CSE 311: Foundations of Computing

Lecture 17: Structural Induction

1. It's not something you can turn off.
2. A part of me is always detached.
3. Abstraction, looking at numbers and patterns.
4. When we should be closest, part of me is still so alone.
5. Counting the touches of her fingertips.
6. Touch, touch, touch, touch, touch, touch, touch, touch, touch, touch, touch, touch.
7. Wait, is that... that's the Fibonacci sequence!
8. 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”, “tth”, “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)

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

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

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

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

\( \text{rev} \) is a function that takes in a binary string and returns a binary string

\[ \text{rev}(\varepsilon) = \varepsilon, \quad \text{rev}(0) = 0, \quad \text{rev}(1) = 1, \quad \text{rev}(B) = B \]

\[ \text{rev}(01) = \text{rev}(\text{rev}(0) + \text{rev}(1)) = \text{rev}(0) + \text{rev}(1) = 0 + 1 = 10 \]

\[ \text{rev}(BSTR BSTR) = \text{rev}(B) + \text{rev}(BSTR) = B + BSTR = BSTR + B \]
Recursively Defined Programs (on Binary Strings)

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

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

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)

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

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

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

**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)

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

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

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

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

Case \( a + b \):

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

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

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

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

\[ = a + b \]

by IH
Recursively Defined Programs (on Binary Strings)

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

rev : B \to B

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

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

**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

Def of rev

By IH!
Recursively Defined Programs (on Binary Strings)

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

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

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

We go by structural induction on \( B \).

**Case** \( \varepsilon \): \( \text{rev}(\text{rev}(\varepsilon)) = \text{rev}(\varepsilon) = \varepsilon \)  

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

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

**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 \) \]  

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

\[ A = \varepsilon | 0 + A | A + 1 \]

\[ \varepsilon \]

\[ A = \varepsilon \]
All Binary Strings with no 1’s before 0’s

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

A BIN is EMPTY or “0 BIN” or “BIN 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>
All Binary Strings with no 1’s before 0’s

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

<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 \( x = \varepsilon \):

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

by def of len

def off \text{len}

by def of \#0

by def of no1
All Binary Strings with no 1’s before 0’s

A = ε \mid 0 + A \mid A + 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 \( x \in A \), \( \text{len}(\text{no1}(x)) = \#0(x) \)

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

**Case A = ε:**

\[
\text{len}(	ext{no1}(ε)) = \text{len}(ε) \quad \text{[Def of no1]}
\]

\[
= 0 \quad \text{[Def of len]}
\]

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

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

<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 \( x \in A \) be arbitrary.

Case \( A = 0 + x \):
\[
\text{len}(\text{no1}(0 + x)) = \text{len}(\varepsilon + \text{no1}(x))
\]
\[
= 1 + \text{len}(\text{no1}(x))
\]
\[
= 1 + \#0(x)
\]
\[
= \#0(0 + x)
\]
All Binary Strings with no 1’s before 0’s

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

<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 \(x \in A\) be arbitrary.

**Case A = 0 + x:**

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

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

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

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

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

<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 \( x \in A \) be arbitrary.
Case A = x + 1:
All Binary Strings with no 1’s before 0’s

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

<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 \( x \in A \) be arbitrary.

**Case** \( A = x + 1 \):

\[
\text{len}(\text{no1}(x + 1)) = \text{len}(\text{no1}(x)) \\
= \#0(x) \\
= \#0(x + 1)
\]

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

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

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
\[
\text{append} : (\text{List}, \text{Int}) \rightarrow \text{List}
\]
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.
Recursively Defined Programs (on Lists)

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

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 = []:
\[
\text{len}(\text{append}(\text{}, i)) = \text{len}(i :: [])
\]
\[
= 1 + \text{len}([])
\]
\[
= 1 + 0
\]
\[
= 1
\]

[Def of append]
[Def of len]
[Def of len]
[Arithmetic]
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 List. Let \( i \) be an integer, and let \( L \) be a list. Suppose \( \text{len}(L) = n \).

**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 List. Let \( i \) be an integer, and let \( L \) be a list. Suppose \( \text{len}(L) = n \).

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

Case $L = x :: L'$:

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

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

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

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

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