Practical - 4

2CS701 – Compiler Construction

Harshit Gajipara

19bce059

**Aim:**

To Implement Left Recursion derivation removal algorithm: Eliminate direct and indirect Left recursion from given grammar for LL(1) parser.

**Code:**

#include <bits/stdc++.h>

using namespace std;

int main()

{

    int n;

    cout << "Enter the number of productions : ";

    cin >> n;

    map<char, vector<string>> map\_with\_recursions, mp;

    vector<string> productions;

    set<char> start\_symbols;

    vector<string> ans;

    cout << "\nEnter the productions (one by one) : \n";

    for (int i = 0; i < n; i++)

    {

        string s;

        cin >> s;

        productions.push\_back(s);

        start\_symbols.insert(s[0]);

        // For each production rule to be analysed, we will be storing it in this string

        string s1 = "";

        // i=0,1,2 are for "E->" so we start with i=3 till all productions for a start symbol

        for (int i = 3; i < s.length(); i++)

        {

            if (s[i] != '|') // scan until new rule

                s1 += s[i];

            else

            {

                if (s1[0] >= 'A' && s1[0] <= 'Z' && s1[0] == s[0]) // string has left recursion

                    map\_with\_recursions[s[0]].push\_back(s1);

                else

                    mp[s[0]].push\_back(s1);

                s1 = "";

            }

        }

        // for last production rule to be stored

        if (s1[0] >= 'A' && s1[0] <= 'Z' && s1[0] == s[0])

            map\_with\_recursions[s[0]].push\_back(s1);

        else

            mp[s[0]].push\_back(s1);

        s1 = "";

    }

    for (auto it = start\_symbols.begin(); it != start\_symbols.end(); it++)

    {

        if (map\_with\_recursions[\*it].size() == 0)

        {

            string ans1 = "";

            ans1 += (\*it);

            ans1 += "-";

            ans1 += ">";

            vector<string> temp = mp[\*it];

            for (int i = 0; i < temp.size(); i++)

            {

                if (i != (temp.size() - 1))

                {

                    ans1 += temp[i];

                    ans1 += "|";

                }

                else

                {

                    ans1 += temp[i];

                }

            }

            ans.push\_back(ans1);

        }

        else

        {

            string temp = "";

            temp += \*it;

            temp += '\'';

            vector<string> temp1 = map\_with\_recursions[\*it];

            vector<string> temp2 = mp[\*it];

            string prod1 = "";

            prod1 += \*it;

            prod1 += "->";

            for (int i = 0; i < temp2.size(); i++)

            {

                if (i != (temp2.size() - 1))

                {

                    string tt = temp2[i];

                    tt += temp;

                    prod1 += tt;

                    prod1 += "|";

                }

                else

                {

                    string tt = temp2[i];

                    tt += temp;

                    prod1 += tt;

                }

            }

            string prod2 = "";

            prod2 += temp;

            prod2 += "->";

            for (int i = 0; i < temp1.size(); i++)

            {

                string tt = temp1[i].substr(1);

                tt += temp;

                prod2 += tt;

                prod2 += "|";

            }

            prod2 += "~";

            ans.push\_back(prod1);

            ans.push\_back(prod2);

        }

    }

    cout << "\nGrammar after eliminating left recursion is : \n";

    for (int i = 0; i < ans.size(); i++)

        cout << ans[i] << endl;

    return 0;

}

/\*

4

S->ABC

A->Aa|Ad|b

B->Bb|e

C->Cc|g

\*/

**Output:**



