Theory of Automata & Formal Languages

Context Free Languages
(Derivation, Parse Tree, Ambiguity)

Prepared By:-

Dinesh Sharma

Asstt. Professor, IT Deptt. KIET Group of Institutions Ghaziabad

Context-Free Grammar (CFG):

A context-free grammar (CFG) G is a 4-tuple

$G=(V, \Sigma, P, S)$ where

1. ***V*** is a finite set called the ***variables***.
2. ***Σ*** is a finite set, disjoint from *V*, called the ***terminals***.
3. ***S*** is a ***start symbol***.
4. ***P*** is a finite set of ***production rules***, with each rule being a variable and a string of variables and terminals: $A \to \alpha$, $A \in V$ and $\alpha \in (V \cup \Sigma)^*$

Derivation

- Strings α yields string β, written $\alpha \overset{*}{\Rightarrow} \beta$, if it is possible to get from α to β using the productions. A derivation of β is the sequence of steps that gets to β.
- A leftmost derivation is where at each stage one replaces the leftmost variable.
- A rightmost derivation is defined similarly.

Given grammar S→S+S | S*S | a | b
Find the leftmost and rightmost derivation for w = a*a+b

Left Most Derivation

w = a*a+b

S→S*S

S→a*S

S→a*S+S

S→a*a+S

S→a*a+b

Rightmost Derivation

w = a*a+b

S→S*S

S→S*S+S

S→S*S+b

S→S*a+b

S→a*a+b

Derivation Tree

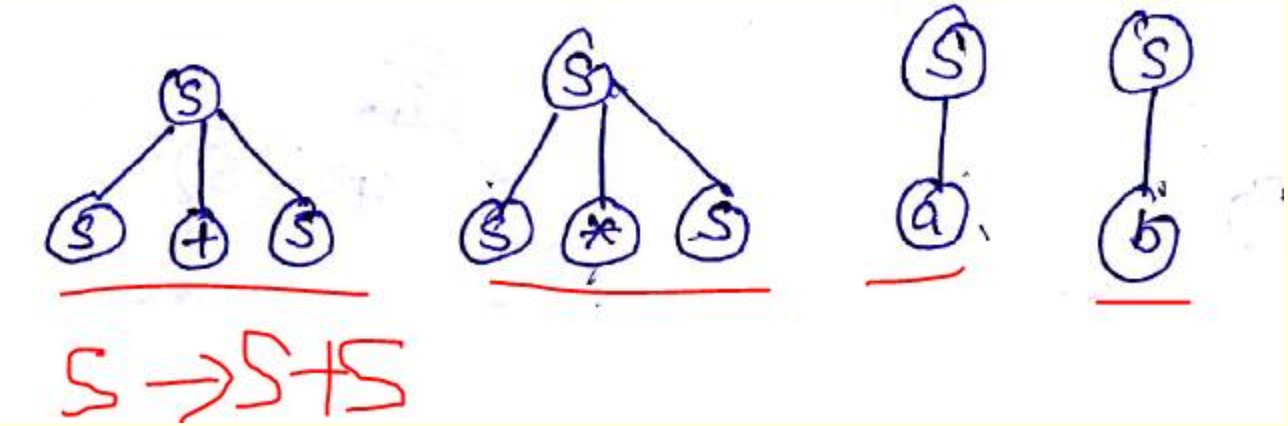
In a derivation tree:

1. the root is the start variable,
2. All internal nodes are labelled with variables,
3. while all leaves are labelled with terminals.

The children of an internal node are labelled from left to right with the right-hand side of the production used

Derivation Tree

- **Given grammar S→S+S | S*S | a | b Each rule of Grammar can be represented by the following derivation tree**



Given grammar S→S+S | S*S | a | b
Create derivation tree for w = a*a+b

Derivation

w = a*a+b

S→S*S

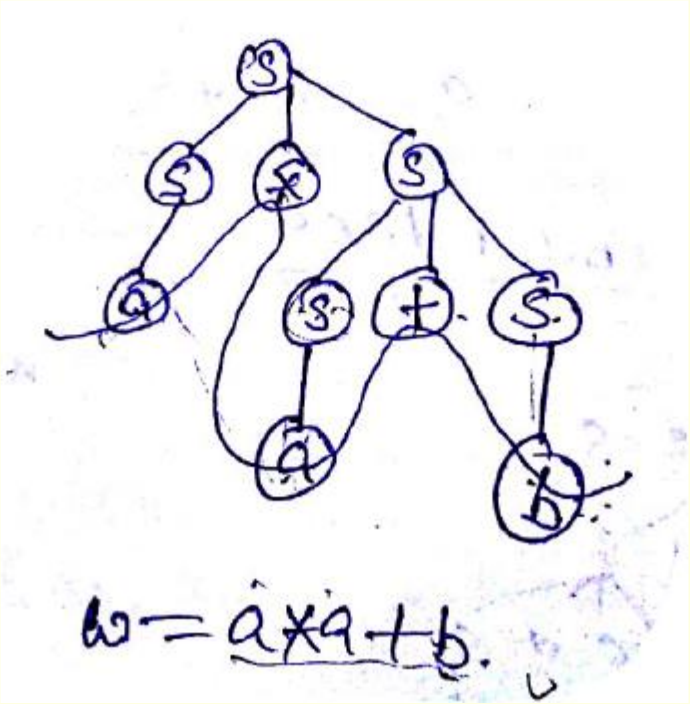
S→S*S+S

S→S*S+b

S→S*a+b

S→a*a+b

Derivation tree



Ambiguous Grammar

- A grammar is ambiguous if it has more than one Parse-Tree for some string. Equivalently, there is more than one right-most (or left-most) derivation for some string.
- Ambiguity is bad: Because multiple derivation trees provides multiple information about a given string w since we cannot decide its syntactical structure uniquely.
- Ambiguity is a property of Grammars, not of Languages

Ambiguous Grammar Example

S→S+S | S*S | a | b

<u>Ist Leftmost derivation</u>

$S \rightarrow S * S$

$\rightarrow a * S$

$\rightarrow a * S + S$

$\rightarrow a * a + S$

$\rightarrow a * a + a$

2nd leftmost derivation

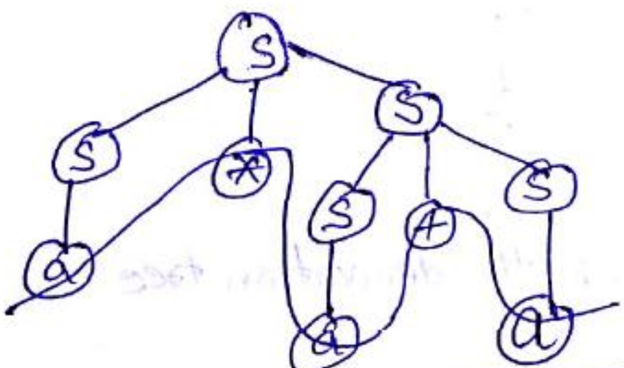
$S \rightarrow S + S$

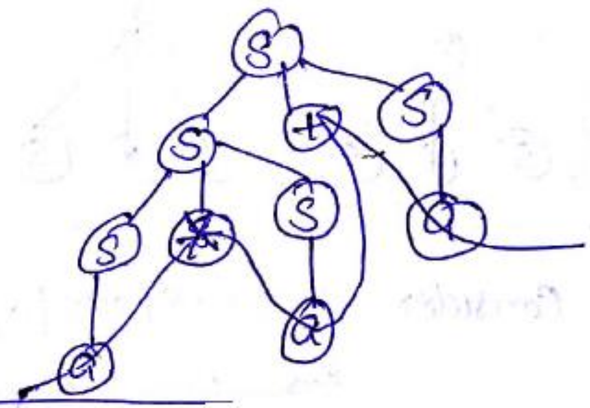
$S \rightarrow S * S + S$

$S \rightarrow a * S + S$

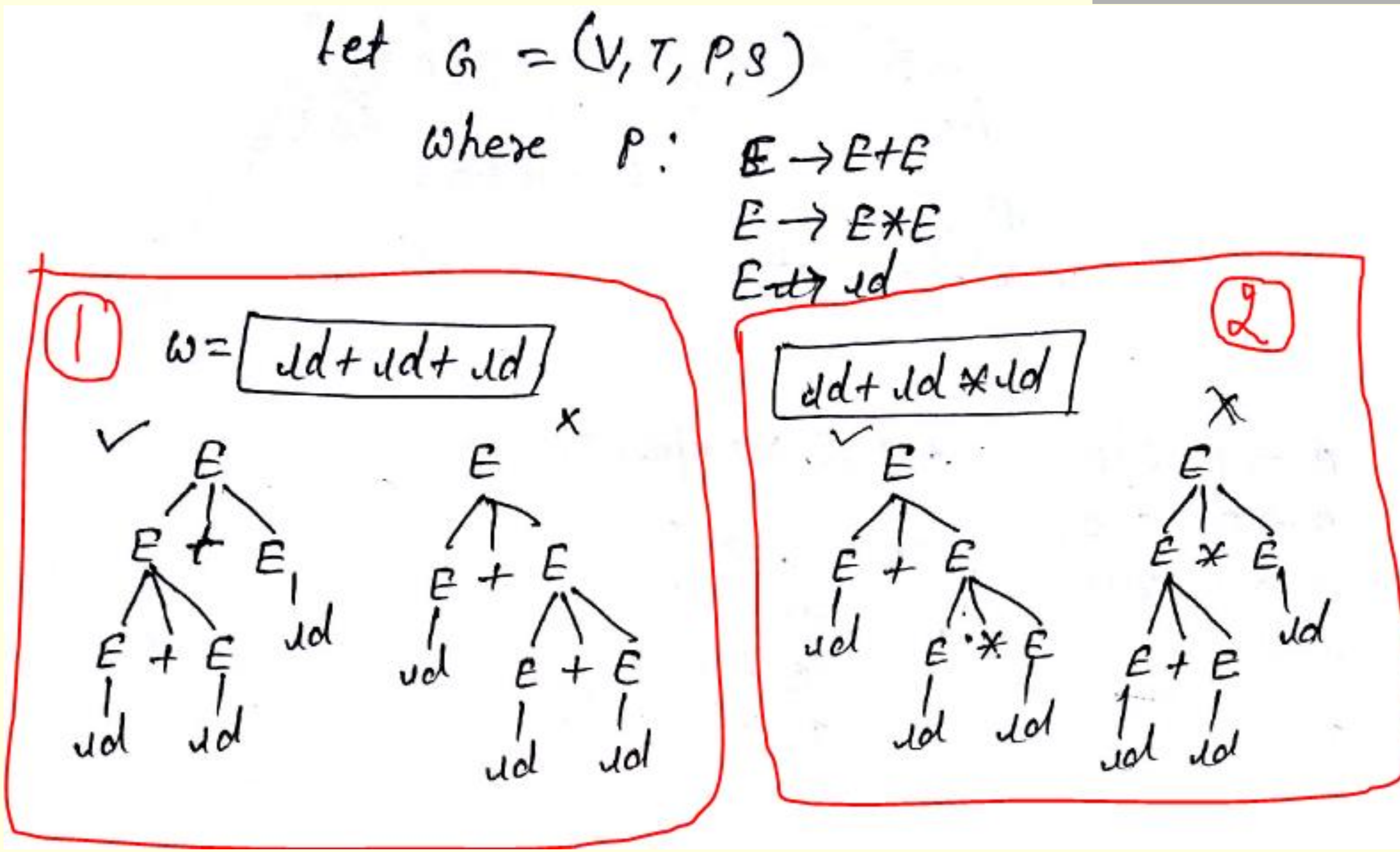
$S \rightarrow a * a + S$

$S \rightarrow a * a + a$





Removing Ambiguity Example 1



Removing Ambiguity : Precedence and Associativity Declarations

→As we can two different parse Tree/ derivation for a given string, the grammar on the previous slide is ambiguous.

→In 1st case we failed to maintain associativity while in 2nd case we failed to maintain precedence

→To maintain associativity we use recursion and to maintain precedence we use levels.

Removing Ambiguity

Ambiguous Grammar	**Equivalent Unambiguous Grammar**
$E \rightarrow E+E \mid E * E \mid id$	$E \rightarrow E+T \mid T$ $T \rightarrow T * F \mid F$ $F \rightarrow id$

Removing Ambiguity Example 2

Ambiguous Grammar

E→E+E | E *E | E ^ E | id

Equivalent Unambiguous Grammar

E→E+T | T

T→ T * F | F

F → G ^ F | G

G →id

Removing Ambiguity Example 3

Ambiguous Grammar

bEXP→bEXP OR bEXP

bEXP→bEXP AND bEXP

bEXP→ NOT bEXP

bEXP→true

bEXP→false

Equivalent Unambiguous Grammar

bEXP→bEXP OR F |F

F→ F AND G | G

G → NOT G | true | false

Removing Ambiguity Example 4

Ambiguous Grammar

R → R + R

R → RR

R → R*

R → a | b | c

Equivalent Unambiguous Grammar

R → R + T

T → TF | F

F → F* | a | b | c

Removing Ambiguity Example 5

$A \rightarrow A \$ B \mid B$
$B \rightarrow B \# C \mid C$
$C \rightarrow C @ D \mid D$
$D \rightarrow d$

$, #, @ are operator

$\$ \cdot> \$$
$\# \cdot> \#$
$@ \cdot> @$
$\$ <\cdot \# <\cdot @$

$E \rightarrow E * F$
$\mid F * E$
$\mid F$

$F \rightarrow F - F$
$\rightarrow id$

$* \cdot> *$

$+ <\cdot +$

$-$ is defined as left as well as right recursive.

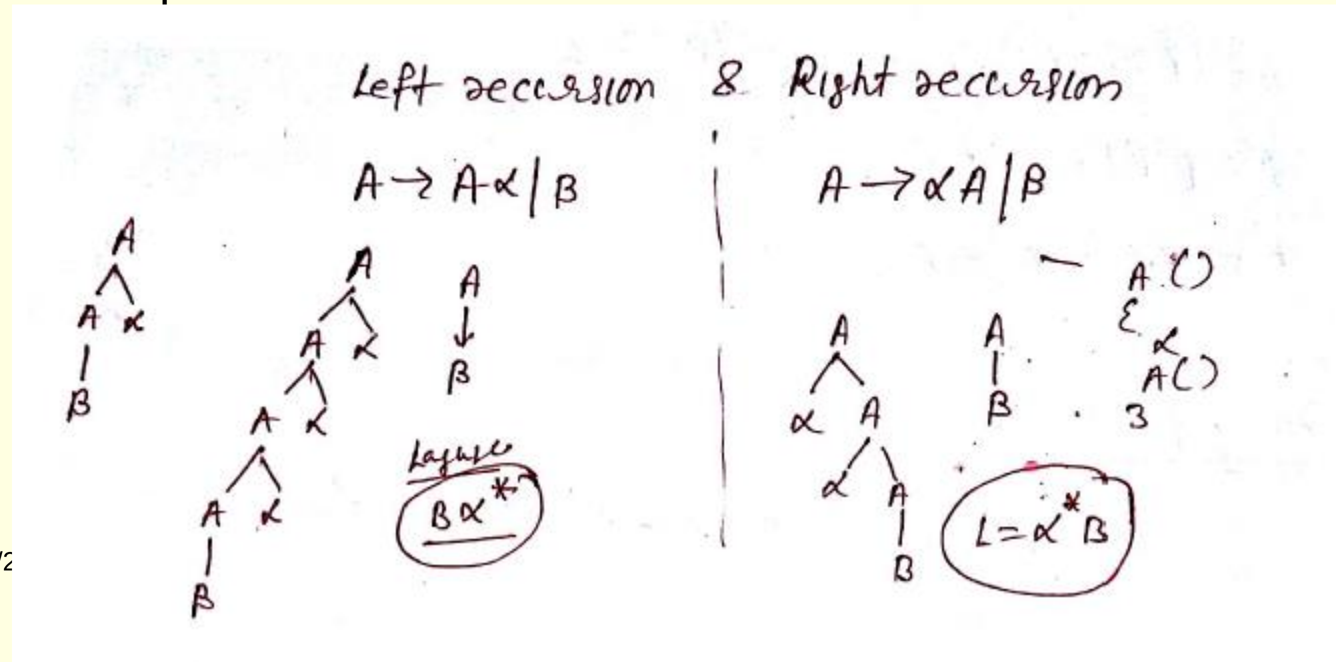
Left Recursive Grammar

A grammar is left recursive if it has a non terminal (variable) S such that their is a derivation

S -> Sα | β

where α is in (V+T)* and β is in (V+T)* (sequence of terminals and non terminals that do not start with S)

Due to the presence of left recursion some top down parsers enter into infinite loop so we have to eliminate left recursion.



Left Recursive Grammar

Lemma Let $G = (V, T, P, S)$ be a CFG. Let the set of A productions be $A \to A\alpha_1 | A\alpha_2 | \ldots | A\alpha_r | \beta_1 | \beta_2 | \ldots | \beta_s$ (β_i's do not start with A). Let Z be a new variable. Let $G_1 = (V \cup \{Z\}, \Sigma, P_1, S)$ where P_1 is defined as follows.

(i). The set of A productions are

$$A \to \beta_1 | \beta_2 | \ldots | \beta_s$$
$$A \to \beta_1 Z | \beta_2 Z | \ldots | \beta_s Z$$

(ii) The set of Z productions are

$$Z \to \alpha_1 Z | \alpha_2 Z | \ldots | \alpha_r Z$$
$$Z \to \alpha_1 | \alpha_2 | \ldots | \alpha_r.$$

(iii) The production for the other variable are same as in P. Then G_1 is a CFG and is equivalent to G.

Removing left recursion

Let the productions is of the form

$A \rightarrow A\alpha1 \mid A\alpha2 \mid A\alpha3 \mid .. \mid A\alpha m \mid \beta1 \mid \beta2 \mid …. \mid \beta n$

Where βi do not begins with an A . then we replace the A-productions by

$A \rightarrow \beta1 A' \mid \beta2 A' \mid ….. \mid \beta n A'$

$A' \rightarrow \alpha1A' \mid \alpha2A' \mid \alpha3A' \mid ….. \mid \alpha mA' \mid \varepsilon$

Removing left recursion

Examples

1

$$\underbrace{E}_{A} \to \underbrace{E}_{A}\underbrace{+T}_{\alpha} / \underbrace{T}_{\beta}$$

$$E \to TE'$$
$$E' \to \varepsilon / +TE'$$

Non left recursive grammar,

2 eliminate left recursion

$$\underbrace{S}_{A} \to \underbrace{S}_{A}\underbrace{0S1S}_{\alpha} / \underbrace{01}_{\beta}$$

$$S \to 01S'$$
$$S' \to \varepsilon / 0S1SS'$$

Ans

3

$$S \to (L) / x$$
$$\underbrace{L}_{A} \to \underbrace{L}_{A}\underbrace{S}_{\alpha} / \underbrace{S}_{B}$$

Remove left recursion

$$L \to SL'$$
$$L' \to \varepsilon / SL'$$
$$S \to (L) / x$$

Ans

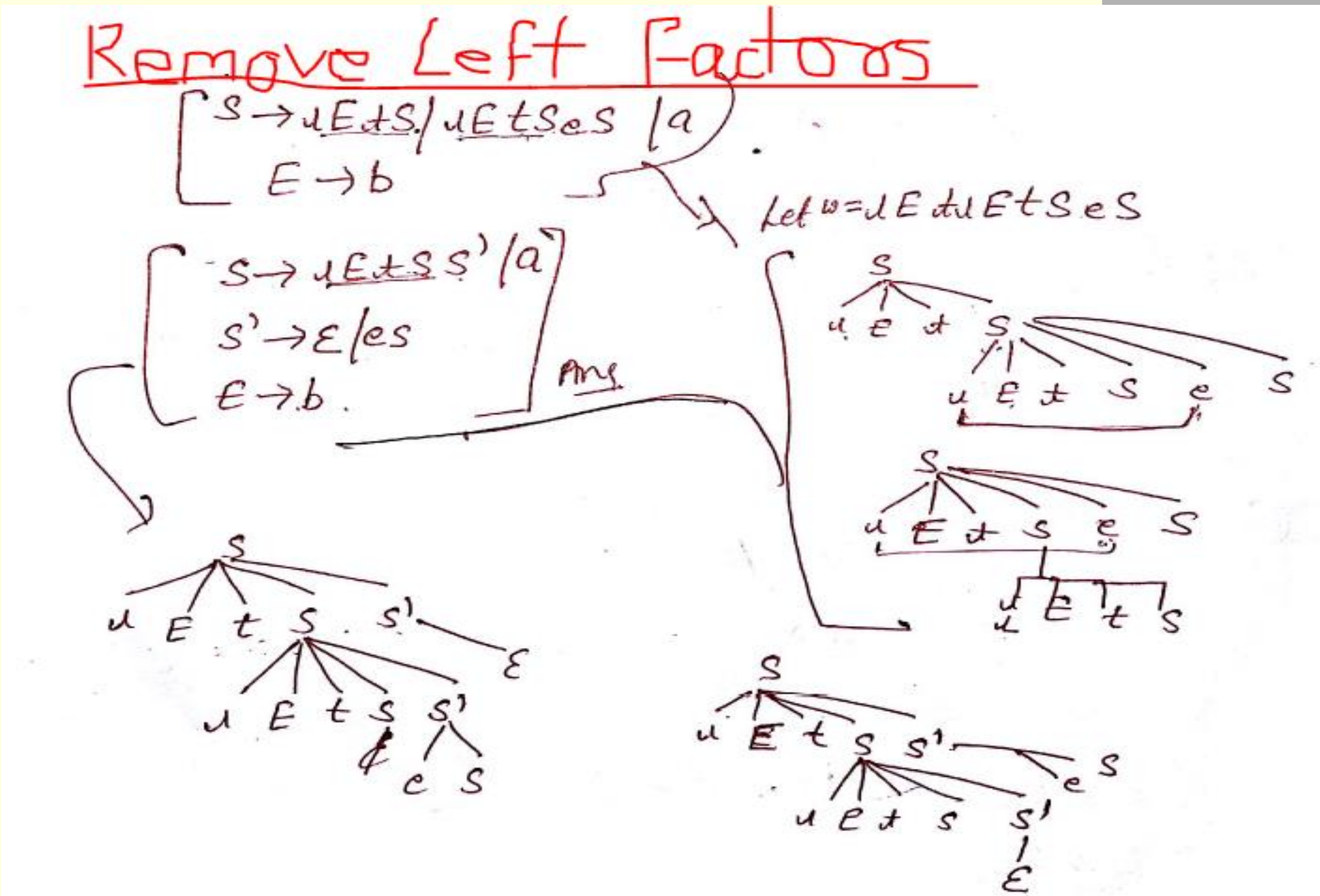
Left Factoring in CFG

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3$$ ← Non deterministic grammer

$$\downarrow$$

$$\boxed{\begin{array}{l} A \rightarrow \alpha A' \\ A' \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \end{array}}$$ ← Deterministic grammar

Left Factoring in CFG

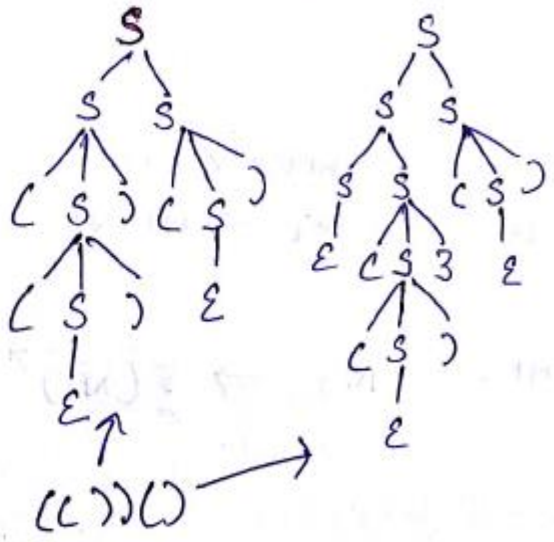


Example

Q Show that the CFG given below generates all strings of balanced parenthesis is ambiguous. Give an equivalent unambiguous G.

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

Sol^n



$$\Rightarrow \begin{array}{l} S \to ST \mid T \\ T \to (S) \mid () \end{array}$$

unambiguous grammar

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:

$L = \{ a^n b^n c^m \mid n,m \geq 1\} \cup \{a^n b^m c^m \mid n,m \geq 1\}$

L is inherently ambiguous

Why?

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

Summary

- Left factoring, Left Recursion and Ambiguity has no relation with each other
- A Grammar can have both left factor and left recursion and still unambigious

Queries

Thanks!!!