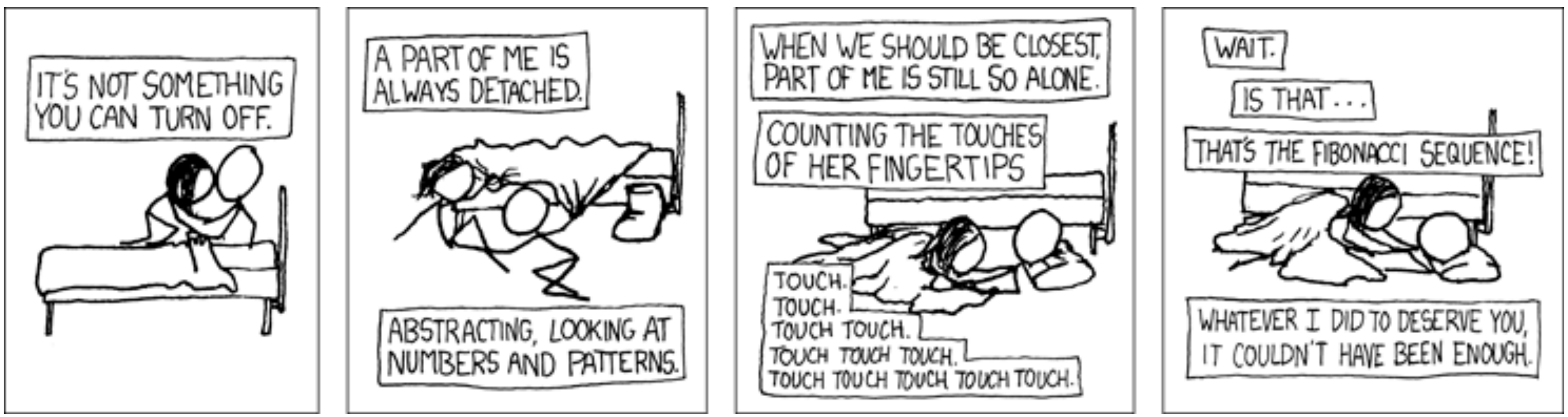
CSE 311

Foundations of Computing I

CSE 311: Foundations of Computing

Lecture 17: Structural Induction



Strings

- An *alphabet* **Σ** is any finite set of characters

- The set **Σ*** is the set of *strings* over the alphabet **Σ**.

Adam

$$\Sigma^* = \varepsilon \mid \Sigma^* \sigma$$

A STRING is EMPTY or "STRING CHAR".

- **The set of strings is made up of:**
  - $\varepsilon \in \Sigma^*$ (ε is the empty string)
  - If $W \in \Sigma^*$, $\sigma \in \Sigma$, then $W\sigma \in \Sigma^*$

Palindromes

Palindromes are strings that are the same backwards and forwards (e.g. “abba”, “tht”, “neveroddoreven”).

$$\textbf{Pal} = \varepsilon \mid \sigma \mid \sigma \, \text{Pal} \, \sigma$$

A PAL is EMPTY or CHAR or “CHAR PAL CHAR”.

$abba = a(bb)a = a(b(\varepsilon)b)a$

Recursively Defined Programs (on Binary Strings)

$$\mathbf{B} = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1$$

A BSTR is EMPTY, 0, 1, or “BSTR0 BSTR1”.

Let’s write a “reverse” function for binary strings.

$1 + (0 + (1 + 0))$

$$\text{rev} : B \to B$$

rev is a function that takes in a binary string and returns a binary string

$rev(\varepsilon) = \varepsilon$

$rev(0) = 0$

$rev(1) = 1$

$rev(a + b) = rev(b) + rev(a)$

Recursively Defined Programs (on Binary Strings)

$$\mathbf{B} = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1$$

A BSTR is EMPTY, 0, 1, or "BSTR0 BSTR1".

Let's write a "reverse" function for binary strings.

$$\begin{aligned}
&\text{rev} : B \to B \\
&\text{rev}(\varepsilon) && = \varepsilon \\
&\text{rev}(0) && = 0 \\
&\text{rev}(1) && = 1 \\
&\text{rev}(a + b) && = \text{rev}(b) + \text{rev}(a)
\end{aligned}$$

Recursively Defined Programs (on Binary Strings)

$\mathbf{B} = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1$

$\text{rev} : B \to B$

$\text{rev}(\varepsilon) = \varepsilon$

$\text{rev}(0) = 0$

$\text{rev}(1) = 1$

$\text{rev}(a + b) = \text{rev}(b) + \text{rev}(a)$

Claim: For all binary strings X, $\text{rev}(\text{rev}(X)) = X$

Case ε: $\text{rev}(\text{rev}(\varepsilon)) = \text{rev}(\varepsilon) = \varepsilon$ **Def of rev**

Case 0: $\text{rev}(\text{rev}(0)) = \text{rev}(0) = 0$ **Def of rev**

Case 1: $\text{rev}(\text{rev}(1)) = \text{rev}(1) = 1$ **Def of rev**

Recursively Defined Programs (on Binary Strings)

$$\mathbf{B} = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1$$

rev : B → B

$\text{rev}(\varepsilon) = \varepsilon$

$\text{rev}(0) = 0$

$\text{rev}(1) = 1$

$\text{rev}(a + b) = \text{rev}(b) + \text{rev}(a)$

Claim: For all binary strings **X**, rev(rev(**X**)) = **X**

Suppose rev(rev(a)) = a and rev(rev(b)) = b for some strings a, b.

Case $a + b$:

rev(rev(a + b)) = rev(rev(b) + rev(a)) = rev(rev(a)) + rev(rev(b)) = a + b by IH

Recursively Defined Programs (on Binary Strings)

$$\mathbf{B} = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1$$

$\text{rev}(\varepsilon) = \varepsilon$
$\text{rev}(0) = 0$
$\text{rev}(1) = 1$
$\text{rev}(a + b) = \text{rev}(b) + \text{rev}(a)$

Claim: For all binary strings **X**, $\text{rev}(\text{rev}(X)) = X$

Suppose $\text{rev}(\text{rev}(a)) = a$ and $\text{rev}(\text{rev}(b)) = b$ for some strings a, b.

Case $a + b$:

$\text{rev}(\text{rev}(a + b)) = \text{rev}(\text{rev}(b) + \text{rev}(a))$	**Def of rev**
$= \text{rev}(\text{rev}(a)) + \text{rev}(\text{rev}(b))$	**Def of rev**
$= a + b$	**By IH!**

Recursively Defined Programs (on Binary Strings)

$$\mathbf{B} = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1$$

rev : B → B
rev(ε) = ε
rev(0) = 0
rev(1) = 1
rev(a + b) = rev(b) + rev(a)

Claim: For all binary strings **X**,
rev(rev(**X**)) = **X**

We go by structural induction on B. Suppose rev(rev(a)) = a and rev(rev(b)) = b for some strings a, b.

Case ε**:** rev(rev(ε)) = rev(ε) = ε — **Def of rev**

Case 0**:** rev(rev(0)) = rev(0) = 0 — **Def of rev**

Case 1**:** rev(rev(1)) = rev(1) = 1 — **Def of rev**

Case $a + b$**:**

rev(rev(a + b)) = rev(rev(b) + rev(a)) — **Def of rev**
= rev(rev(a)) + rev(rev(b)) — **Def of rev**
= a + b — **By IH!**

Since the claim is true for all the cases, it's true for all binary strings.

All Binary Strings with no 1's before 0's

00 Y

01 Y

10 N

1001 N

$$A = \varepsilon \mid 0 + A_0 \mid A_1 + 1$$

All Binary Strings with no 1's before 0's

$$\textbf{A} = \varepsilon \mid 0 + A_0 \mid A_1 + 1$$

A BIN is EMPTY or "0 BIN" or "BIN 1".

len : A → Int	
len(ε)	= 0
len(0 + a)	= 1 + len(a)
len(a + 1)	= 1 + len(a)

#0: A → Int	
#0(ε)	= 0
#0(0 + a)	= 1 + #0(a)
#0(a + 1)	= #0(a)

no1: A → A	
no1 (ε)	= ε
no1(0 + a)	= 0 + no1(a)
no1(a + 1)	= no1(a)

All Binary Strings with no 1's before 0's

$$\mathbf{A} = \varepsilon \mid 0 + A_0 \mid A_1 + 1$$

len : A → Int	#0: A → Int	no1: A → A
len(ε) = 0	#0(ε) = 0	no1 (ε) = ε
len(0 + a) = 1 + len(a)	#0(0 + a) = 1 + #0(a)	no1(0 + a) = 0 + no1(a)
len(a + 1) = 1 + len(a)	#0(a + 1) = #0(a)	no1(a + 1) = no1(a)

Claim: Prove that for all $x \in A$, len(no1(x)) = #0(x)

Case A = ε:

len(no1(ε)) = len(ε) by def of no1

= 0 by def of len

= #0(ε) by def of #0

All Binary Strings with no 1's before 0's

$$\mathbf{A} = \varepsilon \mid 0 + A_0 \mid A_1 + 1$$

len : A → Int	#0: A → Int	no1: A → A
len(ε) = 0	#0(ε) = 0	no1 (ε) = ε
len(0 + a) = 1 + len(a)	#0(0 + a) = 1 + #0(a)	no1(0 + a) = 0 + no1(a)
len(a + 1) = 1 + len(a)	#0(a + 1) = #0(a)	no1(a + 1) = no1(a)

Claim: Prove that for all $x \in A$, len(no1(x)) = #0(x)

We go by structural induction on A. Let A ∈ A be arbitrary.

Case A = ε:

len(no1(ε)) = len(ε) [Def of no1]

= 0 [Def of len]

= #0(ε) [Def of #0]

All Binary Strings with no 1's before 0's

$$\mathbf{A} = \varepsilon \mid 0 + A_0 \mid A_1 + 1$$

len : A → Int	#0: A → Int	no1: A → A
len(ε) = 0	#0(ε) = 0	no1 (ε) = ε
len(0 + a) = 1 + len(a)	#0(0 + a) = 1 + #0(a)	no1(0 + a) = 0 + no1(a)
len(a + 1) = 1 + len(a)	#0(a + 1) = #0(a)	no1(a + 1) = no1(a)

Claim: Prove that for all $x \in A$, len(no1(x)) = #0(x)

We go by structural induction on A. Let A ∈ A be arbitrary.
Suppose len(no1(x)) = #0(x) is true for some x ∈ A.
Case A = 0 + x:

len(no1(0+x)) = len(0 + no1(x)) by def no1
= 1 + len(no1(x)) by def len
= 1 + #0(x) by IH
= #0(0+x)

All Binary Strings with no 1's before 0's

$$\mathbf{A} = \varepsilon \mid 0 + A_0 \mid A_1 + 1$$

len : A → Int	#0: A → Int	no1: A → A
len(ε) = 0	#0(ε) = 0	no1 (ε) = ε
len(0 + a) = 1 + len(a)	#0(0 + a) = 1 + #0(a)	no1(0 + a) = 0 + no1(a)
len(a + 1) = 1 + len(a)	#0(a + 1) = #0(a)	no1(a + 1) = no1(a)

Claim: Prove that for all $x \in A$, len(no1(x)) = #0(x)

We go by structural induction on A. Let A ∈ A be arbitrary.
Suppose len(no1(x)) = #0(x) is true for some x ∈ A.
Case A = 0 + x:

len(no1(0 + x))	= len(0 + no1(x))	[Def of no1]
	= 1 + len(no1(x))	[Def of len]
	= 1 + #0(x)	[By IH]
	= #0(0 + x)	[Def of #0]

All Binary Strings with no 1's before 0's

$$\mathbf{A} = \varepsilon \mid 0 + \mathrm{A}_0 \mid \mathrm{A}_1 + 1$$

len	#0	no1
len : A → Int	#0: A → Int	no1: A → A
len(ε) = 0	#0(ε) = 0	no1 (ε) = ε
len(0 + a) = 1 + len(a)	#0(0 + a) = 1 + #0(a)	no1(0 + a) = 0 + no1(a)
len(a + 1) = 1 + len(a)	#0(a + 1) = #0(a)	no1(a + 1) = no1(a)

Claim: Prove that for all $x \in \mathcal{A}$, len(no1(x)) = #0(x)

We go by structural induction on A. Let A ∈ $\mathcal{A}$ be arbitrary.

Suppose len(no1(x)) = #0(x) is true for some x ∈ $\mathcal{A}$.

Case A = x + 1:

len(no1(x+1)) = len(no1(x)) by def no1

= #0(x) by IH

= #0(x+1) by def of #0

All Binary Strings with no 1's before 0's

$$\mathbf{A} = \varepsilon \mid 0 + \mathrm{A}_0 \mid \mathrm{A}_1 + 1$$

len : A → Int	#0: A → Int	no1: A → A
len(ε) = 0	#0(ε) = 0	no1 (ε) = ε
len(0 + a) = 1 + len(a)	#0(0 + a) = 1 + #0(a)	no1(0 + a) = 0 + no1(a)
len(a + 1) = 1 + len(a)	#0(a + 1) = #0(a)	no1(a + 1) = no1(a)

Claim: Prove that for all $x \in A$, len(no1(x)) = #0(x)

We go by structural induction on A. Let A ∈ A be arbitrary.
Suppose len(no1(x)) = #0(x) is true for some x ∈ A.
Case A = x + 1:

len(no1(x + 1))	= len(no1(x))	[Def of no1]
	= #0(x)	[By IH]
	= #0(x + 1)	[Def of #0]

Recursively Defined Programs (on Lists)

List = [ ] | a :: L

We'll assume a is an integer.

Write a function

$$\text{len} : \text{List} \to \text{Int}$$

that computes the length of a list.

len([]) = 0

len(x :: L) = 1 + len(L)

Finish the function

$$\text{append} : (\text{List}, \text{Int}) \to \text{List}$$

append([], i) = … i :: []

append(a :: L, i) = … a :: append(L, i)

which returns a list with i appended to the end

Recursively Defined Programs (on Lists)

$$\text{List} = [\ ]\ |\ a :: L$$

We'll assume a is an integer.

$\text{len} : \text{List} \rightarrow \text{Int}$

$\text{len}([]) = 0$

$\text{len}(a :: L) = 1 + \text{len}(L)$

$\text{append} : (\text{List}, \text{Int}) \rightarrow \text{List}$

$\text{append}([], i) = i :: []$

$\text{append}(a :: L, i) = a :: \text{append}(L, i)$

Claim: For all lists **L**, and integers **i**, if $\text{len}(L) = n$, then $\text{len}(\text{append}(L, i)) = n + 1$.

Recursively Defined Programs (on Lists)

$$\text{List} = [\ ]\ |\ a :: L$$

$\text{len} : \text{List} \to \text{Int}$
$\text{len}([]) = 0$
$\text{len}(a :: L) = 1 + \text{len}(L)$

$\text{append} : (\text{List}, \text{Int}) \to \text{List}$
$\text{append}([], i) = i :: []$
$\text{append}(a :: L, i) = a :: \text{append}(L, i)$

Claim: For all lists **L**, and integers **i**, if $\text{len}(L) = n$, then $\text{len}(\text{append}(L, i)) = n + 1$.

Recursively Defined Programs (on Lists)

$$\textbf{List} = [\,] \mid a :: L$$

$\text{len} : \text{List} \to \text{Int}$
$\text{len}([]) = 0$
$\text{len}(a :: L) = 1 + \text{len}(L)$

$\text{append} : (\text{List}, \text{Int}) \to \text{List}$
$\text{append}([], i) = i::[]$
$\text{append}(a :: L, i) = a :: \text{append}(L, i)$

Claim: For all lists **L**, and integers **i**, if $\text{len}(L) = n$, then $\text{len}(\text{append}(L, i)) = n + 1$.

We go by structural induction on List. Let i be an integer, and let L be a list. Suppose $\text{len}(L) = n$.

Case L = $[]$:

$$\begin{aligned} \text{len}(\text{append}([], i)) &= \text{len}(i::[]) && \textbf{[Def of append]} \\ &= 1 + \text{len}([]) && \textbf{[Def of len]} \\ &= 1 + 0 && \textbf{[Def of len]} \\ &= 1 && \textbf{[Arithmetic]} \end{aligned}$$

Recursively Defined Programs (on Lists)

$\text{len} : \text{List} \to \text{Int}$
$\text{len}([]) = 0$
$\text{len}(a :: L) = 1 + \text{len}(L)$

$\text{append} : (\text{List}, \text{Int}) \to \text{List}$
$\text{append}([], i) = i::[]$
$\text{append}(a :: L, i) = a :: \text{append}(L, i)$

Claim: For all lists **L**, and integers **i**, if $\text{len}(L) = n$, then $\text{len}(\text{append}(L, i)) = n + 1$.

We go by structural induction on List. Let i be an integer, and let L be a list. Suppose $\text{len}(L) = n$. And Suppose $\text{len}(\text{append}(L', i)) = k + 1$ is true for some list L'.

Case L = $x :: L'$:

Recursively Defined Programs (on Lists)

len : List → Int
len([]) = 0
len(a :: L) = 1 + len(L)

append : (List, Int) → List
append([], i) = i::[]
append(a :: L, i) = a :: append(L, i)

Claim: For all lists **L**, and integers **i**, if len(L) = n, then len(append(L, i)) = n + 1.

We go by structural induction on List. Let i be an integer, and let L be a list. Suppose len(L) = n. **And Suppose** len(append(L', i)) = k + 1 **is true for some list** L'.

Case L = $x :: L'$**:**

len(append(x :: L’, i)) = len(x :: append(L’, i)) **[Def of append]**
= 1 + len(append(L’, i)) **[Def of len]**

We know by our IH that, for all lists smaller than L,
If len(L) = n, then len(append(L, i)) = n + 1

So, if len(L’) = k, **then** len(append(L’, i)) = k + 1

Recursively Defined Programs (on Lists)

We go by structural induction on List. Let i be an integer, and let L be a list. Suppose $\mathrm{len(L) = n}$. And Suppose $\mathrm{len(no1}(L')) = \#0(L')$ is true for some list L'.

Case L = $x :: L'$:

$\mathrm{len(append(x :: L', i)) = len(x :: append(L', i))}$ [Def of append]

$\mathrm{= 1 + len(append(L', i))}$ [Def of len]

We know by our IH that, for all lists smaller than L,

If $\mathrm{len(L) = n}$, then $\mathrm{len(append(L, i)) = n + 1}$

So, if $\mathrm{len(L') = k}$, then $\mathrm{len(append(L', i)) = k + 1}$

$= 1 + k + 1$ [By IH]

Note that $\mathrm{n = len(L) = len(x :: L') = 1 + len(L') = 1 + k}$.

$= 1 + (n - 1) + 1$ [By above]

$= n + 1$ [By above]