Lecture note

Compiler Design

CSE \_313

Topic: Left Recursion And Left Factoring

1. Left Recursion:

#### **1. What is Left Recursion?**

In context-free grammar, **left recursion** occurs when a non-terminal appears on the leftmost side of its own production.

**Definition:**  
 A grammar is left recursive if there exists a non-terminal A such that:

A → Aα | β

* α is a non-empty string
* β is any production that doesn’t start with A

**Problem:** Causes **infinite recursion** in **top-down parsers**.

#### **Example:**

E → E + T | T

T → T \* F | F

F → (E) | id

Here, both E and T are left recursive.

How to Remove Left Recursion:

A → Aα | β (general transformation)

Becomes:

A → β A′

A′ → α A′ | ε

Where A′ is a new non-terminal and ε (epsilon) is the empty string.

1.Example:

Expr → Expr + Term | Term

Solution:

Expr → Term Expr′

Expr′ → + Term Expr′ | ε

2. Example:

Given:

A → A a | A b | c

Solution:

A → c A′

A′ → a A′ | b A′ | ε

#### **2. What is Left Factoring?**

Left factoring is a grammar rewriting technique used to eliminate **common prefixes** in productions, making the grammar easier for parsers.

**Definition:**  
 If:

A → αβ1 | αβ2

Then rewrite as:

A → α A′

A′ → β1 | β2

Example:

Given:

Stmt → if expr then stmt else stmt

| if expr then stmt

Solution:

Stmt → if expr then stmt Stmt′

Stmt′ → else stmt | ε

2. Example:

Given:

S → abcd | abef | ag

Solution:

S → a S′

S′ → b S′′ | g

S′′ → cd | ef