons}(x, \text{cons}(x, \text{echo}(L))) \quad \text{for any } x : \mathbb{Z}, L : \text{List}$$

- **Prove that** $\text{sum}(\text{echo}(S)) = 2\ \text{sum}(S)$ **for any** $S : \text{List}$

Inductive Hypothesis: **assume that** $\text{sum}(\text{echo}(L)) = 2\ \text{sum}(L)$

Inductive Step $(\text{cons}(x, L))$:

$$\begin{aligned}
\text{sum}(\text{echo}(\text{cons}(x, L))) &= \text{sum}(\text{cons}(x, \text{cons}(x, \text{echo}(L)))) && \textbf{def of } \text{echo} \\
&= x + \text{sum}(\text{cons}(x, \text{echo}(L))) && \textbf{def of } \text{sum} \\
&= 2x + \text{sum}(\text{echo}(L)) && \textbf{def of } \text{sum} \\
&= 2x + 2\ \text{sum}(L) && \textbf{Ind. Hyp.} \\
&= 2(x + \text{sum}(L)) && \\
&= 2\ \text{sum}(\text{cons}(x, L)) && \textbf{def of } \text{sum}
\end{aligned}$$

Proof By Cases

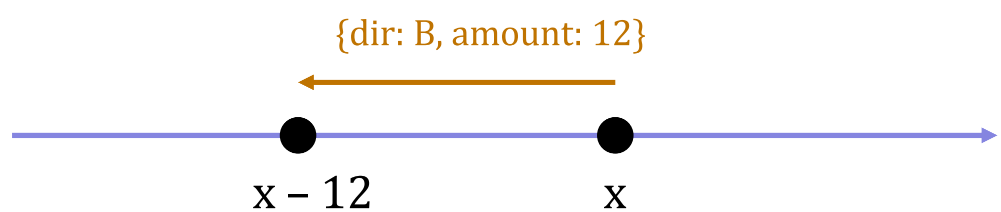
Recall: Pattern Matching

- Define a function by an exhaustive set of patterns

```
type Move   :=  {dir : F | B, amount : ℕ}

func change({dir: F, amount : n})        :=  n        for any n : ℕ
     change({dir: B, amount : n})        :=  -n       for any n : ℕ
```

  - Move **describes movement on the number line**
  - change(m : Move) **says how the position changes**



  - **one of these two rules always applies**

    every Move either has direction as F or B

Proof By Cases

- New code structure means new proof structure

- Can split a proof into cases
  - e.g., $\mathrm{d} = \mathrm{F}$ and $\mathrm{d} = \mathrm{B}$
  - e.g., $\mathrm{n} \geq 0$ and $\mathrm{n} < 0$
  - need to be sure the cases are exhaustive

- Structural induction and Proof By Cases are related
  - one case per constructor
  - structural induction adds the inductive hypothesis part

    we don't get that in proof by cases

More Proof By Cases

func change({dir: F, amount: n}) := n for any $n : \mathbb{N}$
change({dir: B, amount: n}) := -n for any $n : \mathbb{N}$

- **Prove that** $|\text{change}(m)| = n$ **for any** $m = \{\text{dir: d, amount: n}\}$

Will use d : F | B to denote a direction and $n : \mathbb{N}$ a natural number.

Case $d = F$:

$|\text{change}(\{\text{dir: d, amount: n}\})|$
$= |\text{change}(\{\text{dir: F, amount: n}\}|$ **since** $d = F$
$= |n|$ **def of** change
$= n$ **since** $n \geq 0$

Note: we need to know if $d = F$ **or** $d = B$ **to apply the definition!**

More Proof By Cases

func change({dir: F, amount: n}) := n for any $n : \mathbb{N}$
change({dir: B, amount: n}) := -n for any $n : \mathbb{N}$

- **Prove that** $|\text{change}(m)| = n$ **for any** $m = \{\text{dir: d, amount: n}\}$

Will use d : F | B to denote a direction and $n : \mathbb{N}$ a natural number.

Case $d = F$: $|\text{change}(\{\text{dir: d, amount: n}\})| = \ldots = n$

Case $d = B$:

$|\text{change}(\{\text{dir: d, amount: n}\})|$
$= |\text{change}(\{\text{dir: B, amount: n}\}|$ **since** $d = B$
$= |-n|$ **def of** change
$= n$ **since** $n \geq 0$

Since these two cases are exhaustive, the claim holds in general.

Defining Functions by Cases

- **Pattern matching is one way to define with cases**
  - **every list is either** nil **or** $\mathrm{cons}(\mathrm{x}, \mathrm{L})$ **for some** x **and** L**, so we can define** f **in two cases:** $\mathrm{f}(\mathrm{nil})$ **and** $\mathrm{f}(\mathrm{cons}(\mathrm{x}, \mathrm{L}))$

- **Sometimes we want to define with other cases**
  - **e.g., define** $\mathrm{f}(\mathrm{n})$ **where** $\mathrm{n} : \mathbb{Z}$

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

  - **to use the definition on** $\mathrm{f}(\mathrm{m})$**, need to know if** $\mathrm{m} < 0$ **or not**

Proof By Cases

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 $n \geq 0$:

$f(n) =$

$\geq n$

Proof By Cases

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** $n \geq 0$**)**
	$> 2n$	**since** $1 > 0$
	$= n + n$	
	$\geq n$	**since** $n \geq 0$

Proof By Cases

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 $n \geq 0$:

$f(n) = \ldots \geq n$

Case $n < 0$:

$f(n) = 0$ **def of** f **(since** $n < 0$**)**
$> n$ **since** $n < 0$

Since these two cases are exhaustive, $f(n) \geq n$ **holds in general.**