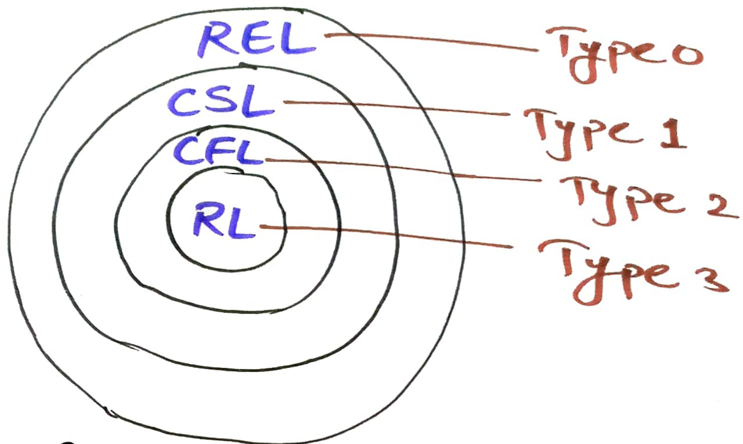
* Types of Formal Languages:-



Chomsky Hierarchy

① Type 0 (or) REL (Recursive Enumerable language)

The language which is generated by REG is known as REL

REL
- (Recursive Set) RS → Algorithm
- RES (Recursive Enumeration Set) → Procedure

Ex:- Program to find sum of two nos. ⇒ RS, REL

② Type 1 (or) CSL :- The language which is generated by CSG is known as CSL

Ex:- $L = \{a^n b^n c^n \mid n \geq 1\}$ $CSL = CSL \not\ni \{\epsilon\}$

Note Every CSL doesn't contain empty string 'ϵ'

② Type 2 (or) CFL :- The language which is generated by CFG is known as CFL

Ex:- $L = \{a^n b^n \mid n \geq 0\}$ ② $L = \{a^m b^n \mid m \leq n\}$

③ Type 3 (or) RL :- The language which is generated by RG is known as RL.

Ex:- $L = \{a^n \mid n \geq 1\}$; $L = \{w \in (a+b)^* \mid |w|_a = \text{even}\}$

Note:-

Type 3 $\subset$ Type 2 $\subset$ Type 1 $\subset$ Type 0

* Regular Grammar :-

The grammar that generate the regular language is known as regular grammar (or)

The grammar G is said to be regular if every production in the form —

$$A \to xB \mid y \quad (\text{or}) \quad A \to Bx \mid y$$

where, $A, B \in V$

$x, y \in T^*$

ex:- ① $S \to 01S \mid 0$ ② $S \to A0 \mid B10$

$A \to A01 \mid 00$

$B \to B11 \mid 00$

* Types of Regular Grammar :-

1) Right Linear Grammar (RLG)

2) Left Linear Grammar (LLG)

RG

RLG	LLG
$A \to xB \mid y$	$A \to Bx \mid y$
$\Downarrow$	$\Downarrow$
Left associativity	Right associativity

Note:- ① Every RL can be generated by either RLG (or) LLG

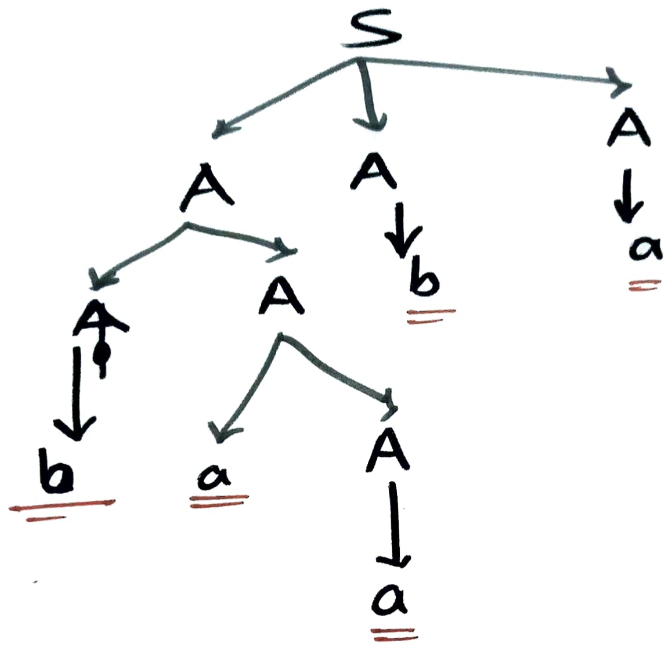
② RLG = LLG (Convert RLG into LLG & vice versa)

③ Regular grammar has one of the associativity either left (or) right, hence it is unambiguous grammar.

1) $S \to AAA \mid AA$

$A \to AA \mid aA \mid Ab \mid a \mid b$

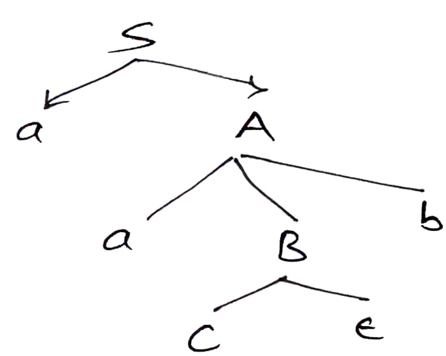
→ $w = baaba$

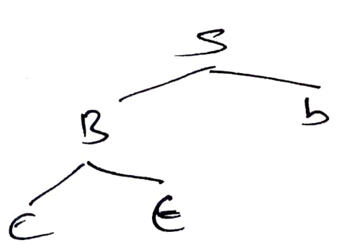


2) $S \to aA \mid Bb$

$A \to aB \mid b$

$B \to c \mid \epsilon$

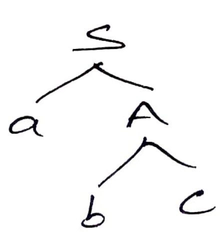


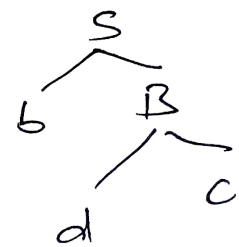


3) $S \to aA \mid bB$

$A \to b \mid c$

$B \to d \mid c$



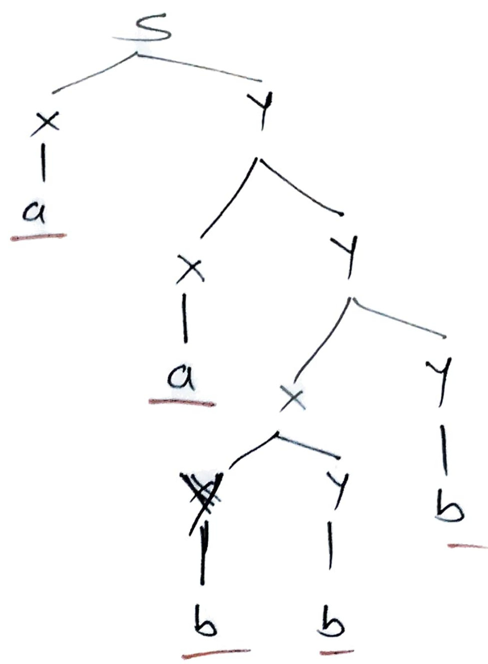


4) $S \to XY$

$X \to YY \mid a$

$Y \to XY \mid b$

$w = aabbb$



5) $E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid a \mid b$

→ Give derivation of $(a+b) * a+b$

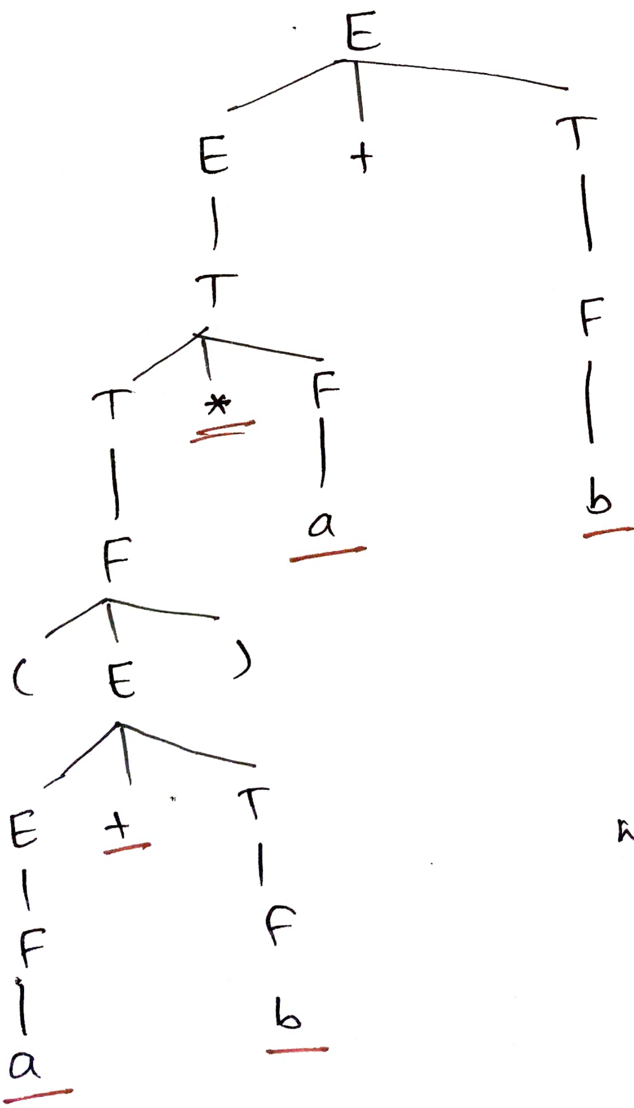
$V = \{E, T, F\}$

$T = \Sigma = \{+, *, (,), a, b\}$

$$P = \{ \begin{array}{l} E \to E+T \mid T \\ T \to T*F \mid F \\ F \to (E) \mid a \mid b \end{array} \}$$

Start Symbol $= E$

$$
\begin{aligned}
E &\rightarrow E+T \\
&\rightarrow T+T \\
&\rightarrow T*F+T \\
&\rightarrow F*F+T \\
&\rightarrow (E)*F+T \\
&\rightarrow (E+T)*F+T \\
&\rightarrow (F+F)*F+F \\
&\rightarrow (a+b)*a+b
\end{aligned}
$$



$w = (a+b)*a+b$

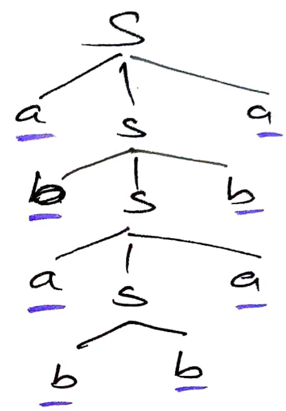
* CFG for Palindromes:-

① $L = \{ w \in (a,b)^* \mid w \text{ is even length palindrome} \mid |w| > 0 \}$

→ $S \to aSa$
$S \to bSb$
$S \to bb$
$S \to aa$

(or) $S \to aSa \mid bSb \mid bb \mid aa$

$w = ababbaba$



$S \to a\underline{S}a$
$\to a\underline{bSb}a$
$\to ab\underline{aSa}ba$
$\to aba\underline{bb}aba$

② $L = \{ w \in (a,b)^* \mid w \text{ is a palindrome of either even length (or) odd length with } |w| > 0 \}$

$S \to aSa \mid bSb \mid bb \mid aa \mid a \mid b$

$S \to aSa$
$\to abSba$
$\to abbbba$

$S \to bSb$
$\to baSab$
$\to baaab$

* CFG for this language

Assignment

1) $(a+b)^* bbb (a+b)^*$

2) $(011+1)^*(01)^*$

3) $0^i 1^{i+k} 0^k$; $i,k \geq 0$

4) $L = \{0^i 1^j 0^k \mid j > i+k\}$

* Ambiguous Grammar:-

A grammar G is said to be Ambiguous if there exists two (or) more derivation tree for a string w (that means two or more left derivation tree)

Ex:- $G = \{\{S\}, \{a, b, +, *\}, P, S\}$ where

P consits of $S \rightarrow S+S \mid S*S \mid a \mid b$

The string $a+a*b$ can be generated

$S \rightarrow S+S$
$\rightarrow a+S$
$\rightarrow a+S*S$
$\rightarrow a+a*S$
$\rightarrow a+a*b$

$S \rightarrow S*S$
$\rightarrow S+S*S$
$\rightarrow a+S*S$
$\rightarrow a+a*S$
$\rightarrow a+a*b$

=> It is ambiguous, it doesn't handle the precedence of operators + & *

② $G = \{ V = \{E, F\}, T = \{a, b, -\}, E, P \}$

where P consists of rules

$E \rightarrow F-E, \; F \rightarrow a, \; E \rightarrow E-F, \; F \rightarrow b, \; E \rightarrow F$

ⓐ Show that G is ambiguous

ⓑ Remove the ambiguity

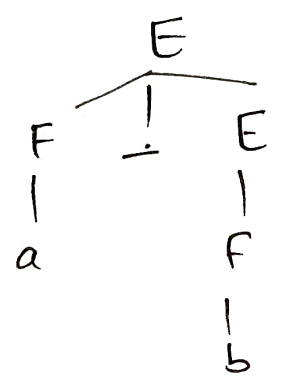
→ ⓐ

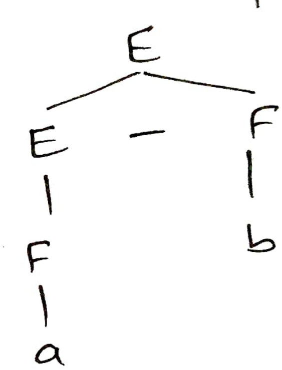
$E \rightarrow F-E \mid E-F \mid F$

$F \rightarrow a \mid b$

$\boxed{w = a-b}$

1)	2)
$E \rightarrow F-E$	$E \rightarrow E-F$
$\rightarrow a-E$	$\rightarrow F-F$
$\rightarrow a-F$	$\rightarrow a-F$
$\rightarrow a-b$	$\rightarrow a-b$





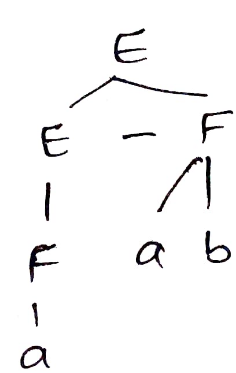
ⓑ The production $E \rightarrow F-E$ makes the evaluation from right to left which is normally not allowed

Thus ambiguity can be removed by deleting/removing the production $E \rightarrow F-E$

Left associativity should be there

$E \rightarrow E-F \mid F$

$F \rightarrow a \mid b$

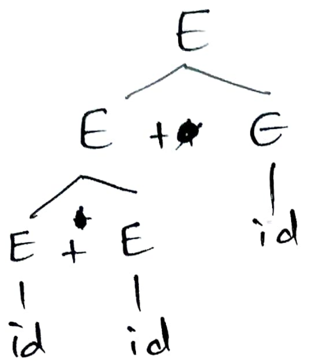


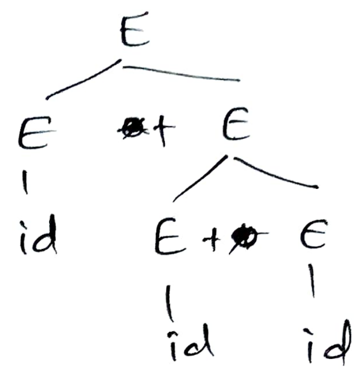
③ $E \rightarrow E+E \mid E*E \mid id$

⇒ ② $w = id + id * id$ ⇐ { Here, operator precedence not taken care }

① $w = id + id + id$

①

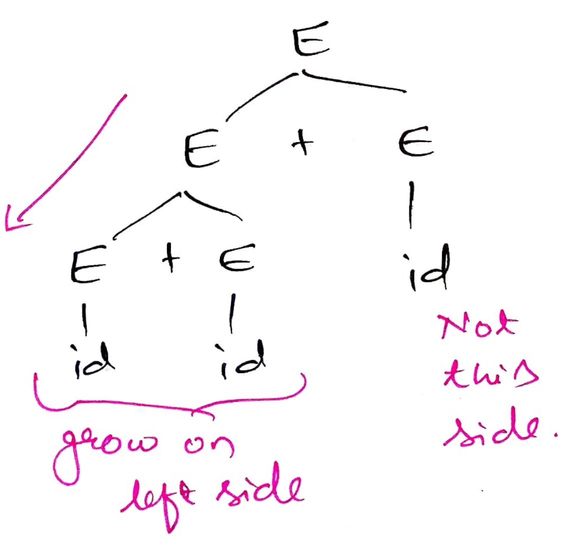




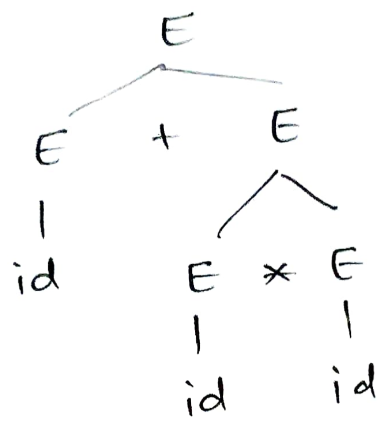
① $w = id + id + id$

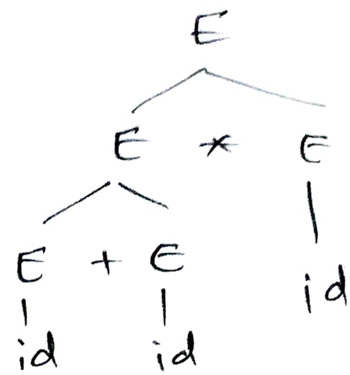
Whenever two operators on other side of operands which operator should associate. If plus (+) operator no problem of associativity but for (÷)(or)(−) operator it can be.

→ If plus operator is on left side then leftmost would be evaluated first.



② id + id * id





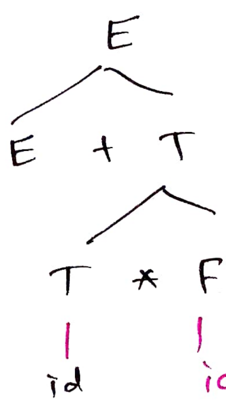
If in count operator should always left associative, the grammar should be left recursive means the leftmost symbol in RHS = LHS then operator is left associative.

$E \rightarrow E + T \mid T$

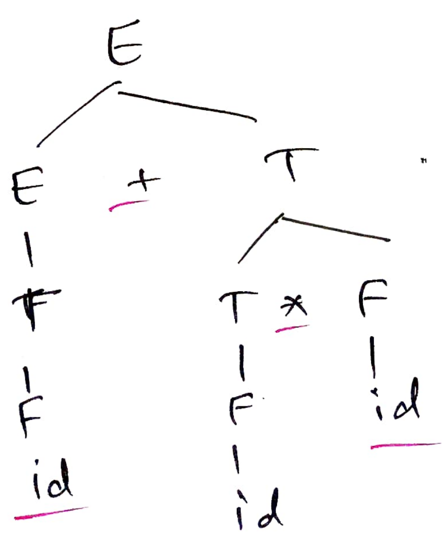
$T \rightarrow T * F \mid F$

$F \rightarrow id$

$\left.\begin{matrix} E = E \\ T = T \end{matrix}\right\}$ Left associativity.



← Unambiguous grammar

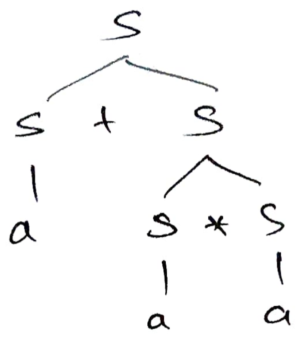


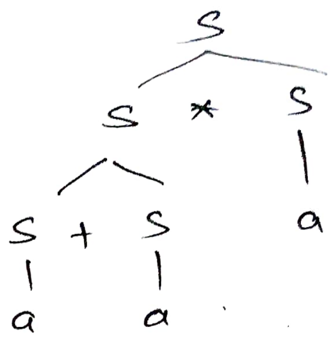
⇒ id + id * id

4 CFG

$S \rightarrow S+S \mid S-S \mid S*S \mid S/S \mid (S) \mid a$

→ derivation as $a+a*a$





- Unambiguious grammar is –

$S \rightarrow S+T \mid S-T \mid T$

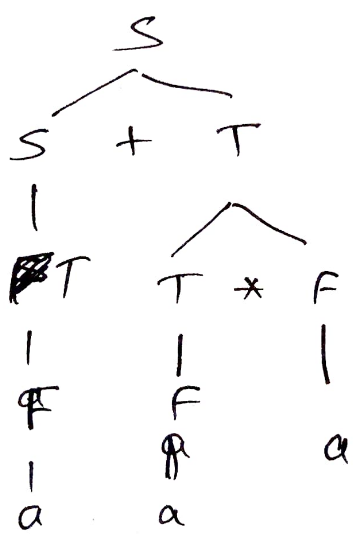
$T \rightarrow T*F \mid T/F$

$F \rightarrow (S) \mid a$

Lef associativity follow

$S = S$

$T = T$



⇒ $a+a*a$

* CFG for Parenthesis match:-

$S \rightarrow (S)$ can generate nested paranthesis

$S \rightarrow SS$ $\Rightarrow$ will allow Parenthesis to grow side ways

$S \rightarrow (S)$	$S \rightarrow SS$
$\rightarrow ((S))$	$\rightarrow (S)(S)$
$\rightarrow (((S)))$	$\rightarrow ((S))((S))$
	$\rightarrow (((S)))(((S)))$
Nesting	Sideways

$\therefore$ The production for above language is –

$P = \{ S \rightarrow (S) \mid SS \mid \epsilon \}$

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

~~Ex~~

$S \rightarrow (S)$	$S \rightarrow SS$
$\rightarrow ((S))$	$\rightarrow (S)(S)$
$\rightarrow (((S)))$	$\rightarrow ((S))((S))$
$\rightarrow ((()))$	$\rightarrow (())(())$