Chapter 3

Grammers & Languages

Chapter - III

* Grammer & Context free Grammer

Grammer - - set of formal rules
- checks correctness of sentence or syntax
- FA & Grammer are classified by chomsky

* Context free Grammer (CFG)

- A Context free grammer G is 4 tuple

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

where

V ⇒ variables, Non Terminal symbol
Σ ⇒ Terminal symbol {a,b} {0,1} etc
S ⇒ start symbol
P ⇒ Productions in grammer
S → A, S → a, A → b, A → ∈

- If L_1 & L_2 are Context free language then it will give
  1) Union
  2) Concatnation
  3) Kleene

$G_1 = (V_1, \Sigma, S_1, P_1)$ & $G_2 = (V_2, \Sigma, S_2, P_2)$

then

$G_U = (V_U, \Sigma, S_U, P_U)$
$G_C = (V_C, \Sigma, S_C, P_C)$
$G^* = (V, \Sigma, S, P)$

Regular Grammers

A Grammer $G=(V, \Sigma, S, P)$ is regular if every production take one of two forms

$$B \to aC$$
$$B \to a$$

$\forall$ B, C $\Rightarrow$ are variables / Non Terminal Symbol
a $\Rightarrow$ Terminal Symbol.

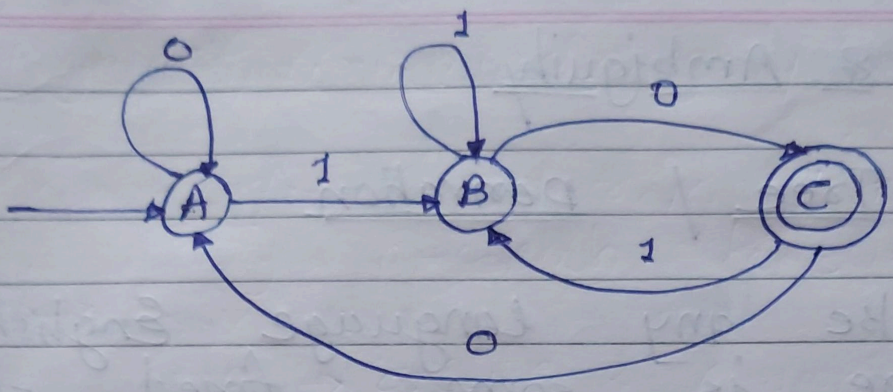
Eg:- Draw the regular language of following $L=\{0,1\}^*\{10\}$

ie set of all string over $\{0,1\}$ & end at 10.

Let take $x = 1100010$ ie string end with 10.

Substring Proceed so far	State.
$\wedge$	A
1	B
11	B
110	C
1100	A
11000	A
110001	B
1100010	C.

State or string end with 10 Contain Accepting or C state.



If we observe the lines of table then

$A \Rightarrow 1B \Rightarrow 11B \Rightarrow 110C \Rightarrow 1100A \Rightarrow 11000A$
$\Rightarrow 110001B \Rightarrow 1100010C.$

If we convert in Grammer then it will

$A \to 1B$
$B \to 1B$
$B \to 0C$
$C \to 0A$
$A \to 0A$
$A \to 1B$
$B \to 0C$

This of the form

$P \to aQ \quad \forall \quad a \in \{0,1\}$

$P \to aQ \Rightarrow \boxed{P \xrightarrow{a} Q} \to$ single production

$\underline{110C \Rightarrow 1100} \Rightarrow$ More than 1 Production

✱3.1✱ Derivation & Ambiguity

Derivation Tree / Derivation

- like any language English or other there is some fixed structure like that in CFG it contain also fixed pattern called derivation tree or parse tree.

Rules

1) Root node in parse / derivation tree is always labeled with start symbol.
2) All the terminal symbols are labeled with some terminal symbols of the grammer. so nodes are called terminal or leaf node
3) If string contain n symbols then it arranged from left to right
4) Non terminal symbols labbled with nonterminal string.
5) If the string contain {∧} empty symbol then it will be ignored.

Eg:- 1) If the Grammer $G = \{V, \Sigma, S, P\}$ in such a way that

① $V = \{S, A, B\}$ → variables

② $T/\Sigma = \{a, b\}$ → terminal symbol

(3) P = { S → AB productions
A → a
B → b
}

(4) S ⇒ starting symbol

Derivation Tree

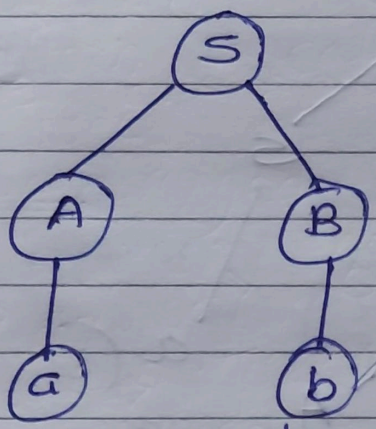


Fig:- Derivation tree for string ab

Eg 2) Draw the derivation tree for string aabbaa using following grammer

G = (Vn, Σ, P, S)

Vn = {S, A}, Σ = {a, b} &

P = {
S → aAS
A → SbA
S → a
A → ba
}

Now S is start symbol. so any string generated will stored at S.

$S \rightarrow aAS$

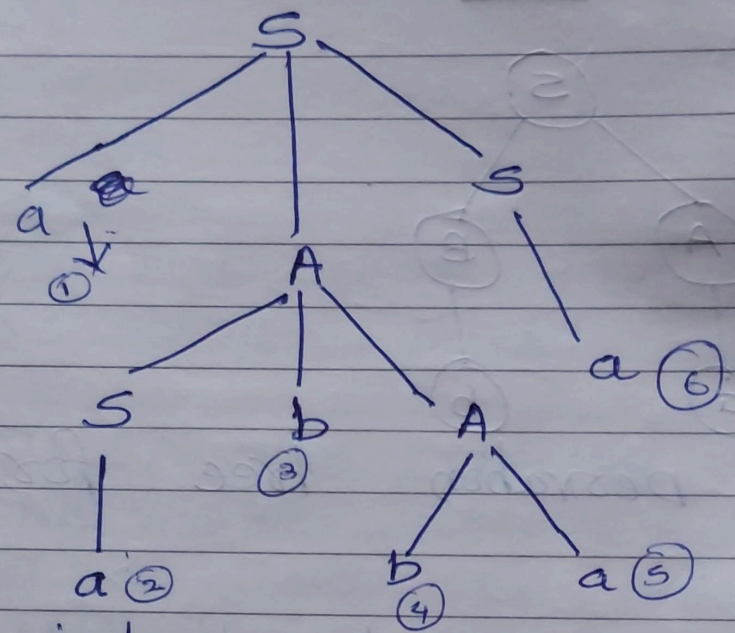


Fig:- Derivation Tree

Reading from left the string is aabbaa hence Ans.

Ambiguity / Ambiguous Grammer

In Case of CFG they called as Ambiguous if

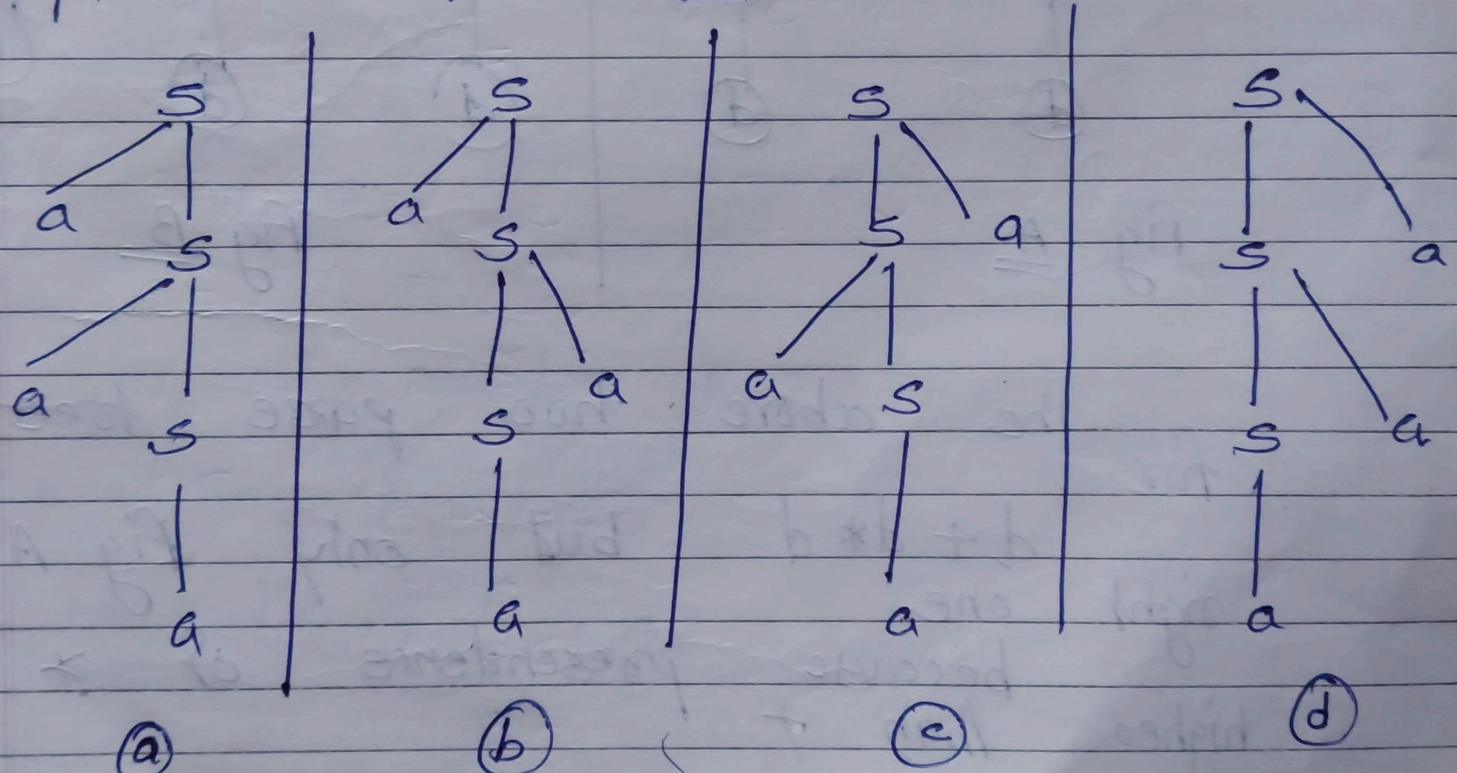
1) It contain terminal string with more than one parse tree.
2) More than one leftmost derivation or leftmost derivation

eg i) If $G = (V, \Sigma, P, S)$ be a grammer with

$G = (\{S\}, \{a\}, P, S\}$

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

$P = \{S \to aS \mid Sa \mid a\}$ find out parse tree for aaa.



by Above 4 way we can represent the aaa so it is ambiguous grammer

Eg 2) If $G = (V, \Sigma, S, P)$ be gramme with

$$G = (\{S\}, \{+, *, d\}, P, S)$$

$$P = \{S \rightarrow S+S*d\}$$

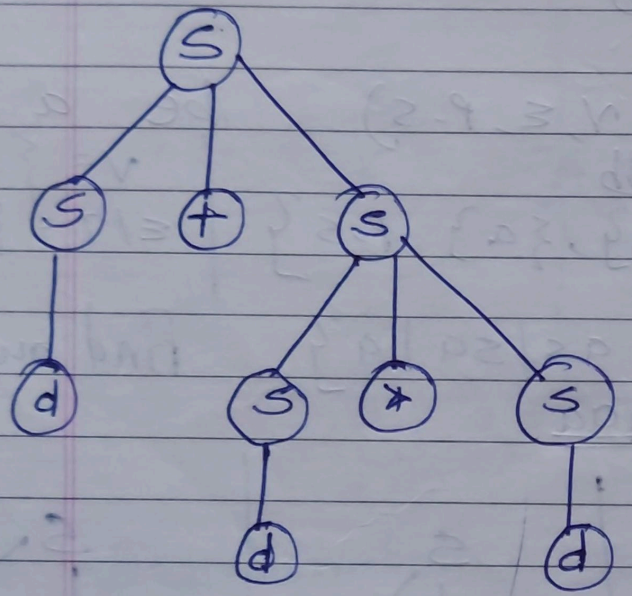


Fig A

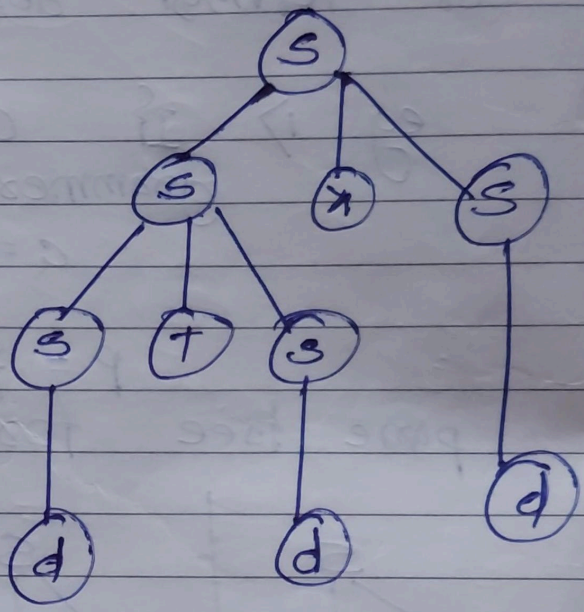
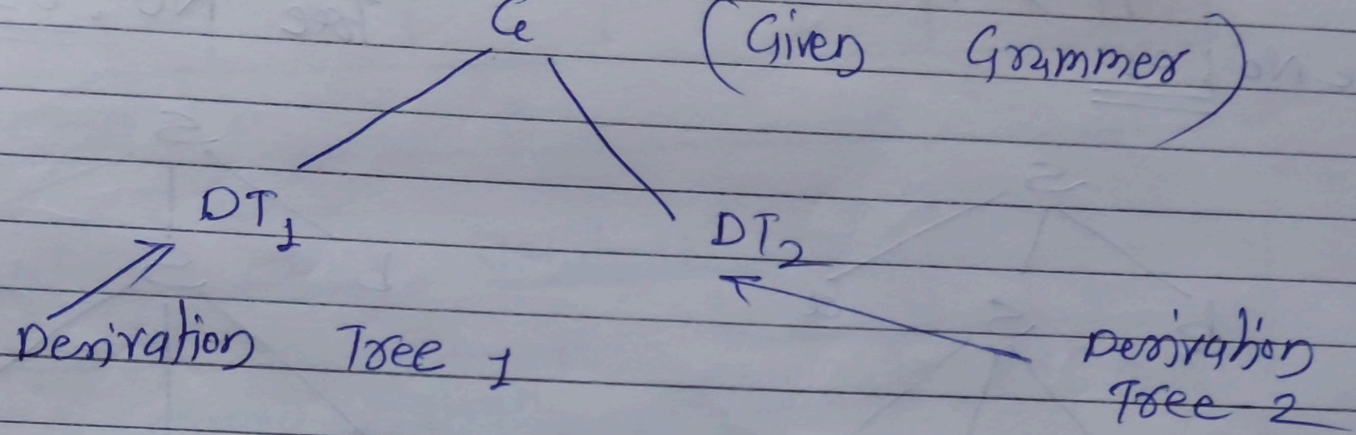


Fig B

For d+d*d the above two parse tree but only Fig A is right one because presendence of * is highes than +

Ambiguous Grammer

If given grammer having more than one derivation tree then it is called as ambiguous grammer.



eg. The following grammer.

$S \rightarrow A$

$S \rightarrow bSa$

$A \rightarrow \epsilon$

$A \rightarrow A \subset A$

If accepts string bb ccc aa draw the derivation tree & check weather it having ambiguous or not.

Now to find the derivation tree we having

- left derivation
- right derivation

Now we can draw following trees.

String ⇒ bbcccaa

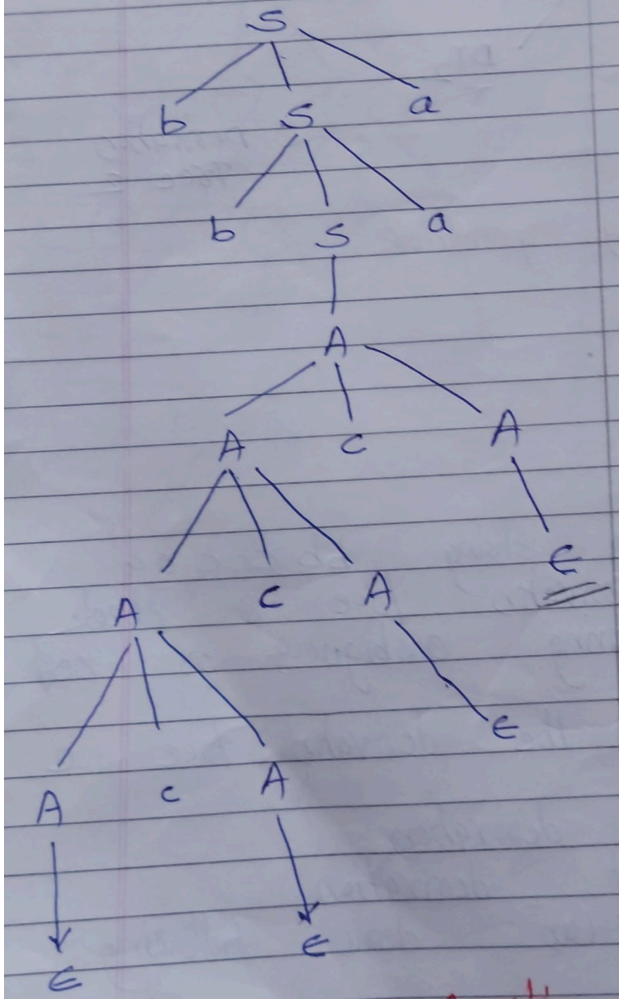
$S \to A$

$S \to bSa$

$A \to \epsilon$

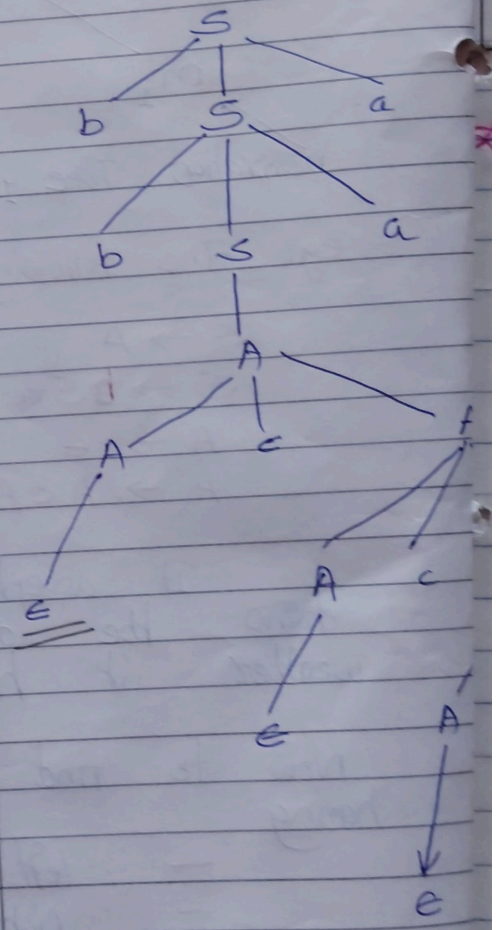
$A \to AcA$

Tree No: 01



bbcccaa Accepting

Tree No: 02



bbcccaa Accepting

Programme : Paper No. :

Roll No. : Date :

Subject : Sign of Supervisor :

Question No.	1	2	3	4	5	6	7	Total	Sign. of Examiner
Marks Obtained									
Maximum Marks									

No. of Supplements : 1 =

* BNF & CNF Notations :-

Convert given CFG into BNF & CNF

1) **BNF:-** Backus Naur Form (BNF)
CFG Notation with minor changes in original form.

Backus Normal form (BNF)

→ It is same as CFG.
→ It can add some convienent symbol like
1) Kleene *
2) Union |
3) ~~Metalanguage~~ / Metalevel parantheses ()

It is metalanguage for CFL
Essential in Compiler Construction
eg

digit := 0|1|2|3|4|5|6|7|8|9

Backus Normal Form

The Compact Notation, used to represent production rule called BNF

e.g. ① <rest> → <letter> <rest>

<rest> → <digit> <rest>

<rest> → ∈

by using BNF it will be

<rest> → <letter> <rest> | <digit> <rest> | ∈

⑪ S → SS

S → a

S → ∈

The BNF will be

S → SS | a | ∈

2) Chomsky Normal form (CNF)

If The given CFG only of the form

Nonterminal → string of ^exactly two Nonterminals

Nonterminal → one terminal

called as chomsky Normal form

eg

A → ab
B → Ab etc.

If they CFL contain ∈ symbol then convert into CFG & then to CNF with removal of ∈ in CNF

Eg :- $G = \{V, T, S, P\}$

$V = \{S, A, B\}$
$T = \{a, b\}$
P ⇒ S → bA / aB
A → bAA / aS / a
B → aBB / bS / a

Now take $b = C_b$ ∀ C_i = Non terminal Symbol
$a = C_a$

P ⇒ S → C_b / C_aB
A → C_bAA / C_aS / a

$B \rightarrow C_aBB / C_bS / a$

Now. first rule / stmt is in CNF

Now

$A \rightarrow C_bAA / C_aS / a$

Now $AA \rightarrow D$

$BB \rightarrow E$

$A \rightarrow C_bD / C_aS / a$

$B \rightarrow C_aE / C_bS / a$

Hence required CNF

Eg:-

1) $S \rightarrow aSa / bSb / a / b / aa / bb.$

2) $S \rightarrow 1A / 0B$

$A \rightarrow 1AA / 0S / 0$

$B \rightarrow 0BB / 1.$

★ Convert CFG into CNF

i) $S \to ASA$

$S \to BSB$

$S \to AA$

$S \to BB$

$S \to a$

$S \to b$

$A \to a$

$B \to b$

Soln - Expect 1st & 2nd production all productions are in CNF

i) $S \to AX_1$

$X_1 \to SA$

ii) $S \to BX_2$

$X_2 \to SB$

Hence required CNF is

$S \to AX_1$

$S \to BX_2$

$S \to AA$

$S \to BB$

$S \to a$

$S \to b$

$A \to a$

$B \to b$

eg.

(a) $S \rightarrow ABA$
$A \rightarrow aA \mid \epsilon$
$B \rightarrow bB \mid \epsilon$

(b) $S \rightarrow ABC$
$A \rightarrow a \mid b$
$B \rightarrow Bb \mid bb$
$C \rightarrow aC \mid cc \mid ba$

1) Steps for Conversion of CFG to CNF

(a) Eliminate ϵ productions

(b) Eliminate unit productions

(c) Eliminate terminal on RHS

(d) Restrict the no of variables on RHS.

☆ Converting CFG to CNF

Let → G be the grammer of the productions.

S → AACD
A → aAb | Λ
C → aC | a
D → aDa | bDb | Λ

Convert into CNF.

1) Eliminating Λ Productions

In above example the nullable variables are A & D.

S → AACD | ACD | AAC | CD | AC | C

A → aAb | ab

C → aC | a

D → aDa | bDb | aa | bb

2) Eliminating Unit Productions

Now from Production i) & iii) it is

S → aC | a

∴ we delete S → C

3) Restricting the right side of production to single terminals or string of two or more variables

$S \rightarrow AACD \mid ACD \mid AAC \mid CD \mid AC \mid X_aC \mid a$

$A \rightarrow X_aAX_b \mid X_aX_b$

$C \rightarrow X_aC \mid a$

$D \rightarrow X_aDX_a \mid X_bDX_b \mid X_aX_a \mid X_bX_b$

$X_a \rightarrow a$

$X_b \rightarrow b$

4) Final step to CNF

$S \rightarrow AT_1 \quad T_1 \rightarrow AT_2 \quad T_2 \rightarrow CD$

$S \rightarrow AU_1 \quad U_1 \rightarrow CD$

$S \rightarrow AV_1 \quad V_1 \rightarrow AC$

$S \rightarrow CD \mid AC \mid X_aC \mid a$

$A \rightarrow X_aW_1 \quad W_1 \rightarrow AX_b$

$A \rightarrow X_aX_b$

$C \rightarrow X_aC \mid a$

$D \rightarrow X_aY_a \quad Y_1 \rightarrow DX_a$

$D \rightarrow X_bZ_1 \quad Z_1 \rightarrow DX_b$

$D \rightarrow X_aX_b \mid X_bX_b$

$X_a \rightarrow a$

$X_b \rightarrow b$

2.* ☆ BNF & CNF Notations
[Boyce Code Normal form
Chomskey Normal form]

▷ Chomskey Normal form

In a CFG the production is of the form $A \to BC$, $A \to a$

& $A \to 2NJ$, $S \to IS$

ie

Nonterminal / variables → string of two non terminals

Nonterminal → Terminal Symbol.

then they called Chomskey Normal form

Eg:-

If two productions $A \to aAb$ & $B \to ab$ &

replace

$X_a \to a$, $X_b \to b$

$A \to X_a A X_b$

$B \to X_a X_b$

Also

$A \to BCDBCE$

would be replace by

$A \to BY_1$

$Y_1 \to CY_2$

$Y_2 \rightarrow DY_3$
$Y_3 \rightarrow BY_4$
$Y_4 \rightarrow CE$

Hence - New variables Y_1, Y_2, Y_3 Y_4 are productions so it is CNF.

- If CFG $G = (V, \Sigma, S, P)$ be in CNF & $G' = (V', \Sigma, S, P')$ then

$$L(G') = L(G) - \{\Lambda\}$$

* BNF

[Backus Naur Form]

It is the one way to represent the Regular language.

- Instead of writing E we writing <exp> in the statement.
- eg
  1) <E> <Addition> will be
     <Exp> <Addition>
  2) <E> <a> <+> <b> will be written eg
     <Exp> <a> <+> <b>

3) if $\langle a \rangle + \langle b \rangle \rightarrow \langle c \rangle$ is the any expression then in backus Naur form it is written as

$$\langle a \rangle + \langle b \rangle = \langle c \rangle$$

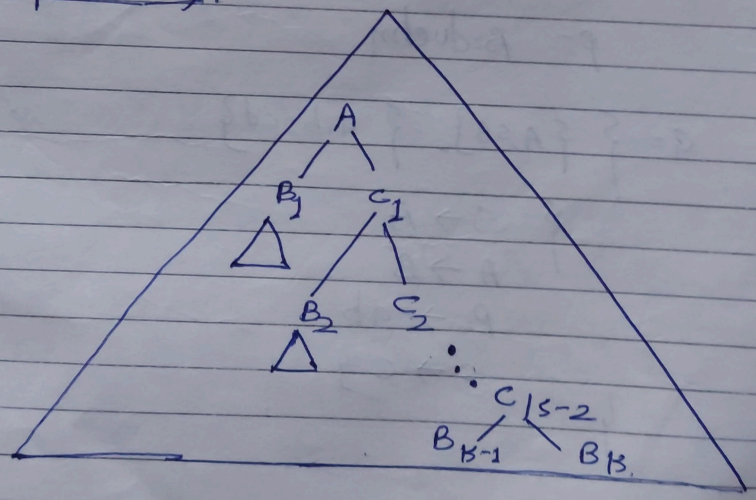
4) To higher expression we can use Extended Backus Naur form to write it.

Eg:- If an example gives an productions in such a way that

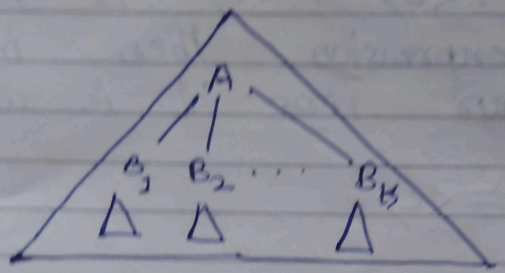
$A \rightarrow B_1 C_1$, $C_1 \rightarrow B_2 C_2$, $C_2 \rightarrow B_3 C_3$, ...

$C_{k-2} \rightarrow B_{k-1} B_k$.

Representation:-



By Using Chomsky Normal form



* Union, Concatenation & *'s of CFL's

$G = V, \Sigma, S, P$

$V, \Sigma / T, S, P$

V = set of variables

T = Terminal Symbol

S = starting symbol

P = Production

$q = \{ \{A, S, B\}, \{a, b, c, d\}$

$\{ S \to A$

$A \to B$

$B \to ab$

$S \to c \}$

$\}$

If L_1 & L_2 are CFL then $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^* also CFL.

Proof : By Constructive method.

Consider $G_1 = \{V_1, \Sigma, S_1, P_1\}$ &

$G_2 = \{V_2, \Sigma, S_2, P_2\}$ generating

L_1 & L_2 respectively

① Union = $G_u = \{V_u, \Sigma, S_u, P_u\}$

generating $L_1 \cup L_2$.

If necessary we rename the elements of V_2 then

$V_1 \cap V_2 = \phi$

$V_u = V_1 \cup V_2 \cup \{S_u\}$

$\because S_u$ = New start symbol not in V_1 & V_2

$P_u = P_1 \cup P_2 \cup \{S_u \rightarrow S_1 | S_2\}$

if x is in L_1 or L_2 then

$S_u \Rightarrow^* x$ in grammer G_u

because we start the derivation in

$S_u \to S_1$ or $S_u \to S_2$ & continue the derivation of x in G_1 or G_2

$\therefore \quad L_1 \cup L_2 \subseteq L(G_u)$

If x is derivable from S_u in G_u the first step in any derivation must be

$S_u \Rightarrow S_1$ or $S_u \Rightarrow S_2$

In the first case all subsequent productions are must be production in G_1 because no variables in V_2 are involved. & thus $x \in L_1$ & second case

$x \in L_2$

$\boxed{L(G_u) \subseteq L_1 \cup L_2}$

ii) Concatenation

A grammer $G_c = (V_c, \Sigma, S_c, P_c)$ generating $L_1 L_2$

$\forall \quad V_1 \cap V_2 = \phi$

$V_c = V_1 \cup V_2 \cup \{S_c\}$

$P_c = P_1 \cup P_2 \cup \{S_c \to S_1 \cdot S_2\}$

if $x \in L_1 L_2$ then $x = x_1 x_2 \ \forall \ x_i \in L_i$ for i we may then derive x in G_c as follows

$S_c \Rightarrow S_1 S_2 \Rightarrow^* x_1 S_2 \Rightarrow^* x_1 x_2 = x$

x_1 in G_1 &
x_2 in G_2

if x is derived from S_c then $S_c \Rightarrow S_1 S_2$ x must be derived from $x = x_1 x_2$

$\forall$ for each i, x_i can be derived from S_i in G_c

$\therefore V_1 \cap L_2 = \phi$, being derivable from S_i in G_c means being derivable from S_i in G_i & so

$$\boxed{x \in L_1 L_2}$$

iii) Kleene :-

A grammer $G^* = \{V, \in, S, P\}$ generating L_1^*

$$V = \{V_1 \cup \{S_*\}\}$$

$\forall \quad S_1 \notin V_1$

L_1^* contains string of the form

$$x = x_1 x_2 \cdots x_k$$

$\forall \quad x_i \in L_1$

Each x_i is derived from S_1, then to derive x from S_* it is enough to be able to derive a string of k S_1's

$S_* \to S_1 S \mid \Lambda$ in P

$\therefore \quad P = P_1 \cup \{S_* \to S_1 S \mid \Lambda\}$

If $x \in L(G^*)$ then on other hand $x = \Lambda$ or x can be derived from some string of the form S_1^k in G^*. 2nd case only productions in G beginning with S_* are those G_1 we come

$$\boxed{x \in L(G_1)^k \subseteq L(G_1)^*}$$

* Eliminating useless variables from a Context free Grammar *

A Context free Grammar is $\{V, \Sigma, S, P\}$ contains some

V = Nonterminal

T = Terminal symbol.

If the given Nonterminal symbol is not in use called as useless symbol or variable.

Now Consider following grammer

$G = \{V, \Sigma, S, P\}$

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

$\Sigma = \{a, b\}$

$S \rightarrow AA \mid 1A0 \mid B \mid 0A1 \mid 01$ —— (I)

A & B are Nonterminals & 0 & 1 are terminals

$S \rightarrow AA \mid 1A0 \mid 0A1 \mid 01$ —— (II)

B is useless symbol

both (I) & (II) are identical

Remove those strings whose string contain at least one non used string

Eg:-

$G = \{ V, T, S, P \}$

$V = \{ S, C, D \}$

$T = \{ d \}$

$P \Rightarrow S \rightarrow CD \mid d$

$D \rightarrow d$

from above production we can say C is useless

so we can remove C

$S \rightarrow d$

$D - d.$

$G = \{ \{S\}, \{d\}, \{S \rightarrow d, D \rightarrow d\} \}$

Now we observe $S \rightarrow d$ doesn't contain any Nonterminal symbol D in it so it become

$\boxed{S \rightarrow d}$ only

by removing $D \rightarrow d$.

* Eliminating ϵ production from a Grammer.

If The CFL contains ϵ transitions in given CFL then convert it into CFG the remove those transitions

☆ Removing Unit Productions

Production of the form

$A \to B$, $B \to S$.

are called unit production

eg.

$S \to A|bb$ — (I)

$A \to B|b$ — (II)

$B \to S|a$ — (III)

from (I) & (III)

$B \to bb|a$ — (IV)

(A) from (II) & (IV)

$A \to bb|a|b$ — (V)

from (I) & (II) & (V)

(A) $S \to b|bb$

The final result will be

$S \to bb$

$A \to b$

$B \to a$

☆ Remove Null / Epsilon Product

$S \to AB$
$A \to aA \mid a$
$B \to bB \mid \epsilon$.

$\Rightarrow$

$S \to AB \mid A$
$A \to aA \mid a$
$B \to bB \mid b$

Now we have epsilon symbol.

$B \to \epsilon$

Initially
$S \to AB$
$A \to aA \mid a$
$B \to bB$

Step II Now replace $B \to \epsilon$ whereever applicable at right side

$S \to A\epsilon$
$A \to aA \mid a$
$B \to b\epsilon$

ie.
$S \to A$
$A \to aA \mid a$
$B \to b$

Hence final result will be

$S \to AB \mid A$
$A \to aA \mid a$
$B \to bB \mid b$