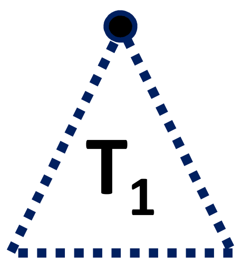
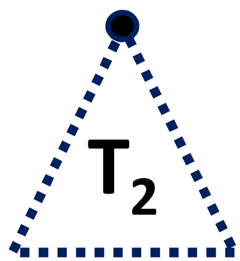
CSE 311: Foundations of Computing

Lecture 18: Strings and Regular Expressions

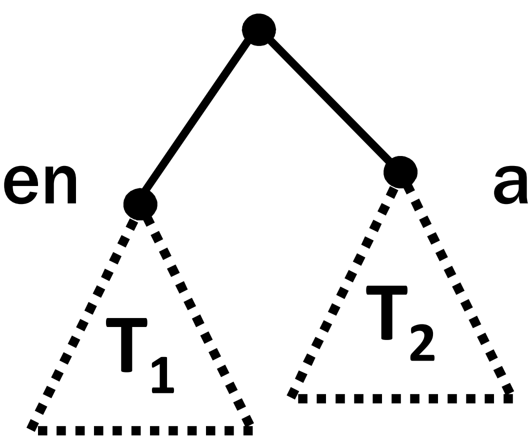
Last time: Rooted Binary Trees

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

If 

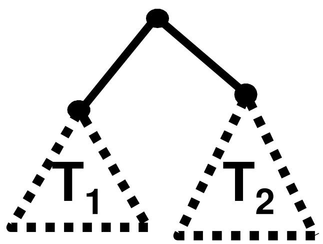
and 

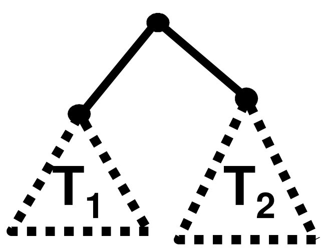
are rooted binary trees,

then 

also is a rooted binary tree.

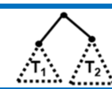
Defining Functions on Rooted Binary Trees

- size(•) := 1
- size (

) := 1 + size(**T**$_1$) + size(**T**$_2$)
- height(•) := 0
- height (

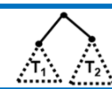
) := 1 + max{height(**T**$_1$), height(**T**$_2$)}

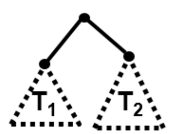
Claim: For every rooted binary tree T, size(T) ≤ $2^{\text{height}(\mathbf{T})+1} - 1$

1. Let P(**T**) be "size(T) ≤ $2^{\text{height}(T)+1}$–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($\mathbf{T_1}$) and P($\mathbf{T_2}$) are true for some rooted binary trees T_1 and T_2, i.e., size($\mathbf{T}_k$) ≤ $2^{\text{height}(\mathbf{T}_k)+1}$ – 1 for k=1,2
4. **Inductive Step:** **Goal: Prove** P(

).

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

1. Let P(T) be "size(T) ≤ $2^{\text{height}(T)+1}-1$". 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) ≤ $2^{\text{height}(T_k)+1} - 1$ for k=1,2
4. Inductive Step: **Goal: Prove P().**

size() = 1 + size(T_1) + size(T_2)   by def'n of size

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

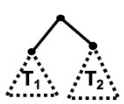
≤ 2 · max($2^{\text{height}(T_1)+1}$, $2^{\text{height}(T_2)+1}$) − 1

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

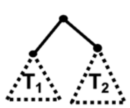
= $2^1 \cdot 2^{\text{height}(}$  $)$ − 1

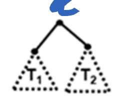
max(2^{a+1}, 2^{b+1}) = $2^{\max(a,b)+1}$

size(•) ::= 1

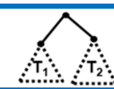
size () ::= 1 + size(T_1) + size(T_2)

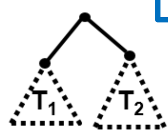
height(•) ::= 0

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

≤ $2^{\text{height}(}$  $)^{+1}$ – 1

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

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($\mathbf{T_1}$) **and** P($\mathbf{T_2}$) **are true for some rooted binary trees** $\mathbf{T_1}$ **and** $\mathbf{T_2}$**, i.e.,** size($\mathbf{T_k}$) $\leq 2^{\text{height}(\mathbf{T_k})+1} - 1$ **for** k=1,2
4. **Inductive Step:** **Goal: Prove** P().

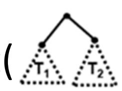
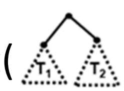
By def, size() $=1+\text{size}(\mathbf{T_1})+\text{size}(\mathbf{T_2})$

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

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

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

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

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

which is what we wanted to show.

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

Strings

- 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

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

Basis:

ε is a palindrome

any $a \in \Sigma$ is a palindrome

Recursive step:

If p is a palindrome,

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

eg arotora

level

deed

ε, a for a ∈ Σ

Σ = {0,1}

0

101

01010

0010100

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 $x \in \Sigma^*$, $a \in \Sigma$

Reversal:

$\varepsilon^R := \varepsilon$

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

Number of c's in a string: $c \in \Sigma$

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

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

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

separate cases for
c vs $a \neq c$

Claim: len(x•y) = len(x) + len(y) **for all** x,y $\in \Sigma^*$

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

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

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

Claim: len(x•y) = len(x) + len(y) for all $x,y \in \Sigma^*$

Let P(y) **be** "len(x•y) = len(x) + len(y) for all x ∈
We prove P(y) **for all** $y \in \Sigma^*$ **by structural induc**

Does this look familiar?

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

Inductive Hypothesis: Assume that P(w) **is true for some arbitrary** $w \in \Sigma^*$**, i.e.,** len(x•w) = len(x) + 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** len(x•wa) = len((x•w)a) **by def of •**
= len(x•w)+1 **by def of len**
= len(x)+len(w)+1 **by I.H.**
= len(x)+len(wa) **by def of len**

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

So, by induction len(x•y) = len(x) + len(y) **for all** $x,y \in \Sigma^*$

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"

- **ε** matches only the **empty string**
- ***a*** matches only 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 (**ε ∪ A ∪ AA ∪ AAA ∪ …**)

Definition of the *language* matched by a regular expression

Examples

xyz² — * bridge closest

001*

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

0*1*

ε, 0011, 0111, 111, 0

~~010~~

all 0's before all 1's

Examples

001*

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

0*1*

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

Examples

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

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

all possible binary strings

Examples

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

{0000, 0010, 1000, 1010}

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

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

- 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

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

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();
```

[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)

Examples

- All binary strings that have an even # of 1's

Examples

- All binary strings that have an even # of 1's

  e.g., *0* (10*10*)**

Examples

- All binary strings that have an even # of 1's

  e.g., $0^* (10^*10^*)^*$

- All binary strings that *don't* contain 101

Examples

- All binary strings that have an even # of 1's

  e.g., $0^* (10^*10^*)^*$

- All binary strings that *don't* contain 101

  e.g., $0^* (1 \cup 000^*)^* 0^*$

  at least two 0s between 1s

Limitations of Regular Expressions

- Not all languages can be specified by regular expressions
- Even some easy things like
  - Palindromes
  - Strings with equal number of 0's and 1's
- But also more complicated structures in programming languages
  - Matched parentheses
  - Properly formed arithmetic expressions
  - etc.

Context-Free Grammars

- A Context-Free Grammar (CFG) is given by a finite set of substitution rules involving
  - A finite set **V** of *variables* that can be replaced
  - Alphabet Σ of *terminal symbols* that can't be replaced
  - One variable, usually **S**, is called the *start symbol*

- The substitution rules involving a variable **A**, written

$$A \rightarrow w_1 \mid w_2 \mid \cdots \mid w_k$$

  where each w_i is a string of variables and terminals
  – that is, $w_i \in (V \cup \Sigma)^*$

How CFGs generate strings

- Begin with start symbol **S**
- If there is some variable **A** in the current string you can replace it by one of the w's in the rules for **A**
  - $A \rightarrow w_1 \mid w_2 \mid \cdots \mid w_k$
  - Write this as $xAy \Rightarrow xwy$
  - Repeat until no variables left
- The set of strings the CFG describes are all strings, containing no variables, that can be *generated* in this manner (after a finite number of steps)

Example Context-Free Grammars

Example: $S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

Example Context-Free Grammars

Example: $S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes

Example Context-Free Grammars

Example: $S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes

Example: $S \rightarrow 0S \mid S1 \mid \varepsilon$

Example Context-Free Grammars

Example: $S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes

Example: $S \to 0S \mid S1 \mid \varepsilon$

*0*1**