CONTEXT –FREE GRAMMAR (CFG)

- A Grammar **G=(V, T, P, S)** is said to be context free, if all productions in P have the form $\alpha \rightarrow \beta$ where $|\alpha| \leq |\beta|$ and α is an element of V (ie, lefthand side contains only nonterminal)
- **Eg 1:** Consider a grammar G = (V , T , P , S) where

$V = \{ S \}$

$T = \{ a , b \}$

$P = \{ S \rightarrow aSbS , S \rightarrow bSaS , S \rightarrow \varepsilon \}$

$S = \{ S \}$

This grammar is an example of a context free grammar.

➢**Eg 2 :** Consider a grammar G = (V , T , P , S) where

$V = \{ S \}$

$T = \{ (,) \}$

$P = \{ S \rightarrow SS , S \rightarrow (S) , S \rightarrow \varepsilon \}$

$S = \{ S \}$

This grammar is an example of a context free grammar.

❖**Applications**

- For defining programming languages
- For parsing the program by constructing syntax tree
- For translation of programming languages
- For describing arithmetic expressions
- For construction of compilers

CONTEXT –FREE LANGUAGES (CFL)

➢The language generated using Context Free Grammar is called as Context Free Language

❖**Properties**

- The context free languages are closed under union.
- The context free languages are closed under concatenation.
- The context free languages are closed under kleen closure.
- The context free languages are not closed under intersection and complement.
- The family of regular language is a proper subset of the family of context free language.
- Each Context Free Language is accepted by a Pushdown automaton.

Remember

If L1 and L2 are two context free languages, then

- L1 ∪ L2 is also a context free language.
- L1.L2 is also a context free language.
- L1* and L2* are also context free languages.
- L1 ∩ L2 is not a context free language.
- L1′ and L2′ are not context free languages.

DERIVATION

- Derivation is a sequence of production rules. It is used to get the input string through these production rules
- At the time of derivation, we have to take two decisions. These are as follows:

1) We have to decide the non-terminal which is to be replaced.
2) We have to decide the production rule by which the non-terminal will be replaced.

We have two options to decide which non-terminal to be placed with production rule.

1. Leftmost Derivation **2. Rightmost Derivation**

❖Leftmost Derivation

➢Find Leftmost derivation of the Following Grammar ?

S → AB | ε

A → aB

B → Sb

Derive the string **abb**

S →AB

→aBB

→aSbB

→aεbB

→abB

→abSb

→abεb

→abb

❖Rightmost Derivation

➢Find Rightmost derivation of the Following Grammar ?

S → AB | ε

A → aB

B → Sb

Derive the string **abb**

S →AB

→ASb

→Aεb

→aBb

→aSbb

→aεbb

→abb

DERIVATION TREE

- Derivation tree is a graphical representation for the derivation of the given production rules for a given CFG.
- It is the simple way to show how the derivation can be done to obtain some string from a given set of production rules.
- The derivation tree is also called a parse tree.

A parse tree contains the following properties:

- The root node is always a node indicating start symbols.
- The derivation is read from left to right.
- The leaf node is always terminal nodes.
- The interior nodes are always the non-terminal nodes.

Eg: 1

Production rules

$E \rightarrow E + E$

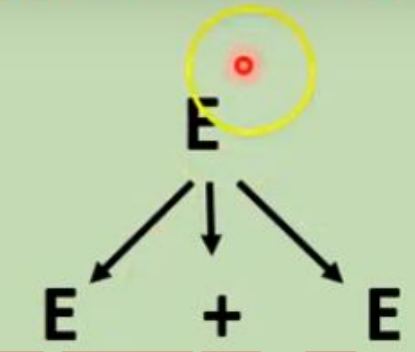
$E \rightarrow E * E$

$E \rightarrow a \mid b \mid c$

Input

a * b + c

DERIVATION TREE



Eg: 1

DERIVATION TREE

Production rules

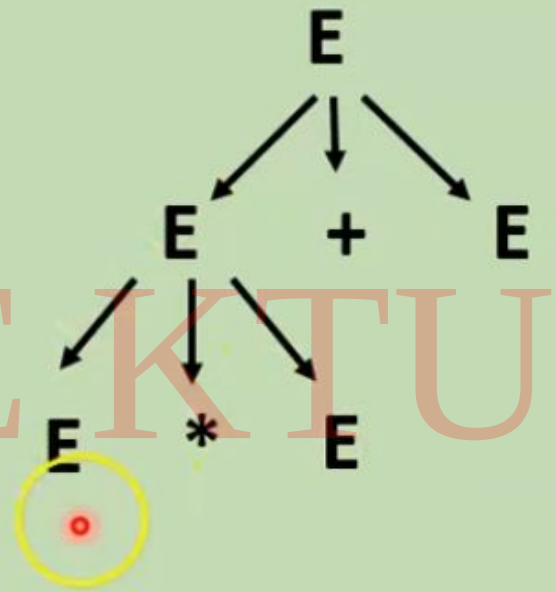
E → E + E

E → E * E

E → a | b | c

Input

a * b + c



Eg: 1

Production rules

$E \rightarrow E + E$

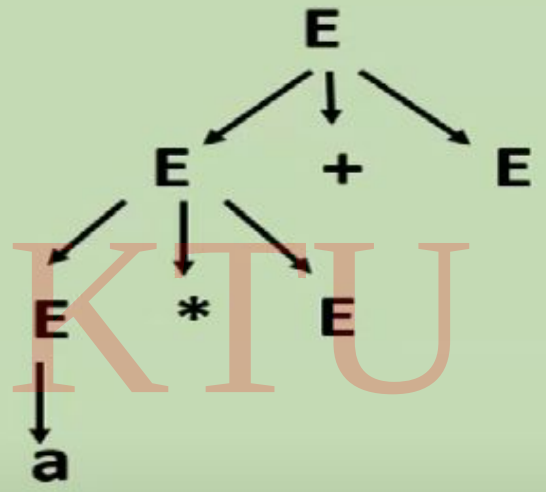
$E \rightarrow E * E$

$E \rightarrow a \mid b \mid c$

Input

a * b + c

DERIVATION TREE



Eg: 1

Production rules

E → E + E

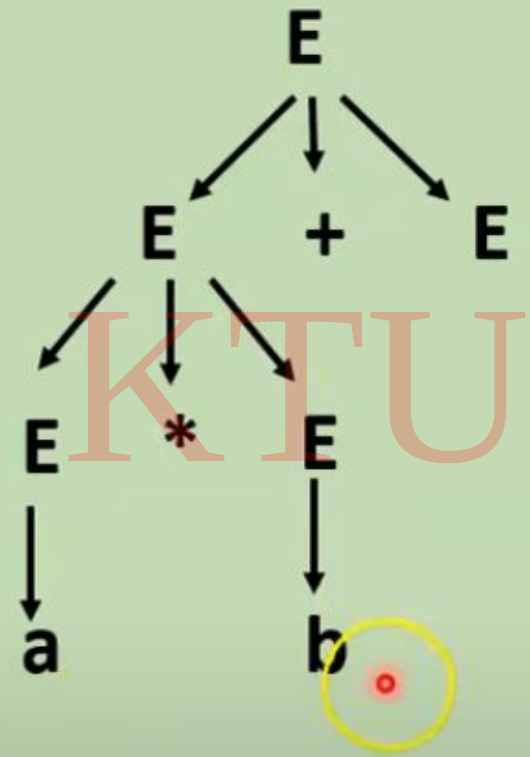
E → E * E

E → a | b | c

Input

a * b + c

DERIVATION TREE



Eg: 1

DERIVATION TREE

Production rules

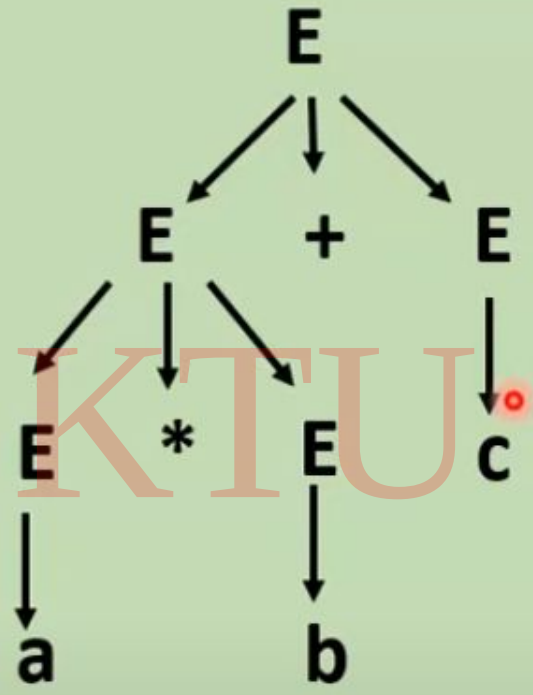
$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow a \mid b \mid c$

Input

a * b + c



Eg: 1

Production rules

$E \rightarrow E + E$

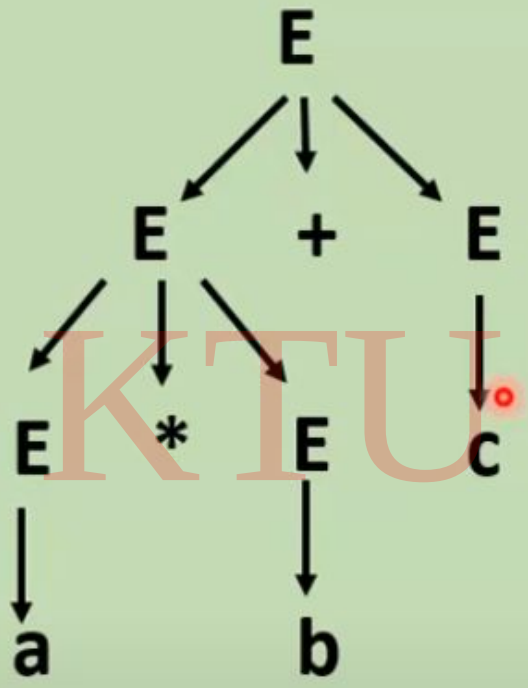
$E \rightarrow E * E$

$E \rightarrow a \mid b \mid c$

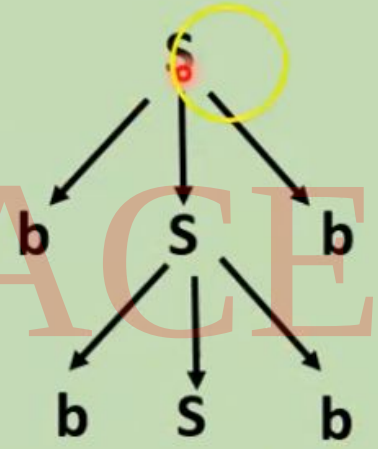
Input

a * b + c

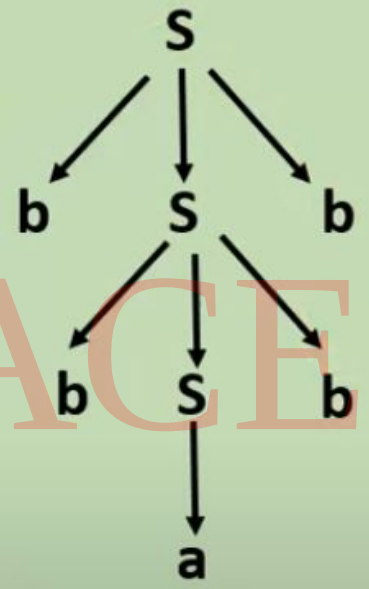
DERIVATION TREE



Eg: 2 - Draw a derivation tree for the string "bbabb" from the CFG given by $S \rightarrow bSb \mid a \mid b$



Eg: 2 - Draw a derivation tree for the string "bbabb" from the CFG given by $S \rightarrow bSb \mid a \mid b$



Home work

- **Eg: 3-** Construct a derivation tree for the string aabbabba for the CFG given by,

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

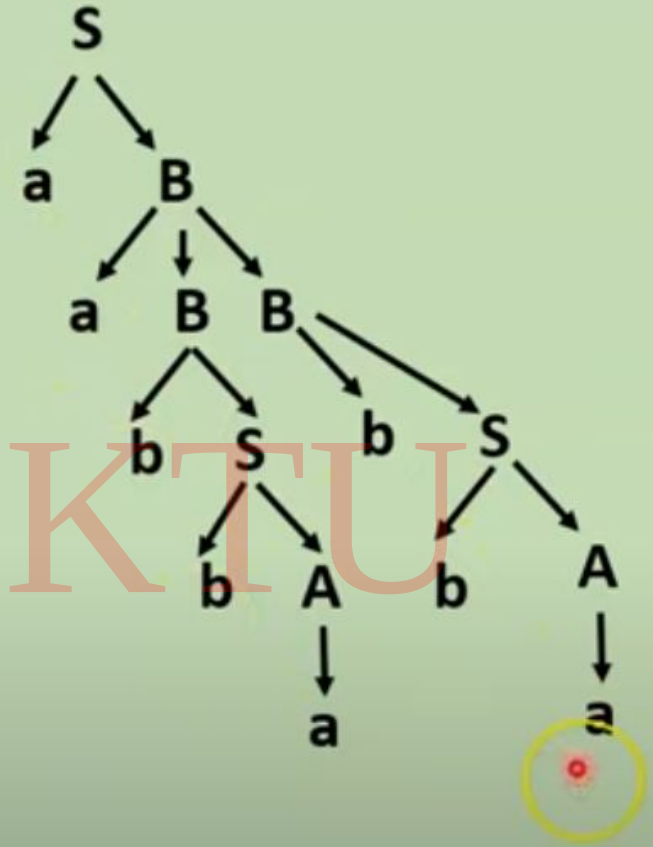
$B \rightarrow b \mid bS \mid aBB$

- **Eg: 3-** Construct a derivation tree for the string **aabbabba** for the CFG given by,

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$



AMBIGUITY IN GRAMMAR

- A grammar is said to be ambiguous if there exists
  - More than one leftmost derivation or
  - More than one rightmost derivation or
  - More than one parse tree for the given input string
- If the grammar is not ambiguous, then it is called unambiguous.
- If the grammar has ambiguity, then it is not good for compiler construction.
- No method can automatically detect and remove the ambiguity, but we can remove ambiguity by re-writing the whole grammar without ambiguity.

Example 1: Let us consider a grammar G with the production rule

$E \rightarrow I$

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$I \rightarrow \varepsilon \mid 0 \mid 1 \mid 2 \mid \ldots \mid 9$

Input String

3 * 2 + 5

- For the string "3 * 2 + 5", the above grammar can generate two parse trees by leftmost derivation
- Since there are two parse trees for a single string "3 * 2 + 5", the grammar G is ambiguous.

Example 1: Let us consider a grammar G with the production rule

$E \rightarrow I$

$E \rightarrow E + E$

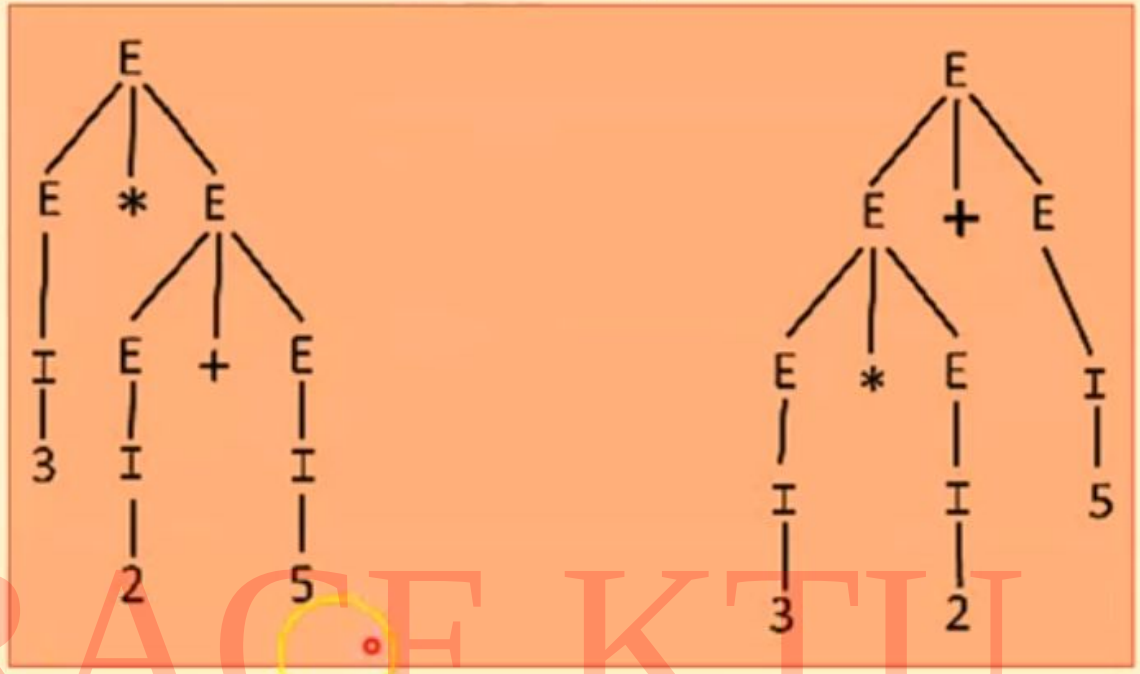
$E \rightarrow E * E$

$E \rightarrow (E)$

$I \rightarrow \varepsilon \mid 0 \mid 1 \mid 2 \mid \ldots \mid 9$

Input String

3 * 2 + 5



- For the string "3 * 2 + 5", the above grammar can generate two parse trees by leftmost derivation
- Since there are two parse trees for a single string "3 * 2 + 5", the grammar G is ambiguous.

Example 2: Check whether the given grammar G is ambiguous or not using the string "id + id - id"

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow id$

First Leftmost derivation

$E \rightarrow E + E$

$\rightarrow id + E$

$\rightarrow id + E - E$

$\rightarrow id + id - E$

$\rightarrow id + id - id$

Second Leftmost derivation

$E \rightarrow E - E$

$\rightarrow E + E - E$

$\rightarrow id + E - E$

$\rightarrow id + id - E$

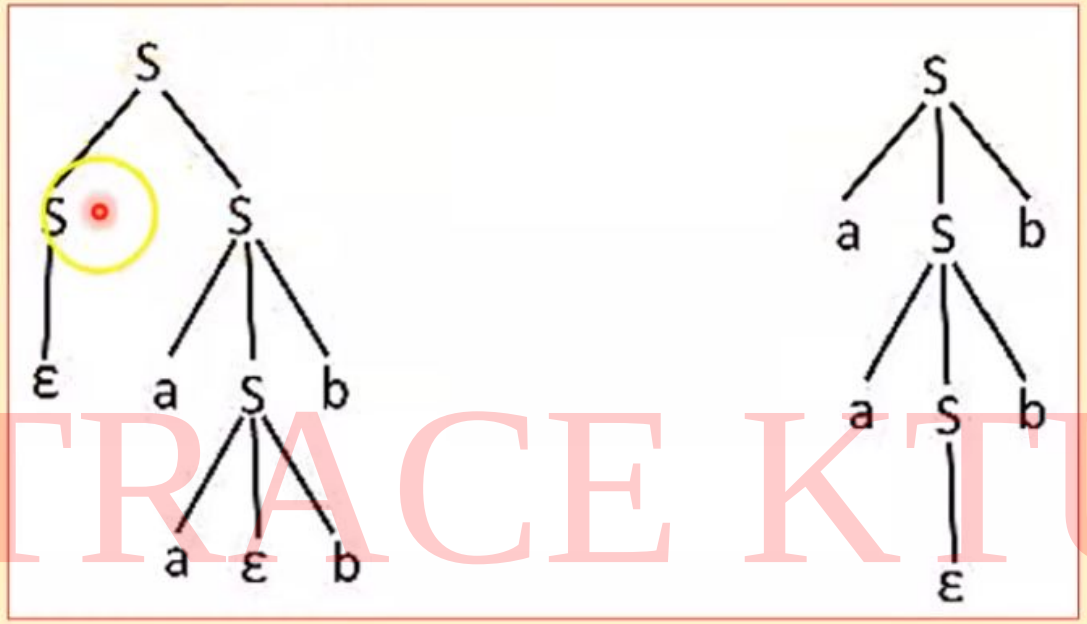
$\rightarrow id + id - id$

➢ Since there are two leftmost derivation for a single string "id + id - id", the grammar G is ambiguous.

Example 3: Check whether the given grammar G is ambiguous or not for the given string "aabb"

$S \rightarrow aSb \mid SS$

$S \rightarrow \varepsilon$



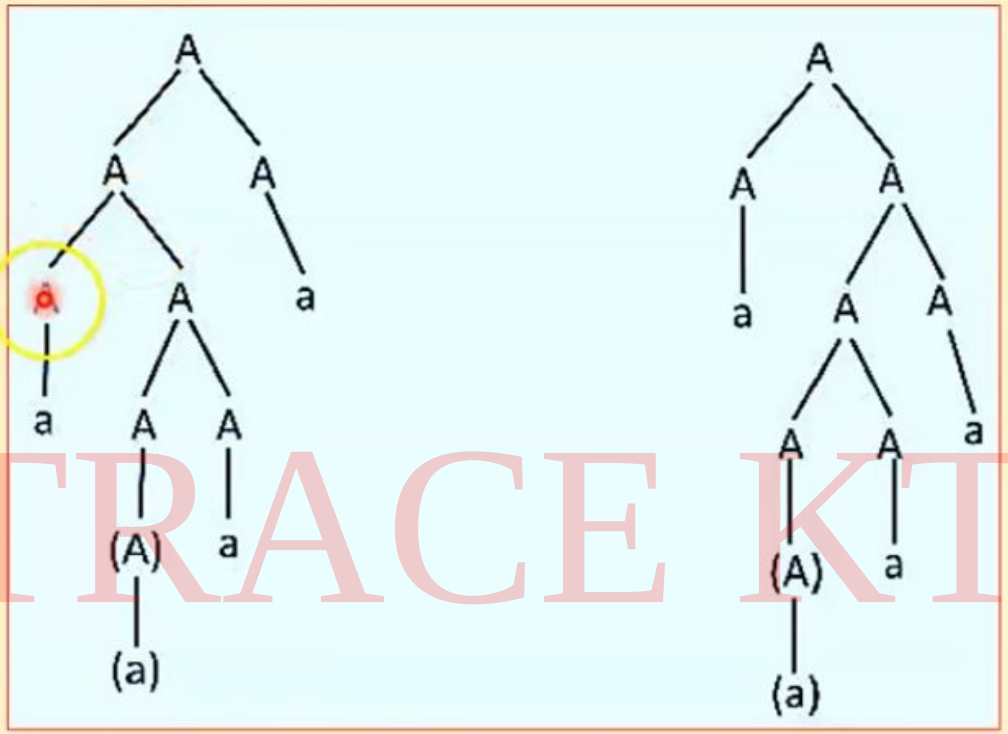
➢Since there are two parse trees for a single string "aabb", the grammar G is ambiguous.

Example 4: Check whether the given grammar G is ambiguous or not for the input string **"a(a)aa"**

$A \rightarrow AA$

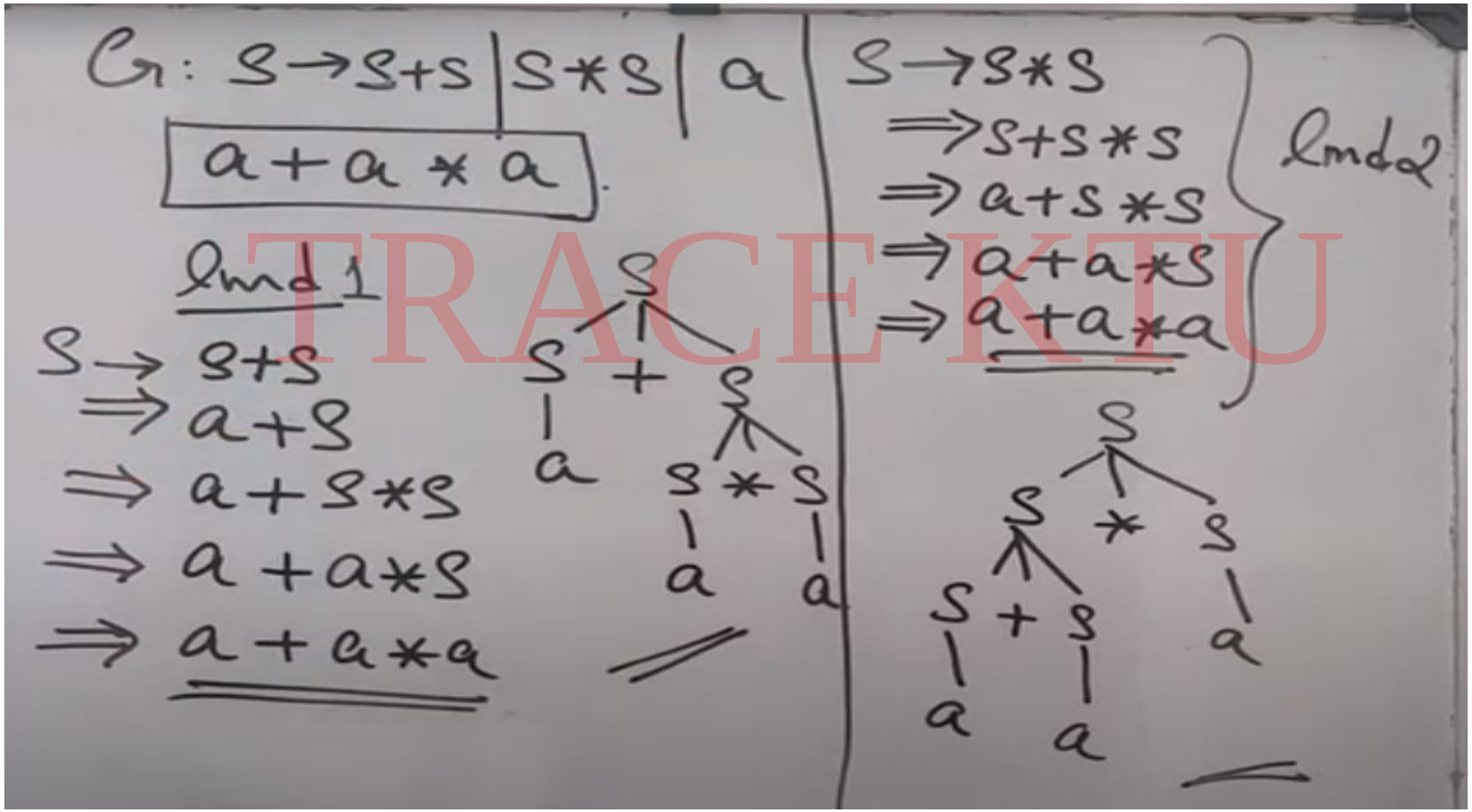
$A \rightarrow (A)$

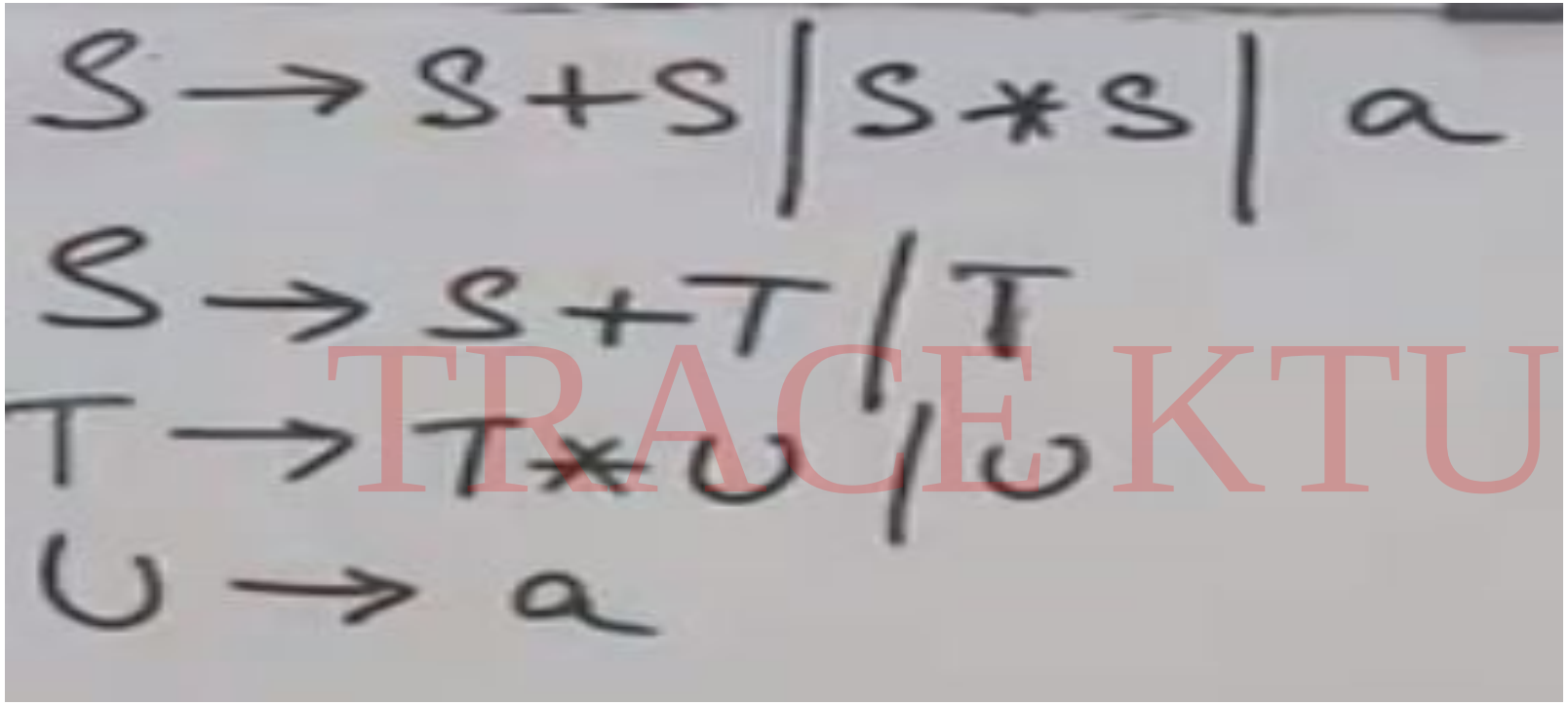
$A \rightarrow a$



➢Since there are two parse trees for a single string "a(a)aa", the grammar G is ambiguous.

How to convert ambiguous to unambiguous





$S \rightarrow S+S \mid S*S \mid a$

$S \rightarrow S+T \mid T$

$T \rightarrow T*U \mid U$

$U \rightarrow a$

$a+a*a$

lmd

$S \rightarrow S+T$

$\Rightarrow T+T$

$\Rightarrow U+T$

$\Rightarrow a+T$

$\Rightarrow a+T*U$

$\Rightarrow a+U*U$

$\Rightarrow a+a*U$

$\Rightarrow a+a*a$

rmd

$S \rightarrow S+T$

$\Rightarrow S+T*U$

$\Rightarrow S+T*a$

$\Rightarrow S+U*a$

$\Rightarrow S+a*a$

$\Rightarrow T+a*a$

$\Rightarrow U+a*a$

$\Rightarrow a+a*a$

$S \rightarrow S+S \mid S*S \mid a$

$S \rightarrow S+T \mid T$

$T \rightarrow T*U \mid U$

$U \rightarrow a$

Ambiguous

$a+a*a$

$S \Rightarrow S+T$

$\Rightarrow T+T$

$\Rightarrow U+T$

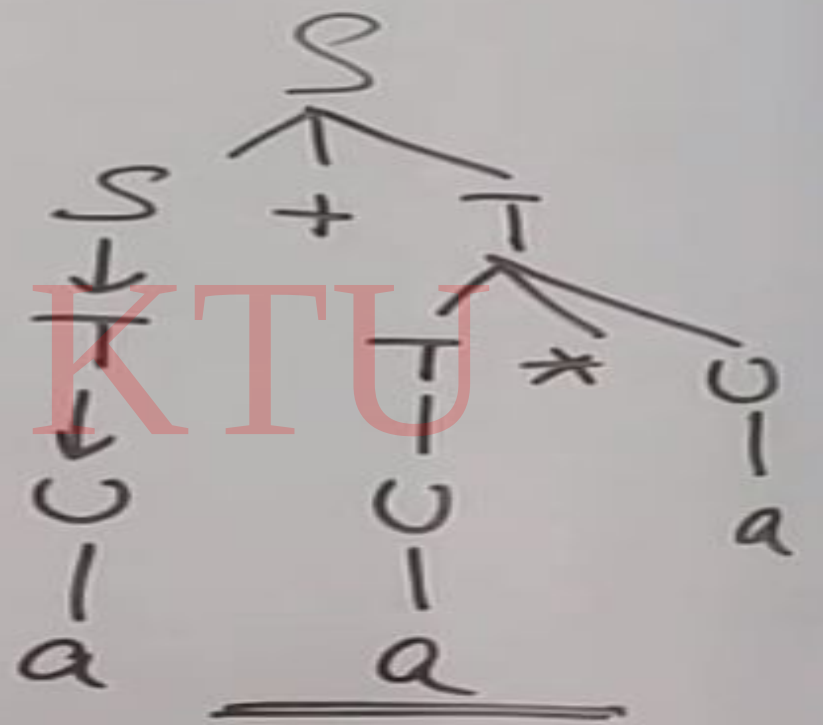
$\Rightarrow a+T$

$\Rightarrow a+T*U$

$\Rightarrow a+U*U$

$\Rightarrow a+a*U$

$\Rightarrow a+a*a$

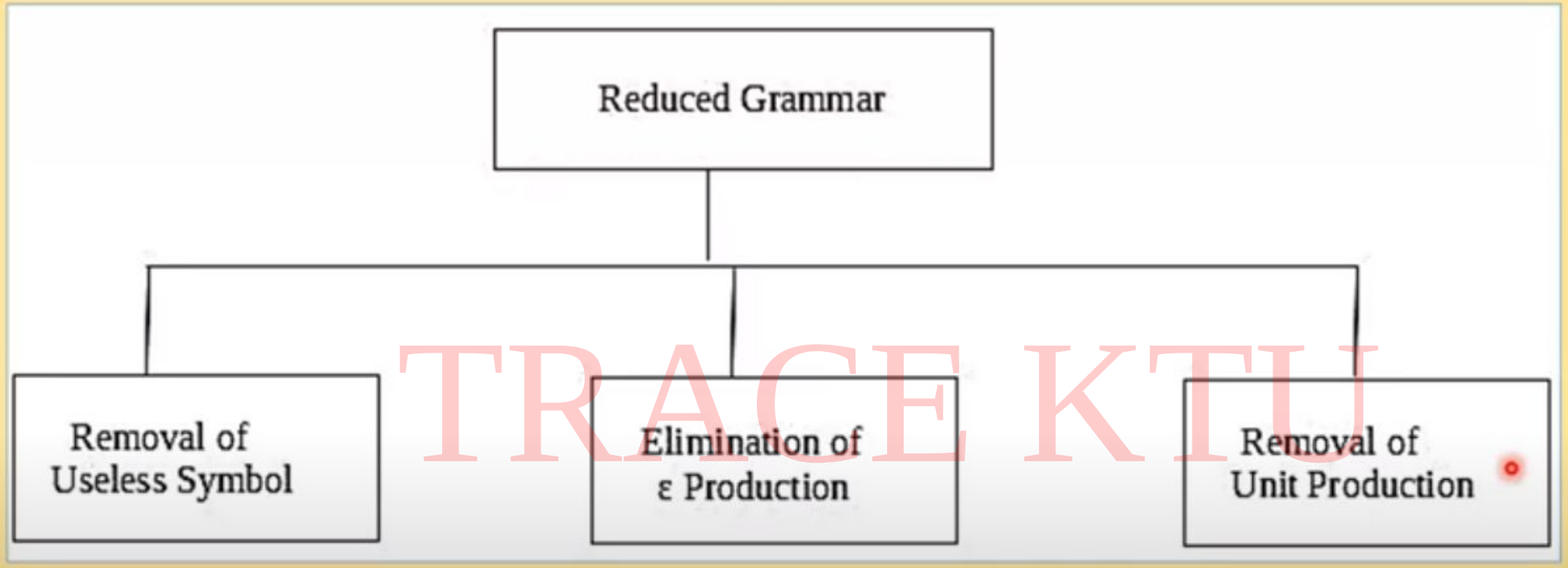


SIMPLIFICATION OF CFG

- All the grammar are not always optimized that means the grammar may consist of some extra symbols(non-terminal).
- Having extra symbols, unnecessary increase the length of grammar.
- Simplification of grammar means reduction of grammar by removing useless symbols.

The properties of reduced grammar are given below

- Each variable (i.e. non-terminal) and each terminal of G appears in the derivation of some word in L.
- There should not be any production as **X → Y** where X and Y are non-terminal.
- If ε is not in the language L then there need not to be the production **X → ε**.


Reduced Grammar
Removal of Useless Symbol
Elimination of ε Production
Removal of Unit Production

❖Removal of Useless Symbols

- A symbol can be useless if it does not appear on the right-hand side of the production rule and does not take part in the derivation of any string. That symbol is known as a useless symbol.
- Similarly, a variable can be useless if it does not take part in the derivation of any string. That variable is known as a useless variable.

For Example:

$T \rightarrow aaB \mid abA \mid aaT$

$A \rightarrow aA$

$B \rightarrow ab \mid b$

$C \rightarrow ad$

- In the above example, the variable 'C' will never occur in the derivation of any string, so the production C → ad is useless.
- So we will eliminate it, and the other productions are written in such a way that variable C can never reach from the starting variable 'T'.
- Production A → aA is also useless because there is no way to terminate it. If it never terminates, then it can never produce a string. Hence this production can never take part in any derivation.
- To remove this useless production A → aA, we will first find all the variables which will never lead to a terminal string such as variable 'A'. Then we will remove all the productions in which the variable 'B' occurs.

Example: Remove the ε production from the following CFG by preserving the meaning of it.

$S \rightarrow XYX$

$X \rightarrow 0X \mid \varepsilon$

$Y \rightarrow 1Y \mid \varepsilon$

Solution:

- While removing ε production, we are deleting the rule $X \rightarrow \varepsilon$ and $Y \rightarrow \varepsilon$.
- To preserve the meaning of CFG we are actually placing ε at the right-hand side whenever X and Y have appeared.

❖Elimination of ε Production

- The productions of type **S → ε** are called ε productions. These type of productions can only be removed from those grammars that do not generate ε.

- **Step 1:** First find out all nullable non-terminal variable which derives ε.
- **Step 2:** For each production A → a, construct all production A → x, where x is obtained from a by removing one or more non-terminal from step 1.
- **Step 3:** Now combine the result of step 2 with the original production and remove ε productions.

Let us take S → XYX

- If the first X at right-hand side is ε. Then S → YX
- Similarly if the last X in R.H.S. = ε. Then S → XY
- If Y = ε then S → XX
- If Y and X are ε then, S → X
- If both X are replaced by ε then, S → Y
- Now, **S → XY | YX | XX | X | Y**

$S \rightarrow XYX$

$X \rightarrow 0X \mid \varepsilon$

$Y \rightarrow 1Y \mid \varepsilon$

❖Removing Unit Productions

The unit productions are the productions in which one non-terminal gives another non-terminal. Use the following steps to remove unit production:

Step 1: To remove $X \rightarrow Y$, add production $X \rightarrow a$ to the grammar rule whenever $Y \rightarrow a$ occurs in the grammar.

Step 2: Now delete $X \rightarrow Y$ from the grammar.

Step 3: Repeat step 1 and step 2 until all unit productions are removed.

Example:

S → 0A | 1B | C

A → 0S | 00

B → 1 | A

C → 01

Thus finally we can write CFG without unit production as

S → 0A | 1B | 01

A → 0S | 00

B → 1 | 0S | 00

C → 01

Solution:

- S → C is a unit production. But while removing S → C we have to consider what C gives. So, we can add a rule to S.

  S → 0A | 1B | 01

- Similarly, B → A is also a unit production so we can modify it as

  B → 1 | 0S | 00

Example: Remove the ε production from the following CFG by preserving the meaning of it.

$S \rightarrow ABAC$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

$C \rightarrow c$

The productions with ε are

$A \rightarrow \varepsilon$, $B \rightarrow \varepsilon$

- Replace A with ε . Consider the first production $S \rightarrow ABAC$

  $S \rightarrow BAC$, $S \rightarrow ABC$, $S \rightarrow BC$

  $S \rightarrow BAC \mid ABC \mid BC$

- Consider the second production $A \rightarrow aA$

  $A \rightarrow a$

After replace A with ε , our grammar will be

S → ABAC|BAC| ABC| BC

A → aA|a

S → ABAC
A → aA | ε
B → bB | ε
C→ c

➢Now Replace B with ε

- Consider the first production S → ABAC |BAC| ABC| BC

  S → AAC |AC|C

- Consider the next production B → bB

  B → b

So the final answer after removing the ε is

$S \rightarrow ABAC|BAC|ABC|BC|AAC|AC|C$

$A \rightarrow aA|a$

$B \rightarrow bB|b$

$C \rightarrow c$

$S \rightarrow ABAC$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

$C \rightarrow c$

Example: Remove the unit production from the following CFG by preserving the meaning of it.

$S \rightarrow XY$

$X \rightarrow a$

$Y \rightarrow Z \mid b$

$Z \rightarrow M$

$M \rightarrow N$

$N \rightarrow a$

Unit productions are

$Y \rightarrow Z$, $Z \rightarrow M$, $M \rightarrow N$

Consider $M \rightarrow N$. Since $N \rightarrow a$, we can add $M \rightarrow a$

Consider $Z \rightarrow M$. Since $M \rightarrow a$, we can add $Z \rightarrow a$

Consider $Y \rightarrow Z$. Since $Z \rightarrow a$, we can write $Y \rightarrow a$

$S \rightarrow XY$

$X \rightarrow a$

$Y \rightarrow a \mid b$

$Z \rightarrow a$

$M \rightarrow a$

$N \rightarrow a$

From the start symbol S, the Nonterminals Z M and N are unreachable. So eliminate it.

So, the final answer is

$S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow a \mid b$

Chomsky Normal Form (CNF)

- CNF stands for Chomsky normal form.
- A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

- Start symbol generating ε. For example, $A \rightarrow \varepsilon$
- A non-terminal generating two non-terminals. For example,

  $S \rightarrow AB$

- A non-terminal generating a terminal. For example, $S \rightarrow a$

Example:

G1 = {S → AB, S → c, A → a, B → b}

G2 = {S → aA, A → a, B → c}

- The production rules of Grammar G1 satisfy the rules specified for CNF, so the grammar G1 is in CNF.
- However, the production rule of Grammar G2 does not satisfy the rules specified for CNF as S → aA contains terminal followed by non-terminal.
- So the grammar G2 is not in CNF.

❖Steps for converting CFG into CNF

Step 1: Eliminate start symbol from the RHS. If the start symbol S is at the right-hand side of any production, create a new production as: $S_1 \rightarrow S$, Where S_1 is the new start symbol.

Step 2: In the grammar, remove the null, unit and useless productions. You can refer to the Simplification of CFG.

Step 3: Eliminate terminals from the RHS of the production if they exist with other non-terminals or terminals. For example, production $S \rightarrow aA$ can be decomposed as:

$S \rightarrow RA$

$R \rightarrow a$

Step 4: Eliminate RHS with more than two non-terminals. For example, S → ASB can be decomposed as:

S → RS

R → AB

Example:

Convert the given CFG to CNF. Consider the given grammar G1:

S → a | aA | BS

A → aBB | ε

B → Aa | b

Solution:

Step 1: We will create a new production $S_1 \rightarrow S$, as the start symbol S appears on the RHS. The grammar will be:

$S_1 \rightarrow S$

$S \rightarrow a \mid aA \mid B$

$A \rightarrow aBB \mid \varepsilon$

$B \rightarrow Aa \mid b$

Step 2: As grammar G1 contains $A \rightarrow \varepsilon$ null production, its removal from the grammar yields:

S1 → S

S → a | aA | B

A → aBB

B → Aa | b | a

Now, as grammar G1 contains Unit production S → B, its removal yield:

S1 → S

S → a | aA | Aa | b

A → aBB

B → Aa | b | a

Also remove the unit production S1 → S, its removal from the grammar yields:

S0 → a | aA | Aa | b

S → a | aA | Aa | b

A → aBB

B → Aa | b | a

Step 3: In the production rule S0 → aA | Aa, S → aA | Aa, A → aBB and B → Aa, terminal a exists on RHS with non-terminals. So we will replace terminal a with X

S0 → a | XA | AX | b

S → a | XA | AX | b

A → XBB

B → AX | b | a

X → a

Step 4: In the production rule A → XBB, RHS has more than two symbols, removing it from grammar yield:

S0 → a | XA | AX | b

S → a | XA | AX | b

A → RB

R → XB

B → AX | b | a

X → a

Hence, for the given grammar, this is the required CNF.

Q) Convert this cfg to cnf?

$S \rightarrow ASB$

$A \rightarrow aAS|a|\varepsilon$

$B \rightarrow SbS|A|bb$

Step 1. As start symbol S appears on the RHS, we will create a new production rule S0->S. Therefore, the grammar will become:

S0->S

S → ASB

A → aAS|a|ε

B → SbS|A|bb

Step 2. As grammar contains null production A-> ε, its removal from the grammar yields:

S0->S

S → ASB|SB

A → aAS|aS|a

B → SbS|A|ε|bb

Now, it creates null production B⟶ ε, its removal from the grammar yields:

S0->S

S → AS|ASB| SB|S

A → aAS|aS|a

B → SbS|A|bb

Now, it creates unit production B->A, its removal from the grammar yields:

```
S0->S
 S → AS|ASB| SB| S
A → aAS|aS|a
B → SbS|bb|aAS|aS|a
```

Also, removal of unit production S0->S from grammar yields:

```
S0-> AS|ASB| SB| S
S → AS|ASB| SB| S
A → aAS|aS|a
B → SbS|bb|aAS|aS|a
```


Also, removal of unit production S->S and S0->S from grammar yields:

```
S0-> AS|ASB| SB
S → AS|ASB| SB
A → aAS|aS|a
B → SbS|bb|aAS|aS|a
```

Step 3. In production rule A->aAS |aS and B-> SbS|aAS|aS, terminals a and b exist on RHS with non-terminates. Removing them from RHS:

S0-> AS|ASB| SB
S → AS|ASB| SB
A → XAS|XS|a
B → SYS|bb|XAS|XS|a
X →a
Y→b

Also, B->bb can't be part of CNF, removing it from grammar yields:

S0-> AS|ASB| SB
S → AS|ASB| SB
A → XAS|XS|a
B → SYS|VV|XAS|XS|a
X → a
Y → b
V → b

Step 4: In production rule S0->ASB, RHS has more than two symbols, removing it from grammar yields:

```
S0-> AS|PB| SB
S → AS|ASB| SB
A → XAS|XS|a
B → SYS|VV|XAS|XS|a
X → a
Y → b
V → b
P → AS
```

Similarly, S->ASB has more than two symbols, removing it from grammar yields:

```
S0-> AS|PB| SB
S → AS|QB| SB
A → XAS|XS|a
B → SYS|VV|XAS|XS|a
X → a
Y → b
V → b
P → AS
Q → AS
```

Similarly, A->XAS has more than two symbols, removing it from grammar yields:

S0-> AS|PB| SB

S → AS|QB| SB

A → RS|XS|a

B → SYS|VV|XAS|XS|a

X → a

Y → b

V → b

P → AS

Q → AS

R → XA

Similarly, B->SYS has more than two symbols, removing it from grammar yields:

S0 -> AS|PB| SB
S → AS|QB| SB
A → RS|XS|a
B → TS|VV|XAS|XS|a
X → a
Y → b
V → b
P → AS
Q → AS
R → XA
T → SY

Similarly, B->XAX has more than two symbols, removing it from grammar yields:

S0-> AS|PB| SB
S → AS|QB| SB
A → RS|XS|a
B → TS|VV|US|XS|a
X → a
Y → b
V → b
P → AS
Q → AS
R → XA
T → SY
U → XA

So this is the required CNF for given grammar.

Greibach Normal Form (GNF)

- GNF stands for Greibach normal form.
- A CFG(context free grammar) is in GNF(Greibach normal form) if all the production rules satisfy one of the following conditions:

- A start symbol generating ε. For example, **S → ε**
- A non-terminal generating a terminal. For example, **A → a**
- A non-terminal generating a terminal which is followed by any number of non-terminals. For example, **S → aASB**.

But in CNF S → AB

Example:

G1 = {S → aAB | aB, A → aA| a, B → bB | b}

G2 = {S → aAB | aB, A → aA | ε, B → bB | ε}

- The production rules of Grammar G1 satisfy the rules specified for GNF, so the grammar G1 is in GNF.
- However, the production rule of Grammar G2 does not satisfy the rules specified for GNF as A → ε and B → ε contains ε(only start symbol can generate ε).
- So the grammar G2 is not in GNF.

❖Steps for converting CFG into GNF

Step 1: Convert the grammar into CNF.

- If the given grammar is not in CNF, convert it into CNF.

Step 2: If the grammar exists left recursion, eliminate it.

- If the context free grammar contains left recursion, eliminate it.

Step 3: In the grammar, convert the given production rule into GNF form.

- If any production rule in the grammar is not in GNF form, convert it.

ELIMINATION OF LEFT RECURSION

- Left recursion form

  $A \rightarrow A\alpha \mid \beta$

It can be removed by

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$

Example

$A \rightarrow Aa \mid b$

$A \rightarrow A\alpha \mid \beta$

$A \rightarrow A\ a \mid b$

$A \rightarrow bA'$

$A' \rightarrow aA' \mid \varepsilon$

Example:

$S \rightarrow XB \mid AA$

$A \rightarrow a \mid SA$

$B \rightarrow b$

$X \rightarrow a$

❖Steps for converting CFG into GNF

Step 1: Convert the grammar into CNF.

- If the given grammar is not in CNF, convert it into CNF. (You can refer the previous video topic to convert the CFG into CNF: Chomsky normal form)

Step 2: If the grammar exists left recursion, eliminate it.

- If the context free grammar contains left recursion, eliminate it.

Step 3: In the grammar, convert the given production rule into GNF form.

- If any production rule in the grammar is not in GNF form, convert it.

Example:

$S \rightarrow XB \mid AA$

$A \rightarrow a \mid SA$

$B \rightarrow b$

$X \rightarrow a$

Solution:

- As the given grammar G is already in CNF and there is no left recursion, so we can skip step 1 and step 2 and directly go to step 3.
- The production rule A → SA is not in GNF, so we substitute S → XB | AA in the production rule A → SA as:

S → XB | AA

A → a | XBA | AAA

B → b

X → a

S → XB | AA
A → a | SA
B → b
X → a

The production rule S → XB and B → XBA is not in GNF, so we substitute X → a in the production rule S → XB and B → XBA as:

S → aB | AA

A → a | aBA | AAA

B → b

X → a

S → XB | AA
A → a | XBA | AAA
B → b
X → a

Now we will remove left recursion (A → AAA), we get:

S → aB | AA

A → aA' | aBAA'

A' → AAA' | ε

B → b

X → a

A → A α | β

A → AAA | a

A → aA'

A' → AAA' | ε

Now we will remove null production A' → ε, we get:

S → aB | AA

A → aA' | aBAA' | a | aBA

A' → AAA' | AA

B → b

X → a

S → aB | AA

A → aA' | aBAA'

A' → AAA' | ε

B → b

X → a

The production rule S → AA is not in GNF, so we substitute

A → aA' | aBAA' | a | aBA in production rule S → AA as:

S → aB | aA'A | aBAA'A | aA | aBAA

A → aA' | aBAA' | a | aBA

A'→ AAA'

A' → aA'A | aBAA'A | aA | aBAA

B → b

X → a

S → aB | AA

A → aA' | aBAA' | a | aBA

A' → AAA' | AA

B → b

X → a

The production rule A' → AAA' is not in GNF, so we substitute

A → aA' | aBAA' | a | aBA in production rule A' → AAA' as:

S → aB | aA'A | aBAA'A | aA | aBAA

A → aA' | aBAA' | a | aBA

A' → aA'AA' | aBAA'AA' | aAA' | aBAAA'

A' → aA'A | aBAA'A | aA | aBAA

B → b

X → a

Hence, this is the GNF form for the grammar G.

S → aB | aA'A | aBAA'A | aA | aBAA

A → aA' | aBAA' | a | aBA

A' → AAA'

A' → aA'A | aBAA'A | aA | aBAA

B → b

X → a