18CSC304J

COMPILER DESIGN

SLR PARSING

MINOR PROJECT REPORT

Submitted by

S Vishnu Tejas (RA2011003010613)

Ch V Kalyan Gupta (RA2011003010622)

Kalimisetty Sashank (RA2011003010649)

Under the guidance of

Dr.G.ABIRAMI

for the course

18CSC304J-Compiler Design

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

S.R.M. Nagar, Kattankulathur, Kancheepuram

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report "**Simple LR Parser**" is the bonafide work of S Vishnu Tejas (RA2011003010613), Ch V Kalyan Gupta (RA2011003010622) and Kalimisetty Sashank (RA2011003010649) who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.G.Abirami	**Dr. M. PUSHPALATHA**
CD Faculty	**HEAD OF THE DEPARTMENT**
Department of Computing Technologies	Department of Computing Technologies

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy**, for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal**, for bringing out novelty in all executions.

We would like to express my heartfelt thanks to **Chairperson, School of Computing Dr. Revathi Venkataraman**, for imparting confidence to complete

my course project

We wish to express my sincere thanks to **Course Audit Professor** for their constant encouragement and support.

We are highly thankful to our Course project Internal guide **Dr.G.Abirami , Compiler Design Faculty , CSE**, for her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to the **Dr. M. PUSHPALATHA, Ph.D HEAD OF THE DEPARTMENT** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project

TABLE OF CONTENT

ABSTRACT :

In compiler design, SLR parsing is a widely used technique for parsing context-free grammars. The SLR parsing algorithm uses a bottom-up approach to parse the input string and construct a parse tree. This report provides an in-depth exploration of the SLR parsing technique, including the theory behind it, its implementation, and its advantages and disadvantages.

INTRODUCTION :

Compilers are essential tools in computer science that translate source code into executable programs. One of the key components of a compiler is the parser, which takes the source code and analyzes its structure to generate a parse tree. There are different types of parsers, including top-down and bottom-up parsers, each with their advantages and disadvantages.

SLR parsing is a bottom-up parsing technique that uses a deterministic finite automaton to parse the input string. The SLR parsing algorithm is efficient and can handle a wide range of context-free grammars. However, it has some limitations, such as its inability to handle left-recursive grammars.

This report provides a comprehensive overview of SLR parsing in compiler design. We will discuss the implementation of the SLR parsing algorithm

METHODOLOGY/TECHNIQUES :

The methodology for SLR parsing involves several techniques and algorithms that work together to parse the input string and construct a parse tree. Here are some of the key techniques used in SLR parsing:

1. Grammar Definition : The first step in SLR parsing is to define the context-free grammar that represents the programming language being parsed. The grammar should be written in a formal notation, such as BNF (Backus-Naur Form).

2. Item Sets : An item set is a set of production rules with a dot (.) placed at various positions within the rule. The item sets represent the state of the parsing process at a particular point.

3. LR(0) Automaton : The LR(0) automaton is a deterministic finite automaton that is constructed from the item sets. It represents the state transitions of the parser based on the input symbol.

4. LR(0) Table Construction : The LR(0) table is constructed from the LR(0) automaton. The table contains the actions to be taken by the parser based on the current state and input symbol. The actions can be either a shift or a reduce operation.

5. Conflict Resolution : Conflicts can arise in the parsing process when the parser encounters a state with multiple possible actions. The conflicts can be resolved using various techniques, such as precedence and associativity rules.

6. Parse Tree Construction : Once the parsing process is complete, the parse tree is constructed from the stack of symbols generated during the parsing process. The parse tree represents the syntactic structure of the input string.

SLR parsing is a powerful technique for parsing context-free grammars, and it is widely used in compiler design. However, it has some limitations, such as its inability to handle left-recursive grammars. To address this limitation, other parsing techniques, such as LR(1) and LALR(1), have been developed.

IMPLEMENTATION :

```
#include<iostream>
#include<string.h>
#include<stdlib.h>
#include<stdio.h>

using namespace std;

char terminals[100]={};int no_t;
char non_terminals[100]={};int no_nt;
char goto_table[100][100];
char reduce[20][20];
char follow[20][20];char fo_co[20][20];
char first[20][20];

struct state{
    int prod_count;
    char prod[100][100]={{}};
};

void add_dots(struct state *I){
    for(int i=0;i<I->prod_count;i++){
        for (int j=99;j>3;j--)
            I->prod[i][j] = I->prod[i][j-1];
        I->prod[i][3]='.';
    }
}

void get_prods(struct state *I){
    cout<<"Enter the number of productions:\n";
    cin>>I->prod_count;
```

```
    cout<<"Enter the number of non terminals:"<<endl;
    cin>>no_nt;
    cout<<"Enter the non terminals one by one:"<<endl;
    for(int i=0;i<no_nt;i++)
        cin>>non_terminals[i];
    cout<<"Enter the number of terminals:"<<endl;
    cin>>no_t;
    cout<<"Enter the terminals (single lettered) one by one:"<<endl;
    for(int i=0;i<no_t;i++)
        cin>>terminals[i];
    cout<<"Enter the productions one by one in form (S->ABc):\n";
    for(int i=0;i<I->prod_count;i++){
        cin>>I->prod[i];
    }
}

bool is_non_terminal(char a){
   if (a >= 'A' && a <= 'Z')
        return true;
    else
        return false;
}

bool in_state(struct state *I,char *a){
    for(int i=0;i<I->prod_count;i++){
        if(!strcmp(I->prod[i],a))
            return true;
    }
    return false;
}
```

```
void closure(struct state *I,struct state *I0){
    char a={};
    for(int i=0;i<I0->prod_count;i++){
        a=char_after_dot(I0->prod[i]);
        if(is_non_terminal(a)){
            for(int j=0;j<I->prod_count;j++){
                if(I->prod[j][0]==a){
                    if(!in_state(I0,I->prod[j])){
                        strcpy(I0->prod[I0->prod_count],I->prod[j]);
                        I0->prod_count++;
                    }
                }
            }
        }
    }
}

void goto_state(struct state *I,struct state *S,char a){
    int time=1;
    for(int i=0;i<I->prod_count;i++){
        if(char_after_dot(I->prod[i])==a){
                if(time==1){
                    time++;
                }
                strcpy(S->prod[S->prod_count],move_dot(I->prod[i],strlen(I->prod[i])));
                S->prod_count++;
        }
    }
}

void print_prods(struct state *I){
```

```
    for(int i=0;i<I->prod_count;i++)
        printf("%s\n",I->prod[i]);
    cout<<endl;
}

int return_index(char a){
    for(int i=0;i<no_t;i++)
        if(terminals[i]==a)
            return i;
    for(int i=0;i<no_nt;i++)
        if(non_terminals[i]==a)
            return no_t+i;
}

void print_shift_table(int state_count){
    cout<<endl<<"********Shift Actions*********"<<endl<<endl;
    cout<<"\t";
    for(int i=0;i<no_t;i++)
        cout<<terminals[i]<<"\t";
    for(int i=0;i<no_nt;i++)
        cout<<non_terminals[i]<<"\t";
    cout<<endl;
    for(int i=0;i<state_count;i++){
        int arr[no_nt+no_t]={-1};
        for(int j=0;j<state_count;j++){
            if(goto_table[i][j]!='~'){
                    arr[return_index(goto_table[i][j])]= j;
            }
        }
        cout<<"I"<<i<<"\t";
        for(int j=0;j<no_nt+no_t;j++){
```

```
            if(i==1&&j==no_t-1)
                cout<<"ACC"<<"\t";
            if(arr[j]==-1||arr[j]==0)
                cout<<"\t";
            else{
            if(j<no_t)
                cout<<"S"<<arr[j]<<"\t";
            else
                cout<<arr[j]<<"\t";

            }
        }
        cout<<"\n";
    }
}

int get_index(char c,char *a){
for(int i=0;i<strlen(a);i++)
    if(a[i]==c)
    return i;
}

void add_dot_at_end(struct state* I){
    for(int i=0;i<I->prod_count;i++){
        strcat(I->prod[i],".");
    }
}

void add_to_first(int n,char b){
   for(int i=0;i<strlen(first[n]);i++)
        if(first[n][i]==b)
```

```
            return;
    first[n][strlen(first[n])]=b;
}

void add_to_first(int m,int n){
    for(int i=0;i<strlen(first[n]);i++){
            int flag=0;
        for(int j=0;j<strlen(first[m]);j++){
            if(first[n][i]==first[m][j])
                flag=1;
        }
        if(flag==0)
            add_to_first(m,first[n][i]);
    }
}

void add_to_follow(int n,char b){
   for(int i=0;i<strlen(follow[n]);i++)
        if(follow[n][i]==b)
            return;
    follow[n][strlen(follow[n])]=b;
}

void add_to_follow(int m,int n){
    for(int i=0;i<strlen(follow[n]);i++){
            int flag=0;
        for(int j=0;j<strlen(follow[m]);j++){
            if(follow[n][i]==follow[m][j])
                flag=1;
        }
        if(flag==0)
```

```
            add_to_follow(m,follow[n][i]);
    }
}

void find_first(struct state *I){
    for(int i=0;i<no_nt;i++){
        for(int j=0;j<I->prod_count;j++){
            if(I->prod[j][0]==non_terminals[i]){
                if(!is_non_terminal(I->prod[j][3])){
                    add_to_first(i,I->prod[j][3]);
                    }


            }
        }
    }
}

int get_index(int *arr,int n){
    for(int i=0;i<no_t;i++){
        if(arr[i]==n)
            return i;
    }
    return -1;
}

void print_reduce_table(int state_count,int *no_re,struct state *temp1){
    cout<<"**********Reduce actions**********"<<endl<<endl;
    cout<<"\t";
    int arr[temp1->prod_count][no_t]={-1};
    for(int i=0;i<no_t;i++){
        cout<<terminals[i]<<"\t";
```

```
    }
    cout<<endl;
    for(int i=0;i<temp1->prod_count;i++){
    int n=no_re[i];
    for(int j=0;j<strlen(follow[return_index(temp1->prod[i][0])-no_t]);j++){
        for(int k=0;k<no_t;k++){
            if(follow[return_index(temp1->prod[i][0])-no_t][j]==terminals[k])
                arr[i][k]=i+1;
        }
    }
    cout<<"I"<<n<<"\t";
    for(int j=0;j<no_t;j++){
        if(arr[i][j]!=-1&&arr[i][j]!=0&&arr[i][j]<state_count)
            cout<<"R"<<arr[i][j]<<"\t";
        else
            cout<<"\t";
    }
    cout<<endl;
}
}

int main(){
    struct state init;
    struct state temp;struct state temp1;
    int state_count=1;
    get_prods(&init);
    temp=init;
    temp1=temp;
    add_dots(&init);

    for(int i=0;i<100;i++)
```

```
    for(int j=0;j<100;j++)
        goto_table[i][j]='~';

struct state I[50];
augument(&I[0],&init);
closure(&init,&I[0]);
cout<<"\nI0:\n";
print_prods(&I[0]);

char characters[20]={};
for(int i=0;i<state_count;i++){
    char characters[20]={};
    for(int z=0;z<I[i].prod_count;z++)
            if(!in_array(characters,char_after_dot(I[i].prod[z])))
            characters[strlen(characters)]=char_after_dot(I[i].prod[z]);



    for(int j=0;j<strlen(characters);j++){
        goto_state(&I[i],&I[state_count],characters[j]);
        closure(&init,&I[state_count]);
        int flag=0;
        for(int k=0;k<state_count-1;k++){
            if(same_state(&I[k],&I[state_count])){
                cleanup_prods(&I[state_count]);flag=1;
                cout<<"I"<<i<<" on reading the symbol "<<characters[j]<<" goes to I"<<k<<".\n";
                goto_table[i][k]=characters[j];;
                break;
            }
        }
        if(flag==0){
            state_count++;
```

```
                    cout<<"I"<<i<<" on reading the symbol "<<characters[j]<<" goes to I"<<state_count-1<<":\n";
                goto_table[i][state_count-1]=characters[j];
                print_prods(&I[state_count-1]);
            }
        }
    }

    int no_re[temp.prod_count]={-1};
    terminals[no_t]='$';no_t++;
    find_first(&temp);
    for(int l=0;l<no_nt;l++){
    for(int i=0;i<temp.prod_count;i++){
        if(is_non_terminal(temp.prod[i][3])){
            add_to_first(return_index(temp.prod[i][0])-no_t,return_index(temp.prod[i][3])-no_t);
        }
    }}

    find_follow(&temp);
    add_to_follow(0,'$');
    for(int l=0;l<no_nt;l++){
        for(int i=0;i<temp.prod_count;i++){
            for(int k=3;k<strlen(temp.prod[i]);k++){
                if(temp.prod[i][k]==non_terminals[l]){
                        if(is_non_terminal(temp.prod[i][k+1])){
                            add_to_follow_first(l,return_index(temp.prod[i][k+1])-no_t);}
                        if(temp.prod[i][k+1]=='\0')
                            add_to_follow(l,return_index(temp.prod[i][0])-no_t);
                            }
                    }
            }
```

```
    }
    print_shift_table(state_count);
    cout<<endl<<endl;
    print_reduce_table(state_count,&no_re[0],&temp1);
}
```

OUTPUT :

```
D:\Downloads\Untitled1.exe
********Shift Actions*********

        +       *       (       )       @       $       E       T       F
I0                      S4              S5              1       2       3
I1      S6                                      ACC
I2              S7
I3
I4                      S4              S5              8       2       3
I5
I6                      S4              S5                      9       3
I7                      S4              S5                              10
I8      S6                      S11
I9              S7
I10
I11


**********Reduce actions**********

        +       *       (       )       @       $
I9      R1                      R1              R1
I2      R2                      R2              R2
I10     R3      R3              R3              R3
I3      R4      R4              R4              R4
I11     R5      R5              R5              R5
I5      R6      R6              R6              R6

--------------------------------
Process exited after 128.2 seconds with return value 0
Press any key to continue . . .
```

RESULT :

Therefore, we have successfully implemented SLR parsing.

CONCLUSION :

In conclusion, SLR parsing is a bottom-up parsing technique that is widely used in compiler design. It is an efficient and effective technique for parsing a wide range of context-free grammars. However, it has some limitations, such as its inability to handle left-recursive grammars.

Despite its limitations, SLR parsing remains a valuable technique for compiler design. It is easy to implement and provides a good balance between efficiency and expressiveness. Furthermore, it serves as a foundation for more advanced parsing techniques, such as LR(1) and LALR(1) parsing.

REFERENCES :

- https://www.geeksforgeeks.org/slr-parser-with-examples/
- https://www.javatpoint.com/slr-1-parsing
- https://www.tutorialspoint.com/construct-the-slr-parsing-table-for-the-following-grammar-also-parse-the-input-string-a-b-plus-a