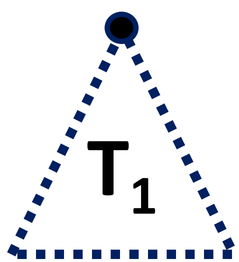
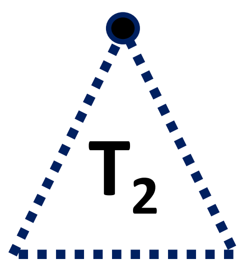
CSE 311: Foundations of Computing

Lecture 18: Strings and Regular Expressions

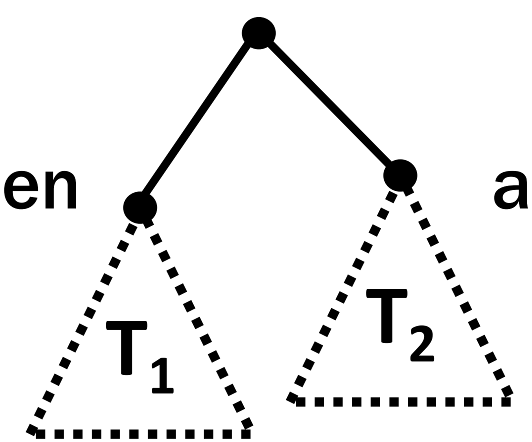
Last time: Rooted Binary Trees

- **Basis:** • is a rooted binary tree
- **Recursive step:**

If 

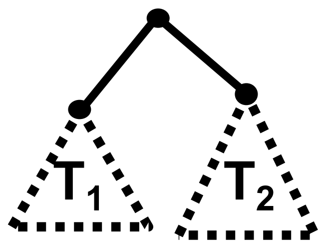
and 

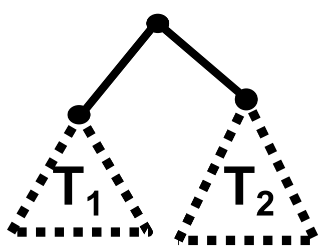
are rooted binary trees,

then 

also is a rooted binary tree.

Defining Functions on Rooted Binary Trees

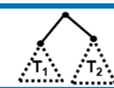
- size(•) := 1
- size (

) := 1 + size($\mathbf{T_1}$) + size($\mathbf{T_2}$)
- height(•) := 0
- height (

) := 1 + max{height($\mathbf{T_1}$), height($\mathbf{T_2}$)}

Claim: **For every rooted binary tree T,** $\text{size}(\mathbf{T}) \le 2^{\text{height}(\mathbf{T})+1} - 1$

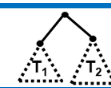
1. **Let** $P(\mathbf{T})$ **be** "$\text{size}(T) \le 2^{\text{height}(T)+1}-1$"**. We prove** $P(\mathbf{T})$ **for all rooted binary trees T by structural induction.**
2. **Base Case:** $\text{size}(\bullet)=1$, $\text{height}(\bullet)=0$, **and** $2^{0+1}-1=2^1-1=1$ **so** $P(\bullet)$ **is true.**
3. **Inductive Hypothesis: Suppose that** $P(\mathbf{T_1})$ **and** $P(\mathbf{T_2})$ **are true for some rooted binary trees** $\mathbf{T_1}$ **and** $\mathbf{T_2}$**, i.e.,** $\text{size}(\mathbf{T}_k) \le 2^{\text{height}(\mathbf{T}_k)+1} - 1$ **for** $k=1,2$
4. **Inductive Step:**

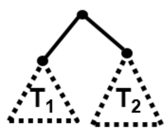
Goal: Prove P().

Claim: For every rooted binary tree T, size(T) $\le 2^{\text{height}(\mathbf{T})+1} - 1$

---

1. Let P(T) be "size(T) $\le 2^{\text{height}(T)+1} - 1$". We prove P(T) for all rooted binary trees T by structural induction.
2. **Base Case:** size(•)=1, height(•)=0, and $2^{0+1}-1=2^1-1=1$ so P(•) is true.
3. **Inductive Hypothesis:** Suppose that P(T_1) and P(T_2) are true for some rooted binary trees T_1 and T_2, i.e., size(T_k) $\le 2^{\text{height}(T_k)+1} - 1$ for k=1,2
4. **Inductive Step:**

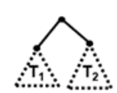
Goal: Prove P( **).**

size() $= s(T_1) + s(T_2) + 1$ by def of size

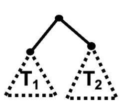
$\le 2^{h(T_1)+1} - 1 + 2^{h(T_2)+1} - 1 + 1$ by IH twice

$= 2^{h(T_1)+1} + 2^{h(T_2)+1} - 1$ algebra

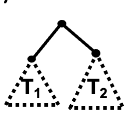
$2^{\max(h(T_1), h(T_2))+1+1} - 1$

= $2^{\text{height}($  $)+1} - 1$ by def of h

size(•) ::= 1

size () ::= 1 + size(T_1) + size(T_2)

height(•) ::= 0

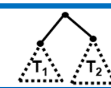
height () ::= 1 + max{height(T_1), height(T_2)}

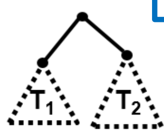
Claim: For every rooted binary tree T, size(T) $\leq 2^{\text{height}(T)+1} - 1$

1. Let P(T) be "size(T) $\leq 2^{\text{height}(T)+1}-1$". We prove P(T) for all rooted binary trees T by structural induction.

2. Base Case: size(•)=1, height(•)=0, and $2^{0+1}-1=2^1-1=1$ so P(•) is true.

3. Inductive Hypothesis: Suppose that P(T_1) and P(T_2) are true for some rooted binary trees T_1 and T_2, i.e., size(T_k) $\leq 2^{\text{height}(T_k)+1} - 1$ for k=1,2

4. Inductive Step: Goal: Prove P().

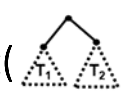
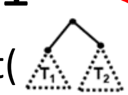
By def, size() $=1+\text{size}(T_1)+\text{size}(T_2)$

$\leq 1+2^{\text{height}(T_1)+1}-1+2^{\text{height}(T_2)+1}-1$ by IH for T_1 and T_2

$= 2^{\text{height}(T_1)+1}+2^{\text{height}(T_2)+1}-1$

$\leq 2\cdot\max(2^{\text{height}(T_1)+1},2^{\text{height}(T_2)+1})-1$

$= 2(2^{\max(\text{height}(T_1),\text{height}(T_2))+1})-1$

$= 2(2^{\text{height}(}$  $^{)}) - 1 = 2^{\text{height}(}$  $^{)+1} - 1$

which is what we wanted to show.

$\max(2^a, 2^b) = 2^{\max(a,b)}$

5. So, the P(T) is true for all rooted binary trees by structural induction.

Strings

ε

- An *alphabet* Σ is any finite set of characters

- The set Σ^* of *strings* over the alphabet Σ
  - example: {0,1}* is the set of *binary strings*

    0, 1, 00, 01, 10, 11, 000, 001, … and “”

- Σ^* is defined recursively by
  - **Basis:** $\varepsilon \in \Sigma^*$ (ε is the empty string, i.e., “”)
  - **Recursive:** if $w \in \Sigma^*$, $a \in \Sigma$, then $wa \in \Sigma^*$

Palindromes

010 $\Sigma = \{0,1\}$ 0110

Palindromes are strings that are the same when read backwards and forwards

1 ε

Basis:

ε is a palindrome

any $a \in \Sigma$ is a palindrome

010 11

0110

Recursive step:

If p is a palindrome,

then apa is a palindrome for every $a \in \Sigma$

Functions on Recursively Defined Sets (on Σ^*)

Length:

$len(\varepsilon) := 0$

$len(wa) := len(w) + 1$ for $w \in \Sigma^*$, $a \in \Sigma$

Concatenation:

$x \bullet \varepsilon := x$ for $x \in \Sigma^*$

$x \bullet wa := (x \bullet w)a$ for $x \in \Sigma^*$, $a \in \Sigma$

Concat(l_1, l_2)

Concat(nil, l) := l

Concat($a::l$, l) :=
$a::$ concat(l, l)

Reversal: w^R

$\varepsilon^R := \varepsilon$

$(wa)^R := a \bullet w^R$ for $w \in \Sigma^*$, $a \in \Sigma$

Number of c's in a string:

$\#_c(\varepsilon) := 0$

$\#_c(wc) := \#_c(w) + 1$ for $w \in \Sigma^*$

$\#_c(wa) := \#_c(w)$ for $w \in \Sigma^*$, $a \in \Sigma$, $a \neq c$

separate cases for c vs $a \neq c$

$(01)^R = 1 \bullet 0^R$

$= 1 \bullet 0 \bullet \varepsilon^R$

$= 1 \bullet 0 \bullet \varepsilon = 10$

Claim: $\text{len}(x \bullet y) = \text{len}(x) + \text{len}(y)$ **for all** $x, y \in \Sigma^*$

Let $P(y)$ **be** "$\text{len}(x \bullet y) = \text{len}(x) + \text{len}(y)$ for all $x \in \Sigma^*$" .
We prove $P(y)$ **for all** $y \in \Sigma^*$ **by structural induction.**

Base Case ($y = \varepsilon$): **Let** $x \in \Sigma^*$ **be arbitrary. Then,** $\text{len}(x \bullet \varepsilon) = \text{len}(x) = \text{len}(x) + \text{len}(\varepsilon)$ **since** $\text{len}(\varepsilon)=0$. **Since** x **was arbitrary,** $P(\varepsilon)$ **holds.**

Inductive Hypothesis: Assume that $P(w)$ **is true for some arbitrary** $w \in \Sigma^*$**, i.e.,** $\text{len}(x \bullet w) = \text{len}(x) + \text{len}(w)$ for all x

Claim: $\text{len}(x \bullet y) = \text{len}(x) + \text{len}(y)$ for all $x,y \in \Sigma^*$

Does this look familiar?

Let $P(y)$ **be** "$\text{len}(x \bullet y) = \text{len}(x) + \text{len}(y)$ for all $x \in$
We prove $P(y)$ **for all** $y \in \Sigma^*$ **by structural indu**

Base Case ($y = \varepsilon$): **Let** $x \in \Sigma^*$ **be arbitrary. Then,** $\text{len}(x \bullet \varepsilon) = \text{len}(x) = \text{len}(x) + \text{len}(\varepsilon)$ **since** $\text{len}(\varepsilon)=0$. **Since** x **was arbitrary,** $P(\varepsilon)$ **holds.**

Inductive Hypothesis: Assume that $P(w)$ **is true for some arbitrary** $w \in \Sigma^*$, **i.e.,** $\text{len}(x \bullet w) = \text{len}(x) + \text{len}(w)$ for all x

Inductive Step: **Goal: Show that** $P(wa)$ **is true for every** $a \in \Sigma$

Let $a \in \Sigma$ **and** $x \in \Sigma^*$. **Then**

$$\begin{aligned} \text{len}(x \bullet wa) &= \text{len}((x \bullet w)a) && \textbf{by def of } \bullet \\ &= \text{len}(x \bullet w)+1 && \textbf{by def of len} \\ &= \text{len}(x)+\text{len}(w)+1 && \textbf{by I.H.} \\ &= \text{len}(x)+\text{len}(wa) && \textbf{by def of len} \end{aligned}$$

Therefore, $\text{len}(x \bullet wa)= \text{len}(x)+\text{len}(wa)$ **for all** $x \in \Sigma^*$, **so** $P(wa)$ **is true.**

So, by induction $\text{len}(x \bullet y) = \text{len}(x) + \text{len}(y)$ **for all** $x,y \in \Sigma^*$

$\Sigma^* =$ lists of letters

$\mathcal{P}(\Sigma) =$ sets of letters

$\Sigma = \{0, 1\}$ 000000

$\{\emptyset, \{0\}, \{1\}, \{0,1\}\}$

Theoretical Computer Science

Languages: Sets of Strings

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

Foreword on Intro to Theory C.S.

- Look at different ways of defining languages
- See which are more expressive than others
  - i.e., which can define more languages

- Later: connect ways of defining languages to different types of (restricted) computers
  - computers capable of recognizing those languages i.e., distinguishing strings in the language from not

- Consequence: computers that recognize more expressive languages are more powerful

Regular Expressions

Regular expressions over Σ

- **Basis:**

  ε is a regular expression (could also include $\varnothing$)

  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*

Each Regular Expression is a "pattern"

$\Sigma = \{0,1\}$

0 matches $\{0\}$

ε matches only the **empty string**

a matches only 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 ($\varepsilon \cup A \cup AA \cup AAA \cup \ldots$)

Definition of the *language* matched by a regular expression

Examples

$\Sigma = \{0, 1\}$

001*

001   0011111111

00

$(001)^*$

ε 001

00100001   001...

0*1*

ε ✓

000

111

001111

Examples

001*

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

0*1*

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

Examples

$\Sigma = \{0, 1\}$

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

0 0 0 0

1 0 1 0

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

$(0 \cup 1)^*$

Examples

$(\boldsymbol{0} \cup \boldsymbol{1})\, \boldsymbol{0}\, (\boldsymbol{0} \cup \boldsymbol{1})\, \boldsymbol{0}$

{0000, 0010, 1000, 1010}

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

All binary strings

Examples

- All binary strings that contain 0110

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

Examples

- All binary strings that contain 0110

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

Examples

$(0 \cup 1)^* 0110$

- All binary strings that contain 0110

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

- All binary strings that begin with a string of doubled characters (00 or 11) followed by 01010 or 10001

immediately

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

followed by anything

Examples

- All binary strings that contain 0110

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

- All binary strings that begin with a string of doubled characters (00 or 11) followed by 01010 or 10001

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

Regular Expressions in Practice

- Used to define the *tokens* of a programming language
  - legal variable names, keywords, etc.

- Used in `grep`, a program that does pattern matching searches in UNIX/LINUX

- 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();
```
//true

[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)