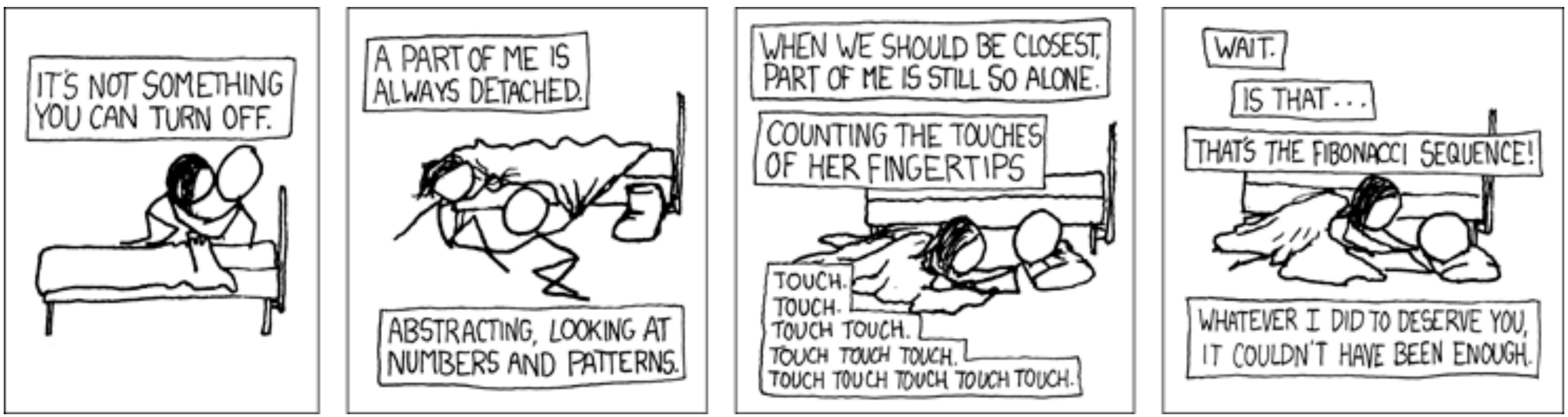
CSE 311: Foundations of Computing

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.

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

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”).

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

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

Recursively Defined Programs (on Binary Strings)

$$B = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1$$

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

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

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

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

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

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

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

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

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

$\text{rev}((01)+(01)) = \text{rev}(01) + \text{rev}(01)$

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

Recursively Defined Programs (on Binary Strings)

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

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

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)

$$B = \varepsilon \mid 0 \mid 1 \mid B + B$$

$\mathbb{N} = 0 \mid n+1$

$rev : B \to B$

$rev(\varepsilon) = \varepsilon$

$rev(0) = 0$

$rev(1) = 1$

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

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

Case ε: $rev(rev(\varepsilon)) = rev(\varepsilon) = \varepsilon$ **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**

Recursively Defined Programs (on Binary Strings)

B = ε | 0 | 1 | B + B

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**

Case $a + b$**:**

rev(rev(a + b)) =

Recursively Defined Programs (on Binary Strings)

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

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**

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!**

Recursively Defined Programs (on Binary Strings)

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

$\mathrm{rev} : \mathrm{B} \to \mathrm{B}$
$\mathrm{rev}(\varepsilon) = \varepsilon$
$\mathrm{rev}(0) = 0$
$\mathrm{rev}(1) = 1$
$\mathrm{rev}(a + b) = \mathrm{rev}(b) + \mathrm{rev}(a)$

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

We go by structural induction on B.

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

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

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

Case $a + b$:

$$
\begin{aligned}
\mathrm{rev}(\mathrm{rev}(a + b)) &= \mathrm{rev}(\mathrm{rev}(b) + \mathrm{rev}(a)) && \textbf{Def of rev} \\
&= \mathrm{rev}(\mathrm{rev}(a)) + \mathrm{rev}(\mathrm{rev}(b)) && \textbf{Def of rev} \\
&= a + b && \textbf{By IH!}
\end{aligned}
$$

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

111

~~10~~

$A = \varepsilon$

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

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

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

A BIN is EMPTY or "0 BIN" or "BIN 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)

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

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

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)

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

Case x = ε:

len(no1(ε)) = len(ε) by def of no1

= 0 def of len

= #0(ε) by def of #0

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

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

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 $x \in A$, len(no1(x)) = #0(x)

We go by structural induction on A. Let $x \in A$ 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

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

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 $x \in A$ be arbitrary.

Case A = 0 + x:

len(no1(0 + x)) = len(0 + no1(x))

= 1 + len(no1(x))

= 1 + #0(x) by IH

= #0(0 + x)

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

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

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)

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

We go by structural induction on A. Let $x \in A$ be arbitrary.

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

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

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 $x \in A$ **be arbitrary.**
Case A = x + 1:

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

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

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)

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

We go by structural induction on A. Let $x \in A$ be arbitrary.

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

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

that computes the length of a list.

Finish the function

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

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

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

which returns a list with i appended to the end

Recursively Defined Programs (on Lists)

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

We'll assume a is an integer.

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

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

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

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

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

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

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)

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

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

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

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)

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

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

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

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

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)

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

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

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

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

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.

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 len(L) = n.

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

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

Note that n = len(L) = len(x :: L') = 1 + len(L') = 1 + k.

	= 1 + (n – 1) + 1	**[By above]**
	= n + 1	**[By above]**