Module 3

CONTEXT FREE GRAMMAR

- Context free Grammar is a formal grammer which is used to generate all possible strings in a given formal language.
- Context free grammar (G) can be defined by 4 tuples:

  $G = (V, T, P, S)$

- V : Variables (NON terminals) : A finite set of NON-terminal symbols. (Capital letters)
- T : Finite set of terminal symbols. (Small letters)
- P : Set of Production Rules
- S : Start Symbol. (used to derive the string)

Example Pallindrome

(* ε : Empty string is a Pallindrome
0 : "0" is Pallindrome
1 : "1" is Pallindrome.

* Grammar contains set of rules to construct Valid Sentences in language.
* CFG, Set of rules to construct strings for context free language.

Example

$$L = \{ wcw^R \mid w \in (a,b)^* \}$$

Production Rules

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow c$

Now check that abbcbba string can be derived from the given CFG.

$S \Rightarrow aSa$

$S \Rightarrow abSba$

$S \Rightarrow abbSbba$

$S \Rightarrow abbcbba$

Formal Definition of CFG

CFG : (V, T, P, S)

V : Finite set of Variables (Non Terminal)

T : finite set of Terminals $(V \cap T = \emptyset)$

P : Production Rules (Substitution Rules)

$\alpha \rightarrow \beta$

α ← Only one Variable (Non-Terminal)

β → $(V \cup T)^*$

[CFG are (Type 2) Grammer]

S : Start Symbol.

$\alpha \rightarrow \beta$

CHOMSKY HIERARCHY

Type 0 :	Unrestricted lang grammer	Recognized by (Turing m/c)	
Type 1 :	Context sensitive	('α' can have one or more symbols)	
Type 2 :	Context free	(Left side has exactly one Non-Terminal symbol)	
Type 3 :	Regular Grammer.		

Example

$S \rightarrow 0S1$

$S \rightarrow \varepsilon$

OR

$S \rightarrow 0S1 \mid \varepsilon$

$S \rightarrow 0\,S\,1$

$\downarrow$

$0S1$

$\downarrow$

$0S1$

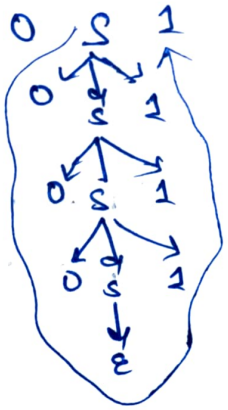
$\downarrow$

$0S1$

$\downarrow$

ε

$\Rightarrow$



$0000\,1111$

$0^n\,1^n \quad n \geq 0$

Example

$S \rightarrow 0A$

$A \rightarrow A0 \mid 0$

Example Ques. Convert CFL $\rightarrow$ CFG

1. $a^n b^n, \; n \geq 0$

$S \rightarrow aSb \mid \varepsilon$

$S \rightarrow aasbb$

$S \rightarrow aaasbbb$

$S \rightarrow aaa\,\varepsilon\,bbb$

(4)

Q.R $a^n b^n$, $n \geq 1$ $\quad S \to aSb \mid ab$

↳ min will be ab instead of λ or ε.

2. $a^n b^{n+2}$, $n \geq 0$

$S \to aSb \mid bb$

3. $a^{2n} b^n$, $n \geq 0$ $\quad$ or $\quad a^{2n} b^n$, $n \geq 1$

$S \to aaSb \mid \lambda$ $\quad$ $S \to aaSb \mid aab$

4. $a^{2n+3} b^n$, $n \geq 0$

$S \to aaSb \mid aaa$

5. $a^m b^n$, $m > n$ $\quad n \geq 0$

$S \to AS_1$

$S_1 \to aS_1 b \mid \lambda$ // means a & b are equal, but we need extra a ($\because m > n$)

$A \to aA \mid a$

OR

$a^m b^n$, $m \geq n$ $\quad n \geq 0$

$S \to AS_1$

$S_1 \to aS_1 b \mid \lambda$

$A \to aA \mid \lambda$

6. $\{ w \mid n_a(w) = n_b(w) \}$

$S \to aSb \mid bSa \mid \lambda$

7. $WW^R \cup w(a+b)w^R \Rightarrow S \to aSa \mid bSb \mid a \mid b \mid \lambda$

$S \to aSa \mid bSb \mid \lambda$ Even Pallindrome

$S \to aSa \mid bSb \mid a \mid b \mid$ Odd Pallindrom

8. $a^m b^m c^n \quad m, n \geq 0$

$S \to S_1 C$

$S_1 \to aS_1 b \mid \lambda$

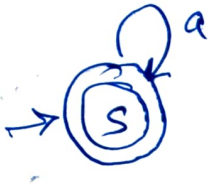
$C \to cc \mid \lambda$

* In formal theory, a context free language is a language generated by some context free Grammar
* The set of all CFL is identical to the set of languages accepted by Pushdown Automata.

Q. CFG for language having any no. of a's.

$V = \{S\} \quad T = \{a\}$

Production Rule $P = \begin{cases} S \to aS \\ S \to \varepsilon \end{cases}$



(6) Q. CFG for language having any no. of a's and b's.

$G = (V, T, P, S)$

$V = \{S\}$ $T = \{a, b\}$

$P = \{ S \to aS \mid bS \mid \varepsilon \}$ or $S \to aS$, $S \to bS$, $S \to \varepsilon$

Q. CFG to generate string consisting of atleast one 'a'.

$P = \{ S \to aA,\ A \to aA \mid \varepsilon \}$

or

$\{ S \to aA,\ A \to \varepsilon \}$

$G = (V, T, P, S)$

$V = \{S\}$ $T = \{a\}$

S: Start Symbol

Q. CFG, for string consisting of multiple of 3 a's.

$G = (V, T, P, S)$ $T = \{a\}$

$V = \{A, B, S\}$

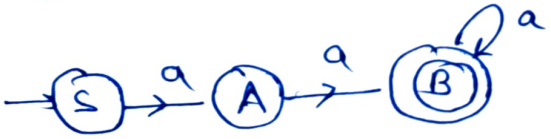
$P = \{ S \to aA,\ A \to aB,\ B \to aS,\ S \to \varepsilon \}$

or $S \to aaaS \mid \varepsilon$

Q. String consisting of atleast two a's.

$G = (V, T, P, S)$ $V = \{S, A, B\}$ $T = \{a\}$

$P = \{ S \to aA,\ A \to aB,\ B \to aB \mid \varepsilon \}$



S: Start Symbol

CFG,

Q: String over a's & b's, with atleast one 'a'.

$G = (V, T, P, S)$

$V = \{S, A\}$ $T = \{a, b\}$

$P = \{ S \to bS$
$S \to aA$
$A \to aA \mid bA \mid \varepsilon \}$

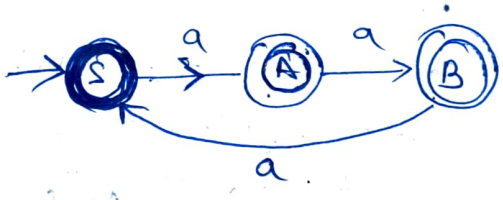
Q. CFG, string over (a, b) s.t, string length is multiple of 3.

$G = (V, T, P, S)$ $V = \{S, A\}$ $T = \{a, b\}$

$$P = \left\{ \begin{matrix} S \to \varepsilon \mid AAAS \\ A \to a \mid b \end{matrix} \right\}$$

Q: CFG, for $L = \{w : |w| \bmod 3 > 0 \quad w \in \{a\}\}$

$G = (V, T, P, S)$ S: start symbol



$V = \{S, A, B\}$ $T = \{a, \}$

$$P = \left\{ \begin{matrix} S \to aA \\ A \to aB \mid \varepsilon \\ B \to aS \mid \varepsilon \end{matrix} \right\}$$

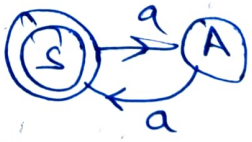
Q. CFG for even No. of a's.

$G = (V, T, P, S)$

$V = \{S, A\}$

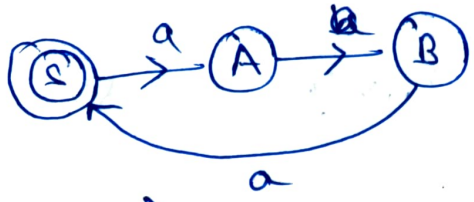
$P = \{ S \to aA \mid \varepsilon$
$A \to aS \}$

'S' is Start Symbol.



(8)

Q. CGF, for $L = \{ w : |w| \bmod 3 = 0,\ w \in \{a\}^* \}$



$G = (V, T, P, S)$

$V = \{S, A, B\}$ S: Start symbol

$T = \{a\}$

$$P = \begin{cases} S \to aA \mid \varepsilon \\ A \to aB \\ B \to aS \end{cases}$$

Q: $L = \{ a^{n+1} b^n : n \geq 0 \}$

$G = (V, T, P, S)$

$V = \{S\}$ $T = \{a, b\}$

S: Start symbol.

$P = \{ S \to aSb \mid a \}$

Q. $L = \{ a^n b^{n+1} : n \geq 0 \}$

$G = (V, T, P, S)$

$V = \{S\}$ $T = \{a, b\}$ S: Start Sy.

$P = \{ S \to aSb \mid b \}$

Q: $L = \{ a^n b^n \}$ $n \geq 0$

$P: \{ S \to aSb \mid \varepsilon \}$

$G = (V, T, P, S)$

$V = \{S\}$ $T = \{a, b\}$

Q. $L = \{ a^n b^n \}$ $n \geq$

$P = \{ S \to aSb \mid ab \}$

Q. $L = \{ a^n b^{n+2} : n \geq 0 \}$

$G = (V, T, P, S)$

$V = \{S\}$ $T = \{a, b\}$ S: start symbol.

$P = \{ S \to aSb \mid bb \}$

Q. $L = \{ a^n b^{2n} : n \geq 0 \}$

$G = (V, T, P, S)$ $V = \{S\}$ $T = \{a, b\}$ S: start symbol

$P = \{ S \to \varepsilon \mid aSbb \}$

Q: CFG, for generating set of all palindrome over $\Sigma = \{a, b\}^*$

$G = (V, T, P, S)$

$V = \{S\}$ $T = \{a, b\}$ S: start symbol

$P = \{ S \to \varepsilon$
$S \to a \mid b$
$S \to aSa \mid bSb \}$

Q: CFG, of all Non-Palindrome over $\{a, b\}^*$

$G = (V, T, P, S)$ $V = \{S, A, B\}$ $T = \{a, b\}$ S: start sy.

$P = \{ S \to aSa \mid bSb$
$S \to A$
$A \to aBb \mid bBa$
$B \to aB \mid bB \mid \varepsilon \}$

$\{ S \to aSa \mid bSb \mid aSb \mid bSa \mid a \mid b \}$

a S a

Q. $L = \{ 0^m 1^m 2^n \mid m \geq 1 \text{ \& } n \geq 0 \}$

A B

$S \rightarrow A \quad B$

A: should produce 'm' no. of 0's followed by 'm' 1's & B → any no. of 2's.

$S \rightarrow AB$

$n \geq 0$ $B \rightarrow \varepsilon \mid 2B$

$m \geq 1$ $A \rightarrow 0A1 \mid 01$

$$P = \begin{cases} S \rightarrow AB \\ A \rightarrow 0A1 \mid 01 \\ B \rightarrow \varepsilon \mid 2B \end{cases}$$

S: start symbol.

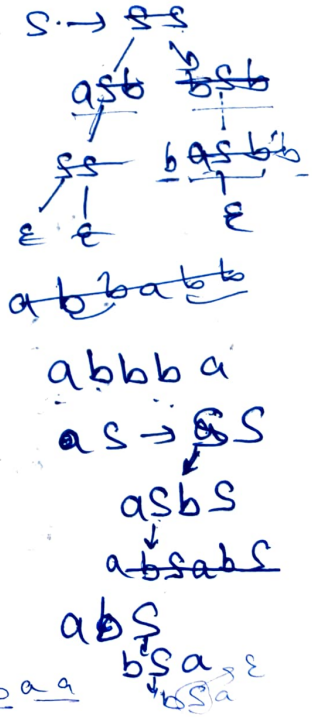
Q. $L = \{ w \mid n_a(w) = n_b(w) \}$

eg: abba, baab, ..

$G = (V, T, P, S)$

$V = \{S\}$ $T = \{a, b\}$ S: Start symb.

$P = \{ S \rightarrow \varepsilon \mid aSb \mid bSa \mid SS \}$



Ques. Obtain a grammar to generate a string balanced parenthesis.

$G = (V, T, P, S)$

$V = \{S\}$ $T = \{(,)\}$ S: Start symbol

$P = \{S \to (S) \mid SS \mid \varepsilon\}$

Ques. Grammar to generate ~~strings~~ language.

$L = \{a^i b^j \mid i \neq j, i \geq 0, j \geq 0\}$

$G = (V, T, P, S)$ $V = \{S, A, B\}$

$T = \{a, b\}$

$P = \{ S \to aSb \mid A \mid B$
$A \to aA \mid a$
$B \to b \mid bB \}$

S: Start Symbol

$L = \{0^i 1^j \mid i \neq j, i \geq 0, j \geq 0\}$

$P =$
$\{S \to 0S1 \mid A \mid B$
$A \to 0A \mid @$
$B \to 1B \mid 1\}$

Ques $L = \{a^n b^m \mid n \geq 0, m > n\}$

$n = 0$ $n = 1$ $n = 2$
$m \geq 1$ $m \geq 2$ $m \geq 3 \ldots$

$L = \{\varepsilon bb^*, abbb^*, aabbbb^*, \ldots\}$

$G = (V, T, P, S)$

$V = \{S, B\}$ S: Start Symbol.

$P = \{S \to aSb \mid B$
$B \to bB \mid b\}$

$P = \{S \to aSb \mid B$
$B \to bB \mid b\}$

(12)

Ques. $L = \{ a^n b^{n-3} \mid n \geq 3 \}$

$n=3$ $n=4$ $n=5$

$L = \{ aaa, aaaab, aaaaabb \ldots \}$

$G = (V, T, P, S)$ $V = \{S, A\}$

$P = \{ S \to aaaA \;|\; A \to aAb \mid \varepsilon \}$

S: start symbol.

aaaaabb

$S \to aaaA$

$aaaaAb$

$aaaaAbb$

$aaaaabb$

Q. $L = \{ a^n b^{2n} \mid n \geq 1 \}$ $P = \{ S \to aSbb \mid abb \}$

Q. $L = \{ a^{2n} b^{3n} \mid n \geq 1 \}$ $P = \{ S \to aaSbbb \mid aabbb \}$

$n=1$ $n=2$

$\{aabbb$

Q. $L = \{ a^n b^{n+3} \mid n \geq 1 \}$ $P = \{ S \to aSb \mid bbb \}$

$\Rightarrow \{ a^n b^n bbb \mid n \geq 1 \}$

Q. $L = \{ a^n b^m \mid n > m, \; m \geq 1 \}$

$m=1$ $m=2$ $m=3$

$L = \{ a^*a^*ab, \; a^*aaabb, \; a^*aaaabbb \}$

$P = \{ S \to aSb \mid A$

$A \to aA \mid a \}$

$B \to$

$S \to AB$

$\to aB$

$S \to A$

aaa

aBb

$aaabb$

$\{ S \to AB$

$B \to aBb \mid ab$

$A \to aA \mid a \}$

Q.

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

$L = \{ \underset{m=1}{a^* ab} \;\; \underset{m=2}{a^* aabb} \;\; \underset{m=3}{a^* aaabbb} \}$

$$P = \begin{cases} S \to aSb \mid A \\ A \to aA \mid \varepsilon \end{cases}$$

Q. $L = \{ a^n b^m \mid m \geq n, \; m \geq 1 \}$

$m=1, \; m=2, \; m=3$

$L = \{ abb^*, \; aabbb^*, \; aaabbbb^* \}$

$$P = \begin{cases} S \to aSb \mid B \\ B \to bB \mid \varepsilon \end{cases}$$

Q: $L = \{ \underbrace{a^n b^n}_{A} \underbrace{c^k}_{B} \mid n, m, k \geq 0 \}$

$$P = \begin{cases} S \to AB \\ A \to aAb \mid \varepsilon \\ B \to cB \mid \varepsilon \end{cases}$$

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

$$P = \begin{cases} S \to AB \\ A \to aA \mid a \\ B \to bBc \mid bc \end{cases}$$

(4)

Q:- $L = \{ \underbrace{a^n}_{A} \underbrace{b^m c^k}_{B} \mid m = k,\ n, m, k \geq 1 \}$

P= { $S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow bBc \mid bc$ }

Q:- $L = \{ a^n b^n c^m \mid n, m \geq 1 \}$ ∪ $L = \{ a^n b^m c^m \mid n, m \geq 1 \}$

$S \rightarrow S_1 \mid S_2$

{ $S_1 \rightarrow AB$

$A \rightarrow aAb \mid ab$

$B \rightarrow cB \mid c$

$S_2 \rightarrow CD$

$C \rightarrow aCb \mid ab$

$D \rightarrow cD \mid c$ }

x x

DERIVATIONS USING GRAMMAR

The Process of obtaining string of terminals or Non-terminals from the start symbol by applying some or all production Rules is called Derivation.

* If only one Production are applied to get string, one-step Derivation.

* If one or more Production rules are applied to get string $\alpha\beta\gamma$ from A then,

$$A \Rightarrow^{+} \alpha\beta\gamma$$

* If zero or more production rules are applied to get string $\alpha\beta\gamma$ from A then we write,

$$A \Rightarrow^{*} \alpha\beta\gamma$$

1. LEFT-MOST DERIVATION

In a derivation process if a left most variable is replaced at every step then derivation is said to be leftmost

2. Right-Most Derivation

In a derivation process if the right most variable is replaced at every step then the derivation is said to be rightmost.

(16)

Ques. Obtain the leftmost derivation for the string aaabbabbba using following grammar.

$$S \rightarrow aB \mid bA$$
$$A \rightarrow aS \mid bAA \mid a$$
$$B \rightarrow bS \mid aBB \mid b$$

Soln Left Most

$S \rightarrow aB$ (Applying $S \rightarrow aB$)

$\Rightarrow aaBB$ ($B \rightarrow aBB$)

$\Rightarrow aaaBBB$ ($B \rightarrow aBB$)

$\Rightarrow aaabBB$ ($B \rightarrow b$)

$\Rightarrow aaabbB$ ($B \rightarrow b$)

$\Rightarrow aaabbaBB$ ($B \rightarrow aBB$)

$\Rightarrow aaabbabS$ ($B \rightarrow bS$)

$\Rightarrow aaabbabbA$ ($S \rightarrow bA$)

$\Rightarrow aaabbabba$ ($A \rightarrow a$)

(17)

Consider the grammar and check $w = aaaabb \in L(G)$ (using leftmost & Right most derivation)

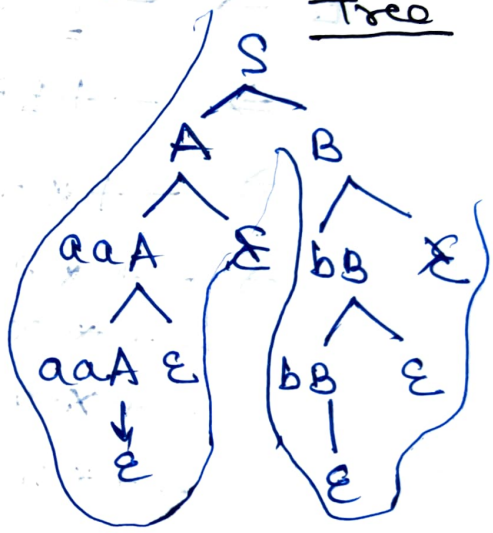
$S \to AB$
$A \to aaA \mid \varepsilon$
$B \to bB \mid \varepsilon$

soln

Leftmost

$S \Rightarrow \underline{A}B \quad (A \to aaA)$
$\Rightarrow aa\underline{A}B \quad (A \to aaA)$
$\Rightarrow aaaa\underline{A}B \quad (A \to \varepsilon)$
$\Rightarrow aaaa\underline{B} \quad (B \to bB)$
$\Rightarrow aaaab\underline{B} \quad (B \to bB)$
$\Rightarrow aaaabb\underline{B} \quad (B \to \varepsilon)$
$aaaabb$

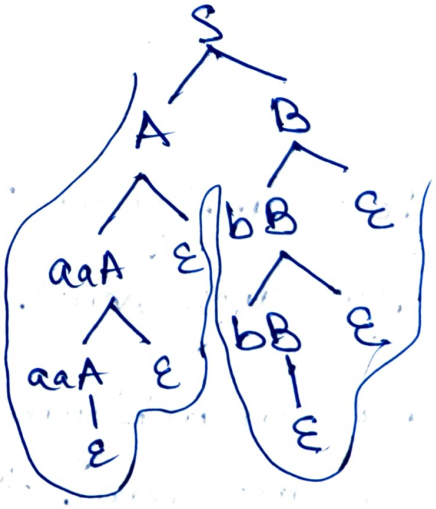
(Parse Tree)
Derivation Tree



aaaabb

Right most

$S \to A\underline{B} \quad (B \to bB)$
$A\underline{bB} \quad (B \to bB)$
$A\underline{bbB} \quad (B \to \varepsilon)$
$\underline{Abb} \quad (A \to aaA)$
$aa\underline{A}bb \quad (A \to aaA)$
$aaaa\underline{A}bb \quad (A \to \varepsilon)$
$aaaabb$



aaaabb

(18)

consider the grammer $E \to +EE \mid *EE \mid -EE \mid x \mid y$

(a) Find LMD & RMD for string $-+*xyxy$

(b) Is the grammar ambiguous ? Justify

(a) Left most Derivation

$E \to \underline{-E}E$

$\Rightarrow -+\underline{E}EE$

$\Rightarrow -+*EEEE$

$\Rightarrow -+*xEEE$

$\Rightarrow -+*xyEE$

$\Rightarrow -+*xyxE$

$\Rightarrow -+*xyxy$

Right most Derivation

$E \to -E\underline{E}$

$\Rightarrow -\underline{E}y$

$\Rightarrow -+E\underline{E}y$

$\Rightarrow -+\underline{E}xy$

$\Rightarrow -+*E\underline{E}xy$

$\Rightarrow -+*\underline{E}yxy$

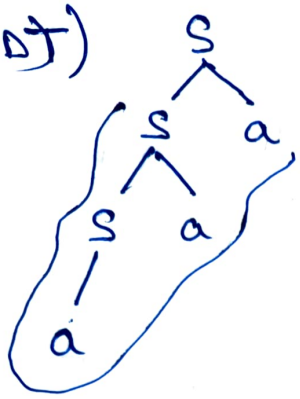
$\Rightarrow -+*xyxy$

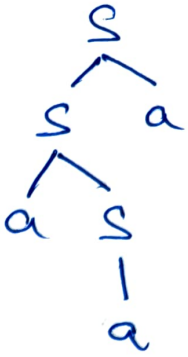
AMBIGUOUS & UNAMBIGUOUS GRAMMAR

- For ambiguous there exists more than one derivation for any word that belongs to Grammer.
- For unambiguous ther exist exactly one derivation for any word that belongs to Grammar.

Example G: $S \to Sa \mid aS \mid a$ W: aaa

(LMDT)



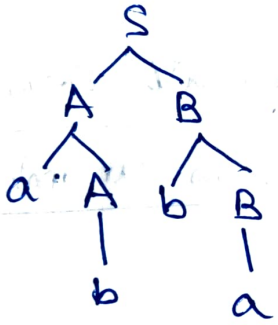


$\Rightarrow$ Ambiguous

Example G_1: $S \to AB$

$A \to aA \mid b$ W: abba

$B \to bB \mid a$



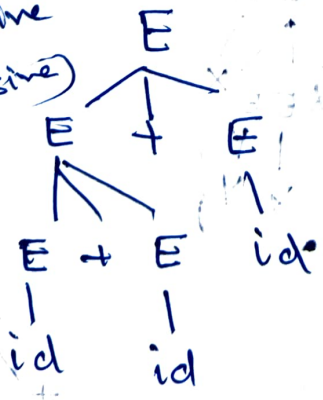
Unambiguous

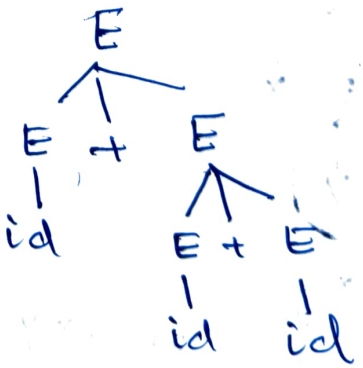
G_2: $E \to E+E \mid id$ W = id+id+id

(left Recursive & Right Recursive)

$\Downarrow$

ambiguous





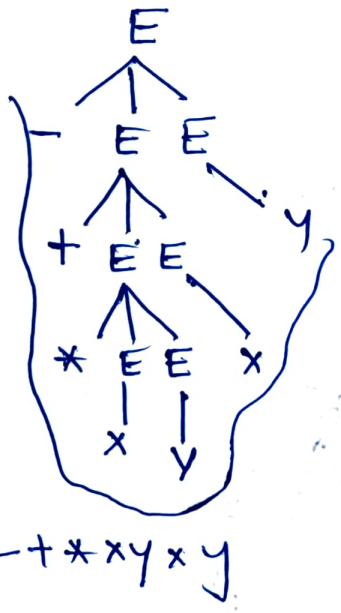
ambiguous

* Ambiguity of Grammar is undecidable
* If a Lang Grammar is Unambiguous
  $\Rightarrow$ R.M.D.T = L.M.D.T

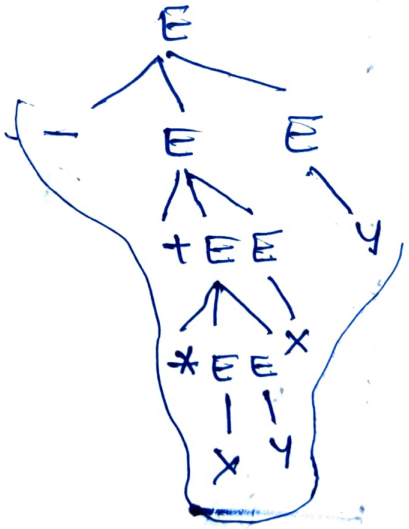
$\Rightarrow$ If a Grammar is Left Recursive & Right recursive $\Rightarrow$ G is ambiguous but Not Viceversa.

Ques (b) $E \rightarrow +EE \mid *EE \mid -EE \mid x \mid y$

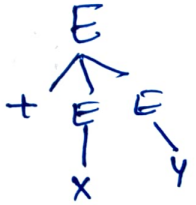
eg:- ~~-+*xyxy~~ If There are two Seperate Derivation tree existing, then language is unambiguous.



-+*xyxy



Here w: +xy



$\Rightarrow$ The Given Grammar is Unambiguous.

PARSE TREE

It is a graphical representation for the derivation of the given production rule for a given CFG.

↳ Start Symbol 'S' represents the root node

↳ The string derived is read from left to right

↳ The Terminal 'T' represent the leaf Node.

↳ The Non-terminal 'V' are the internal Nodes.

Ambiguity [Ambiguous Grammer]

↳ A Grammar is said to be ambiguous if there exist more than one left-most derivation or more than one right-most derivation or more than one parse tree for any given I/P string.

↳ Ambiguity is not best suited for compiler designing, but No method can automatically detect and remove the ambiguity.

Ques: check the grammar is ambiguous or not.

The string is id + id − id

$E \to E+E$

$E \to E-E$

$E \to id$

Left Most

$E \Rightarrow E+E$

$\Rightarrow id+E$

$\Rightarrow id+E-E$

$\Rightarrow id+id-E$

$\Rightarrow id+id-id$

or

Left Most

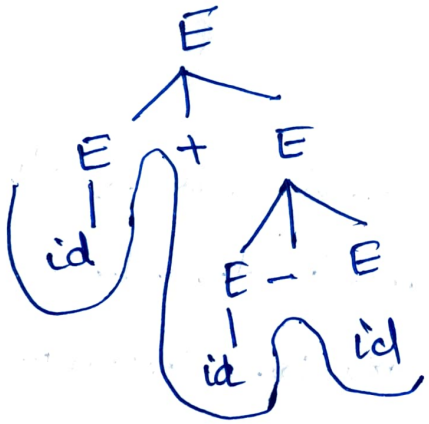
$E \Rightarrow E-E$

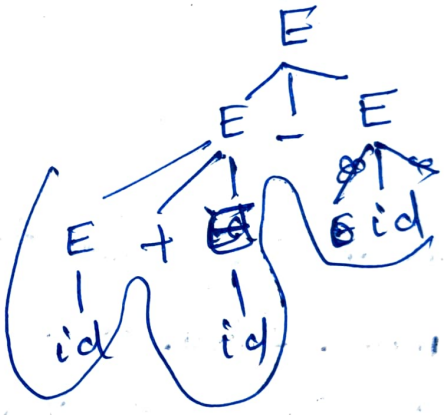
$\Rightarrow E+E-E$

$\Rightarrow id+E-E$

$\Rightarrow id+id-E$

$\Rightarrow id+id-id$





The Grammar is ambiguous.

Ques:

Ques. Given grammer,

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow id$

Obtain derivation Tree of expressions.

(i) $id * id + id$

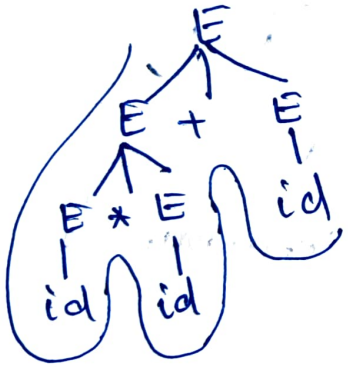
$E \rightarrow E + E$

$\Rightarrow E * E + E$

$\Rightarrow id * E + E$

$\Rightarrow id * id + E$

$\Rightarrow id * id + id$



$id * id + id$

or

$E \rightarrow E * E$

$\Rightarrow id * E$

$\Rightarrow id * E + E$

$\Rightarrow id * id + E$

$\Rightarrow id * id + id$

$\Downarrow$ The Grammer is Ambiguous as more than one LMD.

(ii) $(id + id) * id$

$E \rightarrow E * E$

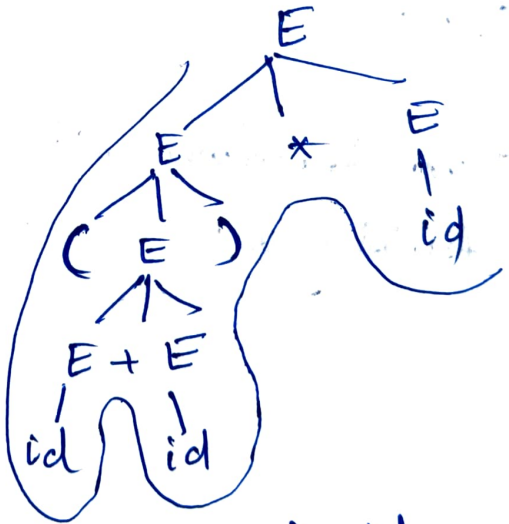
$\Rightarrow (E) * E$

$\Rightarrow (E + E) * E$

$\Rightarrow (id + E) * E$

$\Rightarrow (id + id) * E$

$\Rightarrow (id + id) * id$



$(id + id) * id$

(24)

Check the ambiguity of the Grammar.

$A \rightarrow AA$

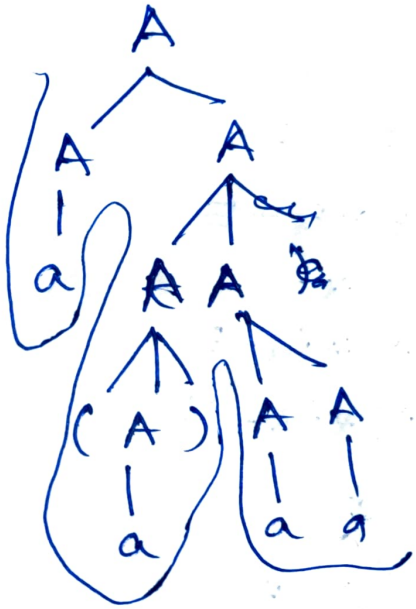
$A \rightarrow (A)$

$A \rightarrow a$

Soln Consider any String "a(a)aa"

a(a)aa

or



a(a)aa

Two derivation exist, hence the Grammar is Ambiguous.

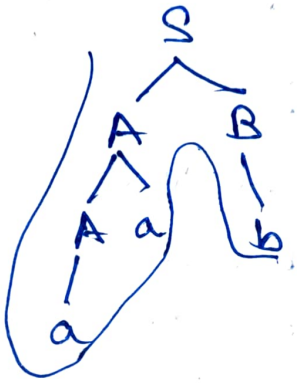
Ques:- $S \to AB \mid aaB$
$A \to a \mid Aa$
$B \to b$

String = aab

Check if the Grammar is ambiguous.

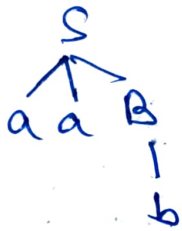
Leftmost:

$S \to AB$
$\Rightarrow AaB$
$\Rightarrow aaB$
$\Rightarrow aab$



or

$S \to aaB$



⇒ Grammar is ambiguous; Since Two Parse Tree.

Ques: $S \to aS \mid X$
$X \to aX \mid a$

check ambiguity.
"aaaa"

$S \to aS$
$\Rightarrow aaS$
$\Rightarrow aaaS$
$\Rightarrow aaaX$
$\Rightarrow aaaa$

or

$S \to X$
$\Rightarrow aX$
$\Rightarrow aaX$
$\Rightarrow aaaX$
$\Rightarrow aaaa$

"Ambiguous", since two derivation exist.

Ques Is the following grammar ambiguous ?

$S \to iC+S \mid iC+SeS \mid a$

$C \to b$

Sol String $ib+ib+aea$

$S \Rightarrow iC+S$

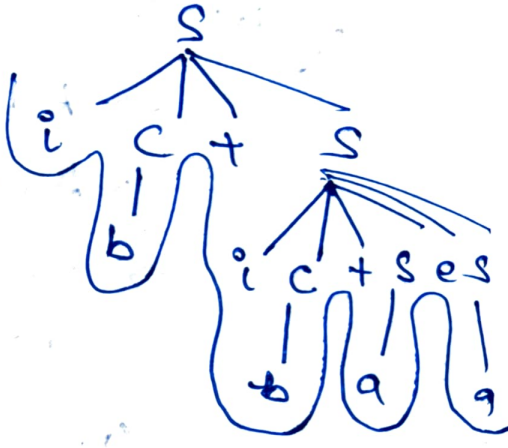
$\Rightarrow ib+S$

$\Rightarrow ib+iC+SeS$

$\Rightarrow ib+ib+SeS$

$\Rightarrow ib+ib+aeS$

$\Rightarrow ib+ib+aea$



OR

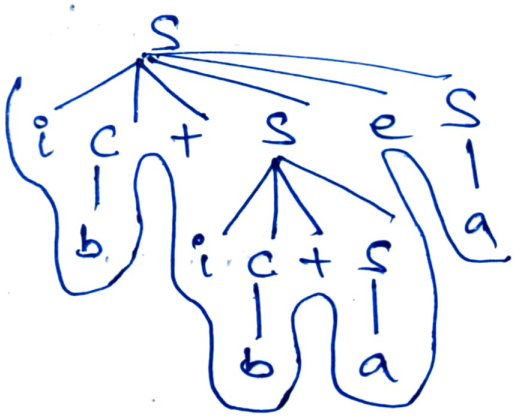
$S \to iC+SeS$

$\Rightarrow ib+iC+SeS$

$\Rightarrow ib+ib+SeS$

$\Rightarrow ib+ib+aeS$

$\Rightarrow ib+ib+aea$



$ib+ib+aea$

$\Rightarrow$ Grammar is Ambiguous.

Ques: $S \to SS \mid (S) \mid ()$ $\Rightarrow$ Check ambiguity.

$(()()())$

or

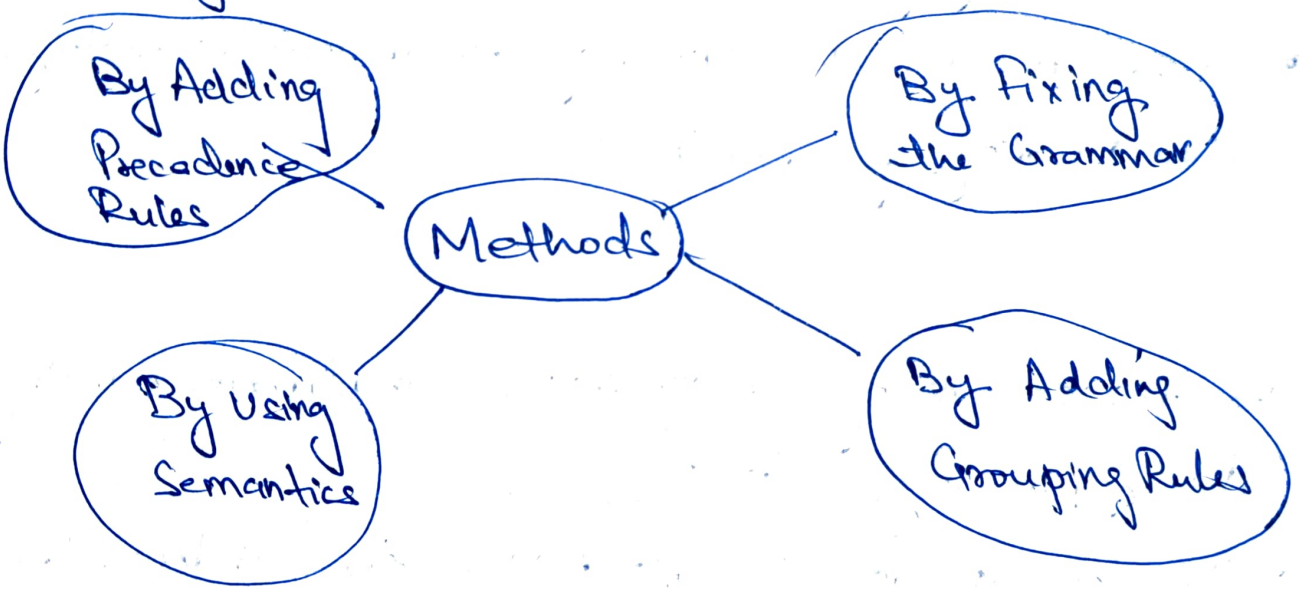
$\Rightarrow$ Ambiguous

Removal of Ambiguity

* There is no general algorithm to remove the ambiguity from Grammar.

↳ To check grammer ambiguity, we try finding a string that has more than one parse tree.

↳ If any such string exists, then grammar is ambiguous otherwise Not.



1. Removing ambiguity by Precedence & Associativity Rules

↳ An ambiguous grammar may be converted into an unambiguous grammer by implementing
(i) Precedence constraints
(ii) Associativity constraints.

Rule - 01 :

The Precedence Constraint is implemented using following rules -

- The level at which production is present defines the priority of operator contained in it.
- The higher the level of production, the lower the priority of operator
- The lower the level of production, the higher the priority of operator.

Rule 02 :

The Associativity Constraint is implemented using the following rules -

- If the operator is left Associative, induce left recursion in its production
- If the operator is right associative, induce right recursion in its production.

Ques: Convert the given Grammar into unambiguous Grammar.

$R \rightarrow R+R \mid R \cdot R \mid R^* \mid a \mid b$

*: Kleene Closure

•: Concatenation

soln. operators : + • * Operands a b

Priority order : (a,b) > * > • > +

* : left Associative

• : left Associative

'+' : left Associative

$E \rightarrow E+T \mid T$ (∵ Add. applied recursively on left side)

$T \rightarrow T \cdot F \mid F$ (∵ Concatenation is handled recursively on left side ie., T.F,

$F \rightarrow F^* \mid G$

$G \rightarrow a \mid b$

OR

$E \rightarrow E+T \mid T$

$T \rightarrow T \cdot F \mid F$

$F \rightarrow F^* \mid a \mid b$

Ques:- Convert the given 'G' to unambiguous.

bexp → bexp or bexp | bexp and bexp | not bexp | T | F

Soln

operators:- or, and, not

operands: T, F

Priority order (T, F) > not > and > or

And : left associative

or : left associative

Not : Right associative

bexp → bexp or M | M

M → M and N | N

N → Not N | G

G → T | F

or

bexp → bexp or M | M

M → M and N | N

N → Not N | T | F

Ques:- $E \to E - E \mid id$

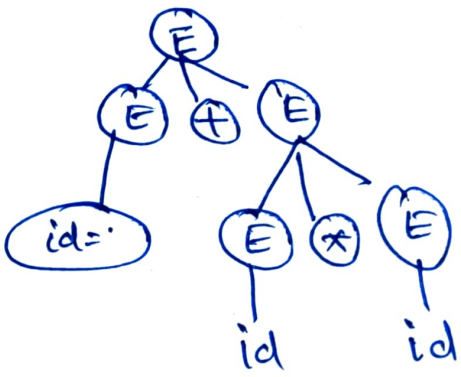
Gramer = {id, id-id, id-id-id ...}

Remove ambiguity.

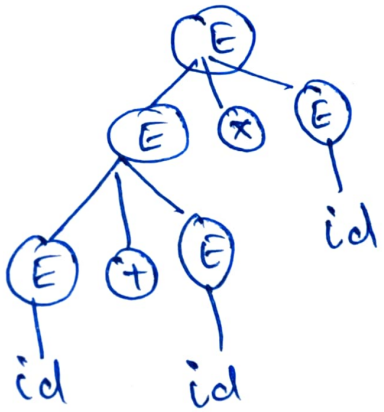
$E \to E - P \mid P$

$P \to id$

Ques:- $E \to E+E \mid E*E \mid id$



id + id * id



id + id * id

'+' has least priority, it has to be at the upper level & has to wait for the result produced by "*" which is at lower level. So first parse tree is correct.

$E \to E + P \mid P$ // + left associative & higher level

$P \to P * Q \mid Q$ // * left associative

$Q \to id$

$E \to E-E \mid E*E \mid E \wedge E \mid id$

$E \to E-P \mid P$ (-) least Priority, left Associativ

$P \to P*Q \mid Q$ * left associative & more priority than (-)

$Q \to R \wedge Q \mid R$ ∧ Right associative & highest Priority

$R \to id$

Ques:- $\{a^n b^n c^m d^m : n \geq 1 \ m \geq 1\} \cup \{a^n b^m c^m d^n : n \geq 1, m \geq 1\}$

$S \to AB \mid C$
$A \to aAb \mid ab$
$B \to cBd \mid cd$
$C \to aCd \mid aDd$
$D \to bDc \mid bc$

n=1 m=1

abcd

aacdd

aDd

bc

aaabcddd

String:- aabbccdd

~~Removing Ambiguity~~.

aB
cD

check if the Grammer is ambigous?

$S \to AB \mid C$
$A \to aB \mid ab$
$B \to cD \mid cd$
$C \to aD$