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. 6

AIM :- To implement a Bottom-Up parser for a given Context Free Grammar.

Parsing: It builds the parse tree from leaves to root. Bottom-up parsing can be defined as an attempt to reduce the input string w to the start symbol of grammar by tracing out the rightmost derivations of w in reverse.

In the following code I have hardcoded Canonical LR(1) parsing table for the given CFG. Input is stored in an array.

Canonical-LR method –

- Use lookahead symbols for items: LR(1) items
- Results in a large collection of items

CFG Used:

```
// E->T | T + E
// T-> int | int * T|(E)
```

Source Code:

```
#include<bits/stdc++.h>
using namespace std;

// E->T | T + E
// T-> int | int * T|(E)
// Epsilon 0 || E 1 || T 2 || int 3 || * 4 || + 5 || ( 6 || ) 7 || $ 8
vector<string> val =
{"Ep","E","T","int","*","+","(",")","$","","s","r"};

void pre(unordered_map<int,unordered_map<int,pair<char,int>>>&
ActionTable,
unordered_map<int,unordered_map<int,int>>& GotoTable,
unordered_map<string,int>& states,
```

```
vector<vector<int>>& rules
){
    states["E"] = 1; states["T"] = 2;
    states["int"] = 3; states["*"] = 4;states["+"] = 5;
    states["("] = 6;states[")"] = 7;states["$"]= 8;
    states[""]=0; states["s"]= 9; states["r"] = 10;

    rules.push_back({0,1});        //E'->E
    rules.push_back({1,1});        //E->T
    rules.push_back({1,3});    //E->T + E
    rules.push_back({2,1});        //T-> int
    rules.push_back({2,3});    //T->int * T
    rules.push_back({2,3});    //T->(E)

    ActionTable[0][6]=make_pair('s',3); ActionTable[0][3] =
make_pair('s',4); GotoTable[0][2] = 2; GotoTable[0][1] = 1;
    ActionTable[1][8] = make_pair('A',100);
    ActionTable[2][5] = make_pair('s',5); ActionTable[2][8] =
make_pair('r',1);
    ActionTable[3][6] = make_pair('s',12);ActionTable[3][3] =
make_pair('s',9);GotoTable[3][2] = 8;GotoTable[3][1] = 6;
    ActionTable[4][5] = make_pair('r',3);ActionTable[4][4] =
make_pair('s',17); ActionTable[4][8] = make_pair('r',3);
    ActionTable[5][6] = make_pair('s',3);ActionTable[5][3] =
make_pair('s',4); GotoTable[5][2] = 2;GotoTable[5][1] = 6;
    ActionTable[6][7] = make_pair('s',7);ActionTable[6][8] =
make_pair('r',2);
    ActionTable[7][5] = make_pair('r',5);ActionTable[7][8] =
make_pair('r',5);
    ActionTable[8][5] = make_pair('s',13);ActionTable[8][7] =
make_pair('r',1);
    ActionTable[9][5] = make_pair('r',3);ActionTable[9][4] =
make_pair('s',10);ActionTable[9][7] = make_pair('r',3);
    ActionTable[10][6] = make_pair('s',12);GotoTable[10][2] = 11;
    ActionTable[11][4] = make_pair('r',4);ActionTable[11][7] =
make_pair('r',4);
    ActionTable[12][6] = make_pair('s',12);ActionTable[12][3] =
make_pair('s',9);GotoTable[12][2] = 8;GotoTable[12][1] = 15;
    ActionTable[13][6] = make_pair('s',12);ActionTable[13][3] =
make_pair('s',9);GotoTable[13][2] = 8;GotoTable[13][1] = 14;
    ActionTable[14][7] = make_pair('r',2);
```

```
    ActionTable[15][7] = make_pair('s',16);
    ActionTable[16][7] = make_pair('r',5);
    ActionTable[17][6] = make_pair('s',3);ActionTable[17][3] =
make_pair('s',4);GotoTable[17][2] = 18;
    ActionTable[18][5] = make_pair('r',4);ActionTable[18][8] =
make_pair('r',4);
}

int main(){
    unordered_map<int,unordered_map<int,pair<char,int>>> ActionTable;
    unordered_map<int,unordered_map<int,int>> GotoTable;
    unordered_map<string,int> states;
    vector<vector<int>> rules;

    pre(ActionTable,GotoTable,states,rules);

    vector <pair<int,int>> st;

    int i =0,error = 0;

    // Tokenizing the input (space separated input expected)
    string input,s="";
    vector<int> arr;
    getline(cin,input);

    for(int i = 0; i< input.size();i++){
        if(input[i] == ' '){
            arr.push_back(states[s]);
            s= "";
        }else{
            s.push_back(input[i]);
        }
    }
    arr.push_back(states[s]);
    arr.push_back(states["$"]);

    // parsing
    int sig = 0;
    st.push_back( make_pair(states["$"],0) );
```

```
    while(!st.empty() && i < arr.size()){
        pair<int,int> t = st.back();
        for(auto rex: st) cout << val[rex.first]<<" "<<rex.second<<" ";
            cout << endl;

        if(ActionTable[t.second][arr[i]].second == 0){
            break;
        }
        if(ActionTable[t.second][arr[i]].first == 'A'){
            cout << "Accepted" <<endl;
            sig = 1;
            break;
        }
        // cout << ActionTable[t.second][arr[i]].first<<"
"<<ActionTable[t.second][arr[i]].second<<endl;
        if(ActionTable[t.second][arr[i]].first == 's'){


st.push_back(make_pair(arr[i],ActionTable[t.second][arr[i]].second));
            i++;

        }else if(ActionTable[t.second][arr[i]].first == 'r'){

            int stz =  rules[ ActionTable[t.second][arr[i]].second
][0]; // State that will reduce to
            int len = rules[ ActionTable[t.second][arr[i]].second ][1];
// number of elements to be erased
            while(len--){
                st.pop_back();
            }
            // cout << "Here\n";
            // for(auto rex: st) cout << rex.first<<" "<<rex.second<<"
";
            // cout << endl;

            int xx = GotoTable[st.back().second][stz];
            st.push_back(make_pair(stz ,xx));
        }

    }
```

```
    if(sig == 0)
        cout <<"NO Match Found"<<endl;
}
```

Input / Output :

```
D:\sem -6\compiler design\LAB\LAB -7  Bottom up parser && first and follow>a.exe
int * int + ( ( int ) )
$ 0
$ 0 int 4
$ 0 int 4 * 17
$ 0 int 4 * 17 int 4
$ 0 int 4 * 17 T 18
$ 0 T 2
$ 0 T 2 + 5
$ 0 T 2 + 5 ( 3
$ 0 T 2 + 5 ( 3 ( 12
$ 0 T 2 + 5 ( 3 ( 12 int 9
$ 0 T 2 + 5 ( 3 ( 12 T 8
$ 0 T 2 + 5 ( 3 ( 12 E 15
$ 0 T 2 + 5 ( 3 ( 12 E 15 ) 16
$ 0 T 2 + 5 ( 3 T 8
$ 0 T 2 + 5 ( 3 E 6
$ 0 T 2 + 5 ( 3 E 6 ) 7
$ 0 T 2 + 5 T 2
$ 0 T 2 + 5 E 6
$ 0 E 1
Accepted
```

Algorithm :

1) We construct a canonical LR(1) parser table using canonical LR(0) item sets .(we have hard coded the table in this code)
2) Then input tokens are taken one by one and appropriate shift or reduce action or goto action is taken and carried out on the stack.
3) If in the end we get Accept state from action table then that string can be reduced to starting state else not
4) Also if for some input and corresponding item number on stack we don't have an action it is assumed that string cannot be reduced to starting symbol by the grammar

Learning :

1) Making a canonical LR(0) item sets with lookaheads can be tough
2) There can be a lot of states in the CLR parser table as compared to the LR parser table.
3) Not all CLR can be converted to LALR parsers.