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”.

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.

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

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
Since the claim is true for all the cases, it's true for all

Recursively Defined Programs (on Binary Strings)

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

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

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

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

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

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

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

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

\[ \text{len}(\text{no1}(\text{rev}(\text{rev}(\epsilon)))) = \text{len}(\text{rev}(\text{rev}(\epsilon))) \]

\[ = 0 \]

\[ = \text{len}(\epsilon) \]

\[ = \text{len}(\text{no1}(\epsilon)) \]

\[ = \text{no1}(\text{rev}(\text{rev}(\epsilon))) \]

\[ = 0 \]

\[ = \text{len}(\epsilon) \]

\[ = \text{len}(\text{no1}(\epsilon)) \]

\[ = \text{no1}(\text{rev}(\text{rev}(\epsilon))) \]

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

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

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

\[ = a + b \]

By IH!

Since the claim is true for all the cases, it's true for all binary strings.

Recursively Defined Programs (on Binary Strings)

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

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

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

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

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

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

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

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

**Case** \( \epsilon : \)

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

**Def of rev**

**Case** \( 0 : \)

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

**Def of rev**

**Case** \( 1 : \)

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

**Def of rev**

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

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

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

\[ = a + b \]

By IH!

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

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

\[ \text{len} : A \rightarrow \text{Int} \]

\[ \text{len}(\epsilon) = 0 \]

\[ \text{len}(0 + a) = 1 + \text{len}(a) \]

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

\[ \text{#0} : A \rightarrow \text{Int} \]

\[ \text{#0}(\epsilon) = 0 \]

\[ \text{#0}(0 + a) = 1 + \text{#0}(a) \]

\[ \text{#0}(a + 1) = \text{#0}(a) \]

\[ \text{no1} : A \rightarrow A \]

\[ \text{no1}(\epsilon) = \epsilon \]

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

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

**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** \( A = \epsilon : \)

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

\[ = 0 \]  [Def of len]

\[ = \text{#0}(\epsilon) \]  [Def of #0]

**Case** \( 0 + x : \)

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

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

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

\[ = \text{#0}(0 + x) \]  [Def of #0]

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

\[ \text{len}(\text{no1}(a + 1)) = \text{len}(\text{no1}(0 + a)) \]  [Def of no1]

\[ = \text{#0}(x + 1) \]  [Def of #0]

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

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

**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

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

\[ \text{append}([\text{}], i) = \ldots \]

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

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)

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 [Arithmetic]

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

Note that n = len(L) = len(x :: L) = 1 + len(L) = 1 + k.

= 1 + (n − 1) + 1 [By above]
= n + 1 [By above]