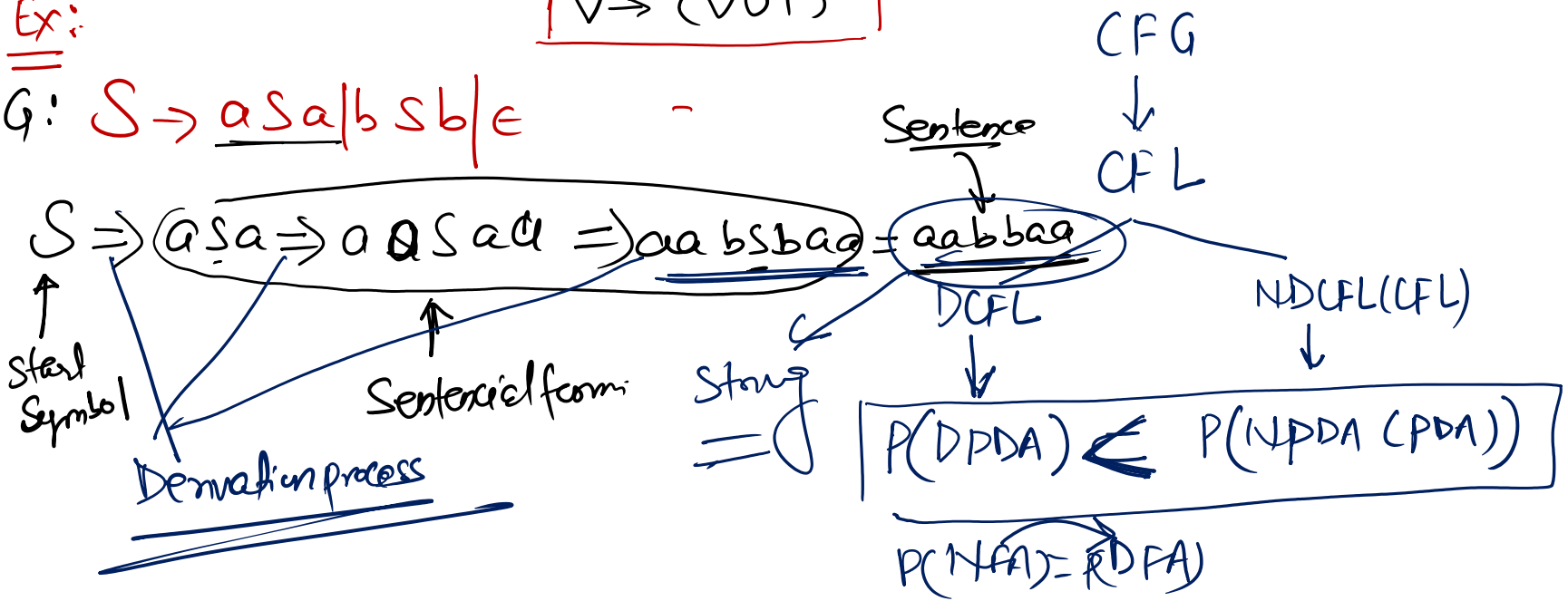
Context Free Grammar (CFG) :- $A \to \alpha$, where $A \in V$, $\alpha \in (V \cup T)^*$

$V \to (V \cup T)^*$

Ex:

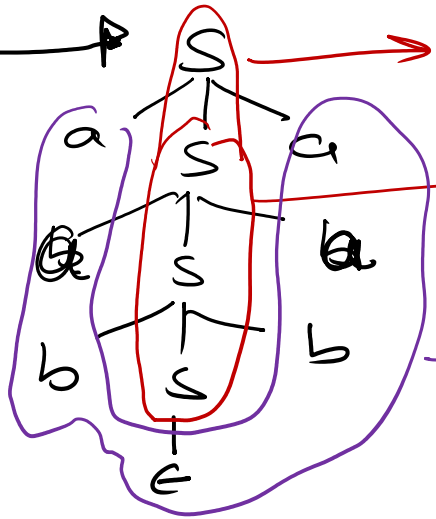
G: $S \to aSa \mid bSb \mid \epsilon$



Derivation Tree or Parse Tree :- Graphical Representation of Derivation Process

$S \rightarrow aSa \mid bSS \mid \epsilon$

$S \Rightarrow aSa \Rightarrow aaSSaa \Rightarrow aaSSbaa \Rightarrow aaS\epsilon baa \Rightarrow aabbaa$



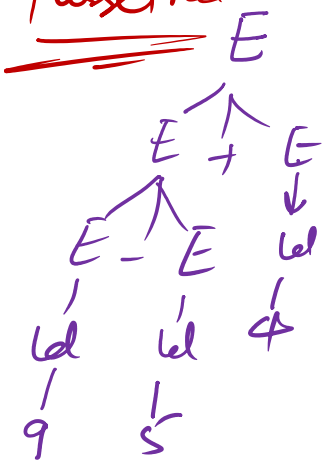
Root node Always Start Symbol

Non terminal

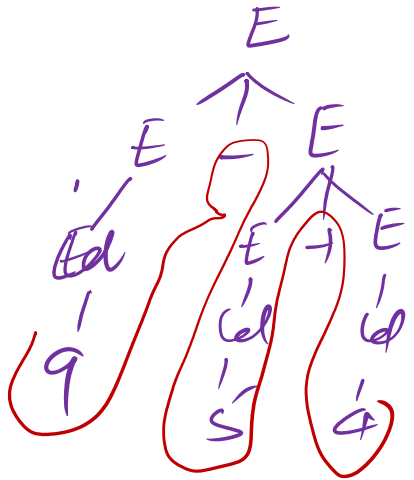
Terminal Symbol

$E \rightarrow E+E \mid E-E \mid id$ $\quad \omega = 9-5+4$

Parse Tree



or



9-5+4

9-5+4

Derivation Process

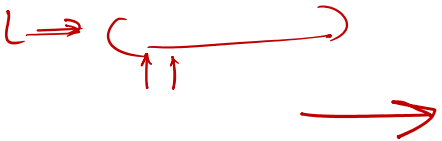
Leftmost Derivation process → LMD

Right most Derivation process → RMD

Derivation process :

There are two types of derivation process,

1. Left most derivation (LMD)
2. Right most derivation (RMD)



Left most derivation (LMD):

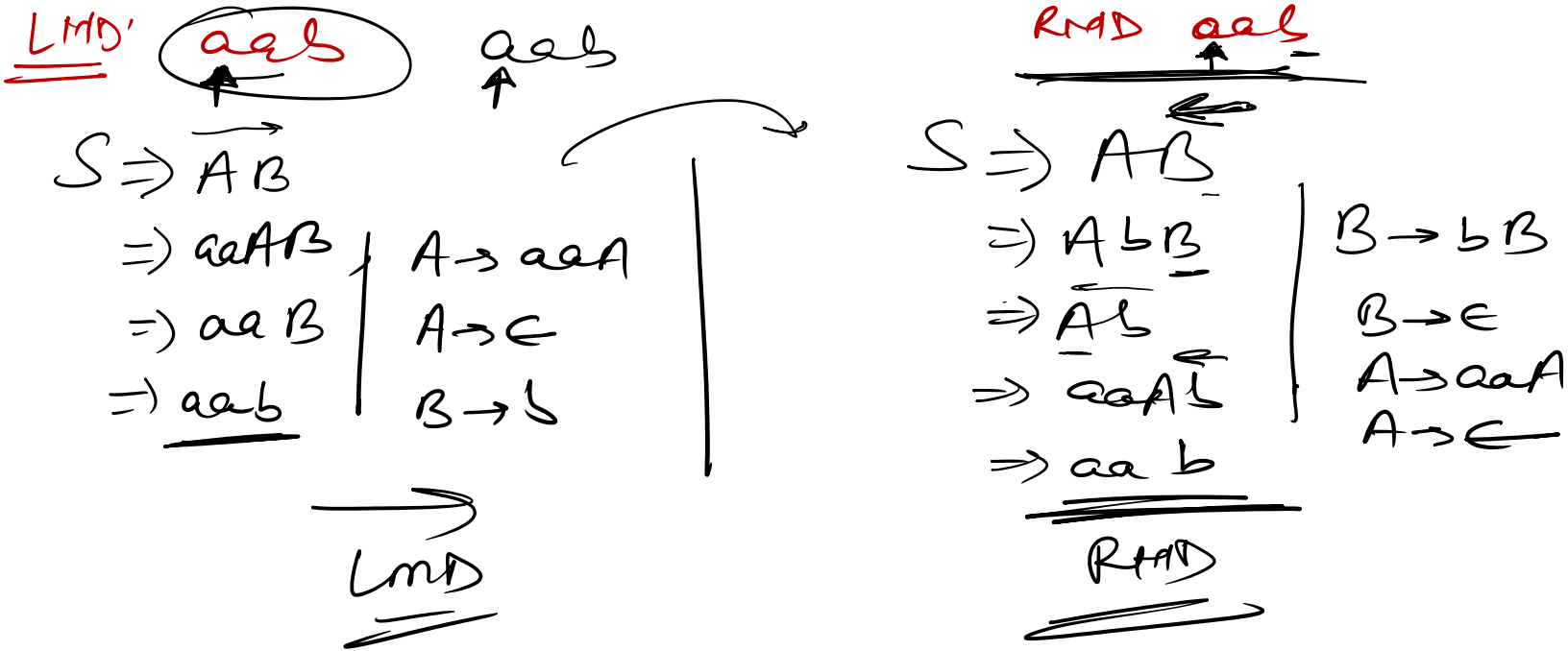
A derivation $S \Rightarrow^{*} w$ is called a leftmost derivation if we apply a production only to the leftmost variable at every step.

Right most derivation (RMD):

A derivation $S \Rightarrow^{*} w$ is called a rightmost derivation if we apply a production to the right most variable at every step.

Example: Consider a grammar G for the language, $L = \{a^{2n}b^m, m, n \geq 0\}$

G: S → AB, A → aaA | ε, B → bB | ε, find the LMD and RMD for string w = aab

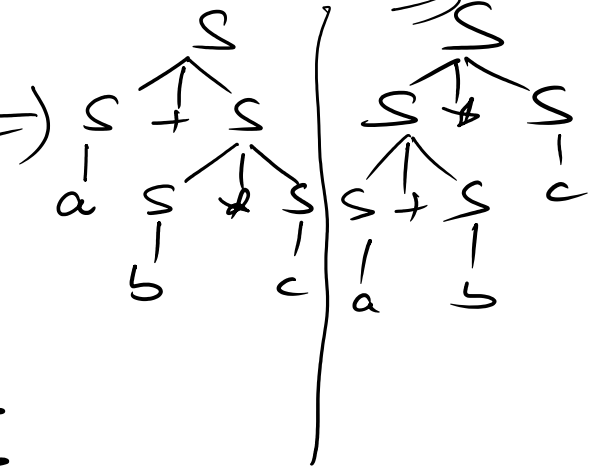


Example: G: S → S+S | S*S | a | b | c, find the LMD and RMD for string w = a + b * c

w = a + b * c

LMD:

S ⇒ S+S
⇒ a+S
⇒ a+S*S
⇒ a+b*S
⇒ a+b*c



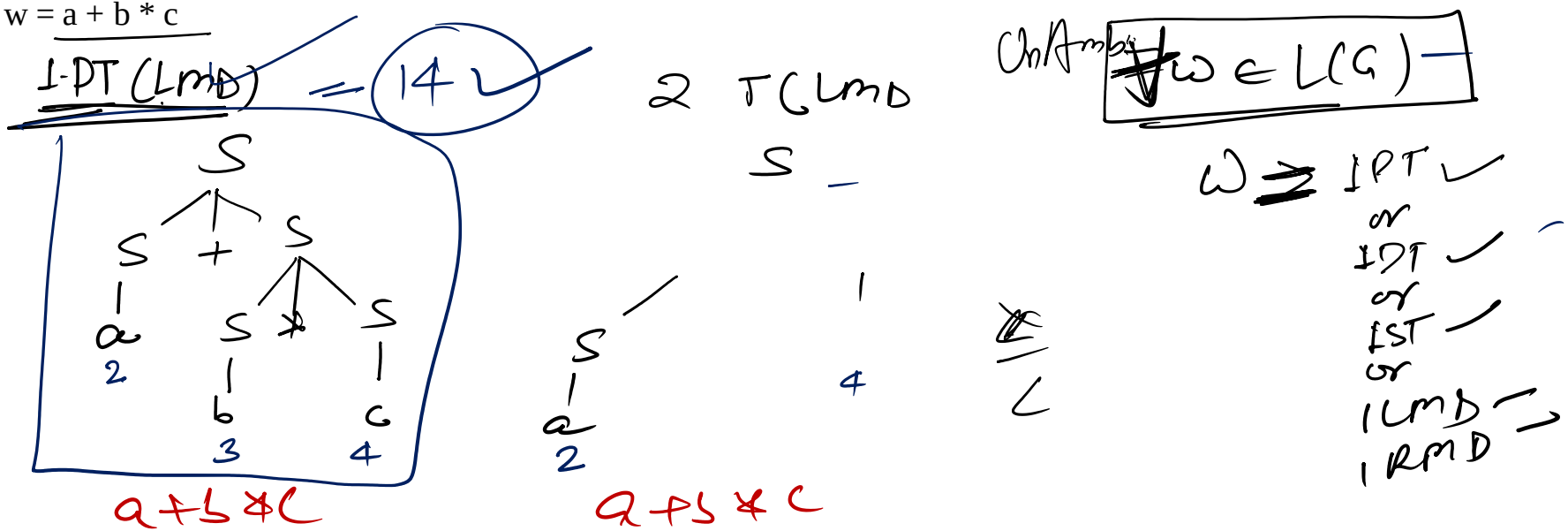
w = a + b * c

S ⇒ S*S
⇒ S*c
⇒ S+S*c
⇒ S+b*c
⇒ a+b*c

Ambiguous Grammar

A grammar G, is ambiguous grammar if ∃w ∈ L (G), such that w has > one Parse tree either using 2 LMD or 2 RMD i.e., a grammar G is ambiguous if there is more than one Parse Tree or LMD / RMD for a string w ∈ L (G).

Example: G: S → S+S | S*S | a | b | c, Grammar G is ambiguous because there exist two different LMD for a string w = a + b * c

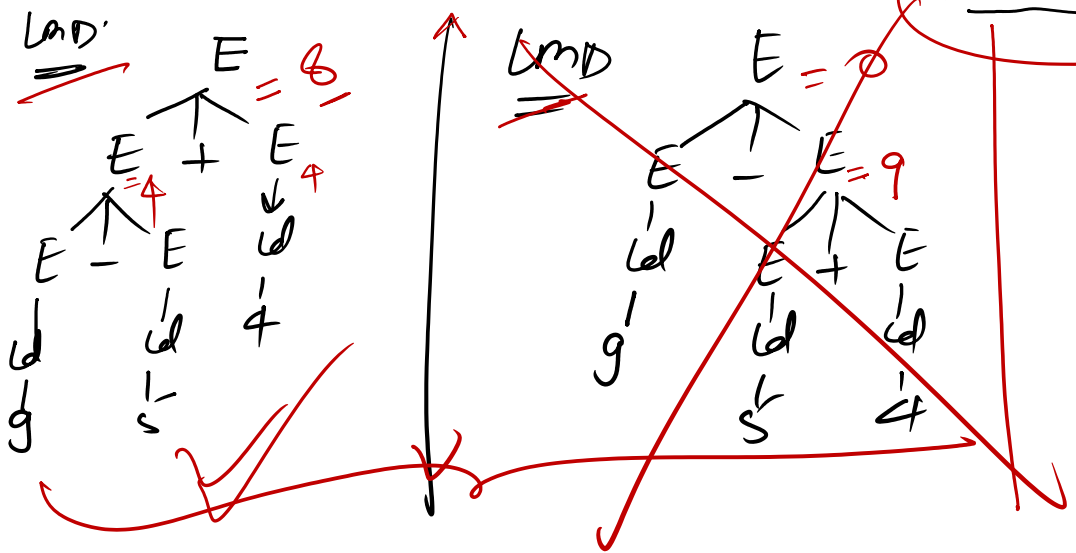


NOTE:

1- A grammar G is unambiguous if there exist exactly one parse tree or LMD / RMD for all the string, $w \in L(G)$

2- If a grammar G is ambiguous, it doesn't mean language (L) is ambiguous.

Example: State whether given grammar is ambiguous or not. G: E → E+E | E-E | id



⇒ 9-5+4

9-5+4 ≠ 8

$L = \{a^n \mid n \geq 0\}$

G_1: $S \to aS \mid a \mid \epsilon$ → Ambiguous

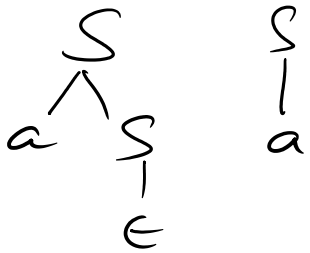
G_2: $S \to aS \mid Sa \mid \epsilon$ → Ambiguous

G_3: $S \to Sa \mid a \mid \epsilon$ → Ambiguous

G_4: $S \to aS \mid \epsilon$ → Unambiguous

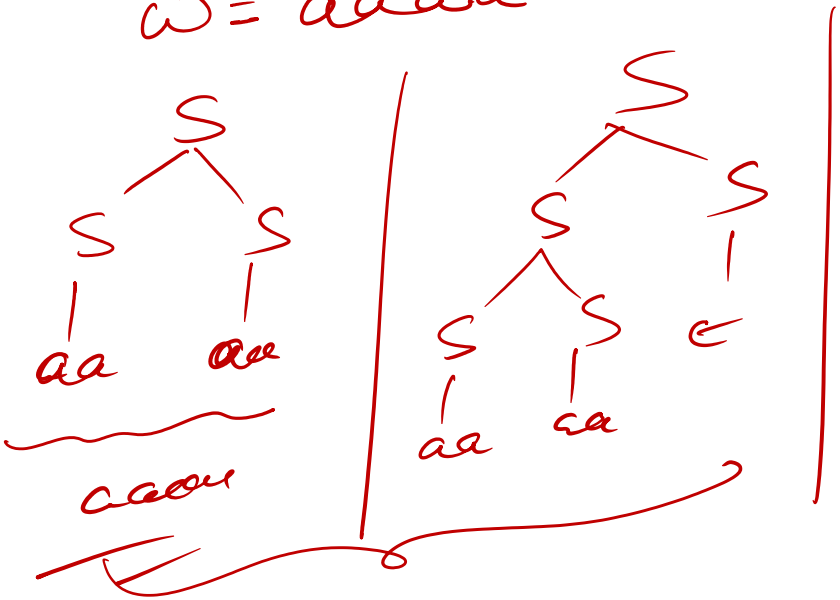
G_n.

a



$S \rightarrow aa \mid bb \mid ss \mid \epsilon$ Ambiguous Grammar

$\omega = aaaa$



RG → Ambiguous (NFA) ↘
RG → Unambiguous (DFA)

RL → Unambiguous

DCFL → Always Unambiguous

NOTE:

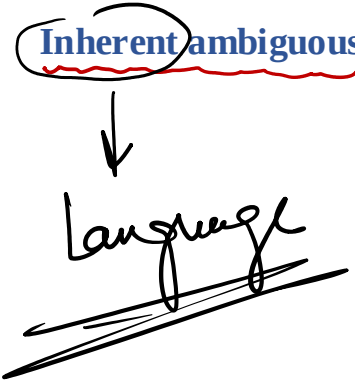
- Regular Grammar:
  - Ambiguous (NFA)
  - Unambiguous (DFA)
- Regular language is always unambiguous.
- DCFL is always unambiguous language
- Ambiguity starts from CFL

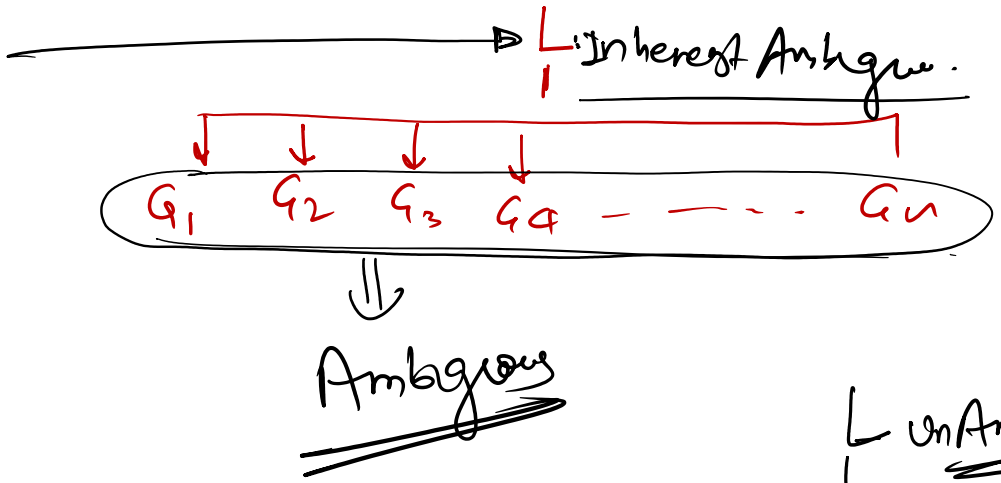
$G \rightarrow L(G) = \{\omega_1, \omega_2, \omega_3, \omega_4 \ldots\}$

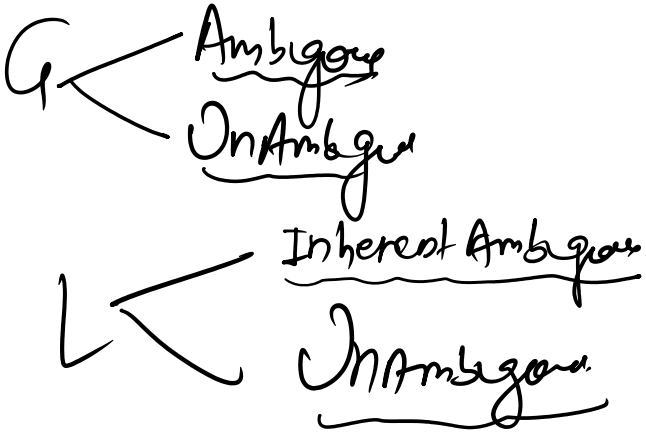
$\exists \omega \in L(G)$

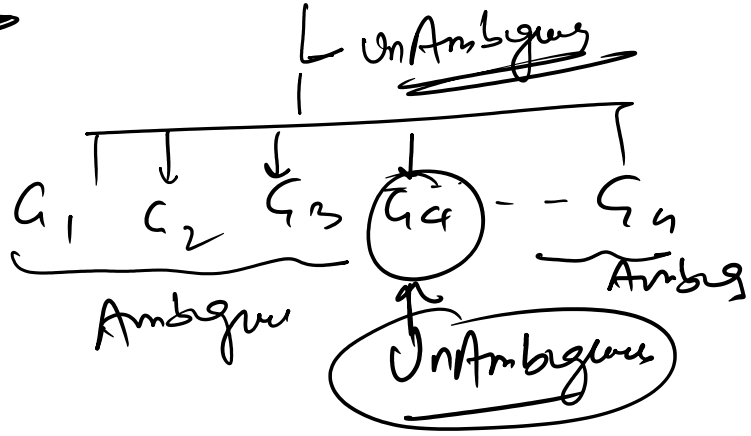
$\rightarrow$ > 1 PT

Inherent ambiguous language:

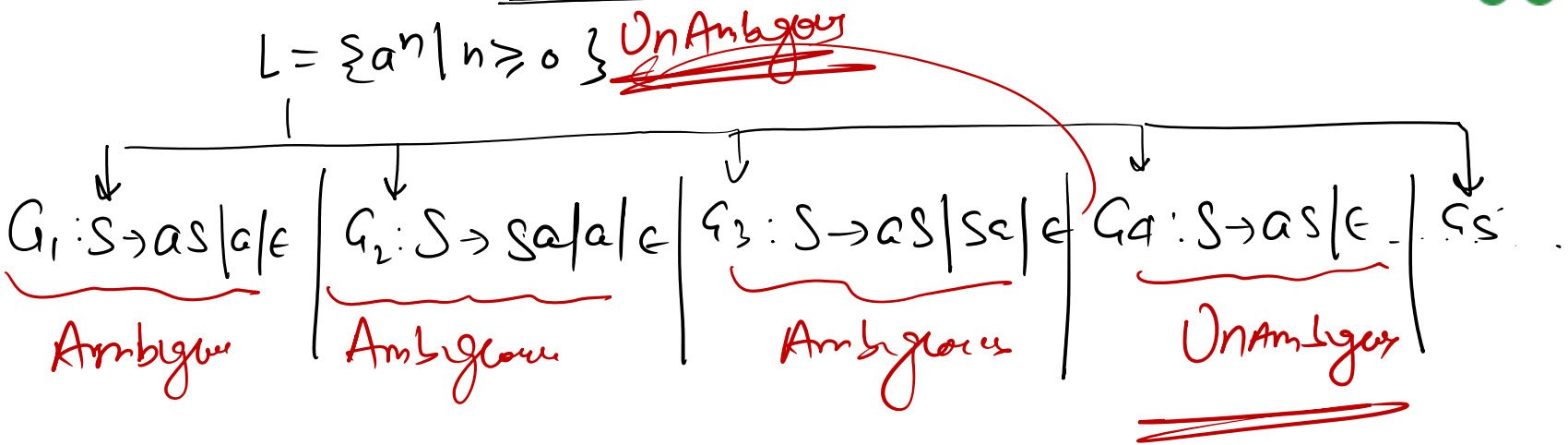








Example: Check whether the given language L = $\{a^n \mid n \geq 0\}$ is an inherently ambiguous or not



Example: Given language L = {$a^m b^n c^k$ | m, n, k ≥ 1, either m = n or n = k } is Inherent Ambiguous?

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

$L = L_1 \text{ or } L_2$

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

A B C D

$L_1 = \{a^n b^n c^k \mid n, k \geq 1\}$

AND

$L_2 = \{a^m b^k c^k \mid m, k \geq 1\}$

$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1$

G:

$S \to AB \mid CD$

$A \to aAb \mid ab$

$B \to cB \mid c$

$C \to aC \mid a$

$D \to bDc \mid bc$

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

L_3 L_4

Inherent Ambiguous

$L_3 \cap L_4 = \{a^n b^n c^n d^n \mid n \geq 1$

Remove Ambiguity: →

It is not always possible to convert an ambiguous grammar into an unambiguous grammar because ambiguity finding & removal both are undecidable

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

Remove Ambiguity by precedence & associativity rules:

An ambiguous grammar may be converted into an unambiguous grammar by implementing:

- Precedence Constraints
- Associativity Constraints

These constraints are implemented using the following rules:

Rule 1: The precedence constraint is implemented using the following rules:

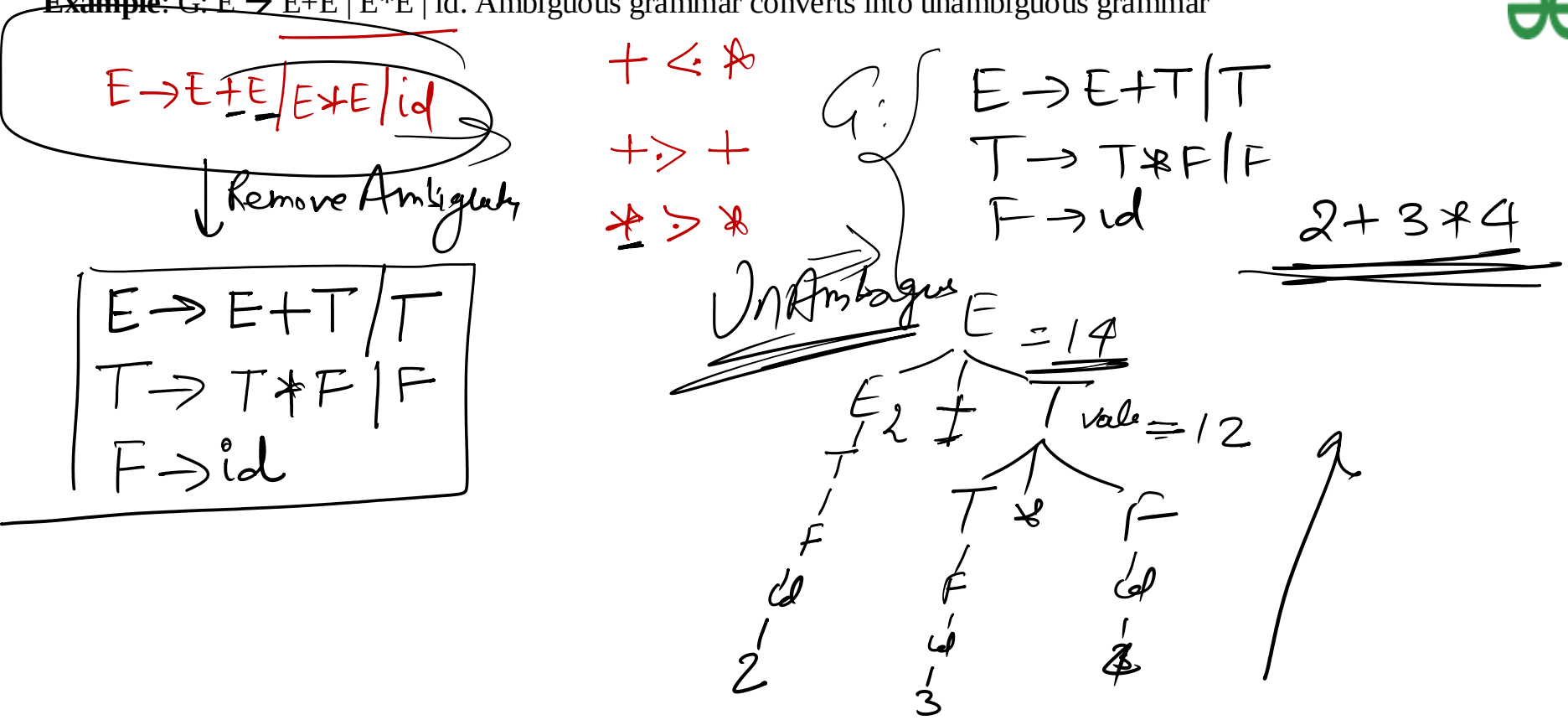
- The level at which the production is present defines the priority of the operator contained in it.
- The top level of the production, the lower the priority of the operator.
- The bottom level of production, the higher the priority of the operator.

Rule 2: The associativity constraint is implemented using the following rule:

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

If the operator is right associative, induce right recursion in its production

Example: G: E → E+E | E*E | id. Ambiguous grammar converts into unambiguous grammar



Example: Convert the following ambiguous grammar into unambiguous grammar.

G: E → E+E | E*E | E^E | id

E → E+E

E → E*E

E → E^E

E → id

Unambiguous Grammar

Apply Rule 1 & 2

E → E + T | T (+) left associative

T → T * F | F (*) left associative

F → G ^ F | G (^) Right associative

G → id

The priority order: **id** . > ^ . > * . > +

Associativity: + and * are left associative and ^ operator is right associative

Example: Find the precedence and associativity

i) G1: A → A $ B | B
B → B # C | C
C → D @ C | D
D → d

$, #, @

@ .> # .> $

@ <. @

.>

$.> $

ii) G2: E → E * F | F + E | F
F → F − F | id

− .> [* = * = +]

* .> *

+ <. +

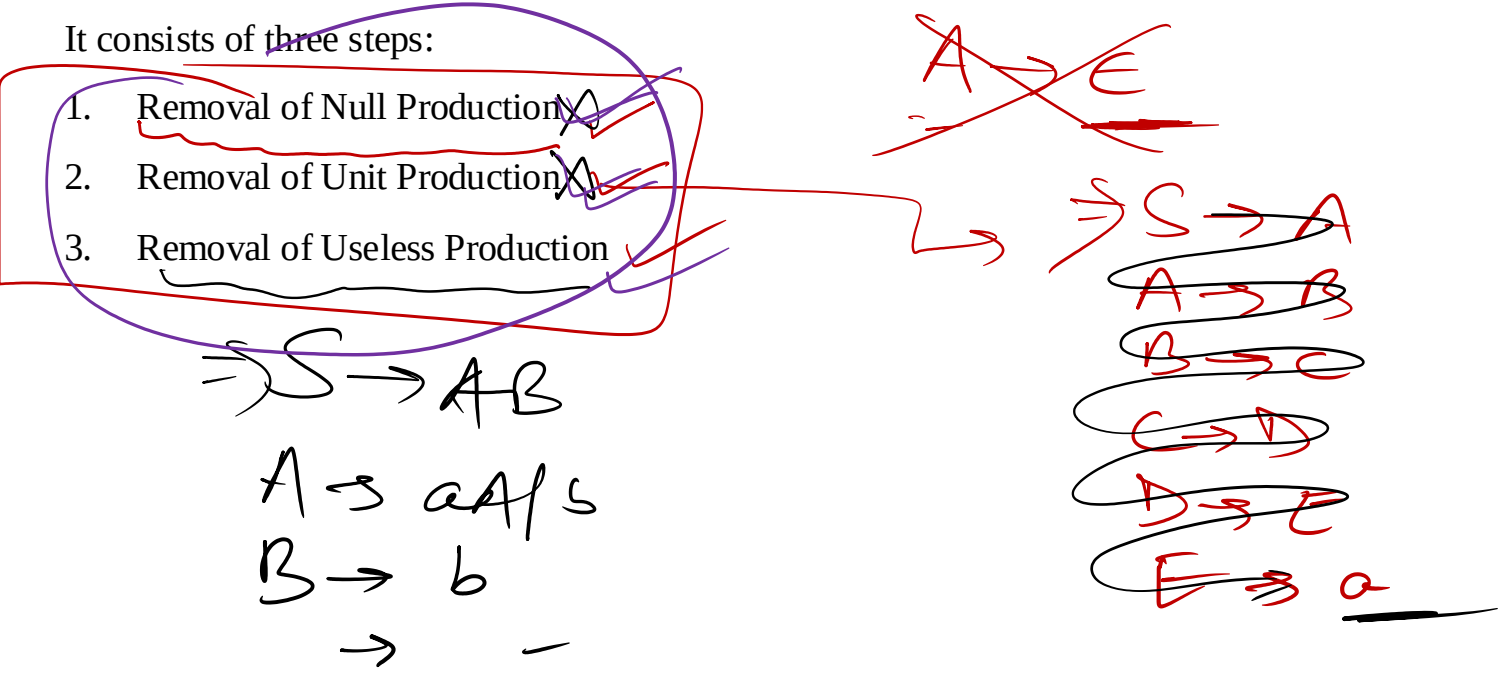
− −

Simplification of CFG

In CFG, it may happen that all the production rules and symbols are not needed for the derivation of strings. Elimination of these productions and symbols is called simplification of CFG.

It consists of three steps:

1. Removal of Null Production
2. Removal of Unit Production
3. Removal of Useless Production



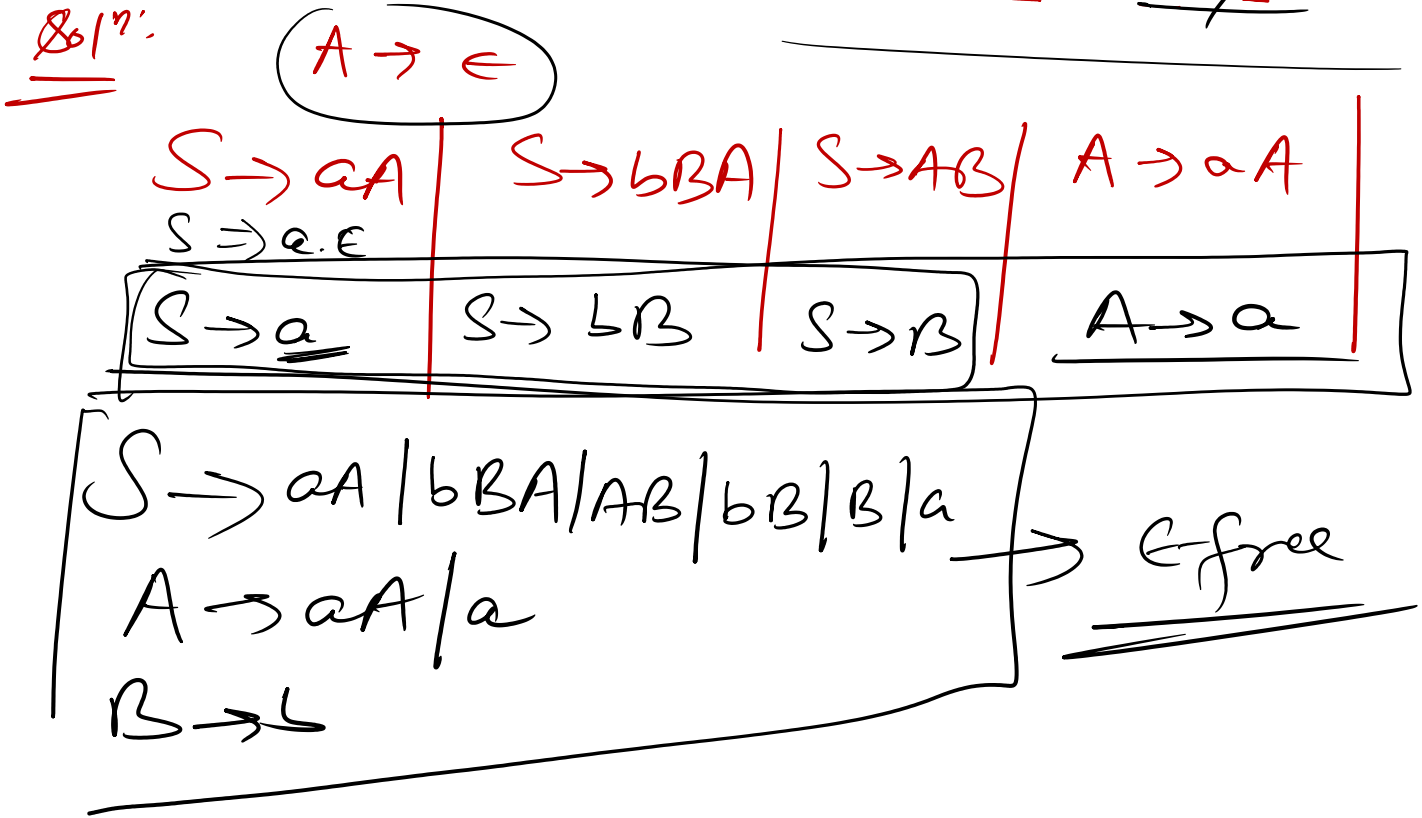
Removal of Null Production

Any production of the form X → ε should be removed

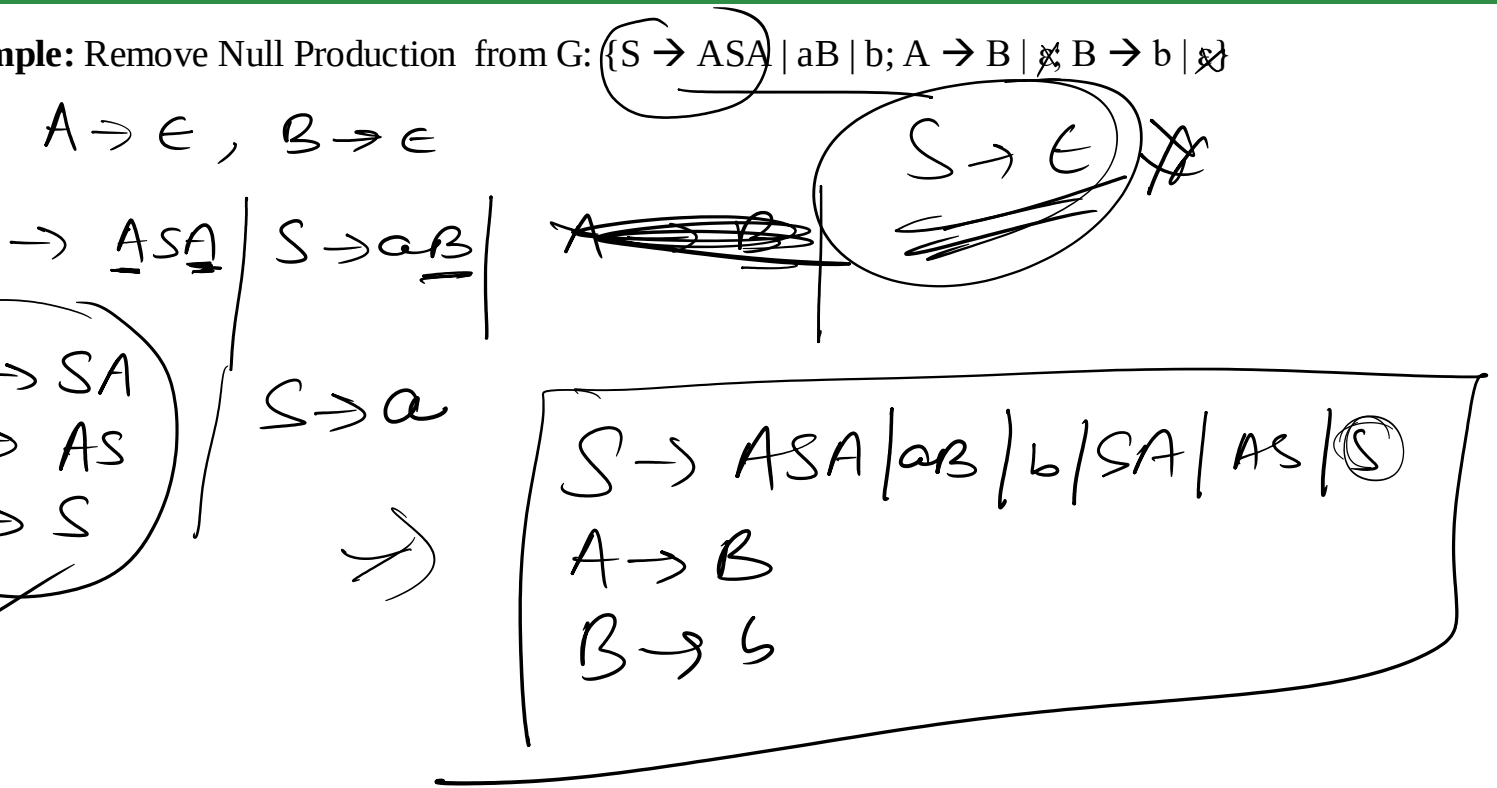
Removal Procedure

1. Find all variables which derive ε. E.g., X→ ε
2. If variable X is present in any production, then remove X from the production.
3. Combine original production with above step & remove ε - productions.

Example: Remove Null Production from G: {S → aA| bBA | AB; A → aA | ∈ B → b}



Example: Remove Null Production from G: {S → ASA | aB | b; A → B | ∈; B → b | ∈}



Q: $S \to aS \mid bA \mid \epsilon$, $A \to aAb \mid \epsilon$

$\epsilon \in L(G)$?

X

$S \to \epsilon$ $A \to \epsilon$

$S \to aS \mid bA \mid \epsilon$ | $A \to aAb$

$S \to b$ | $A \to ab$

$S \to aS \mid bA \mid b \mid \epsilon$, $A \to aAb \mid ab$

Ans

Removal of Unit Production

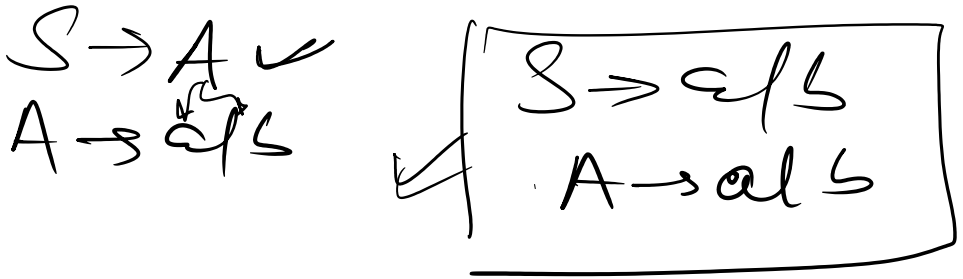
- Any rule in the form <variable> → <variable> is called unit production (X→Y)
- Substitute all Y products if Y is generating any terminal symbol.

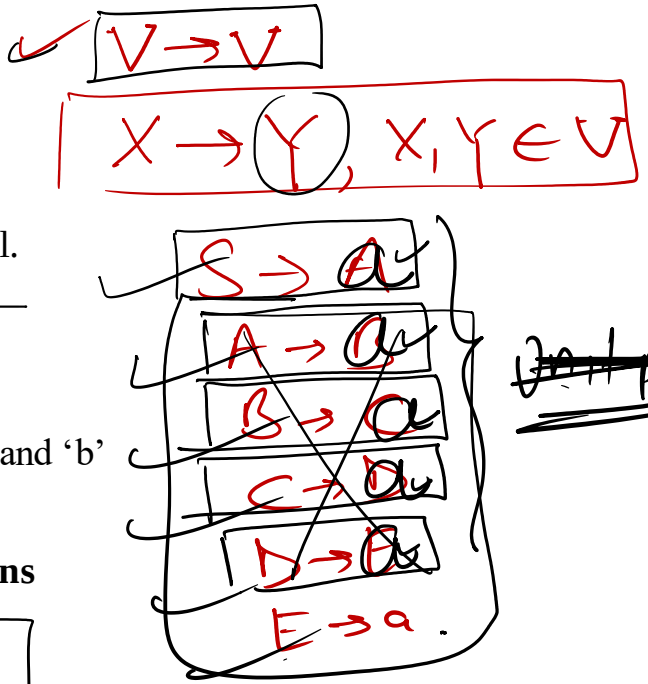
Example: {S → A; A → a | b}

Solution:

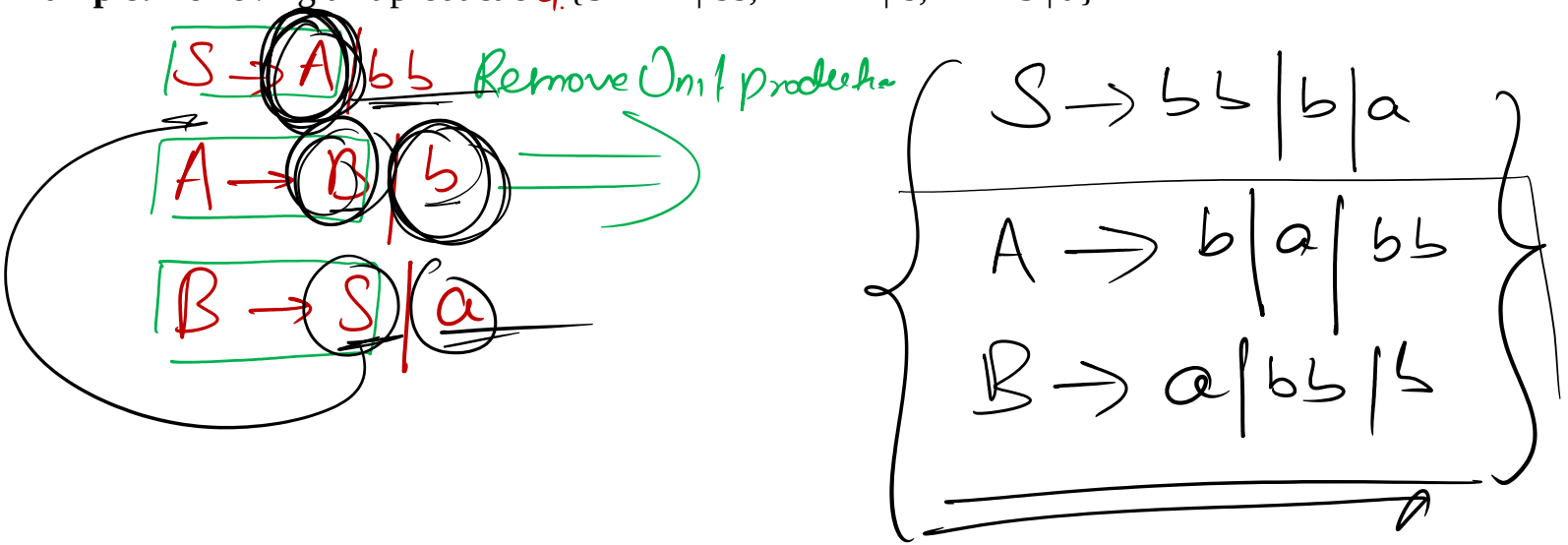
Substitute variable A in the production S → A with the terminal 'a' and 'b' because A → a; A → b

After removing unit production, we have: **{S → a | b, A → a | b } Ans**





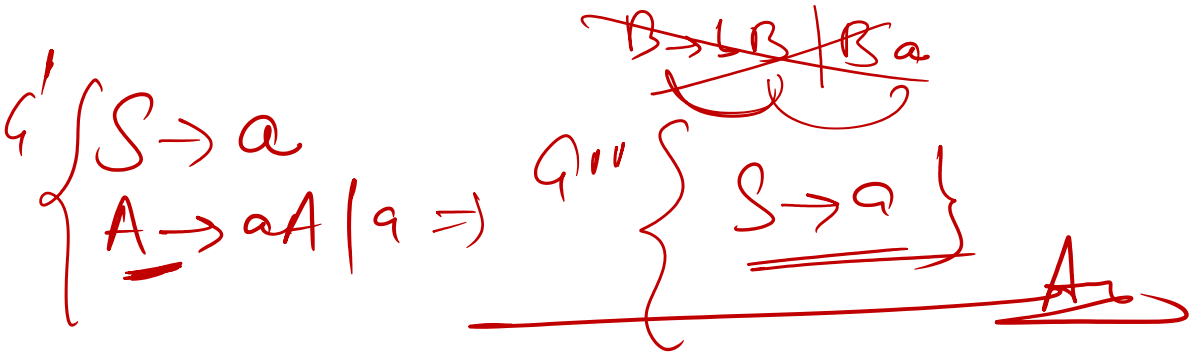
Example: Removing unit production: {S → A | bb; A → B | b; B → S | a}

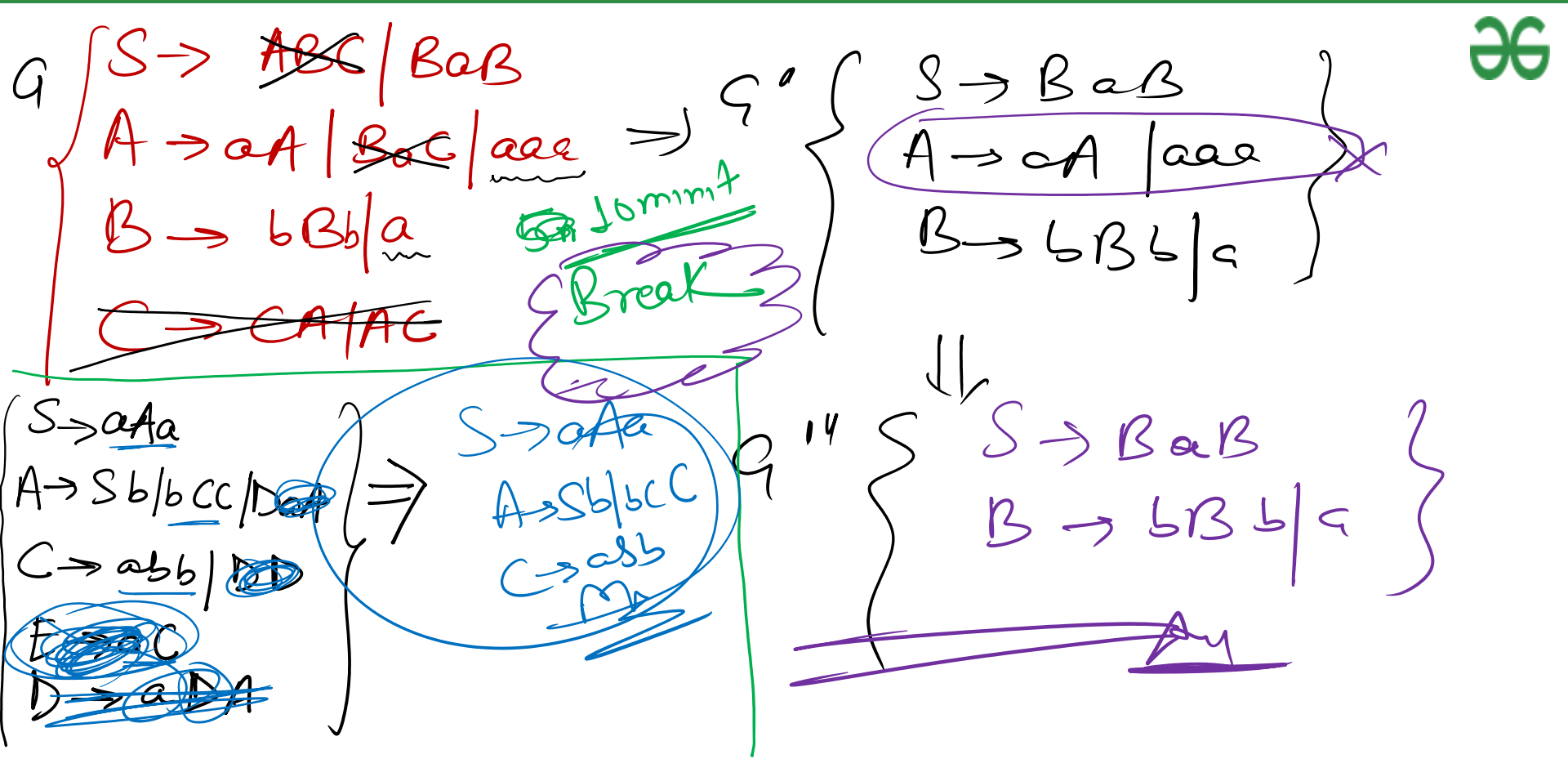


Removal of Useless Symbol and Production

- Variables / Non Terminals, which don't derive any string are called as useless symbols.
- The production rule generating useless symbol becomes useless production.

Example: {S → AB | a; A → aA | a; B → bB | BS}




G
S → ABC | BaB
A → aA | BaC | aaa
B → bBb | a
C → CA | AC
10 min
Break
G'
S → BaB
A → aA | aaa
B → bBb | a
S → aAa
A → Sb | bCC | DaA
C → abb | DD
E → aC
D → aDA
S → aAa
A → Sb | bCC
C → abb
G''
S → BaB
B → bBb | a

Chomsky Normal Form (CNF)

- A CFG is in CNF if the productions are in following forms

<variable> → <Terminal>

<variable> → <variable><variable>

S → ε (Start symbol generate ε)

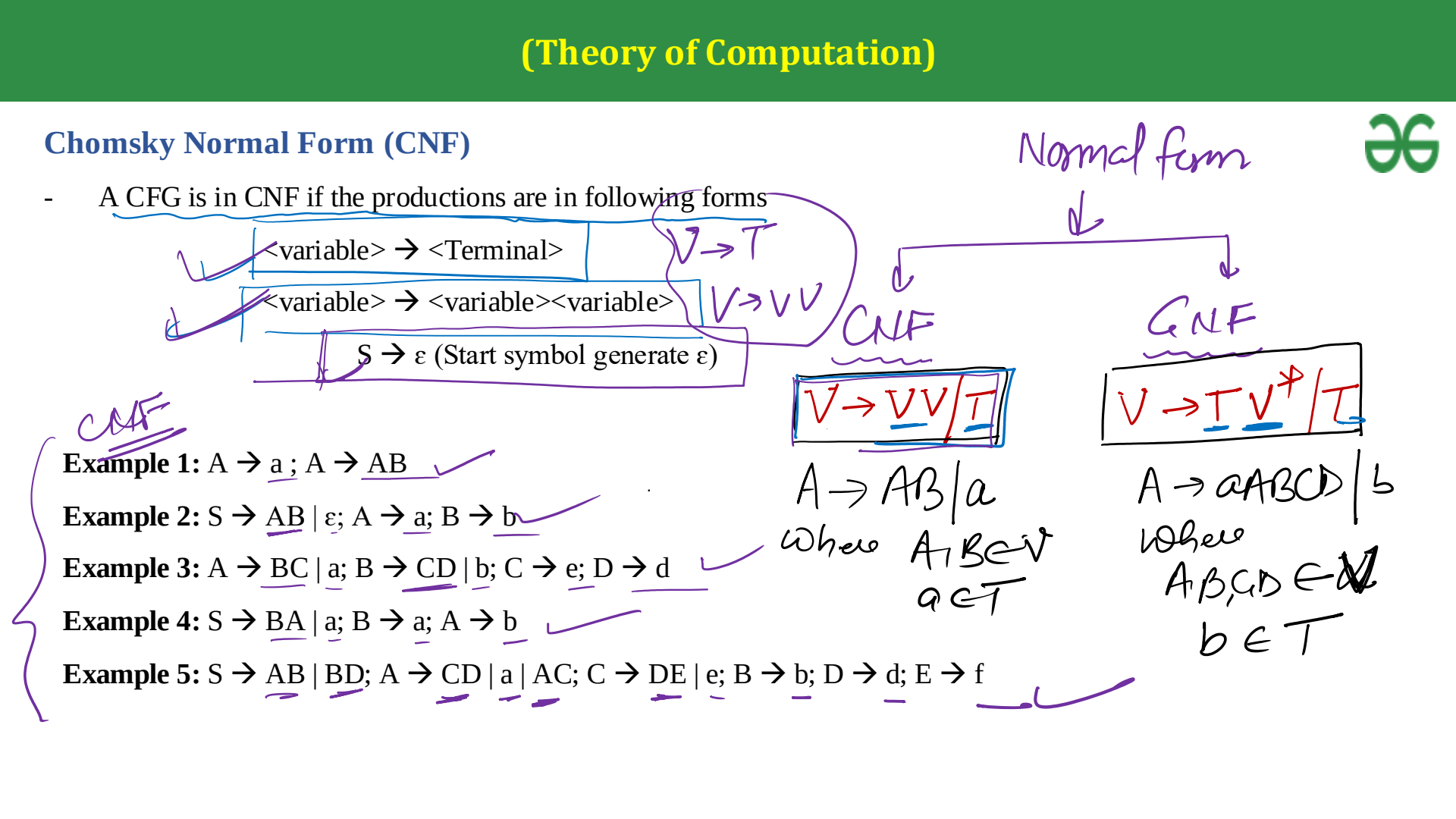
Example 1: A → a ; A → AB

Example 2: S → AB | ε; A → a; B → b

Example 3: A → BC | a; B → CD | b; C → e; D → d

Example 4: S → BA | a; B → a; A → b

Example 5: S → AB | BD; A → CD | a | AC; C → DE | e; B → b; D → d; E → f



$G:\{S \to aSbA \mid abA,\ A \to BaAb \mid a,\ B \to Ba \mid ab\}$

C D

X X

$V \to VV \mid a$

Step1: Remove Null & Unit Production

Step2: CNF

$$\begin{cases} S \to CD \mid CaD \\ C \to CaS,\ Ca \to a \\ D \to CbA,\ Cb \to b \\ A \to EF \mid a \\ E \to BCa \\ F \to ACb \\ B \to BCa \mid CaCb \end{cases}$$

$G\{S \to aS \mid \epsilon\}$

1) Remove ϵ & Unit

$S' \to S$, $S \to aS \mid \epsilon$

$S' \to aS \mid \epsilon$, $S \to aS \mid \epsilon$

$S' \to aS \mid a \mid \epsilon$, $S \to aS \mid a$

$S \to \epsilon$

$S \to A \to B$

$A \to a$

$S' \to AS \mid a \mid \epsilon$, $S \to AS \mid a$, $A \to a$

CNF

Properties of CNF :

9

Steps = 2 * n - 1 ✓ (n → length of string)

1 - Every derivation Tree in CNF is a Binary Tree

2 - If Grammar in CNF and String Length is n (|w| = n) Then No. of derivation steps = 2n-1

Ex: G: { S → AB | a, A → AA | a, B → BS | b }, ω = aaabc |= 5

S ⇒ AB — 1
⇒ AAB — 2
⇒ aAB — 3
⇒ aAAB — 4
⇒ aaAB — 5
⇒ aaaB — 6
⇒ aaaBS — 7
⇒ aaabS — 8
⇒ aaabc — 9

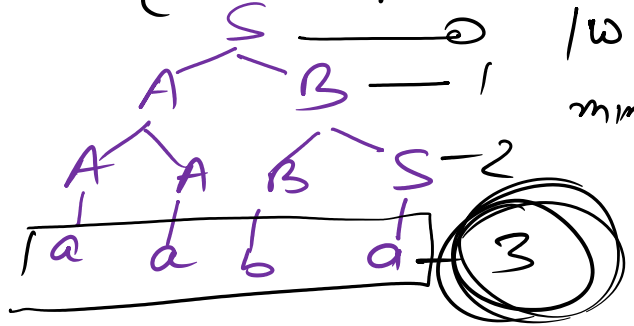
9 steps

2n-1 ⇒ 2×5-1 ⇒ 9

3- If the Grammar is in CNF, Then the minimum height of the parse tree of a string of length n is $h_{min} \geq \log_2 n + 1$

4) max height $h_{max} \leq n$

Ex: $G = \{S \to AB \mid a,\ A \to AA \mid a,\ B \to BS \mid b\}$

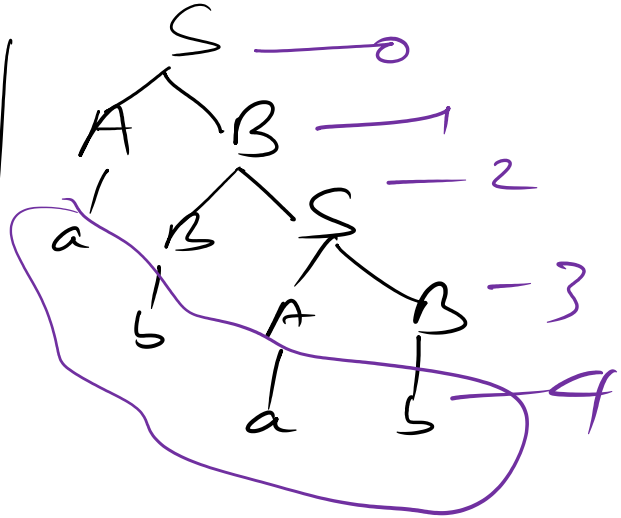


$|w| = 4$

min height $\geq \log_2 n + 1$

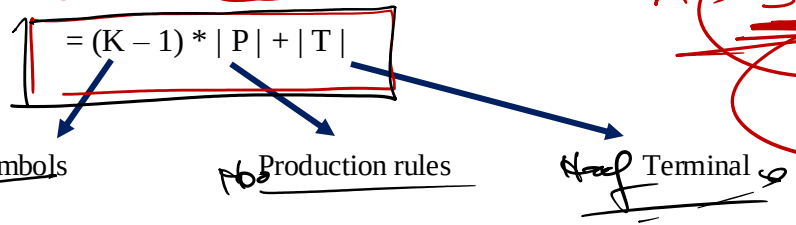
$\geq \log_2 4 + 1$

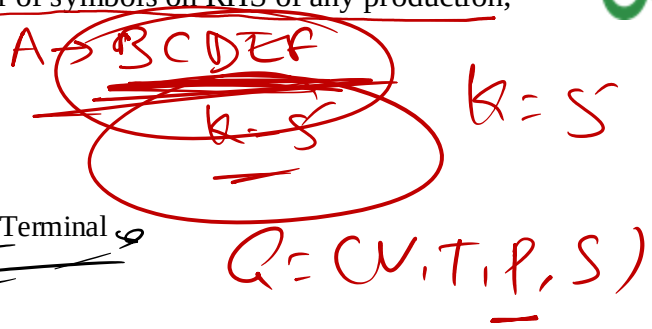
≥ 3



NOTE:

Let G be CFG without NULL production & UNIT production. K be the maximum number of symbols on RHS of any production, then equivalent CNF contains maximum number of productions.





Note: For generating string w of length 'n' requires '2n-1' production or steps in CNF

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

$\Downarrow$ CNF

$G' = (V', T, P', S)$

$|P| =$ No. of production Rules

$|T| =$ No. of Terminal Symbols

$k =$ max No. of symbols on RHS of Any Produ...

$|P'| = ?$

Greibach Normal Form (GNF)

- A CFG is in GNF if the productions are in the following forms

  <Variable> → <Terminal>

  <Variable> → <Single Terminal><Variables List>

  S → ε (Start Symbol)

Example 1: A → a; B → aBB | a

Example 2: A → aABBA | b; B → a

Example 3: S → aAABBB; A → a; B → b

Example 4: S → bAAAABBB; A → a; B → b

Note: For generating string w of length 'n' requires 'n' production or steps in GNF.

Ex: $G = \{S \to A_a S \mid aa,\ A \to ab Sa \mid b\}$

Soln:

$C_a \to a$, $C_b \to b$

$S \to A C_a S \mid a C_a$, $A \to a C_b S C_a \mid b$

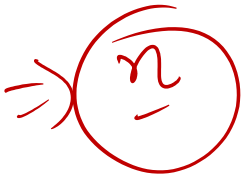
$S \to a C_b S C_a C_a S \mid b C_a S \mid a C_a$

$A \to a C_b S C_a \mid b$

$\Rightarrow$ GNF

Ex: $G = \{ S \to aAB \mid b,\ A \to bAS \mid a,\ B \to b \}$

$w = abaabb = S$



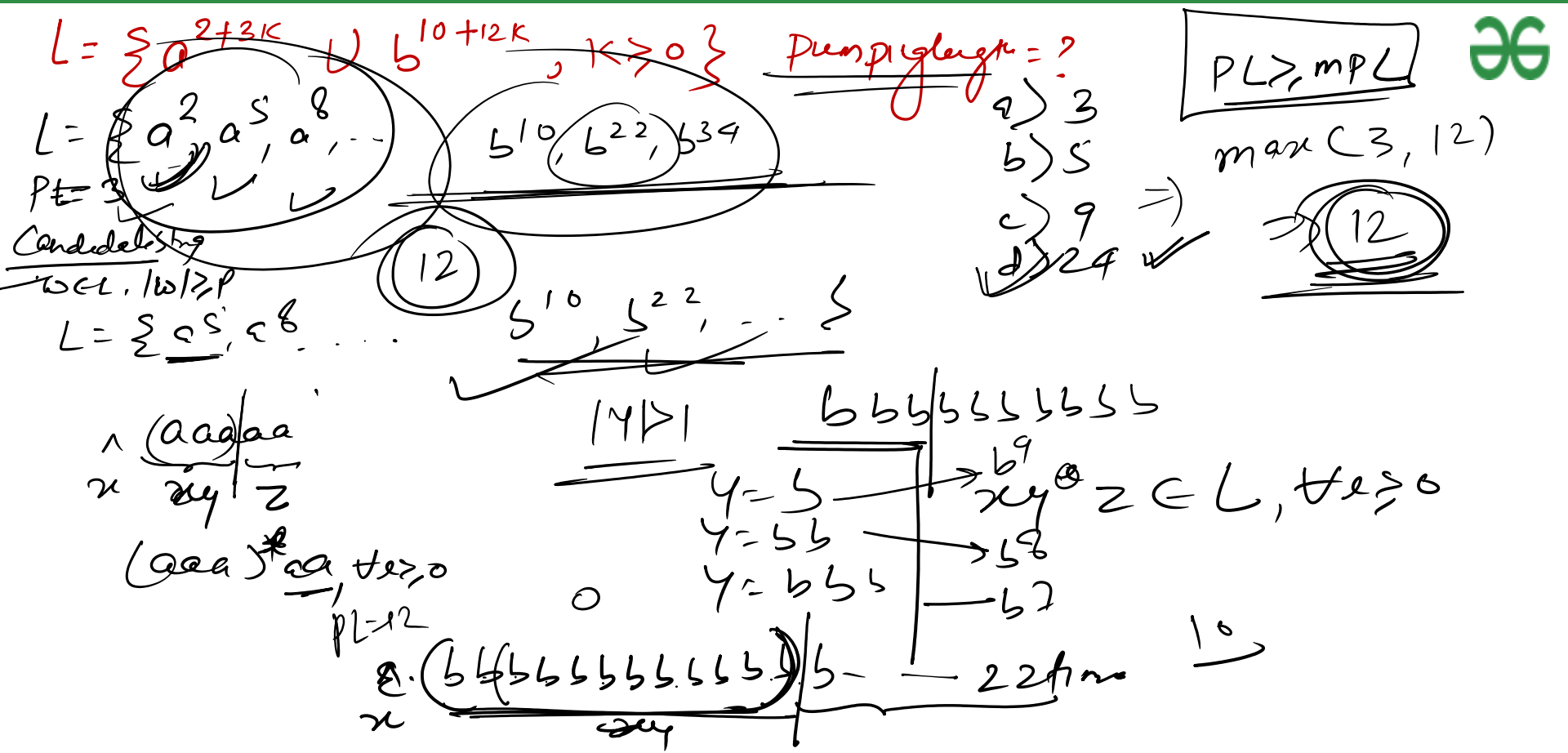
$S \Rightarrow aAB$ — (1)

$\Rightarrow abASB$ — (2)

$\Rightarrow abaSB$ — (3)

$\Rightarrow ababB$ — (4)

$\Rightarrow abaabb$ — (5)


L = {a^{2+3K} ∪ b^{10+12K}, K≥0}
Pumping length = ?
a) 3
b) 5
c) 9
d) 24
PL ≥ mPL
max(3, 12)
12
L = {a^2, a^5, a^8, ...}
L = {a^5, a^8, ...}
{b^10, b^22, b^34}
{b^10, b^22, ...}
|w| ≥ P
|y| ≥ 1
aaaaa
x xy z
(aaa)* aa, k≥0
PL = 12
bbbbbbbbbb
y = b
y = bb
y = bbb
b^9
b^8
b^7
xy^k z ∈ L, k≥0
10
22 times