United College of Engineering and Research, Naini, Prayagraj

Computer Science and Engineering Department

Compiler Design

Left Recursion:-

A grammar becomes left-recursive if it has any non-terminal 'A' whose derivation contains 'A' itself as the left-most symbol. Left-recursive grammar is considered to be a problematic situation for top-down parsers. Top-down parsers start parsing from the Start symbol, which in itself is non-terminal. So, when the parser encounters the same non-terminal in its derivation, it becomes hard for it to judge when to stop parsing the left non-terminal and it goes into an infinite loop.

Ex.

(1) $A \rightarrow A\alpha \mid \beta$ (immediate left recursion)

(2) $S \rightarrow A\alpha \mid \beta$ (indirect left recursion)

$A \rightarrow Sd$

Removal of Left Recursion:-

One way to remove left recursion is to use the following technique:

The production

$$\boxed{A \rightarrow A\alpha \mid \beta} \longrightarrow \boxed{\begin{array}{c} A \rightarrow \beta A' \\ A' \rightarrow \alpha A' \mid \varepsilon \end{array}}$$

Ex. Remove left recursion from the following grammar.

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow id \mid (E)$

Solution:-

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \epsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \epsilon$
$F \rightarrow id \mid (E)$

Second method is to use the following algorithm, which should eliminate all direct and indirect left recursions.

START

Arrange non-terminals in some order like A1, A2, A3,…, A_n

for each i from 1 to n
{
for each j from 1 to i-1
{
replace each production of form $A_i \Rightarrow A_j \boldsymbol{\gamma}$
with $A_i \Rightarrow \delta 1\boldsymbol{\gamma} \mid \delta 2\boldsymbol{\gamma} \mid \delta 3\boldsymbol{\gamma} \mid \ldots \mid \boldsymbol{\gamma}$
where $A_j \Rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_n$ are current A_j productions
}
}
eliminate immediate left-recursion

END

Example

The production set

S => Aα | β
A => Sd

after applying the above algorithm, should become

S => Aα | β
A => Aαd | βd

and then, remove immediate left recursion using the first technique.

A => βdA'
A' => αdA' | ε

Left Factoring:-

If RHS of more than one production starts with the same symbol, then such a grammar is called as **Grammar With Common Prefixes**.

<u>Example-</u>

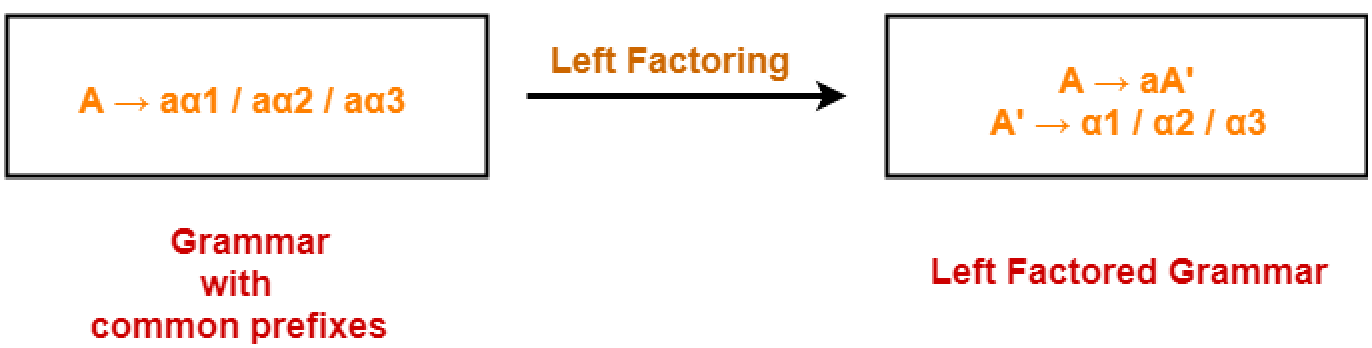
$$A \rightarrow \alpha\beta_1 / \alpha\beta_2 / \alpha\beta_3$$

- This kind of grammar creates a problematic situation for Top down parsers.
- Top down parsers can not decide which production must be chosen to parse the string in hand.

To remove this confusion, we use left factoring.

Left factoring *is a process by which the grammar with common prefixes is transformed to make it useful for Top down parsers.*

<u>Example-</u>



Ex. Do left factoring in the following grammar-

$S \rightarrow iEtS / iEtSeS / a$

$E \rightarrow b$

Sol.

The left factored grammar is-

$S \rightarrow iEtSS' / a$

S’ → eS / ∈

E → b

Ex. Do left factoring in the following grammar-

A → aAB / aBc / aAc

Step-01:

A → aA’

A’ → AB / Bc / Ac

Again, this is a grammar with common prefixes.

Step-02:

A → aA’

A’ → AD / Bc

D → B / c

This is a left factored grammar.