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:
  - $\varepsilon \in \Sigma^*$ (ԑ 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)

$$B = \varepsilon \mid 0 \mid 1 \mid B_0 + B_1$$

A BSTR is empty, 0, 1, or "BSTR0 BSTR1".

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

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

$\text{rev}$ 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)

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

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

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

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

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

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

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

Def of rev
\[ = a + b \]

By IH!

All Binary Strings with no 1’s before 0’s

\[ \begin{array}{c}
0 \mid 0 \\
0 \mid 1 \\
1 \mid 0 \\
1 \mid 1 \\
\end{array} \]

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

A BIN is EMPTY or “0 BIN” or “BIN 1.”

Recursively Defined Programs (on Binary Strings)

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

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

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

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

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

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

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

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

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

\[ A = \varepsilon \big| 0 + A_0 \big| A_1 + 1 \]

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

Case \( A = \varepsilon \):

\[ \text{len}(\text{no1}(\varepsilon)) = \text{len}(\varepsilon) = 0 \] by def \( \varepsilon \) \& \( \text{no1} \)

\[ = 0 \] by def \( \text{len} \)

\[ = \#0(\varepsilon) \] by def \( \#0 \)

Case \( A = 0 + x \):

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

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

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

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

\[ = \#0(0 + x) \] by def \( \#0 \)

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

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

Case \( A = \varepsilon \):

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

Case \( A = 0 + x \):

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

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

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

\[ = \#0(0 + x) \] by def \( \#0 \)

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

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

Case \( A = \varepsilon \):

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

Case \( A = x + 1 \):

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

\[ = \#0(x) \] by def \( \#0 \)

\[ = \#0(x + 1) \] by def \( \#0 \)
Recursively Defined Programs (on Lists)

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

We'll assume \( a \) is an integer.

Write a function

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

that computes the length of a list.

Finish the function

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

which returns a list with \( i \) appended to the end.

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 \( L \). 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 \( L = x :: L' \):
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. And Suppose len(no1(L')) = #0(L') is true for some list L'.

Case L = x :: L':

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

[Def of append]

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

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

\[ = 1 + k + 1 \]

[By IH]

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

\[ = 1 + (n - 1) + 1 \]

[By above]

\[ = n + 1 \]

[By above]