CSE 311

Foundations of Computing I

* All slides are a combined effort between previous instructors of the course

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

$$\mathbf{A} = \varepsilon \mid 0 + \mathrm{A} \mid \mathrm{A} + 1$$

len : A → Int	
len(ε)	= 0
len(0 + a)	= 1 + len(a)
len(a + 1)	= 1 + len(a)

#0: A → Int	
#0(ε)	= 0
#0(0 + a)	= 1 + #0(a)
#0(a + 1)	= #0(a)

no1: A → A	
no1 (ε)	= ε
no1(0 + a)	= 0 + no1(a)
no1(a + 1)	= no1(a)

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

We go by structural induction on A. Let A ∈ $\mathcal{A}$ be arbitrary.
Suppose len(no1(x)) = #0(x) is true for some x ∈ $\mathcal{A}$.
Case A = x + 1:

len(no1(x + 1))	= len(no1(x))	[Def of no1]
	= #0(x)	[By IH]
	= #0(x + 1)	[Def of #0]

Structural Induction

How to prove $\forall(x \in S)\ P(x)$ is true:

- **Base Case:** Show that P(u) is true for all specific elements of $u \in S$ mentioned in the *Basis step*
- **Inductive Hypothesis:** Assume that P is true for some arbitrary values of each of the existing named elements mentioned in the *Recursive step*
- **Inductive Step:** Prove that P(w) holds for each of the new elements constructed in the *Recursive step* using the named elements mentioned in the Inductive Hypothesis
- **Conclude** that $\forall(x \in S)\ P(x)$

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

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

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

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

which returns a list with i appended to the end

Recursively Defined Programs (on Lists)

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

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**,

len(append(L, i)) = 1 + len(L).

Recursively Defined Programs (on Lists)

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

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

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

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

Let i be an integer, and let L be a list. We go by structural induction on L.

Case L = []:

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

$= 1 + \text{len}([])$ **[Def of len]**

Recursively Defined Programs (on Lists)

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**,
then len(append(L, i)) = 1 + if len(L).

Let i be an integer, and let L be a list. We go by structural induction on L.

Suppose "len(append(L', i)) = len(L') + 1" is true for some list L'.

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]**
= 1 + (1 + len(L')) **[By IH]**
= 1 + len(x::L') **[Def of len]**

The Whole Proof!

Let i be an integer, and let L be a list. We go by structural induction on L.

Case L = [ ]:

len(append([ ], i))	= len(i::[ ])	**[Def of append]**
	= 1 + len([ ])	**[Def of len]**

Suppose "len(append(L', i)) = len(L') + 1" **is true for some list** L'**.**

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]**
	= 1 + (1 + len(L'))	**[By IH]**
	= 1 + len(x::L')	**[Def of len]**

Since the claim is true for all cases of the definition of List, it's true for all lists.

CSE 311: Foundations of Computing

Lecture 18: Regular expressions

Languages: Sets of Strings

- Sets of strings that satisfy special properties are called *languages*. Examples:
  - English sentences
  - Syntactically correct Java/C/C++ programs
  - Σ^* = All strings over alphabet Σ
  - Palindromes over Σ
  - Binary strings that don't have a 0 after a 1
  - Legal variable names. keywords in Java/C/C++
  - Binary strings with an equal # of 0's and 1's

Regular Expressions

Regular expressions over Σ

- **Basis**:

  $\varnothing$, ε are regular expressions

  a is a regular expression for any $a \in \Sigma$

- **Recursive step**:
  - If **A** and **B** are regular expressions then so are:

    $(A \cup B)$

    (AB)

    A^*

REGEX = $\varnothing$ | ε | a | REGEX $\cup$ REGEX | REGEX REGEX | REGEX *

Each Regular Expression is a "pattern"

ε matches the **empty string**

a matches the one character string *a*

(A ∪ B) matches all strings that either **A** matches or **B** matches (or both)

(AB) matches all strings that have a first part that **A** matches followed by a second part that **B** matches

A* matches all strings that have any number of strings (even 0) that **A** matches, one after another

Examples

001*

{00, 001, 0011, 00111, ...}

0*1*

Any number of 0's followed by any number of 1's

Examples

$(\mathbf{0} \cup \mathbf{1})\mathbf{0}(\mathbf{0} \cup \mathbf{1})\mathbf{0}$

{0000, 0010, 1000, 1010}

$(\mathbf{0}^*\mathbf{1}^*)^*$

All binary strings

Examples

$(\mathit{0} \cup \mathit{1})^*\mathit{0110}(\mathit{0} \cup \mathit{1})^*$

Strings that contain "0110"

$(\mathit{00} \cup \mathit{11})^*(\mathit{01010} \cup \mathit{10001})(\mathit{0} \cup \mathit{1})^*$

Strings that begin with pairs of characters followed by "01010" or "10001"

Regular Expressions in Practice

- Used to define the “tokens”: e.g., legal variable names, keywords in programming languages and compilers
- Used in **`grep`**, a program that does pattern matching searches in UNIX/LINUX
- Pattern matching using regular expressions is an essential feature of PHP
- We can use regular expressions in programs to process strings!

Regular Expressions in Java

- Pattern p = Pattern.compile("a*b");
- Matcher m = p.matcher("aaaaab");
- boolean b = m.matches();

`[01]`	a 0 or a 1	`^` start of string	`$` end of string				
`[0-9]`	any single digit	`\.`	period	`\,`	comma	`\-`	minus
`.`	any single character						

ab	a followed by b	(**AB**)
(a|b)	a or b	(**A** $\cup$ **B**)
a?	zero or one of a	(**A** $\cup$ ε)
a*	zero or more of a	**A***
a+	one or more of a	**AA***

- e.g. `^[\-+]?[0-9]*(\.|\,)?[0-9]+$`

  General form of decimal number e.g. 9.12 or -9,8 (Europe)