Lecture 17: Structural Induction

It's not something you can turn off.

A part of me is always detached.

Abstracting, looking at numbers and patterns.

When we should be closest, part of me is still so alone.

Counting the touches of her fingertips.


Wait.

Is that...

That's the Fibonacci sequence!

Whatever I did to deserve you, it couldn't have been enough.
Strings

• An alphabet $\Sigma$ is any finite set of characters

• The set $\Sigma^*$ is the set of strings over the alphabet $\Sigma$.

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

A STRING is EMPTY or “STRING CHAR”.

• The set of strings is made up of:
  – $\varepsilon \in \Sigma^*$ ($\varepsilon$ 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”.

\[ abba = a(bb)a = a(\varepsilon)b(\varepsilon)a \]
Recursively Defined Programs (on Binary Strings)

\[ B = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1 \]

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

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

\[ \text{rev} : B \to B \]

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

\[ \text{rev}(\varepsilon) = \varepsilon \]
\[ \text{rev}(0) = 0 \]
\[ \text{rev}(1) = 1 \]
\[ \text{rev}(a + b) = \text{rev}(b) + \text{rev}(a) \]
Recursively Defined Programs (on Binary Strings)

\[ B = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1 \]

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

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

\[
\text{rev} : B \rightarrow B \\
\text{rev}(\varepsilon) = \varepsilon \\
\text{rev}(0) = 0 \\
\text{rev}(1) = 1 \\
\text{rev}(a + b) = \text{rev}(b) + \text{rev}(a)
\]
Recursively Defined Programs (on Binary Strings)

\[ B = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1 \]

\[ \text{rev} : B \rightarrow B \]

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

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

Case \( \varepsilon \): \( \text{rev}(\text{rev}(\varepsilon)) = \text{rev}(\varepsilon) = \varepsilon \) \hspace{1cm} \text{Def of rev}

Case \( 0 \): \( \text{rev}(\text{rev}(0)) = \text{rev}(0) = 0 \) \hspace{1cm} \text{Def of rev}

Case \( 1 \): \( \text{rev}(\text{rev}(1)) = \text{rev}(1) = 1 \) \hspace{1cm} \text{Def of rev}
Recursively Defined Programs (on Binary Strings)

\[ B = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1 \]

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 \( X \), \( \text{rev}(\text{rev}(X)) = X \)

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

**Case** \( a + b \):

\[ \text{rev}(\text{rev}(a + b)) = (\text{rev}(b) + \text{rev}(a)) = \text{rev}(a + b) \]

so \[ \varepsilon + 0 = \varepsilon \]

and \[ 1 + 1 = 0 \]
Recursively Defined Programs (on Binary Strings)

\[ B = \varepsilon | 0 | 1 | B_0 + B_1 \]

\[
\begin{align*}
\text{rev}(\varepsilon) &= \varepsilon \\
\text{rev}(0) &= 0 \\
\text{rev}(1) &= 1 \\
\text{rev}(a + b) &= \text{rev}(b) + \text{rev}(a)
\end{align*}
\]

**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** \( a + b \):

\[
\begin{align*}
\text{rev}(\text{rev}(a + b)) &= \text{rev}(\text{rev}(b) + \text{rev}(a)) \\
&= \text{rev}(\text{rev}(a)) + \text{rev}(\text{rev}(b)) \\
&= a + b
\end{align*}
\]

Def of rev: \text{Def of rev} 

By IH!
Recursively Defined Programs (on Binary Strings)

\[ B = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1 \]

**Claim:** For all binary strings \( X \),

\[ \text{rev} (\text{rev} (X)) = X \]

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

**Case** \( \varepsilon \):

\[ \text{rev} (\text{rev} (\varepsilon)) = \text{rev} (\varepsilon) = \varepsilon \]

**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** \( a + b \):

\[ \text{rev} (\text{rev} (a + b)) = \text{rev} (\text{rev} (b) + \text{rev} (a)) \]

\[ = \text{rev} (\text{rev} (a)) + \text{rev} (\text{rev} (b)) \]

\[ = a + b \]

**Def of rev**

**Def of rev**

**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  
01  
10  
1001  

$A = 3\mid 0^+A_3\upharpoonright A_1+1$
All Binary Strings with no 1’s before 0’s

\[ A = \varepsilon \mid 0 + A_0 \mid A_1 + 1 \]

A BIN is EMPTY or “0 BIN” or “BIN 1”.

<table>
<thead>
<tr>
<th>len : A \rightarrow \text{Int}</th>
<th>#0 : A \rightarrow \text{Int}</th>
<th>no1 : A \rightarrow A</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(\varepsilon) = 0</td>
<td>#0(\varepsilon) = 0</td>
<td>no1(\varepsilon) = \varepsilon</td>
</tr>
<tr>
<td>len(0 + a) = 1 + len(a)</td>
<td>#0(0 + a) = 1 + #0(a)</td>
<td>no1(0 + a) = 0 + no1(a)</td>
</tr>
<tr>
<td>len(a + 1) = 1 + len(a)</td>
<td>#0(a + 1) = #0(a)</td>
<td>no1(a + 1) = no1(a)</td>
</tr>
</tbody>
</table>
### All Binary Strings with no 1’s before 0’s

\[ A = \varepsilon \mid 0 + A_0 \mid A_1 + 1 \]

<table>
<thead>
<tr>
<th>len: A → Int</th>
<th>#0: A → Int</th>
<th>no1: A → A</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(\varepsilon) = 0</td>
<td>#0(\varepsilon) = 0</td>
<td>no1(\varepsilon) = \varepsilon</td>
</tr>
<tr>
<td>len(0 + a) = 1 + len(a)</td>
<td>#0(0 + a) = 1 + #0(a)</td>
<td>no1(0 + a) = 0 + no1(a)</td>
</tr>
<tr>
<td>len(a + 1) = 1 + len(a)</td>
<td>#0(a + 1) = #0(a)</td>
<td>no1(a + 1) = no1(a)</td>
</tr>
</tbody>
</table>

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

**Case** \( A = \varepsilon \):

\[
\text{len} (\text{no1}(\varepsilon)) = \text{len}(\varepsilon) \\
= 0 \\
= #0(\varepsilon)
\]

by def of \( \text{no1} \) 

by def of \( \text{len} \) 

by def of \( #0 \)
All Binary Strings with no 1’s before 0’s

\[ A = \varepsilon \mid 0 + A_0 \mid A_1 + 1 \]

<table>
<thead>
<tr>
<th>len : A → Int</th>
<th>#0 : A → Int</th>
<th>no1 : A → A</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(\varepsilon) = 0</td>
<td>#0(\varepsilon) = 0</td>
<td>no1(\varepsilon) = \varepsilon</td>
</tr>
<tr>
<td>len(0 + a) = 1 + len(a)</td>
<td>#0(0 + a) = 1 + #0(a)</td>
<td>no1(0 + a) = 0 + no1(a)</td>
</tr>
<tr>
<td>len(a + 1) = 1 + len(a)</td>
<td>#0(a + 1) = #0(a)</td>
<td>no1(a + 1) = no1(a)</td>
</tr>
</tbody>
</table>

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

We go by structural induction on \( A \). Let \( A \in A \) be arbitrary.

**Case \( A = \varepsilon \):**

\[
\text{len}(\text{no1}(\varepsilon)) = \text{len}(\varepsilon) \quad \text{[Def of no1]}
\]
\[
= 0 \quad \text{[Def of len]}
\]
\[
= \#0(\varepsilon) \quad \text{[Def of \#0]}
\]
All Binary Strings with no 1’s before 0’s

\[ A = \varepsilon \mid 0 + A_0 \mid A_1 + 1 \]

<table>
<thead>
<tr>
<th>len : A → Int</th>
<th>#0: A → Int</th>
<th>no1: A → A</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(\varepsilon) = 0</td>
<td>#0(\varepsilon) = 0</td>
<td>no1(\varepsilon) = \varepsilon</td>
</tr>
<tr>
<td>len(0 + a) = 1 + len(a)</td>
<td>#0(0 + a) = 1 + #0(a)</td>
<td>no1(0 + a) = 0 + no1(a)</td>
</tr>
<tr>
<td>len(a + 1) = 1 + len(a)</td>
<td>#0(a + 1) = #0(a)</td>
<td>no1(a + 1) = no1(a)</td>
</tr>
</tbody>
</table>

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

We go by structural induction on A. Let \( A \in A \) be arbitrary. Suppose \( \text{len}(\text{no1}(x)) = \#0(x) \) is true for some \( x \in A \).

**Case A = 0 + x:**

\[ \text{len}(\text{no1}(0 + x)) = \text{len}(0 + \text{no1}(x)) \]

\[ = 1 + \text{len}(\text{no1}(x)) \]

\[ = 1 + \#0(x) \]

\[ = \#0(0 + x) \]
All Binary Strings with no 1’s before 0’s

\[ A = \varepsilon \mid 0 + A_0 \mid A_1 + 1 \]

<table>
<thead>
<tr>
<th>len : A → Int</th>
<th>#0 : A → Int</th>
<th>no1 : A → A</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(ε) = 0</td>
<td>#0(ε) = 0</td>
<td>no1(ε) = ε</td>
</tr>
<tr>
<td>len(0 + a) = 1 + len(a)</td>
<td>#0(0 + a) = 1 + #0(a)</td>
<td>no1(0 + a) = 0 + no1(a)</td>
</tr>
<tr>
<td>len(a + 1) = 1 + len(a)</td>
<td>#0(a + 1) = #0(a)</td>
<td>no1(a + 1) = no1(a)</td>
</tr>
</tbody>
</table>

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

We go by structural induction on A. Let \( A \in A \) be arbitrary.

Suppose \( \text{len}(\text{no1}(x)) = \text{#0}(x) \) is true for some \( x \in A \).

Case \( A = 0 + x \):

\[
\text{len}(\text{no1}(0 + x)) = \text{len}(0 + \text{no1}(x)) \quad [\text{Def of no1}]
\]

\[
= 1 + \text{len}(\text{no1}(x)) \quad [\text{Def of len}]
\]

\[
= 1 + \text{#0}(x) \quad [\text{By IH}]
\]

\[
= \text{#0}(0 + x) \quad [\text{Def of #0}]
\]
All Binary Strings with no 1’s before 0’s

\[ A = \varepsilon \mid 0 + A_0 \mid A_1 + 1 \]

<table>
<thead>
<tr>
<th>len : A \rightarrow \text{Int}</th>
<th>#0: A \rightarrow \text{Int}</th>
<th>no1: A \rightarrow A</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(\varepsilon) = 0</td>
<td>#0(\varepsilon) = 0</td>
<td>no1(\varepsilon) = \varepsilon</td>
</tr>
<tr>
<td>len(0 + a) = 1 + len(a)</td>
<td>#0(0 + a) = 1 + #0(a)</td>
<td>no1(0 + a) = 0 + no1(a)</td>
</tr>
<tr>
<td>len(a + 1) = 1 + len(a)</td>
<td>#0(a + 1) = #0(a)</td>
<td>no1(a + 1) = no1(a)</td>
</tr>
</tbody>
</table>

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

We go by structural induction on \( A \). Let \( A \in A \) be arbitrary. Suppose \( \text{len}(\text{no1}(x)) = \#0(x) \) is true for some \( x \in A \).

Case \( A = x + 1 \):

\[ \text{len}(\text{no1}(x + 1)) = \text{len}(\text{no1}(x)) \]
\[ = \#0(x) \quad \text{by IH} \]
\[ = \#0(x + 1) \quad \text{by def of } \text{no1} \]
All Binary Strings with no 1’s before 0’s

\[ A = \varepsilon \mid 0 + A_0 \mid A_1 + 1 \]

<table>
<thead>
<tr>
<th>len: A → Int</th>
<th>#0: A → Int</th>
<th>no1: A → A</th>
</tr>
</thead>
<tbody>
<tr>
<td>len((\varepsilon)) = 0</td>
<td>#0((\varepsilon)) = 0</td>
<td>no1((\varepsilon)) = (\varepsilon)</td>
</tr>
<tr>
<td>len(0 + a) = 1 + len(a)</td>
<td>#0(0 + a) = 1 + #0(a)</td>
<td>no1(0 + a) = 0 + no1(a)</td>
</tr>
<tr>
<td>len(a + 1) = 1 + len(a)</td>
<td>#0(a + 1) = #0(a)</td>
<td>no1(a + 1) = no1(a)</td>
</tr>
</tbody>
</table>

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

We go by structural induction on A. Let A \(\in A\) be arbitrary.

Suppose \(\text{len}(\text{no1}(x)) = \#0(x)\) is true for some \(x \in A\).

Case A = x + 1:

\[
\begin{align*}
\text{len}(\text{no1}(x + 1)) &= \text{len}(\text{no1}(x)) \\
&= \#0(x) \\
&= \#0(x + 1)
\end{align*}
\]

[Def of \text{no1}] [By IH] [Def of \#0]
Recursively Defined Programs (on Lists)

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

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

\[
\begin{align*}
\text{append}([], i) &= \cdots \quad i :: C \\text{J} \\
\text{append}(a :: L, i) &= \cdots \quad a :: \text{append} (L, i)
\end{align*}
\]

which returns a list with \( i \) appended to the end.
Recursively Defined Programs (on Lists)

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

We’ll assume \( a \) is an integer.

\[
\begin{align*}
\text{len} : \text{List} &\rightarrow \text{Int} \\
\text{len}([]) & = 0 \\
\text{len}(a :: \text{L}) & = 1 + \text{len}(\text{L})
\end{align*}
\]

\[
\begin{align*}
\text{append} : (\text{List}, \text{Int}) &\rightarrow \text{List} \\
\text{append}([], i) & = i :: [] \\
\text{append}(a :: \text{L}, i) & = a :: \text{append}(\text{L}, i)
\end{align*}
\]

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

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

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

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

- \( \text{append}([\ ], i) = i :: [\ ] \)
- \( \text{append}(a :: L, i) = a :: \text{append}(L, i) \)

\textbf{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)

**List** = [ ] | a :: L

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

**append** : (List, Int) → 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 = []:
\[\text{len}(\text{append}([], i)) = \text{len}(i :: [])\]
\[= 1 + \text{len}([])\]
\[= 1 + 0\]
\[= 1\]
Recursively Defined Programs (on Lists)

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

append : (\text{List}, \text{Int}) \rightarrow \text{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** \(L = x :: L'\):
Recursively Defined Programs (on Lists)

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

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

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

We go by structural induction on \(\text{List}\). 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** \(L = x :: L'\):

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

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

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

So, if \(\text{len}(L') = k\), then \(\text{len}(\text{append}(L', i)) = k + 1\)
Recursively Defined Programs (on Lists)

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

**Case** \( L = x :: L' \):

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

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

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

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

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

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