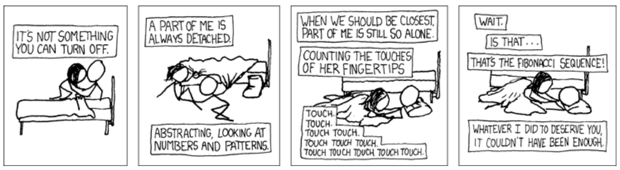
CSE 311: Foundations of Computing

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").

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

A PAL is EMPTY or CHAR or "CHAR PAL CHAR".

Recursively Defined Programs (on Binary Strings)

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

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

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

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

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

Recursively Defined Programs (on Binary Strings)

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

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

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

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

Recursively Defined Programs (on Binary Strings)

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

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

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

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)

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)

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**

We go by structural induction on 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

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 = 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))	[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

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 = 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

len : List → Int

that computes the length of a list.

Finish the function

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

which returns a list with i appended to the end

Recursively Defined Programs (on Lists)

List = [ ] | a :: L

We'll assume a is an integer.

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.

Recursively Defined Programs (on Lists)

List = [ ] | a :: L

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 = []:

len(append([], i))	= len(i::[])	[Def of append]
	= 1 + 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 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]