Lecture 17: Structural Induction
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)

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

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} is a function that takes in a binary string and returns a binary string.
Recursively Defined Programs (on Binary Strings)

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

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)

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

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

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

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

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

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

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

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

\[\text{rev(0)} = 0\]

\[\text{rev(1)} = 1\]

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

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

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

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

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

\[
= a + b\]

By IH!
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 \( X \), \( \text{rev}(\text{rev}(X)) = X \)

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

**Case \( \varepsilon \):** \( \text{rev}(\text{rev}(\varepsilon)) = \text{rev}(\varepsilon) = \varepsilon \) [Def of \( \text{rev} \)]

**Case 0:** \( \text{rev}(\text{rev}(0)) = \text{rev}(0) = 0 \) [Def of \( \text{rev} \)]

**Case 1:** \( \text{rev}(\text{rev}(1)) = \text{rev}(1) = 1 \) [Def of \( \text{rev} \)]

**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 \( \text{rev} \)]

[Def of \( \text{rev} \)]

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

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

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

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

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

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

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

<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>Function</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len</code></td>
<td><code>A</code></td>
<td><code>Int</code></td>
</tr>
<tr>
<td><code>len(\varepsilon)</code></td>
<td>= 0</td>
<td></td>
</tr>
<tr>
<td><code>len(0 + a)</code></td>
<td>= 1 + <code>len(a)</code></td>
<td></td>
</tr>
<tr>
<td><code>len(a + 1)</code></td>
<td>= 1 + <code>len(a)</code></td>
<td></td>
</tr>
<tr>
<td><code>#0</code></td>
<td><code>A</code></td>
<td><code>Int</code></td>
</tr>
<tr>
<td><code>#0(\varepsilon)</code></td>
<td>= 0</td>
<td></td>
</tr>
<tr>
<td><code>#0(0 + a)</code></td>
<td>= 1 + <code>#0(a)</code></td>
<td></td>
</tr>
<tr>
<td><code>#0(a + 1)</code></td>
<td>= <code>#0(a)</code></td>
<td></td>
</tr>
<tr>
<td><code>no1</code></td>
<td><code>A</code></td>
<td><code>A</code></td>
</tr>
<tr>
<td><code>no1(\varepsilon)</code></td>
<td>= <code>\varepsilon</code></td>
<td></td>
</tr>
<tr>
<td><code>no1(0 + a)</code></td>
<td>= <code>0 + no1(a)</code></td>
<td></td>
</tr>
<tr>
<td><code>no1(a + 1)</code></td>
<td>= <code>no1(a)</code></td>
<td></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 \( x \in A \) be arbitrary.

**Case** \( A = x + 1 \):
\[
\text{len}(\text{no1}(x + 1)) = \text{len}(\text{no1}(x)) \quad \text{[Def of no1]}
\]
\[
= \#0(x) \quad \text{[By IH]}
\]
\[
= \#0(x + 1) \quad \text{[Def of #0]}
\]
Recursively Defined Programs (on Lists)

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

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

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

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

**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)

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

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

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

**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 = []:**

\[
\begin{align*}
\text{len}(\text{append}([ ], i)) & = \text{len}(i :: [ ])
\end{align*}
\]

[Def of append]

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

[Def of len]

\[
= 1 + 0
\]

[Def of len]

\[
= 1
\]

[Arithmetic]
Recursively Defined Programs (on Lists)

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

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

\textbf{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 \(\text{len}(L) = n\).

\textbf{Case} \(L = x :: L'\):

\[\begin{align*}
\text{len}(\text{append}(x :: L', i)) & = \text{len}(x :: \text{append}(L', i)) \\
& = 1 + \text{len}(\text{append}(L', i))
\end{align*}\]

[Def of append]

[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 $\text{len}(L) = n$.

Case $L = x :: L'$:

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

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

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

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

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

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

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