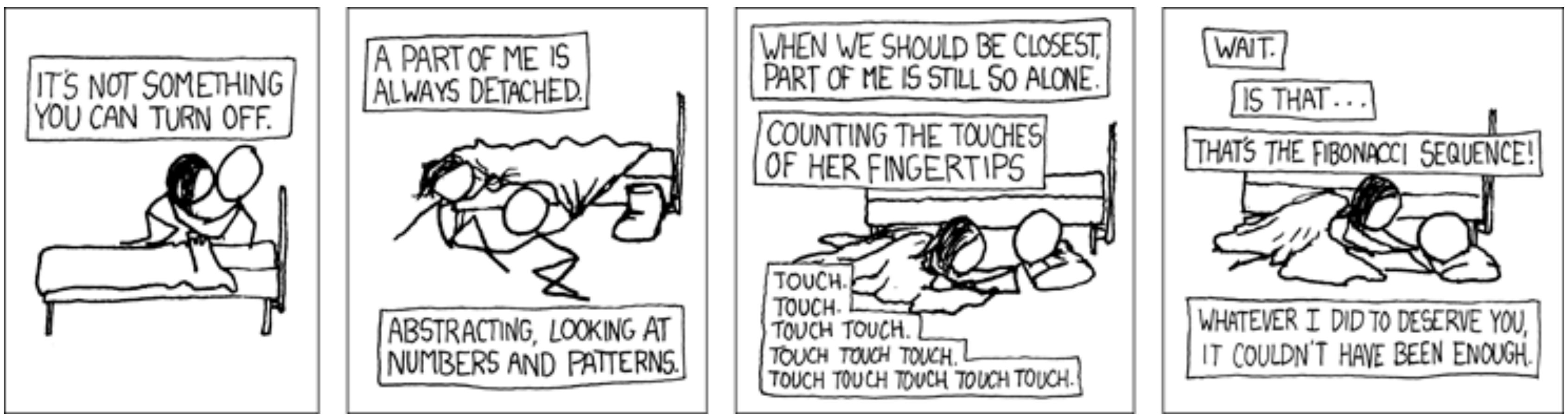
CSE 311: Foundations of Computing

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.

$$\Sigma^* = \varepsilon \,\Big|\, \Sigma^* \sigma$$

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)

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

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

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

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

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

Recursively Defined Programs (on Binary Strings)

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

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

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

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

Recursively Defined Programs (on Binary Strings)

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

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

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

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

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

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

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)

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

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

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

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

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

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

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

Case $a + b$**:**

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

Recursively Defined Programs (on Binary Strings)

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

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

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

We go by structural induction on B.

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**

Case $a + b$**:**

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

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

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

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 = 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

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

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 = 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:

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.

Finish the function

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

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

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

which returns a list with i appended to the end

Recursively Defined Programs (on Lists)

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

We'll assume a is an integer.

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

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

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

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

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

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

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)

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

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

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

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 = []:

$\text{len}(\text{append}([], i))$	$= \text{len}(i :: [])$	**[Def of append]**
	$= 1 + \text{len}([])$	**[Def of len]**
	$= 1 + 0$	**[Def of len]**
	$= 1$	**[Arithmetic]**

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 $\mathrm{len}(L) = n$.

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

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

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

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

$$= 1 + k + 1 \quad \text{[By IH]}$$

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

$$= 1 + (n - 1) + 1 \quad \text{[By above]}$$
$$= n + 1 \quad \text{[By above]}$$