PRACTICAL FILE

BE (CSE) 6th Semester

COMPILER DESIGN (CS654)

March 2021 – May 2021

Submitted By

Jatin Ghai
(UE183042)

Submitted To

Dr. Akashdeep
Assistant Professor, Computer Science and Engineering

Computer Science and Engineering
University Institute of Engineering and Technology
Panjab University, Chandigarh – 160014, INDIA
2021

Experiment No. 7

AIM :- To evaluate first and follow for a given grammar.

First and Follow sets can provide the actual position of any terminal in the derivation. This is done to create the parsing table where the decision of replacing T[A, t] = α with some production rule.

First(A) is a set of terminal symbols that begin in strings derived from A.
Follow(A) is the set of terminals a that can appear to the right of A.

Source Code:

```
#include<bits/stdc++.h>

using namespace std;

int main(){

    int nProds;

    unordered_map<char,int> symbol;

    map< char,vector<string> > prod;

    vector<char> order;

    unordered_map<char,set<char>> FRT,FLW;

    unordered_map<char, set<char> > later;

    int count = 1;

    string lhs,rhs;


    cout << "Enter number of productions\n";

    cin >> nProds;

    cout << "Enter productions, Use # for epsilon and use only
single character for symbols\n";

    for(int i = 0; i < nProds; i++){

        cin >> lhs >>rhs;
```

```
        if( symbol.find(lhs[0]) == symbol.end() )
            order.push_back(lhs[0]);
            symbol[lhs[0]] = i;
        prod[lhs[0]].push_back(rhs);
    }
    cout << endl;


    for(int i = order.size()-1; i >= 0; i--){
        symbol[order[i]] = 0;
        for(auto x: prod[order[i]]){
            if( x[0] == '#'){
                FRT[order[i]].insert('#');
            }else{
                if(!isupper(x[0]) ){
                    FRT[order[i]].insert(x[0]);
                }
                else{
                    int j =0;
                    int eps =1;
                    // T FB
                    while(j < x.size() && isupper(x[j]) && eps){
                        eps = 0;
                        for(auto y : FRT[x[j]]){
                            if(y == '#'){
                                eps = 1;
                            }
                            FRT[order[i]].insert(y);
                        }
```

```
                        j++;


                        if(j == x.size() || !isupper(x[j]) || eps ==
0){
                            cout << order[i]<<endl;
                            FRT[order[i]].erase('#');
                            for(auto y : FRT[x[j-1]]){
                                if(y == '#'){
                                    FRT[order[i]].insert(y);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    cout << "First of Non terminals\n";
    for(auto a: FRT){
        cout << a.first << " ";
        for(auto y: a.second){
            cout << y<<" ";
        }
        cout << endl;
    }
    cout << endl;


    FLW[order[0]].insert('$');
```

```
    int eps = 0;


    for(auto x: prod){
        for(auto y: x.second){
            if(y[0] == '#') continue;
            // y  each rhs of productions of x.first
            if( isupper(y[y.size()-1]) && y[y.size()-1] != x.first)
{ // adding follow of lhs as follow of rhs[last] if it is non
terminal
                later[y[y.size()-1]].insert(x.first);
                symbol[y[y.size()-1]] = 1;
            }

            for(int i =0; i < y.size()-1; i++){ //n2 to calculate
whole prod

                if(! isupper(y[i])) continue; // not for terminal

                eps = 1;
                int j = i+1;
                for(j; j <y.size() && eps == 1; j++){
                    eps = 0;
                    if(!isupper(y[j])){ // found terminal,terminate
                        FLW[y[i]].insert(y[j]);
                        break;
                    }
                    for(auto rex: FRT[y[j]]){
                        // cout << y[i] <<" "<<rex << " first of
"<< y[j]<<endl;
```

```
                    if(rex == '#'){
                        eps = 1;
                        continue;
                    }
                    FLW[y[i]].insert(rex);
                }
            }
            if(eps == 1 && j == y.size() && y[i] != x.first){
                later[y[i]].insert(x.first);
                symbol[y[i]] = 1;
            }
        }
    }
}

queue<char> q;
for(auto x: symbol){
    if(x.second == 0) q.push(x.first);
}

int sig = 0,sig2 = 0;
while(!q.empty()){
    char ch = q.front();
    // cout << "q.front "<< ch<<endl;
    q.pop();
    for(auto y: later){
        sig = 0;
        sig2 = 0;
```

```
            // cout << y.first <<endl;
            for(auto x: y.second){
                sig2 = 1;
                // cout << x<<" ";
                if(ch == x){
                    for(auto z: FLW[ch]){
                        // cout << y.first <<" "<<z<<endl;
                        FLW[y.first].insert(z);
                    }
                    // cout << y.first <<
                    later[y.first].erase(ch);
                }else{
                    sig = 1;
                }
            }
            if(sig == 0 && sig2 == 1) {
                q.push(y.first);
            }
        }
    }
    cout << "Follow of Non terminals\n";
    for(auto a: FLW){
        cout << a.first << " ";
        for(auto y: a.second){
            cout << y<<" ";
       }cout << endl;
    }
}
```

Input / Output :

```
PS D:\sem -6\compiler design\LAB\LAB -7  Bottom up parser && first and follow> .\a.exe
Enter number of productions
8
Enter productions, Use # for epsilon and use only single charcter for symbols
E TA
A +TA
A #
T FB
B *FB
B #
F (E)
F i

First of Non terminals
E ( i
A # +
T ( i
F ( i
B # *

Follow of Non terminals
B $ ) +
A $ )
E $ )
F $ ) * +
T $ ) +
```

```
10
Enter productions, Use # for epsilon and use only single charcter for symbols
S ABCDE
A a
A #
B b
B #
C c
D d
D #
E e
E #

First of Non terminals
A # a
B # b
C c
S a b c
E # e
D # d

Follow of Non terminals
D $ e
C $ d e
B c
E $
S $
A b c
```

Algorithm :

1) First we store all the production corresponding to each LHS in a map.
2) Then for each production we will find FIRST, starting with the last most production.
3) Results of each production are stored in a map.
4) To find the FIRST we iterate in production and the FIRST of the first symbol(A)(except epsilon) in production is added to the FIRST of LHS. In case there is Epsilon in the A then we add FIRST of the next symbol(B)(except epsilon) as well and this goes till the end. Incase B is the last symbol in production, and includes epsilon then FIRST of A include epsilon as well.
5) To find the FOLLOW we iterate in all the productions and for each symbol (let's say it A) in the production their FOLLOW set includes the FIRST set of the symbol followed by that(A) symbol (let's call the following symbol as B). In case the FIRST of B has epsilon in it then the FOLLOW of A also has FIRST set(except epsilon) of symbol next to B and this goes till the FIRST set of corresponding symbols include epsilon. Incase B is the last symbol in production then FOLLOW of A include FOLLOW of LHS.
6) The FOLLOW of the start symbol always contains '$', since input is always followed by '$'.

Learning :

1) FIRST and FOLLOW sets are very useful heuristics to decide the production to be applied.
2) If grammar is left recursive directly or indirectly, the program will fail.