CSE 311: Foundations of Computing

Topic 9: Languages

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Theoretical Computer Science

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 $wa \in \Sigma^*$

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

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 $a \in \Sigma$

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 $\cup$ **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 (**ε** $\cup$ **A** $\cup$ **AA** $\cup$ **AAA** $\cup$ **…**)

Definition of the *language* matched by a regular expression

Language of a Regular Expression

The language defined by a regular expression:

$$\mathrm{L}(\varepsilon) = \{\varepsilon\}$$

$$\mathrm{L}(a) = \{a\}$$

$$\mathrm{L}(A \cup B) = L(A) \cup L(B)$$

$$\mathrm{L}(AB) = \{x \ : \ \exists y \in L(A), \exists z \in L(B)\ (x = y \bullet z)\}$$

$$\mathrm{L}(A^*) = \bigcup_{n=0}^{\infty} L(A^n)$$

A^n defined recursively by

$$A^0 = \emptyset$$

$$A^{n+1} = A^n A$$

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

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

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

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

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

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

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

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., $0^*(1 \cup 1000^*)^*(\varepsilon \cup 10)$

  at least two 0s between 1s

Finite languages vs Regular Expressions

- All finite languages have a regular expression.

  (a language is finite if its elements can be put into a list)

  Why?

- Given a list of strings $s_1, s_2, \ldots, s_n$

  Construct the regular expression

$$s_1 \cup s_2 \cup \ldots \cup s_n$$

(Could make this formal by induction on n)

Finite languages vs Regular Expressions

- Every regular expression that does not use * generates a finite language.

  Why?

- Prove by structural induction on the syntax of regular expressions!

Star-free implies finite

Let A be a regular expression that does not use *. Then L(A) is finite.

Proof: We proceed by structural induction on A.

Case ε: L(ε) = {ε}, which is finite

Case *a*: L(*a*) = {*a*}, which is finite

Case A ∪ B:

L(A ∪ B) = L(A) ∪ L(B)

By the IH, each is finite, so their union is finite.

Star-free implies finite

Let A be a regular expression that does not use *. Then L(A) is finite.

Proof: We proceed by structural induction on A.

Case AB:

$$\mathrm{L(AB)} = \{x \;:\; \exists y \in L(A), \exists z \in L(B)\ (x = y \bullet z)\}$$

By the IH, $\mathrm{L(A)}$ and $\mathrm{L(B)}$ are finite.

Every element of $\mathrm{L(AB)}$ is covered by a pair (y, z) where $y \in L(A)$ and $z \in L(B)$, so $\mathrm{L(AB)}$ is finite.

(No case for A*!)

Finite languages vs Regular Expressions

Key takeaways:

- Regular expressions can represent all finite languages
- To prove a language is represented by a regular expression, just describe the regular expression.
- Regular expressions are more powerful than finite languages (e.g., 0* is an infinite language)
- To prove something about *all* regular expressions, use structural induction on the syntax.

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	($\mathbf{A} \cup \mathbf{B}$)
a**?**	zero or one of a	($\mathbf{A} \cup \varepsilon$)
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)

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 as

$$\mathbf{A} \rightarrow w_1 \mid w_2 \mid \cdots \mid w_k$$

where each w_i is a string of variables and terminals

– that is $w_i \in (\mathbf{V} \cup \Sigma)^*$

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**
  - $\mathbf{A} \rightarrow w_1 \mid w_2 \mid \cdots \mid w_k$
  - Write this as $x\mathbf{A}y \Rightarrow xwy$
  - 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: $\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

Example Context-Free Grammars

Example: $\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes

Example Context-Free Grammars

Example: $\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes

Example: $\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

Example Context-Free Grammars

Example: $\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes

Example: $\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

0^*1^*

Example Context-Free Grammars

Grammar for $\{0^n 1^n : n \geq 0\}$

(i.e., matching `0*1*` but with same number of 0's and 1's)

Example Context-Free Grammars

Grammar for $\{0^n 1^n : n \geq 0\}$

(i.e., matching `0*1*` but with same number of 0's and 1's)

$$\mathbf{S} \rightarrow 0\mathbf{S}1 \mid \varepsilon$$

Example Context-Free Grammars

Grammar for $\{0^n 1^n : n \geq 0\}$

(i.e., matching `0*1*` but with same number of 0's and 1's)

$$\mathbf{S} \rightarrow 0\mathbf{S}1 \mid \varepsilon$$

Grammar for $\{0^n 1^{2n} : n \geq 0\}$

Example Context-Free Grammars

Grammar for $\{0^n 1^n : n \geq 0\}$

(i.e., matching `0*1*` but with same number of 0's and 1's)

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

Grammar for $\{0^n 1^{2n} : n \geq 0\}$

$$S \rightarrow 0S11 \mid \varepsilon$$

Example Context-Free Grammars

Grammar for $\{0^n 1^n : n \geq 0\}$

(i.e., matching 0*1* but with same number of 0's and 1's)

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

Grammar for $\{0^n 1^{n+1} 0 : n \geq 0\}$

Example Context-Free Grammars

Grammar for $\{0^n 1^n : n \geq 0\}$

(i.e., matching 0*1* but with same number of 0's and 1's)

$$\mathbf{S} \to 0\mathbf{S}1 \mid \varepsilon$$

Grammar for $\{0^n 1^{n+1} 0 : n \geq 0\}$

$$\mathbf{S} \to \mathrm{A}10$$
$$\mathbf{A} \to 0\mathbf{A}1 \mid \varepsilon$$

Example Context-Free Grammars

Example: $\mathbf{S} \rightarrow \mathbf{(S)} \mid \mathbf{SS} \mid \varepsilon$

Example Context-Free Grammars

Example: $\mathbf{S} \rightarrow \mathbf{(S)} \mid \mathbf{SS} \mid \varepsilon$

The set of all strings of matched parentheses

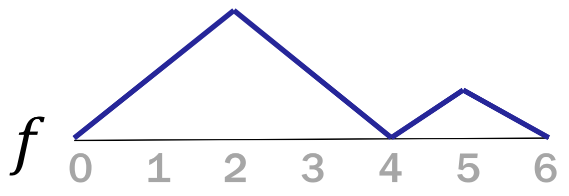
Example Context-Free Grammars

Example: $S \to (S) \mid SS \mid \varepsilon$

The set of all strings of matched parentheses

Suppose S generates x. Define $f(k)$ to be number of “(”s – “)”s in first k characters of x

E.g., for x = (())()

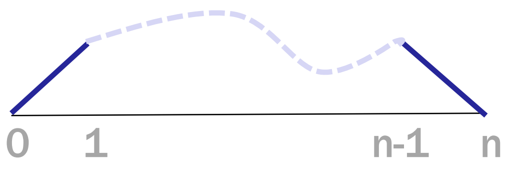


Example Context-Free Grammars

Three possibilities for $f(\mathrm{k})$ for $k \in \{1, \dots, n-1\}$

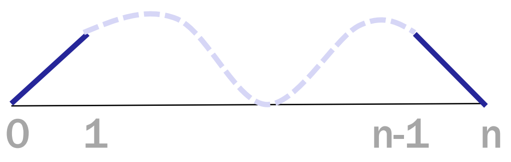
- $f(k) > 0$ for all such k

  S $\rightarrow$ (S)



- $f(k) = 0$ for some such k

  S $\rightarrow$ SS



Example Context-Free Grammars

Binary strings with equal numbers of 0s and 1s
(not just 0^n1^n, also 0101, 0110, etc.)

$$S \to SS \mid 0S1 \mid 1S0 \mid \varepsilon$$

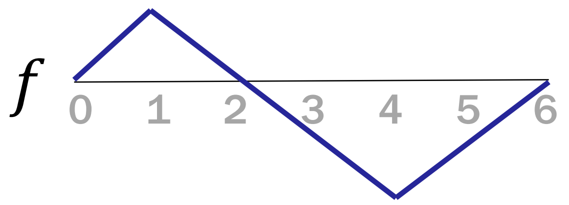
Example Context-Free Grammars

Binary strings with equal numbers of 0s and 1s
(not just 0^n1^n, also 0101, 0110, etc.)

$$S \rightarrow SS \mid 0S1 \mid 1S0 \mid \varepsilon$$

Suppose S generates x. Define $f(k)$ to be #0s – #1s in the first k characters of x.

E.g., for x = 011100



Example Context-Free Grammars

Binary strings with equal numbers of 0s and 1s
(not just 0^n1^n, also 0101, 0110, etc.)

$$S \rightarrow SS \mid 0S1 \mid 1S0 \mid \varepsilon$$

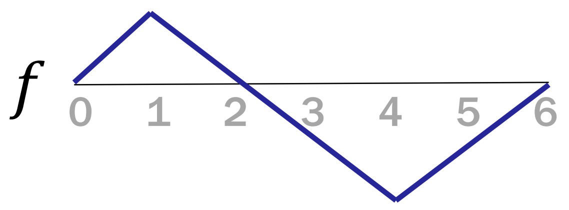
Suppose S generates x. Define $f(k)$ to be #0s – #1s in the first k characters of x.

If k-th character is 0, then $f(k) = f(k-1) + 1$
If k-th character is 1, then $f(k) = f(k-1) - 1$

Example Context-Free Grammars

Let $x \in (0 \cup 1)^*$. Define $f_x(k)$ to be the number 0s minus the number of 1s in the k characters of x.

E.g., for x = 011100



$f(k) = 0$ when first k characters have #0s = #1s

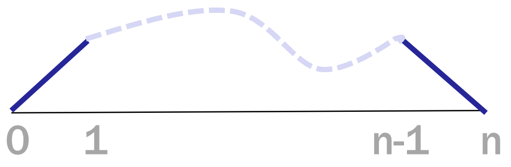
– starts out at 0 $\quad f(0) = 0$

– ends at 0 $\quad f(n) = 0$

Example Context-Free Grammars

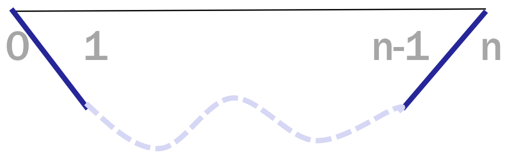
Three possibilities for $f(\mathrm{k})$ for $k \in \{1, \dots, n-1\}$

- $f(k) > 0$ for all such k



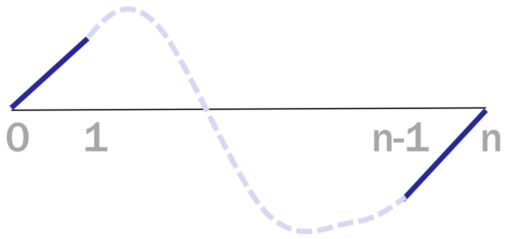
S $\rightarrow$ 0S1

- $f(k) < 0$ for all such k



S $\rightarrow$ 1S0

- $f(k) = 0$ for some such k



S $\rightarrow$ SS

Simple Arithmetic Expressions

$$E \rightarrow E+E \mid E*E \mid (E) \mid x \mid y \mid z \mid 0 \mid 1 \mid 2 \mid 3 \mid 4$$
$$\mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Generate $(2*x) + y$

Simple Arithmetic Expressions

$E \to E+E \mid E*E \mid (E) \mid x \mid y \mid z \mid 0 \mid 1 \mid 2 \mid 3 \mid 4$
$\mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Generate $(2*x) + y$

$E \Rightarrow E+E \Rightarrow (E)+E \Rightarrow (E*E)+E \Rightarrow (2*E)+E \Rightarrow (2*x)+E \Rightarrow (2*x)+y$

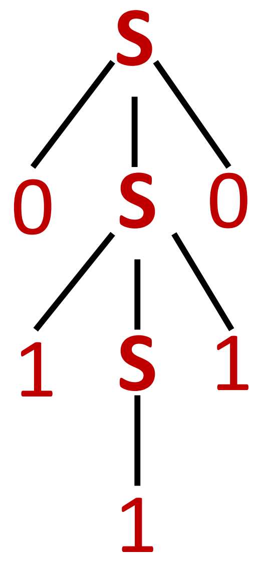
Parse Trees

Suppose that grammar G generates a string x

- A *parse tree* of x for G has
  - Root labeled S (start symbol of G)
  - The children of any node labeled A are labeled by symbols of w left-to-right for some rule $A \rightarrow w$
  - The symbols of x label the leaves ordered left-to-right

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

Parse tree of 01110



Two ways to Define Binary Palindromes

Recursively-Defined Set

Basis:

ε is a palindrome
any $a \in \{0, 1\}$ is a palindrome

Recursive step:

If p is a palindrome,
then apa is a palindrome for every $a \in \{0, 1\}$

Grammar $\quad$ $\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

CFGs and recursively-defined sets of strings

- A CFG with the start symbol **S** as its *only* variable recursively defines the set of strings of terminals that **S** can generate

- A CFG with more than one variable is a simultaneous recursive definition of the sets of strings generated by *each* of its variables
  - sometimes necessary to use more than one

CFGs and Regular Expressions

Theorem: For any set of strings (language) A described by a regular expression, there is a CFG that recognizes A.

Proof idea:

P(A) is “A is recognized by some CFG”

Structural induction based on the recursive definition of regular expressions...

Regular Expressions over Σ

- **Basis:**
  - ε is a regular expression
  - ***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*

CFGs are more general than REs

- CFG to match RE **ε**

  S $\rightarrow$ ε

- CFG to match RE ***a*** (for any $a \in \Sigma$)

  S $\rightarrow$ a

CFGs are more general than REs

Suppose CFG with start symbol $\mathbf{S_1}$ matches RE **A**
CFG with start symbol $\mathbf{S_2}$ matches RE **B**

- CFG to match RE **A** $\cup$ **B**

  $\mathbf{S} \rightarrow \mathbf{S_1} \mid \mathbf{S_2}$ + rules from original CFGs

- CFG to match RE **AB**

  $\mathbf{S} \rightarrow \mathbf{S}_1\, \mathbf{S}_2$ + rules from original CFGs

CFGs are more general than REs

Suppose CFG with start symbol $\mathbf{S_1}$ matches RE **A**

- CFG to match RE $\mathbf{A}^*$ (= $\varepsilon \cup \mathbf{A} \cup \mathbf{AA} \cup \mathbf{AAA} \cup \ldots$)

  $\mathbf{S} \rightarrow \mathbf{S_1}\,\mathbf{S} \mid \varepsilon$ + rules from CFG with $\mathbf{S_1}$