Premier University Chittagong

Department of Computer Science and Engineering

Course Title : Compiler Construction Lab

Course Code: CSE 454

Report No : 06

Report Title : Detection and Elimination of Left Recursion in Context-Free Grammar

Submission Date: 17.09.2025

Submitted By

Name : Rayanul Kader Chowdhury Abid
ID : 210401020 2162
Semester : 8^{th} A Section

Submitted To

Ms. Tanni Dhoom
Assistant Professor
Department of Computer Science and Engineering

Experiment No: 06

Experiment Name: Detection and Elimination of Left Recursion in Context-Free Grammar

Objective: To write a C++ program that takes grammar productions as input, detects **left recursion**, and transforms the grammar into an equivalent **left recursion-free form**.

Algorithm:

- **Start the program.**
- Input the number of productions `n`.
- Input each production (in the form `A->...`).
- For each production:
    - Split the production into **LHS (non-terminal)** and **RHS (rules)**.
    - Break the RHS into multiple rules using `|` as a separator.
    - Check each rule:
        - If the RHS starts with the LHS, it indicates **left recursion** → store in **α (alpha)** set.
        - Otherwise, store in **β (beta)** set.
- If no alpha rules exist → **no left recursion**.
- If left recursion exists:
    - Create a new non-terminal `A'`.
    - Rewrite the production as:

      A → β1A' | β2A' | ...

      A' → α1A' | α2A' | ... | ε

Code:

```
#include <iostream>

#include <string>

#include <vector>

using namespace std;


int main() {
```

```
int n;
cout << "Enter number of productions: ";
cin >> n;
cin.ignore();

vector<string> productions(n);
cout << "Enter productions :" << endl;
for (int i = 0; i < n; i++) {
    getline(cin, productions[i]);
}

for (int i = 0; i < n; i++) {
    string prod = productions[i];

    int arrowPos = prod.find("->");
    if (arrowPos == string::npos) {
        cout << "Invalid production: " << prod << endl;
        continue;
    }

    string lhs = prod.substr(0, arrowPos);
    string rhs = prod.substr(arrowPos + 2);

    vector<string> rules;
    string temp = "";
    for (char c : rhs) {
        if (c == '|') {
            rules.push_back(temp);
```

```
        temp = "";
    } else {
        temp += c;
    }
}
if (!temp.empty()) rules.push_back(temp);

vector<string> alpha, beta;
for (string r : rules) {
    if (r.find(lhs) == 0) {
        alpha.push_back(r.substr(lhs.size()));
    } else {
        beta.push_back(r);
    }
}

if (alpha.empty()) {
    cout << "No left recursion in " << lhs << endl;
} else {
    cout << "Left recursion found in " << lhs << endl;
    cout << "Eliminated form:" << endl;

    string newLHS = lhs + "'";
    cout << lhs << " -> ";
    for (int j = 0; j < beta.size(); j++) {
        cout << beta[j] << newLHS;
        if (j != beta.size() - 1) cout << " | ";
    }
```

```
            cout << endl;
            cout << newLHS << " -> ";
            for (int j = 0; j < alpha.size(); j++) {
                cout << alpha[j] << newLHS;
                if (j != alpha.size() - 1) cout << " | ";
            }
            cout << " | epsilon hobe" << endl;
        }
    }
    return 0;
}
```

Input:

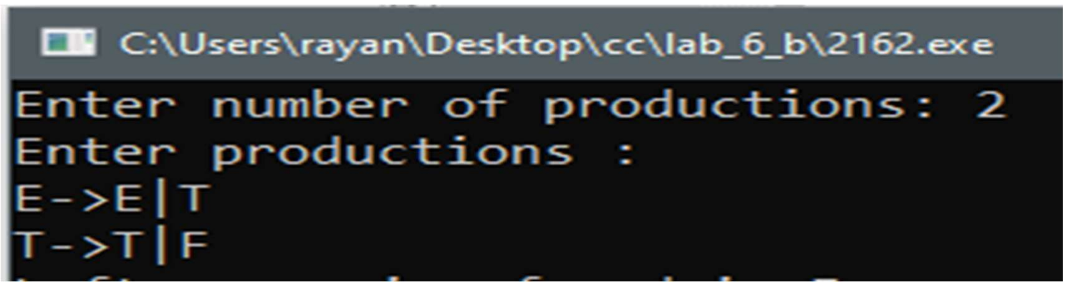

Figure 5.1: Input

Output:

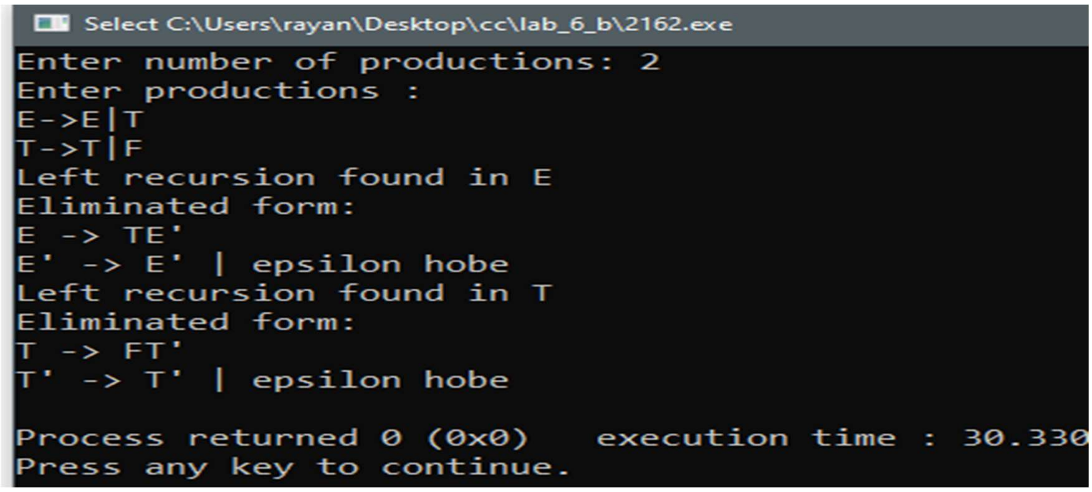

Figure 5.2: Output

Discussion:

This program detects **immediate left recursion** in context-free grammar productions. Left recursion occurs when a production rule has the form:

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

where `A` is a non-terminal, α is a sequence of grammar symbols, and β is a sequence that does not start with `A`.

Left recursion makes **top-down parsers (like recursive descent parsers)** enter infinite recursion. Hence, it must be eliminated for parser implementation.

The algorithm replaces left-recursive productions with an equivalent grammar using a new non-terminal, ensuring that the grammar can be parsed by top-down methods.

This is an important step in **compiler design** during the parsing phase.