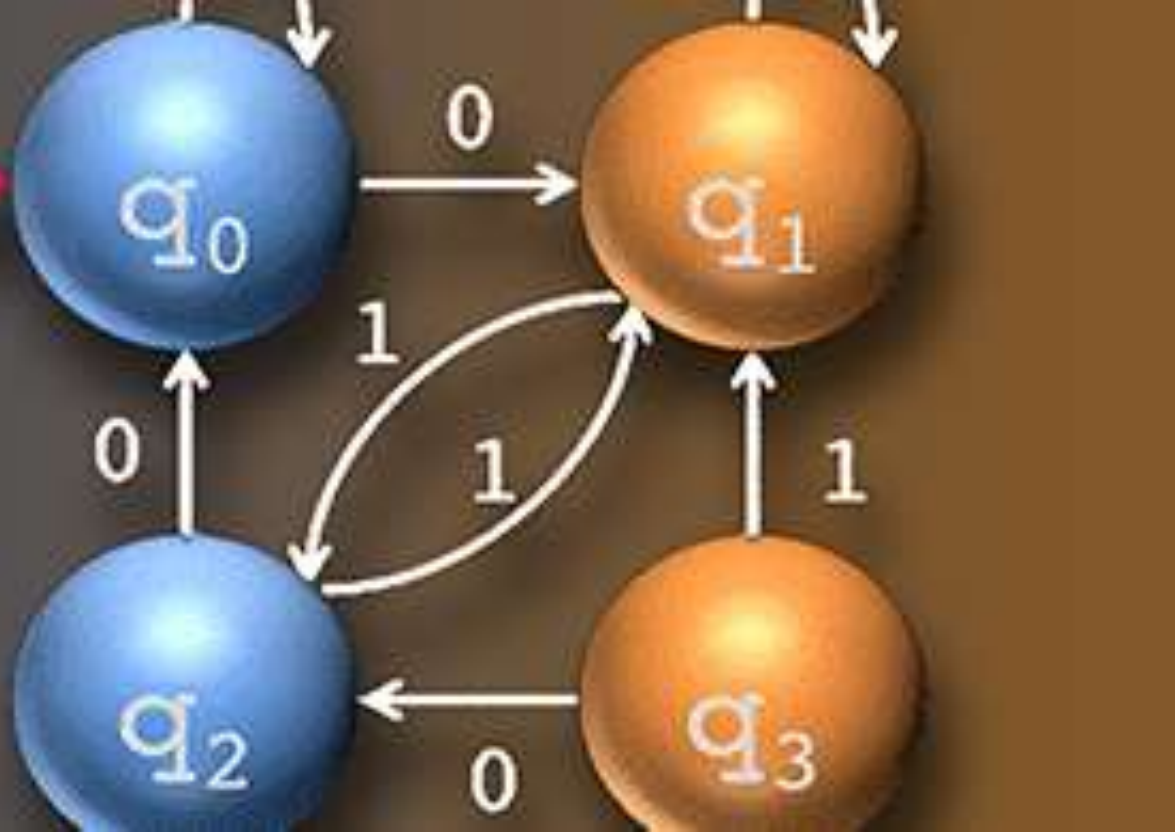
CSE 305

Theory of COMPUTATION



Lecture 24

Context-Free Grammars (2)

Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

www.mijanrahman.com

Contents

Context-Free Grammars

Derivations Using Grammars

- Derivation is a sequence of production rules. It is used to get the input string through these production rules.
- During parsing we have to take two decisions. These are as follows:
  - We have to decide the non-terminal which is to be replaced.
  - 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 replaced with production rule:
  I. Left-most Derivation, and
  II. Right-most Derivation

Derivations Using Grammars

- **Left-most Derivation:**
- In the left most derivation, the input is scanned and replaced with the production rule from left to right. So in left most derivatives we read the input string from left to right.
- **Example:** Consider the following grammar-

  S → aB / bA

  S → aS / bAA / a

  B → bS / aBB / b
- Let us consider a string w = **aaabbabbba**
- Now, let us derive the string w using leftmost derivation.

Leftmost Derivation-

S → a**B**

→ aa**B**B (Using B → aBB)

→ aaa**B**BB (Using B → aBB)

→ aaab**B**B (Using B → b)

→ aaabb**B** (Using B → b)

→ aaabba**B**B (Using B → aBB)

→ aaabbab**B** (Using B → b)

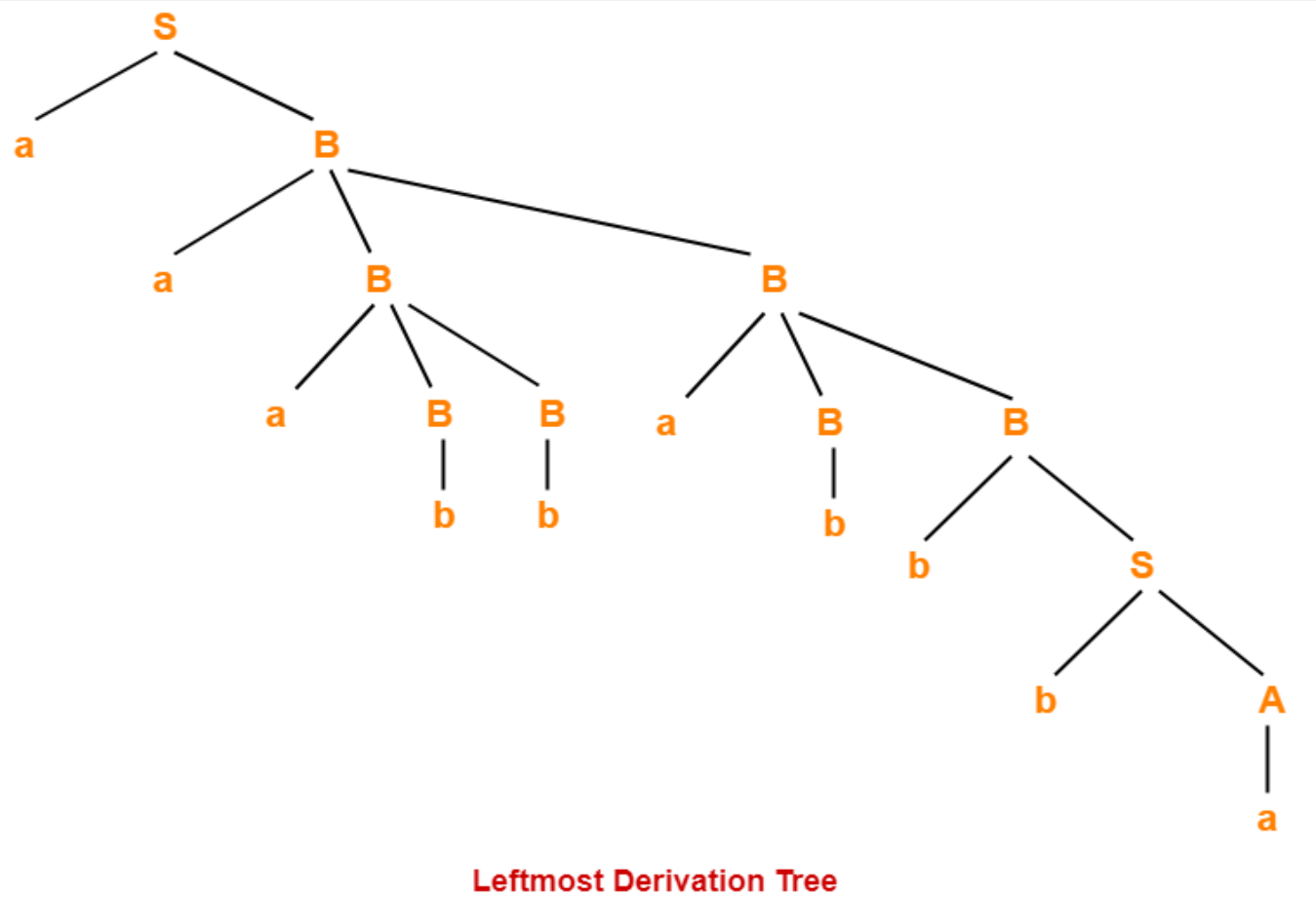
→ aaabbabb**S** (Using B → bS)

→ aaabbabbb**A** (Using S → bA)

→ aaabbabbba (Using A → a)

Derivations Using Grammars

- **Left-most Derivation:**



Leftmost Derivation Tree

Leftmost Derivation-

S → a**B**

→ aa**B**B (Using B → aBB)

→ aaa**B**BB (Using B → aBB)

→ aaab**B**B (Using B → b)

→ aaabb**B** (Using B → b)

→ aaabba**B**B (Using B → aBB)

→ aaabbab**B** (Using B → b)

→ aaabbabb**S** (Using B → bS)

→ aaabbabbb**A** (Using S → bA)

→ aaabbabbba (Using A → a)

Derivations Using Grammars

- **Right-most Derivation:**
- In rightmost derivation, the input is scanned and replaced with the production rule from right to left. So in rightmost derivation, we read the input string from right to left.
- **Example:** Consider the following grammar-

  S → aB / bA

  S → aS / bAA / a

  B → bS / aBB / b
- Let us consider a string w = **aaabbabbba**
- Now, let us derive the string w using right-most derivation.

Rightmost Derivation-

S → a**B**

→ aaB**B** (Using B → aBB)

→ aaBaB**B** (Using B → aBB)

→ aaBaBb**S** (Using B → bS)

→ aaBaBbb**A** (Using S → bA)

→ aaBa**B**bba (Using A → a)

→ aa**B**abbba (Using B → b)

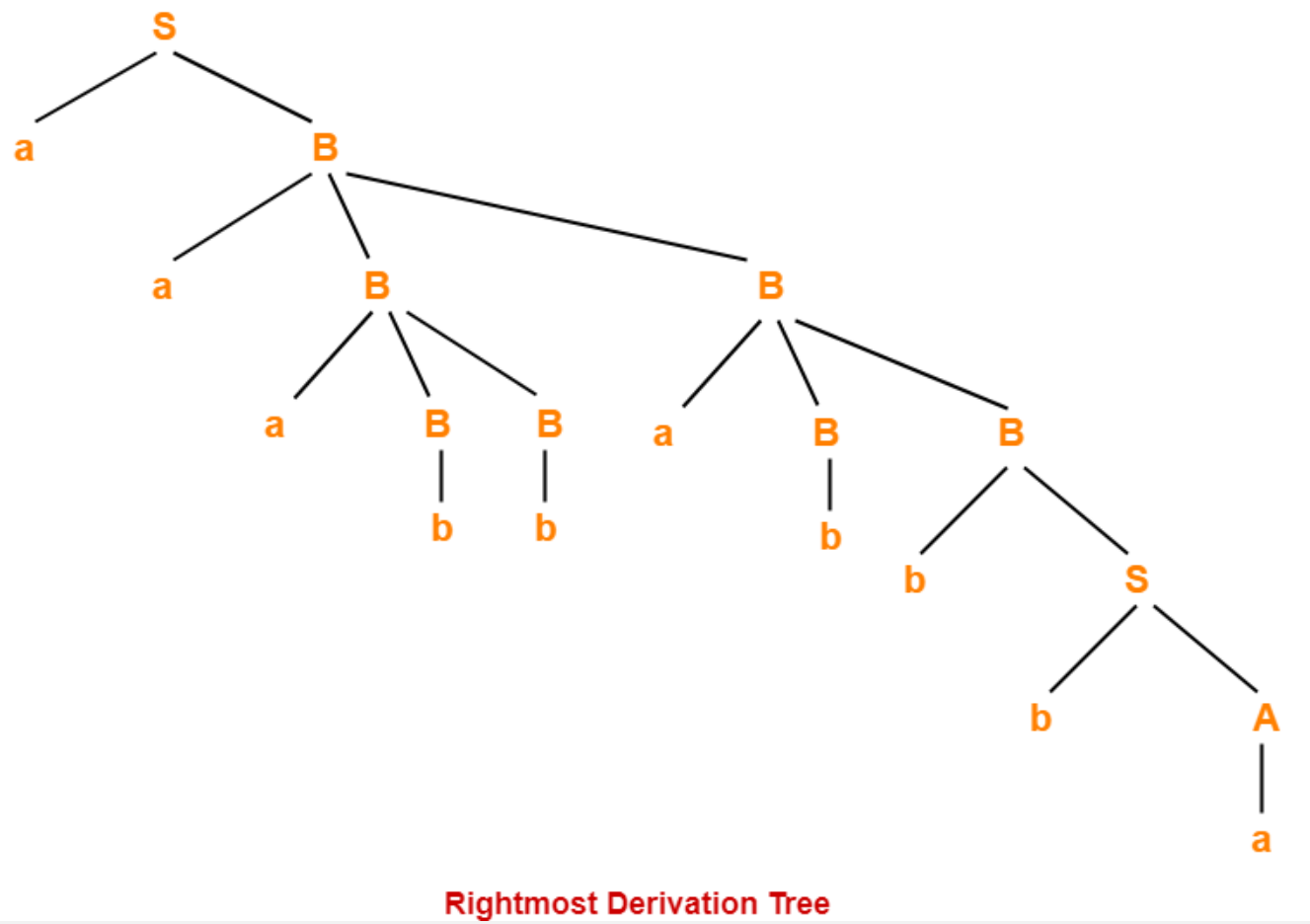
→ aaaB**B**abbba (Using B → aBB)

→ aaa**B**babbba (Using B → b)

→ aaabbabbba (Using B → b)

Derivations Using Grammars

- **Right-most Derivation:**



Rightmost Derivation Tree

Rightmost Derivation-

$S \rightarrow aB$

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

$\rightarrow aaBaBB$ (Using $B \rightarrow aBB$)

$\rightarrow aaBaBbS$ (Using $B \rightarrow bS$)

$\rightarrow aaBaBbbA$ (Using $S \rightarrow bA$)

$\rightarrow aaBaBbba$ (Using $A \rightarrow a$)

$\rightarrow aaBabbba$ (Using $B \rightarrow b$)

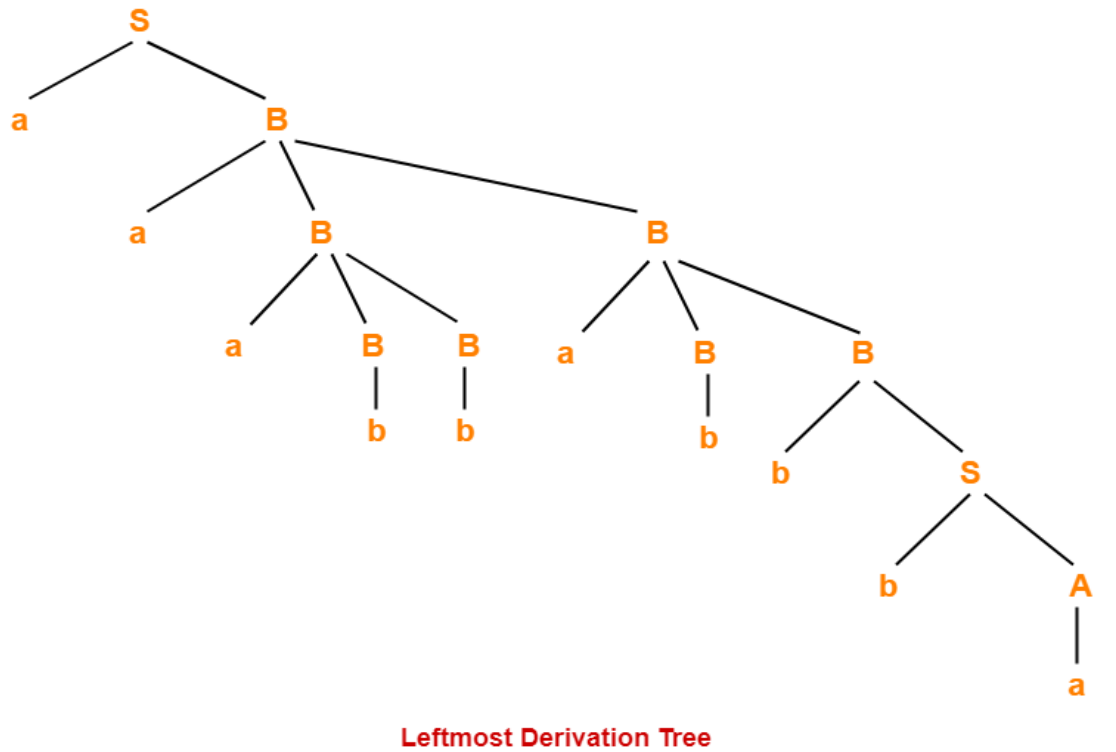
$\rightarrow aaaBBabbba$ (Using $B \rightarrow aBB$)

$\rightarrow aaaBbabbba$ (Using $B \rightarrow b$)

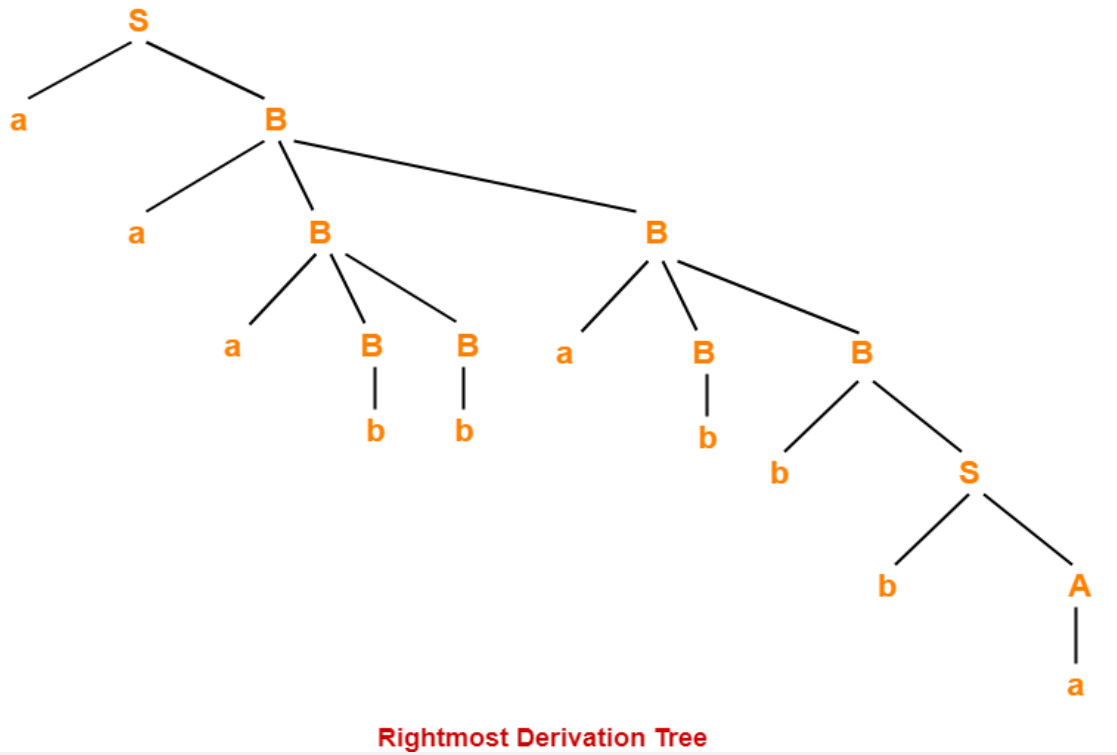
$\rightarrow aaabbabbba$ (Using $B \rightarrow b$)

Derivations Using Grammars

- **Left-most derivation tree (Parse tree) = Right-most derivation tree (Parse tree).**
- Hence, the given grammar is unambiguous.



Leftmost Derivation Tree



Rightmost Derivation Tree

Removal of Ambiguity

- We can remove ambiguity solely on the basis of the following two properties –

1. Precedence:

- If different operators are used, we will consider the precedence of the operators. The three important characteristics are:
  1. The level at which the production is present denotes the priority of the operator used.
  2. The production at **higher levels** will have **operators with less priority**. In the parse tree, the nodes which are at top levels or close to the root node will contain the lower priority operators.
  3. The production at **lower levels** will have operators with **higher priority**. In the parse tree, the nodes which are at lower levels or close to the leaf nodes will contain the higher priority operators.

Removal of Ambiguity

2. Associativity:

- If the same precedence operators are in production, then we will have to consider the associativity.
    - If the associativity is left to right, then we have to prompt a left recursion in the production. The parse tree will also be left recursive and grow on the left side. +, -, *, / are left associative operators.
    - If the associativity is right to left, then we have to prompt the right recursion in the productions. The parse tree will also be right recursive and grow on the right side. ^ is a right associative operator.

Removal of Ambiguity

- If the grammar is ambiguous then it is desirable to find an equivalent unambiguous CFG.
- Now we will discuss how to remove the ambiguity from the grammar. If a given grammar is in the following form:

$$X \longrightarrow \alpha X \beta X \gamma \mid \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots\ldots \mid \alpha_n$$

- Then the ambiguity from this grammar can be removed by using following equations:

$$X \longrightarrow \alpha X \beta X_1 \gamma \mid X_1$$

$$X_1 \longrightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots\ldots \mid \alpha_n$$

- If the grammar has more than one ambiguous production than the same method can be applied repetitively.

Removal of Ambiguity

- **Example: Remove the ambiguity from the following grammar:**

$$E \longrightarrow E * E \mid E + E \mid (E) \mid id$$

SOLUTION:

- Now compare the given production rules

$$E \longrightarrow E * E \mid E + E \mid (E) \mid id$$

With $X \longrightarrow \alpha X \beta X \gamma \mid \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots\ldots \mid \alpha_n$

- We find the following similarity between them:

$$X = E,\ \alpha = \text{empty}\ (\epsilon),\ \beta = *,\ \gamma = \text{empty}\ (\epsilon),\ \alpha_1 = E + E,\ \alpha_2 = (E),\ \alpha_3 = id$$

- So, the ambiguity can be removed by using following rules:

$$X \longrightarrow \alpha X \beta A_1 \gamma \mid A_1$$

$$X_1 \longrightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots\ldots \mid \alpha_n$$

Removal of Ambiguity

SOLUTION:

- We will get

$$E \longrightarrow E*T \mid T \qquad (1)$$

$$T \longrightarrow E+E \mid (E) \mid id \qquad (2)$$

- Now, if we put Eq.(1) in Eq.(2), we will obtain:

$$T \longrightarrow T+T \mid (T) \mid id \qquad (3)$$

- Again apply the same method to remove ambiguity, compare Eq.(3) with Eq.(4):

$$X \longrightarrow \alpha X \beta X \gamma \mid \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots\ldots \mid \alpha_n \qquad (4)$$

Removal of Ambiguity

$$T \longrightarrow T + T \mid (T) \mid id$$

$$X \longrightarrow \alpha X \beta X \gamma \mid \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots\ldots \mid \alpha_n$$

SOLUTION:

- Thus, after comparison of Eq.(3) with Eq.(4), we get:

$$A = T, \alpha = \text{ empty}, \beta = +, \gamma = \text{empty}, \alpha_1 = (E), \alpha_2 = id$$

- Now the solution is:

$$T \longrightarrow T + F \mid F \qquad (5)$$

$$F \longrightarrow (T) \mid id \qquad (6)$$

- So, the final solution may be written as:

$$E \longrightarrow E * T \mid T$$

$$T \longrightarrow E + E \mid (E) \mid id$$

$$T \longrightarrow T + F \mid F$$

$$F \longrightarrow (T) \mid id$$

The given grammar:

$$E \longrightarrow E * E \mid E + E \mid (E) \mid id$$

Removal of Left Recursion

- **Left Recursion**
  - A CFG is called left recursive grammar, if the production starts with the same symbol as on its left side. In other words, if the first symbol on the RHS of the production is similar to the symbol on its LHS then the grammar is said to be left recursive.

$$X \longrightarrow X\alpha_1 \mid X\alpha_2 \mid X\alpha_3 \mid \ldots\ldots \mid X\alpha_n \mid \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots\ldots \mid \beta_m$$

where α and β are any combination of terminals and non-terminals.

- **Removal of Left Recursion**
  - The left recursion from the corresponding grammar can be removed by using the following equations:

$$X \longrightarrow \beta_1 X_1 \mid \beta_2 X_1 \mid \beta_3 X_1 \mid \ldots\ldots \mid \beta_m X_1$$

$$X_1 \longrightarrow \alpha_1 X_1 \mid \alpha_2 X_1 \mid \alpha_3 X_1 \mid \ldots\ldots \mid \alpha_n X_1 \mid \epsilon$$

where X_1 is assumed to be a new non-terminal.

Removal of Left Recursion

- **Example: Remove the Left Recursion from the following grammar:**

$$E \longrightarrow E * E \mid E + E \mid (E) \mid id$$

- **SOLUTION**

Compare the given production $E \longrightarrow E * E \mid E + E \mid (E) \mid id$

with the equation: $X \longrightarrow X\alpha_1 \mid X\alpha_2 \mid X\alpha_3 \mid \ldots\ldots \mid X\alpha_n \mid \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots\ldots \mid \beta_m$

We will find the following similarities between them:

$X = E,\ \alpha_1 = *E,\ \alpha_2 = + E,\ \beta_1 = (E),\ \beta_2 = id$

Using:

$$X \longrightarrow \beta_1 X_1 \mid \beta_2 X_1 \mid \beta_3 X_1 \mid \ldots\ldots \mid \beta_m X_1$$
$$X_1 \longrightarrow \alpha_1 X_1 \mid \alpha_2 X_1 \mid \alpha_3 X_1 \mid \ldots\ldots \mid \alpha_n X_1 \mid \epsilon$$

- **So, after the Remove the Left Recursion, the solution will be:**

$$X \longrightarrow (E)\, X1 \mid id\, X_1$$
$$X_1 \longrightarrow *EX_1 \mid + EX_1 \mid \epsilon$$

Left Factoring

- **The process in which uncommon parts of two or more productions are grouped into one production is called the left factoring.**
- For example, any production of the form:

$$X \longrightarrow A\alpha_1$$
$$X \longrightarrow A\alpha_2$$
$$X \longrightarrow A\alpha_n$$

Or $X \longrightarrow A\alpha_1 \,|A\alpha_2 \,|A\alpha_3 \,|.....|A\ \alpha_n$

Can be replaced by:

$$X \longrightarrow AB$$
$$B \longrightarrow \alpha_1 \,|\alpha_2 \,|\alpha_3 \,|.....|\alpha_n$$

Left Factoring

- **Example:** Do left factoring in the following grammar-

  $A \rightarrow aAB / aBc / aAc$

Solution:

- Step-1:

  $A \rightarrow aA'$

  $A' \rightarrow AB / Bc / Ac$

  Again, this is a grammar with common prefixes.

- Step-2:

  $A \rightarrow aA'$

  $A' \rightarrow AD / Bc$

  $D \rightarrow B / c$

  This is a left factored grammar.

Left Factoring

- **Example:** Do left factoring in the following grammar-

  S → bSSaaS / bSSaSb / bSb / a

Solution:

- Step-1:

  S → bSS' / a

  S' → SaaS / SaSb / b

  Again, this is a grammar with common prefixes.

- Step-2:

  S → bSS' / a

  S' → SaA / b

  A → aS / Sb

  This is a left factored grammar.

?

THE END