**LAB SESSION 5: ELIMINATION OF LEFT-RECURSION IN CONTEXT FREE GRAMMAR**

**AIM**: To implement a program in python to eliminate left recursion from a given CFG.

**PROBLEM DEFINITION:** Develop a python program to eliminate left-recursion from a given Context Free Grammar. Accept the Grammar from a file.

**THEORY:** Context-free grammars play a vital role in compilers and programming languages. CFGs are used to define how valid programs should be organized. We use recursion in parsing, whereby examining a sequence of tokens to its grammatical structure, is greatly dependent on these grammars.

**Left Recursion**

A context-free grammar is said to be left recursive if it contains a production rule where the non-terminal on the left-hand side of the rule also appears as the first symbol on the right-hand side. In other words, the grammar is trying to define a non-terminal in terms of itself, creating a recursive loop.

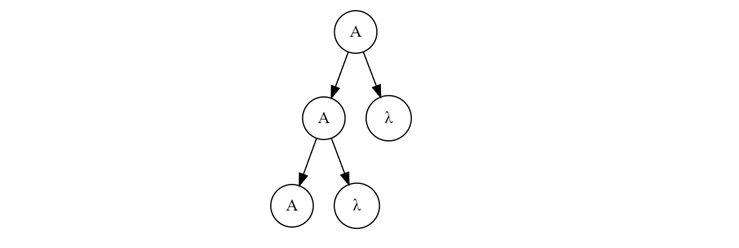
This can be represented formally as: A→Aα|β

Where: A is a non-terminal symbol.

α represents a sequence of terminals and/or non-terminals.

* β represents another sequence of terminals and/or non-terminals.

The most important part here is the presence of A on both sides of the production rule, with it appearing first on the right-hand side. To visualize this, consider the following parse tree −



It is generated by a left-recursive grammar. As the grammar recursively expands the non-terminal 'A' on the left, the tree grows indefinitely downwards on the left side. This continuous expansion makes it unsuitable for top-down parsing, as the parser could get trapped in an infinite loop, trying to expand 'A' repeatedly.

**Problem of Left Recursion for Top-Down Parsing**

The top-down parsing works by starting with the start symbol of the grammar and attempting to derive the input string by applying production rules. When encountering a left-recursive rule, the parser keeps expanding the same non-terminal, leading to an infinite loop. This inability to handle left recursion directly is a significant drawback of top-down parsing methods.

### **Eliminating Left Recursion**

To solve this we can eliminate immediate left recursion from a grammar without altering the language it generates. The general approach involves introducing a new non-terminal and rewriting the recursive rules. Consider a simplified arithmetic expression grammar −

E→E+T|T

T→T∗F|F

F→(E)|id

* Eliminating Left Recursion in `E`: E→TE′

E′→+TE′|ε

* Eliminating Left Recursion in `T`:
* T→FT′
* T′→∗FT′|ε
* The final transformed grammar, free from left recursion, becomes −
* E→TE′
* E′→+TE′|ε
* T→FT′
* T′→∗FT′|ε
* F→(E)|id

**PROGRAM**

**OUTPUT**

**CONCLUSION**