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

$$A = \varepsilon \mid 0 + A \mid A + 1$$

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

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

We go by structural induction on A. Let A ∈ A be arbitrary.
Suppose len(no1(x)) = #0(x) is true for some x ∈ 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)

List = [ ] | a :: L

We'll assume a is an integer.

len([]) = 0
len(x :: L) = 1 + len(L)

[1, 2, 3]   1 :: (2 :: (3 :: []))

Write a function

len : List → Int

that computes the length of a list.

append(1 :: [], 2) = 1 :: 2 :: []

Finish the function

append : (List, Int) → List

append([], i) = i :: [] ⇝ [i]

append(a :: L, i) = a :: append(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**,

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

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

Let L be an arb. list and i be an arb. int. We go by structural induction on L.

Case L = []:
len(append([], i)) = len(i :: []) by def of app.
= 1 + len([]) by def of len
= 1 + 0
= 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**,
then len(append(L, i)) = 1 + len(L).

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]**

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 + 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))
= 1 + len(append(L', i)) by IH
= 1 + (1 + len(L')) by len.
= 1 + len($x :: L'$)

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 = $x :: L'$**:**

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

$0^* = \{\varepsilon, 0, 00, 000, \dots\}$

Regular expressions over Σ

- **Basis:**
  - ∅, ε are regular expressions
  - *a* is a regular expression for any $a \in \Sigma$

0^* zero or more occ. of the thing before

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

    (A ∪ B)

    (AB)

    A*

$\emptyset = \{\}$

$\varepsilon = \{""\}$

REGEX = ∅ | ε | *a* | **REGEX ∪ 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***
- **0*1***

Examples

001*

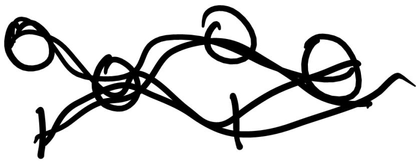
{00, 001, 0011, 00111, ...}

0*1*

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

Examples

$(0 \cup 1)0(0 \cup 1)0$



$(0^*1^*)^*$

$\{0,1\}^*$

$(0^*1^*)^*$

$\varepsilon \cup (0^*1^*) \cup (0^*1^*0^*1^*) \cup \ldots$

Examples

$(0 \cup 1)0(0 \cup 1)0$

{0000, 0010, 1000, 1010}

$(0^*1^*)^*$

All binary strings

Examples

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

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

Examples

$(0 \cup 1)^*0110(0 \cup 1)^*$

Strings that contain “0110”

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

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)