

**CSE
31F**

Foundations of Computing I

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

Recursively Defined Programs (on Lists)

List = [] | **a :: L**

We'll assume a is an integer.

len : List \rightarrow Int

a :: [] \approx [a]

len([]) = 0

len(a :: L) = 1 + len(L)

append : (List, Int) \rightarrow 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).

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

len : List \rightarrow Int

(1) len([]) = 0

(2) len(a :: L) = 1 + len(L)

append : (List, Int) \rightarrow List

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

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

Claim: For all lists L, and integers i,
~~then~~ len(append(L, i)) = 1 + len(L).

Case L = []: len(append([], i)) = len(i :: [])

= 1 + len([])

= 1 + 0

= 1 + len([])

def of app.

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

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)

$\text{len} : \text{List} \rightarrow \text{Int}$

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

$\text{len}(a :: L) = 1 + \text{len}(L)$

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

(1) $\text{append}([], i) = i :: []$

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

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 .

Suppose “ $\text{len}(\text{append}(L', i)) = \text{len}(L') + 1$ ” is true for some list L' .

CASE $L = x :: L'$:

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

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

$$= 1 + (\text{len}(L') + 1)$$

by IH

$$= \text{len}(x :: L') + 1 \quad \text{by len.}$$

Recursively Defined Programs (on Lists)

$\text{len} : \text{List} \rightarrow \text{Int}$

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

$\text{len}(a :: L) = 1 + \text{len}(L)$

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

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

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

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 .

Suppose “ $\text{len}(\text{append}(L', i)) = \text{len}(L') + 1$ ” is true for some list L' .

Case $L = x :: L'$:

Recursively Defined Programs (on Lists)

$\text{len} : \text{List} \rightarrow \text{Int}$

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

$\text{len}(a :: L) = 1 + \text{len}(L)$

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

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

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

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 .

Suppose “ $\text{len}(\text{append}(L', i)) = \text{len}(L') + 1$ ” is true for some list L' .

Case $L = x :: L'$:

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

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

$= 1 + (1 + \text{len}(L'))$ [By IH]

$= 1 + \text{len}(x :: L')$ [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 $L = []$:

$$\begin{aligned} \text{len}(\text{append}([], i)) &= \text{len}(i::[]) && \text{[Def of append]} \\ &= 1 + \text{len}([]) && \text{[Def of len]} \end{aligned}$$

Suppose “ $\text{len}(\text{append}(L', i)) = \text{len}(L') + 1$ ” is true for some list L' .

Case $L = x :: L'$:

$$\begin{aligned} \text{len}(\text{append}(x::L', i)) &= \text{len}(x::\text{append}(L', i)) && \text{[Def of append]} \\ &= 1 + \text{len}(\text{append}(L', i)) && \text{[Def of len]} \\ &= 1 + (1 + \text{len}(L')) && \text{[By IH]} \\ &= 1 + \text{len}(x::L') && \text{[Def of len]} \end{aligned}$$

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 \boxed{a} $a^* \varepsilon$

- Basis:**

\emptyset , ε are regular expressions

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

$$a^* = \{\varepsilon, a, aa, \dots\}$$

- Recursive step:**

– If **A** and **B** are regular expressions then so are:

(A U B)

$a \cup b$

(AB)

(adam) U (bob)

A*

(Adam)

~~((a U A)adam)*~~

REGEX = \emptyset | ε | a | REGEX U REGEX | REGEX REGEX | REGEX *

Each Regular Expression is a “pattern”

ϵ matches the **empty string**

a matches the one character string a

$(A \cup B)$ 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

$n(\)^{\#}$

001*

Y	N	
0011 00	0001	00 followed by n 1's $n \geq 0$

0*1*

Σ	327 100
----------	------------

Examples

001^*

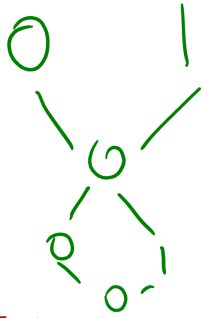
{00, 001, 0011, 00111, ...}

0^*1^*

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

Examples

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



$\{0000, 1010, 1000, 0010\}$

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

$\{2, 5\}$

Examples

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

{0000, 0010, 1000, 1010}

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

All binary strings

Examples

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

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

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"

10001

$(10)^*$

~~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 ∪ B)**

`a?` zero or one of a **(A ∪ 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)