

# Context-Free Languages & Grammars (CFLs & CFGs)



---

Reading: Chapter 5



# Not all languages are regular

---

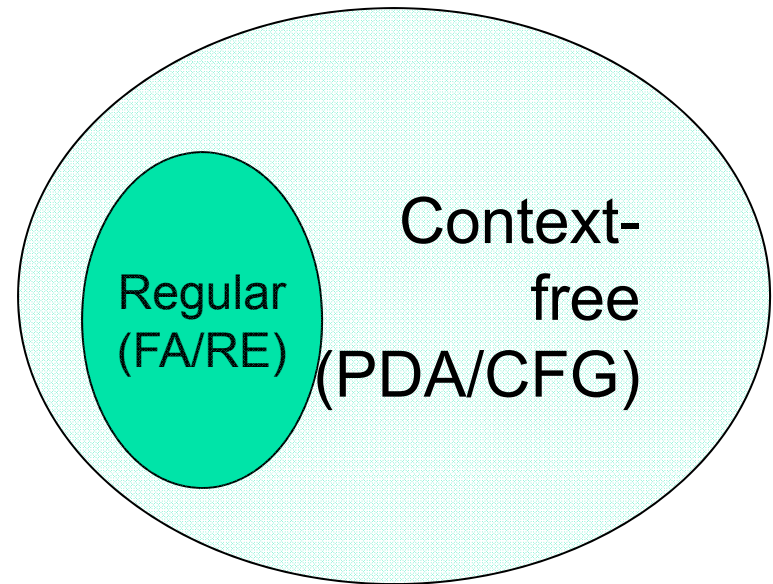
- So what happens to the languages which are not regular?
- Can we still come up with a language recognizer?
  - i.e., something that will accept (or reject) strings that belong (or do not belong) to the language?



# Context-Free Languages

---

- A language class larger than the class of regular languages
- Supports natural, recursive notation called “context-free grammar”
- Applications:
  - Parse trees, compilers
  - XML





# An Example

---

- A palindrome is a word that reads identical from both ends

- E.g.,  $\xrightarrow{\hspace{1cm}} \xleftarrow{\hspace{1cm}}$  madam,  $\xrightarrow{\hspace{1cm}} \xleftarrow{\hspace{1cm}}$  redivider,  $\xrightarrow{\hspace{1cm}} \xleftarrow{\hspace{1cm}}$  malayalam,  $\xrightarrow{\hspace{1cm}} \xleftarrow{\hspace{1cm}}$  010010010

- Let  $L = \{ w \mid w \text{ is a binary palindrome} \}$
- Is  $L$  regular?

- No.

- Proof:

- Let  $w = 0^N 1 0^N$  (assuming  $N$  to be the p/l constant)
    - By Pumping lemma,  $w$  can be rewritten as  $xyz$ , such that  $xy^kz$  is also  $L$  (for any  $k \geq 0$ )
    - But  $|xy| \leq N$  and  $y \neq \epsilon$
    - $\implies y = 0^+$
    - $\implies xy^kz$  will NOT be in  $L$  for  $k=0$
    - $\implies$  Contradiction

# But the language of palindromes...

is a CFL, because it supports recursive substitution (in the form of a CFG)

- This is because we can construct a “grammar” like this:

1.  $A \Rightarrow \varepsilon$
2.  $A \Rightarrow 0$
3.  $A \Rightarrow 1$
4.  $A \Rightarrow 0A0$
5.  $A \Rightarrow 1A1$

Terminal

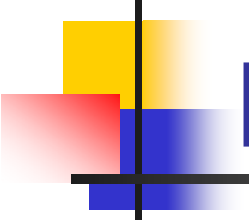
Variable or non-terminal

Same as:

$A \Rightarrow 0A0 \mid 1A1 \mid 0 \mid 1 \mid \varepsilon$

Productions

How does this grammar work?



# How does the CFG for palindromes work?

An input string belongs to the language (i.e., accepted) iff it can be generated by the CFG

G:

$A \Rightarrow 0A0 \mid 1A1 \mid 0 \mid 1 \mid \varepsilon$

- Example:  $w=01110$
- G can generate w as follows:

1.  $A \Rightarrow 0A0$
2.  $\Rightarrow 01A10$
3.  $\Rightarrow 01110$

## Generating a string from a grammar:

1. Pick and choose a sequence of productions that would allow us to generate the string.
2. At every step, substitute one variable with one of its productions.



# Context-Free Grammar: Definition

---

- A context-free grammar  $G=(V,T,P,S)$ , where:
  - $V$ : set of variables or non-terminals
  - $T$ : set of terminals (= alphabet  $\cup \{\varepsilon\}$ )
  - $P$ : set of *productions*, each of which is of the form
$$V \Rightarrow \alpha_1 \mid \alpha_2 \mid \dots$$
    - Where each  $\alpha_i$  is an arbitrary string of variables and terminals
  - $S \Rightarrow$  start variable

CFG for the language of binary palindromes:

$G=(\{A\},\{0,1\},P,A)$

$P: A \Rightarrow 0 A 0 \mid 1 A 1 \mid 0 \mid 1 \mid \varepsilon$



# More examples

---

- Parenthesis matching in code
- Syntax checking
- In scenarios where there is a general need for:
  - Matching a symbol with another symbol, or
  - Matching a count of one symbol with that of another symbol, or
  - Recursively substituting one symbol with a string of other symbols





## Example #2

---

- Language of balanced paranthesis  
e.g.,  $()(((((())((()))))((()))))\dots$
- CFG?

$\begin{array}{l} \underline{G}: \\ S \Rightarrow (S) \mid SS \mid \varepsilon \end{array}$
---

How would you “interpret” the string “ $(((((())((()))))((())))$ ” using this grammar?



## Example #3

---

- A grammar for  $L = \{0^m 1^n \mid m \geq n\}$

- CFG?

G:  
 $S \Rightarrow 0S1 \mid A$   
 $A \Rightarrow 0A \mid \varepsilon$

How would you interpret the string “00000111”  
using this grammar?



## Example #4

---

A program containing **if-then(-else)** statements

**if** *Condition* **then** *Statement* **else** *Statement*

(Or)

**if** *Condition* **then** *Statement*

CFG?



## More examples

---

- $L_1 = \{0^n \mid n \geq 0\}$
- $L_2 = \{0^n \mid n \geq 1\}$
- $L_3 = \{0^i 1^j 2^k \mid i=j \text{ or } j=k, \text{ where } i, j, k \geq 0\}$
- $L_4 = \{0^i 1^j 2^k \mid i=j \text{ or } i=k, \text{ where } i, j, k \geq 1\}$



# Applications of CFLs & CFGs

---

- Compilers use parsers for syntactic checking
- Parsers can be expressed as CFGs
  1. Balancing paranthesis:
    - $B \Rightarrow BB \mid (B) \mid \textit{Statement}$
    - $\textit{Statement} \Rightarrow \dots$
  2. If-then-else:
    - $S \Rightarrow SS \mid \textit{if Condition then Statement else Statement} \mid \textit{if Condition then Statement} \mid \textit{Statement}$
    - $\textit{Condition} \Rightarrow \dots$
    - $\textit{Statement} \Rightarrow \dots$
  3. C paranthesis matching  $\{ \dots \}$
  4. Pascal *begin-end* matching
  5. YACC (Yet Another Compiler-Compiler)



# More applications

---

- Markup languages

- Nested Tag Matching

- HTML

- `<html> ...<p> ... <a href=...> ... </a> </p> ... </html>`

- XML

- `<PC> ... <MODEL> ... </MODEL> .. <RAM> ...  
</RAM> ... </PC>`



# Tag-Markup Languages

Roll  $\Rightarrow$  **<ROLL>** Class Students **</ROLL>**

Class  $\Rightarrow$  **<CLASS>** Text **</CLASS>**

Text  $\Rightarrow$  Char Text | Char

Char  $\Rightarrow$  **a | b | ... | z | A | B | .. | Z**

Students  $\Rightarrow$  Student Students |  $\epsilon$

Student  $\Rightarrow$  **<STUD>** Text **</STUD>**

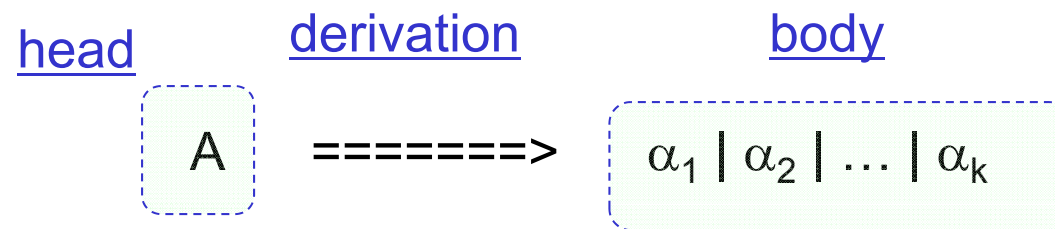
Here, the left hand side of each production denotes one non-terminals  
(e.g., “Roll”, “Class”, etc.)

Those symbols on the right hand side for which no productions (i.e.,  
substitutions) are defined are terminals (e.g., ‘a’, ‘b’, ‘|’, ‘<’, ‘>’, “ROLL”,  
etc.)



# Structure of a production

---



The above is same as:

1.  $A \Longrightarrow \alpha_1$
2.  $A \Longrightarrow \alpha_2$
3.  $A \Longrightarrow \alpha_3$
- ...
- K.  $A \Longrightarrow \alpha_k$





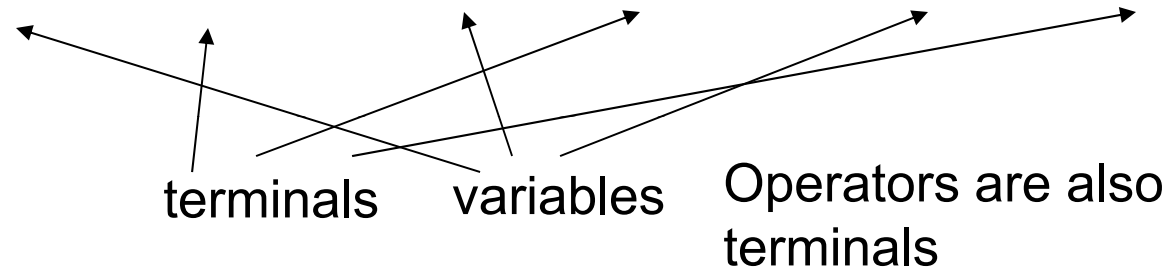
# CFG conventions

---

- Terminal symbols  $\leq a, b, c \dots$
- Non-terminal symbols  $\leq A, B, C, \dots$
- Terminal or non-terminal symbols  $\leq X, Y, Z$
- Terminal strings  $\leq w, x, y, z$
- Arbitrary strings of terminals and non-terminals  $\leq \alpha, \beta, \gamma, \dots$

# Syntactic Expressions in Programming Languages

*result = a \* b + score + 10 \* distance + c*



## Regular languages have only terminals

- Reg expression =  $[a-z][a-z0-1]^*$
- If we allow only letters a & b, and 0 & 1 for constants (for simplification)
  - Regular expression =  $(a+b)(a+b0+1)^*$

# String membership

How to say if a string belong to the language defined by a CFG?

1. Derivation
  - Head to body
2. Recursive inference
  - Body to head

Both are equivalent forms

Example:

- $w = 01110$
- Is  $w$  a palindrome?

G:  
 $A \Rightarrow 0A0 \mid 1A1 \mid 0 \mid 1 \mid \varepsilon$

$A \Rightarrow 0A0$   
 $\Rightarrow 01A10$   
 $\Rightarrow 01110$



# Simple Expressions...

---

- We can write a CFG for accepting simple expressions
- $G = (V, T, P, S)$ 
  - $V = \{E, F\}$
  - $T = \{0, 1, a, b, +, *, (, )\}$
  - $S = \{E\}$
  - $P$ :
    - $E \Rightarrow E + E \mid E * E \mid (E) \mid F$
    - $F \Rightarrow aF \mid bF \mid 0F \mid 1F \mid a \mid b \mid 0 \mid 1$



# Generalization of derivation

---

- Derivation is *head*  $\implies$  *body*
- $A \implies X$  (A derives X in a single step)
- $A \implies^*_G X$  (A derives X in a multiple steps)
- Transitivity:  
IF  $A \implies^*_G B$ , and  $B \implies^*_G C$ , THEN  $A \implies^*_G C$



# Context-Free Language

---

- The language of a CFG,  $G=(V,T,P,S)$ , denoted by  $L(G)$ , is the set of terminal strings that have a derivation from the start variable  $S$ .
  - $L(G) = \{ w \text{ in } T^* \mid S \Rightarrow^*_G w \}$



# Left-most & Right-most Derivation Styles

G:

$E \Rightarrow E + E \mid E * E \mid (E) \mid F$   
 $F \Rightarrow aF \mid bF \mid 0F \mid 1F \mid \varepsilon$

Derive the string  $a^*(ab+10)$  from G:

$E \xRightarrow{*}_G a^*(ab+10)$

Left-most  
derivation:

Always  
substitute  
leftmost  
variable

■ E  
 ■  $\Rightarrow E * E$   
 ■  $\Rightarrow F * E$   
 ■  $\Rightarrow aF * E$   
 ■  $\Rightarrow a * E$   
 ■  $\Rightarrow a * (E)$   
 ■  $\Rightarrow a * (E + E)$   
 ■  $\Rightarrow a * (F + E)$   
 ■  $\Rightarrow a * (aF + E)$   
 ■  $\Rightarrow a * (abF + E)$   
 ■  $\Rightarrow a * (ab + E)$   
 ■  $\Rightarrow a * (ab + F)$   
 ■  $\Rightarrow a * (ab + 1F)$   
 ■  $\Rightarrow a * (ab + 10F)$   
 ■  $\Rightarrow a * (ab + 10)$

■ E  
 ■  $\Rightarrow E * E$   
 ■  $\Rightarrow E * (E)$   
 ■  $\Rightarrow E * (E + E)$   
 ■  $\Rightarrow E * (E + F)$   
 ■  $\Rightarrow E * (E + 1F)$   
 ■  $\Rightarrow E * (E + 10F)$   
 ■  $\Rightarrow E * (E + 10)$   
 ■  $\Rightarrow E * (F + 10)$   
 ■  $\Rightarrow E * (aF + 10)$   
 ■  $\Rightarrow E * (abF + 0)$   
 ■  $\Rightarrow E * (ab + 10)$   
 ■  $\Rightarrow F * (ab + 10)$   
 ■  $\Rightarrow aF * (ab + 10)$   
 ■  $\Rightarrow a * (ab + 10)$

Right-most  
derivation:

Always  
substitute  
rightmost  
variable



# Leftmost vs. Rightmost derivations

---

Q1) For every leftmost derivation, there is a rightmost derivation, and vice versa. True or False?

True - will use parse trees to prove this

Q2) Does every word generated by a CFG have a leftmost and a rightmost derivation?

Yes – easy to prove (reverse direction)

Q3) Could there be words which have more than one leftmost (or rightmost) derivation?

Yes – depending on the grammar





# How to prove that your CFGs are correct?

---

(using induction)



# CFG & CFL

$$\frac{G_{\text{pal}}}{A \Rightarrow 0A0 \mid 1A1 \mid 0 \mid 1 \mid \varepsilon}$$

- Theorem: A string  $w$  in  $(0+1)^*$  is in  $L(G_{\text{pal}})$ , if and only if,  $w$  is a palindrome.
  
- Proof:
  - Use induction
    - on string length for the IF part
    - On length of derivation for the ONLY IF part



# Parse trees

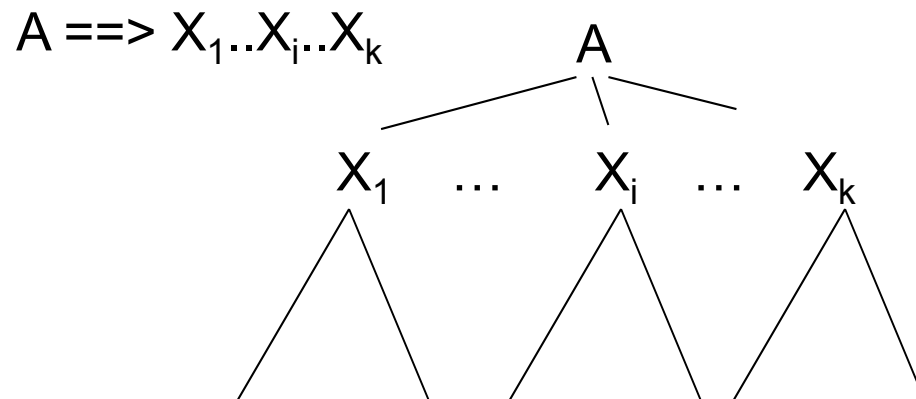
---

# Parse Trees

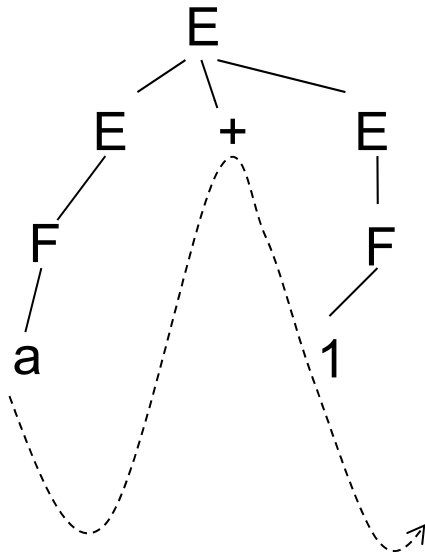
- Each CFG can be represented using a *parse tree*:
  - Each internal node is labeled by a variable in  $V$
  - Each leaf is terminal symbol
  - For a production,  $A \Rightarrow X_1 X_2 \dots X_k$ , then any internal node labeled  $A$  has  $k$  children which are labeled from  $X_1, X_2, \dots, X_k$  from left to right

---

Parse tree for production and all other subsequent productions:



# Examples



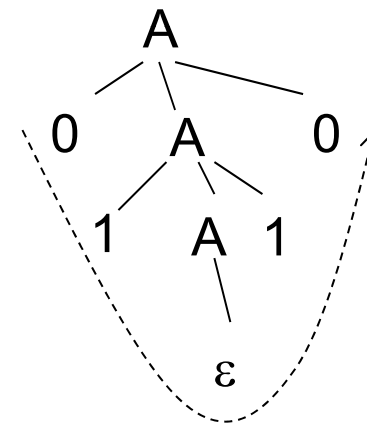
Parse tree for a + 1

G:

$E \Rightarrow E + E \mid E * E \mid (E) \mid F$

$F \Rightarrow aF \mid bF \mid 0F \mid 1F \mid 0 \mid 1 \mid a \mid b$

Recursive inference



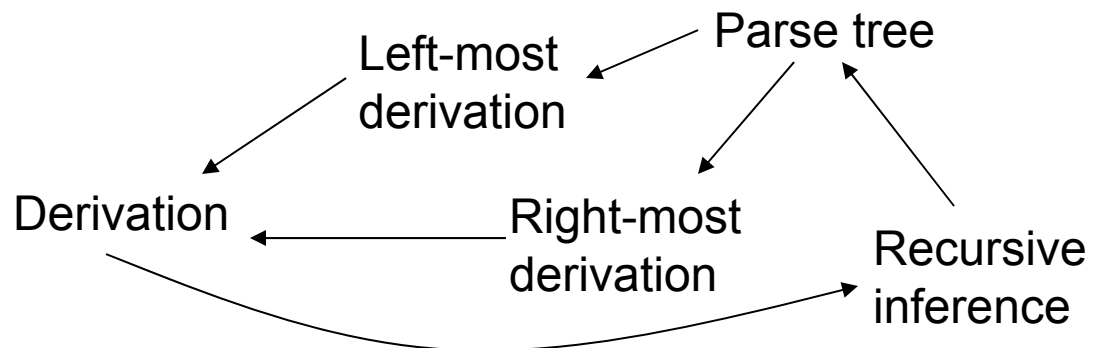
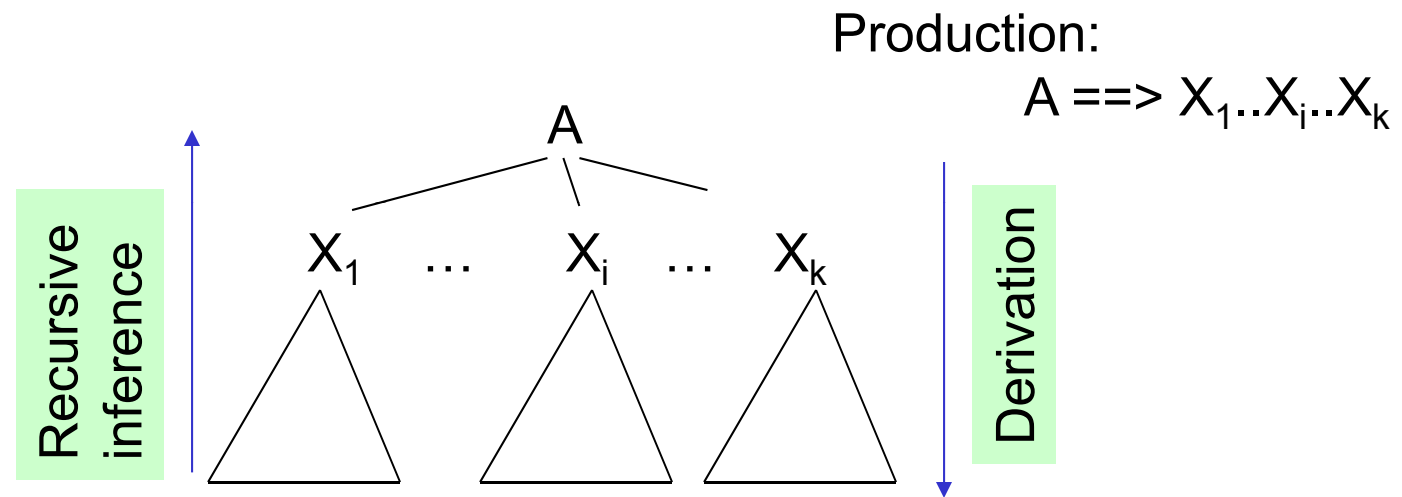
Parse tree for 0110

Derivation

G:

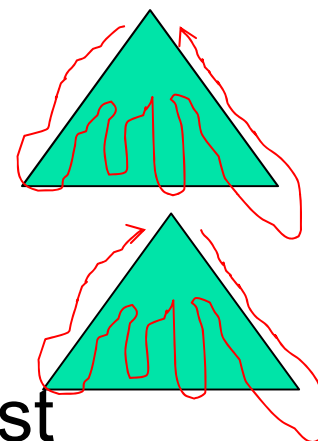
$A \Rightarrow 0A0 \mid 1A1 \mid 0 \mid 1 \mid \varepsilon$

# Parse Trees, Derivations, and Recursive Inferences

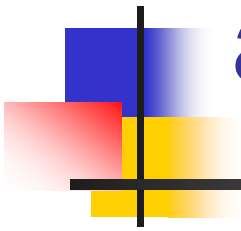


# Interchangeability of different CFG representations

- Parse tree  $\Rightarrow$  left-most derivation
  - DFS left to right
- Parse tree  $\Rightarrow$  right-most derivation
  - DFS right to left
- $\Rightarrow$  left-most derivation  $\equiv$  right-most derivation
- Derivation  $\Rightarrow$  Recursive inference
  - Reverse the order of productions
- Recursive inference  $\Rightarrow$  Parse trees
  - bottom-up traversal of parse tree



# Connection between CFLs and RLs





What kind of grammars result for regular languages?

# CFLs & Regular Languages

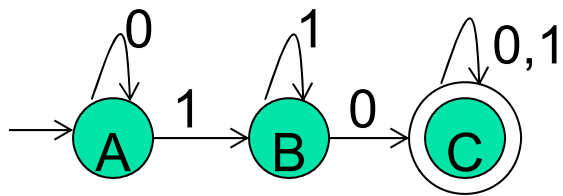
- A CFG is said to be *right-linear* if all the productions are one of the following two forms:  $A \Rightarrow wB$  (or)  $A \Rightarrow w$

Where:

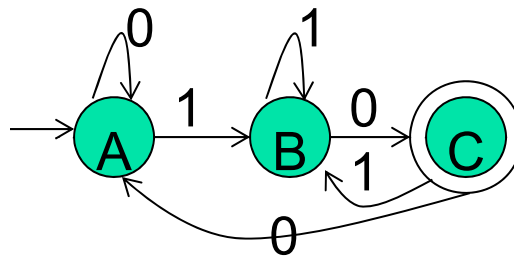
- A & B are variables,
- w is a string of terminals

- Theorem 1: Every right-linear CFG generates a regular language
- Theorem 2: Every regular language has a right-linear grammar
- Theorem 3: Left-linear CFGs also represent RLs

# Some Examples



Right linear CFG?



Right linear CFG?

➤  $A \Rightarrow 01B \mid C$   
 $B \Rightarrow 11B \mid 0C \mid 1A$   
 $C \Rightarrow 1A \mid 0 \mid 1$

Finite Automaton?



# Ambiguity in CFGs and CFLs

---

# Ambiguity in CFGs

- A CFG is said to be *ambiguous* if there exists a string which has more than one left-most derivation

Example:

$S \Rightarrow AS \mid \varepsilon$

$A \Rightarrow A1 \mid 0A1 \mid 01$

Input string: 00111

Can be derived in two ways

LM derivation #1:

$S \Rightarrow AS$

$\Rightarrow 0A1S$

$\Rightarrow 0A11S$

$\Rightarrow 00111S$

$\Rightarrow 00111$

LM derivation #2:

$S \Rightarrow AS$

$\Rightarrow A1S$

$\Rightarrow 0A11S$

$\Rightarrow 00111S$

$\Rightarrow 00111$

# Why does ambiguity matter?

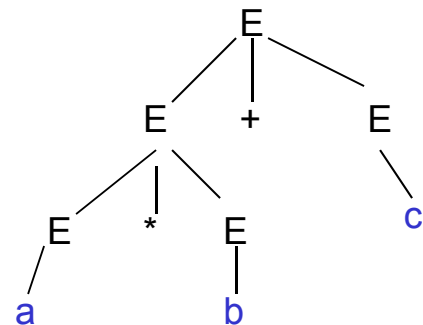
$E \Rightarrow E + E \mid E * E \mid (E) \mid a \mid b \mid c \mid 0 \mid 1$

Values are  
different !!!

*string* =  $a * b + c$

• LM derivation #1:

•  $E \Rightarrow E + E \Rightarrow E * E + E$   
 $\Rightarrow a * b + c$

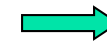
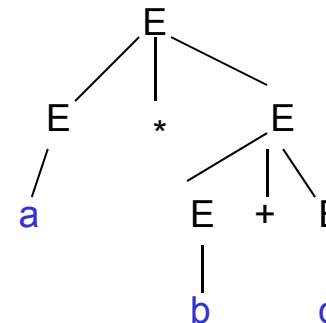
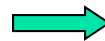


$(a*b)+c$

---

• LM derivation #2

•  $E \Rightarrow E * E \Rightarrow a * E \Rightarrow$   
 $a * E + E \Rightarrow a * b + c$



$a*(b+c)$

The calculated value depends on which  
of the two parse trees is actually used.



# Removing Ambiguity in Expression Evaluations

- It MAY be possible to remove ambiguity for some CFLs
  - E.g., in a CFG for expression evaluation by imposing rules & restrictions such as precedence
  - This would imply rewrite of the grammar

Modified unambiguous version:

- Precedence:  $()$ ,  $*$ ,  $+$

$$\begin{aligned} E &\Rightarrow E + T \mid T \\ T &\Rightarrow T * F \mid F \\ F &\Rightarrow I \mid (E) \\ I &\Rightarrow a \mid b \mid c \mid 0 \mid 1 \end{aligned}$$

Ambiguous version:

$$E \Rightarrow E + E \mid E * E \mid (E) \mid a \mid b \mid c \mid 0 \mid 1$$

How will this avoid ambiguity?



# Inherently Ambiguous CFLs

---

- However, for some languages, it may not be possible to remove ambiguity
- A CFL is said to be *inherently ambiguous* if every CFG that describes it is ambiguous

## Example:

- $L = \{ a^n b^n c^m d^m \mid n, m \geq 1 \} \cup \{ a^n b^m c^m d^n \mid n, m \geq 1 \}$
- L is inherently ambiguous
- Why?

Input string:  $a^n b^n c^n d^n$



# Summary

---

- Context-free grammars
- Context-free languages
- Productions, derivations, recursive inference, parse trees
- Left-most & right-most derivations
- Ambiguous grammars
- Removing ambiguity
- CFL/CFG applications
  - parsers, markup languages