College of Engineering Trivandrum

Compiler Design Lab

Abhishek Manoharan
S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

September 30, 2020

CS431 - Compiler Design Lab · 2020 ·

Exp 9

1 First and Follow

1.1 Aim

Write program to find Simulate First and Follow of any given grammar

1.2 Theory

First

FIRST is applied to the r.h.s. of a production rule, and tells us all the terminal symbols that can start sentences derived from that r.h.s.

Follow

FOLLOW is used only if the current non-terminal can derive ε; then we're interested in what could have followed it in a sentential form. (NB: A string can derive ε if and only if ε is in its FIRST set.

1.3 Algorithm

Algorithm 1: Algorithm for precedence parsing

```
FIRST ( X ) for all grammar symbols X
    If X is terminal , FIRST ( X ) = { X }.
    If X -> e is a production , then add e to FIRST ( X ) .
    If X is a non - terminal , and X -> Y1 Y2 ... Yk is a production , and e is in all of
    FIRST ( Y1 ) , ... , FIRST ( Yk ) , then add e to FIRST ( X ) .
    If X is a non - terminal , and X -> Y1 Y2 ... Yk is a production , then add a to FIRST ( X
     ) if for some i , a is in FIRST ( Yi ) , and e is in all of FIRST ( Y1 ) ,... , FIRST (
    Yi -1) .
FOLLOW ( A ) for all non - terminals A
    If $ is the input end - marker , and S is the start symbol , $ e FOLLOW ( S ) .
    If there is a production , A -> aBb , then ( FIRST ( b ) - e ) subset_of FOLLOW ( B ) .
    If there is a production , A -> aB , or a production A -> aBb , where e belongs to FIRST (
     b ) , then FOLLOW ( A ) subset of FOLLOW ( B ) .
```

1.4 Code

```
#include <bits/stdc++.h>
using namespace std;
vector<vector<string>> get_production(unordered_map<char, int> &non_term, int *num)
{
    int non = -1;
    string s;
    vector<vector<string>> production(100);
    getline(cin, s);
    while (s != "")
    {
        int non_index;
        char left = s[0];
        if (non_term.find(s[0]) == non_term.end())
        {
            non_term[s[0]] = ++non;
            non_index = non;
        }
        else
        {
            non_index = non_term[s[0]];
        }
        string right = s.substr(4, s.size() - 4);
```

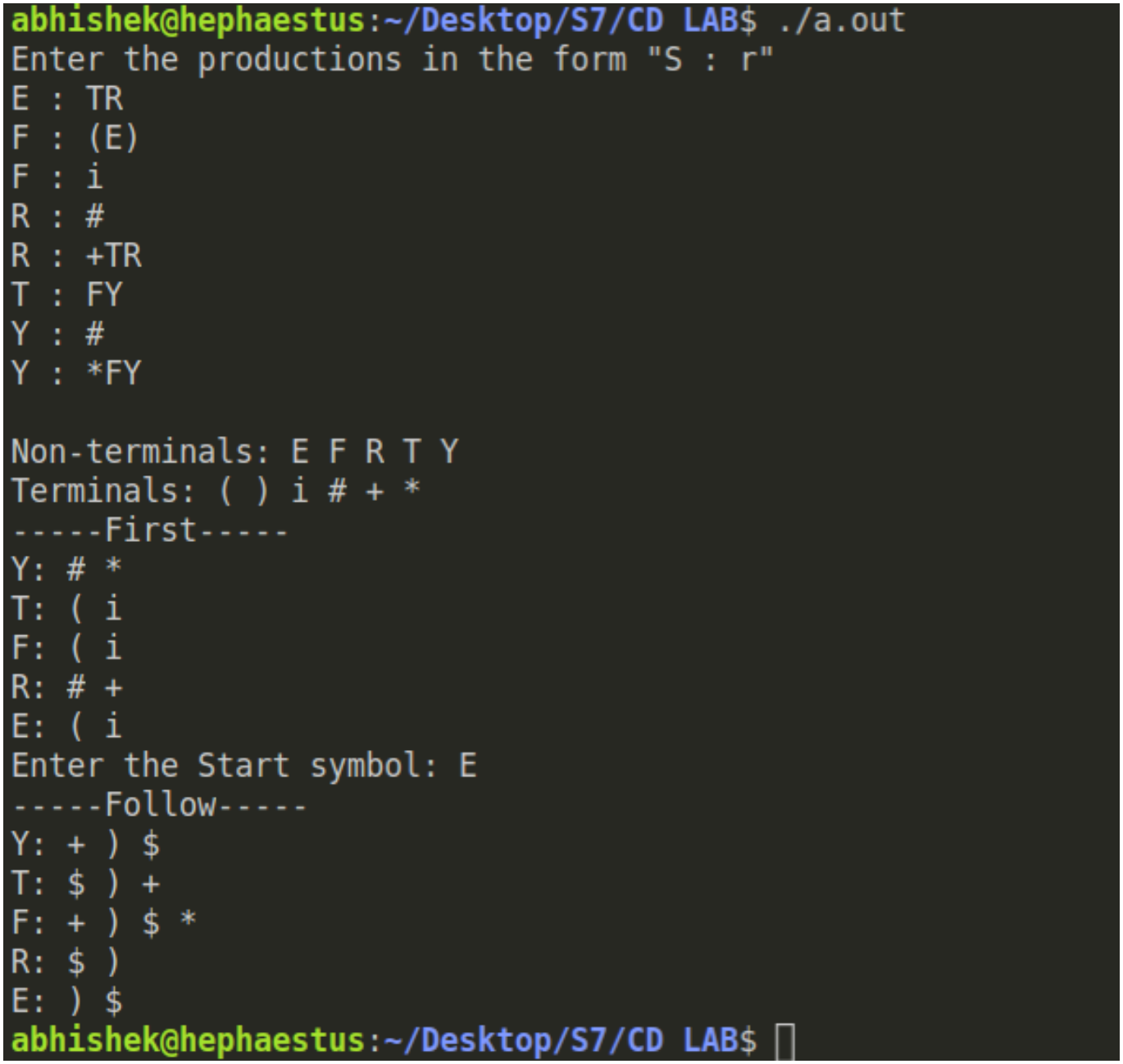
```
        production[non].push_back(right);
        //cout << "right side " << right << endl;
        getline(cin, s);
    }
    *num = non;
    return production;
}

unordered_set<char> split_string(string s)
{
    int n = s.size();
    unordered_set<char> result;
    for (int i = 0; i < n; ++i)
    {
        if (s[i] != ' ')
        {
            result.insert(s[i]);
        }
    }
    return result;
}

vector<char> find_first(char c, vector<vector<string>> production, vector<vector<char>> &First
    , unordered_map<char, int> umap)
{
    vector<char> res;
    if (umap.find(c) == umap.end())
    {
        res.push_back(c);
        return res;
    }
    int num = umap[c];
    if (First[num].size() != 0)
    {
        return First[num];
    }
    int n = production[num].size();
    for (int i = 0; i < n; ++i) // iterate through each production
    {
        string m = production[num][i];
        int right_size = m.size();
        for (int j = 0; j < right_size; ++j) // iterate through each charecter in production
        {
            if (umap.find(m[j]) == umap.end()) //if right side of production  is a terminal
            {
                if (find(res.begin(), res.end(), m[j]) == res.end())
                {
                    res.push_back(m[j]);
                }

                break;
            }
            else // Non terminal
            {
                vector<char> temp = find_first(m[j], production, First, umap); // finding
    first of  j th non terminal
                //cout << "called for first of " << m[j] << endl;
                int first_char = temp.size();
                int flag = 1;
                for (int k = 0; k < first_char; ++k)
                {
                    if (temp[k] == '#')
                    {
                        // cout << "Epsilon found in first of " << m[j] << endl;
                        flag = 0;
                    }
                    if (find(res.begin(), res.end(), temp[k]) == res.end())
                    {
                        if (temp[k] != '#')
                        {
                            res.push_back(temp[k]);
                        }
                        else
                        {
                            if (j == right_size - 1)
                            {
```

```
                                res.push_back(temp[k]);
                            }
                        }
                    }
                }
                if (flag == 1)
                {
                    break;
                }
            }
        }
    }
    First[num] = res;
    return res;
}
unordered_set<char> find_follow(char c, vector<vector<string>> production, vector<
    unordered_set<char>> &follow, unordered_map<char, int> umap, vector<vector<char>> first)
{
    if (!follow[umap[c]].empty())
    {
        return follow[umap[c]];
    }
    //cout << "called follow of " << c << endl;
    unordered_set<char> res;
    if (umap[c] == 0)
    {
        //cout << "added $ in follow of " << c << endl;
        res.insert('$');
    }
    int n = production.size();
    for (int i = 0; i < n; ++i)
    {
        for (auto x : production[i])
        {                       // considering each production
            int m = x.size(); //read rhs charecter by charecter
            for (int j = 0; j < m; ++j)
            {
                if (x[j] == c) // if we find charecter in right side of production
                {
                    //cout << c << " found in production " << x << endl;
                    if (j == m - 1)
                    { //last element
                        //  cout << c << " is the edning charecter" << endl;
                        char check;
                        for (auto y : umap)
                        {
                            if (y.second == i)
                            {
                                check = y.first;
                            }
                        }
                        if (check != c)
                        {
                            unordered_set<char> sample = find_follow(check, production, follow
    , umap, first);
                            for (auto y : sample)
                            {
                                //cout << y << " inserted in follow of " << c << endl;

                                res.insert(y);
                            }
                            //cout << endl;
                        }
                    }
                    else
                    {
                        for (int k = j + 1; k < m; ++k)
                        {
                            int flag = 0;
                            if (umap.find(x[k]) == umap.end())
                            { // checking whether char is termi if so add and stop
                                // cout << "since found non terminal " << x[k] << "stop here
    added it "
                                //<< "in follow of " << c << endl;
                                res.insert(x[k]);
                                flag = 1;
```

```
                            }
                            else
                            { // if it is a non terminal then add its first
                                int first_b = first[umap[x[k]]].size();
                                for (int l = 0; l < first_b; ++l)
                                {
                                    if (first[umap[x[k]]][l] != '#')
                                    {
                                        res.insert(first[umap[x[k]]][l]);
                                        //cout << first[umap[x[k]]][l] << " Added to follow of " << c << endl;
                                        if (l == first_b - 1) // first[b] has #
                                        {
                                            char check;
                                            for (auto y : umap)
                                            {
                                                if (y.second == i)
                                                {
                                                    check = y.first;
                                                }
                                            }
                                            if (check != c)
                                            {
                                                unordered_set<char> sample = find_follow(check, production, follow, umap, first);
                                                for (auto y : sample)
                                                {
                                                    //cout << y << "added to follow of " << c << endl;
                                                    res.insert(y);
                                                }
                                            }
                                        }
                                    }
                                    else
                                    {
                                        flag = 1;
                                    }
                                }
                            }
                            if (flag == 1)
                            {
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
    return res;
}
int main()
{
    int non = -1;
    string s;
    vector<vector<string>> production(100);
    cout << "Enter the productions in the form \"S : r\" " << endl;
    unordered_map<char, int> non_term;
    production = get_production(non_term, &non);
    unordered_set<char> terminals;
    unordered_set<char> non_terminals;
    cout << "Non-terminals: ";
    getline(cin, s);
    non_terminals = split_string(s);
    // for (auto x : non_terminals)
    // {
    //     cout << x << " ";
    // }
    // cout << endl;
    cout << "Terminals: ";
    getline(cin, s);
    vector<vector<char>> First(non + 1);
    terminals = split_string(s);
    // for (auto x : terminals)
    // {
```

```
    //      cout << x << " ";
    // }
    // cout << endl;
    // cout << "number of non term is : " << non << endl;
    for (auto x : non_terminals)
    {
        First[non_term[x]] = find_first(x, production, First, non_term);
    }
    cout << "-----First----- " << endl;
    for (auto x : non_terminals)
    {
        cout << x << ": ";
        for (auto y : First[non_term[x]])
        {
            cout << y << " ";
        }
        cout << endl;
    }
    vector<unordered_set<char>> follow(non + 1);
    cout << "Enter the Start symbol: ";
    char c;
    cin >> c;
    cout << "-----Follow-----" << endl;
    for (auto x : non_terminals)
    {
        if (follow[non_term[x]].empty())
            follow[non_term[x]] = find_follow(x, production, follow, non_term, First);
    }
    for (auto x : non_terminals)
    {
        cout << x << ": ";
        for (auto y : follow[non_term[x]])
        {
            cout << y << " ";
        }
        cout << endl;
    }

    return 0;
}
```

1.5 Output




```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
-----First-----
Y: # *
T: ( i
F: ( i
R: # +
```

```
E: ( i
Enter the Start symbol: E
-----Follow-----
Y: + ) $
T: $ ) +
F: + ) $ *
R: $ )
E: ) $
```

1.6 Result

Implemented the program to find FIRST and FOLLOW. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.