Name: Saurabh Khandagale

Roll No.46

# PRACTICAL No. 4

## Predictive Parser

Predictive parser is a recursive descent parser, which has the capability to predict which production is to be used to replace the input string. The predictive parser does not suffer from backtracking.To accomplish its tasks, the predictive parser uses a look-ahead pointer, which points to the next input symbols. To make the parser back-tracking free, the predictive parser puts some constraints on the grammar and accepts only a class of grammar known as LL(k) grammar.

## Recursive Descent Parsing

Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right. It uses procedures for every terminal and non-terminal entity. This parsing technique recursively parses the input to make a parse tree, which may or may not require back-tracking. But the grammar associated with it (if not left factored) cannot avoid back-tracking. A form of recursive-descent parsing that does not require any back-tracking is known as **predictive parsing**

## Bottom-up Parsing

Bottom-up parsing starts from the leaf nodes of a tree and works in upward direction till it reaches the root node. Here, we start from a sentence and then apply production rules in reverse manner in order to reach the start symbol. The image given below depicts the bottom-up parsers available.

## Shift-Reduce Parsing

Shift-reduce parsing uses two unique steps for bottom-up parsing. These steps are known as shift-step and reduce-step.

Shift step: The shift step refers to the advancement of the input pointer to the next input symbol, which is called the shifted symbol. This symbol is pushed onto the stack. The shifted symbol is treated as a single node of the parse tree.

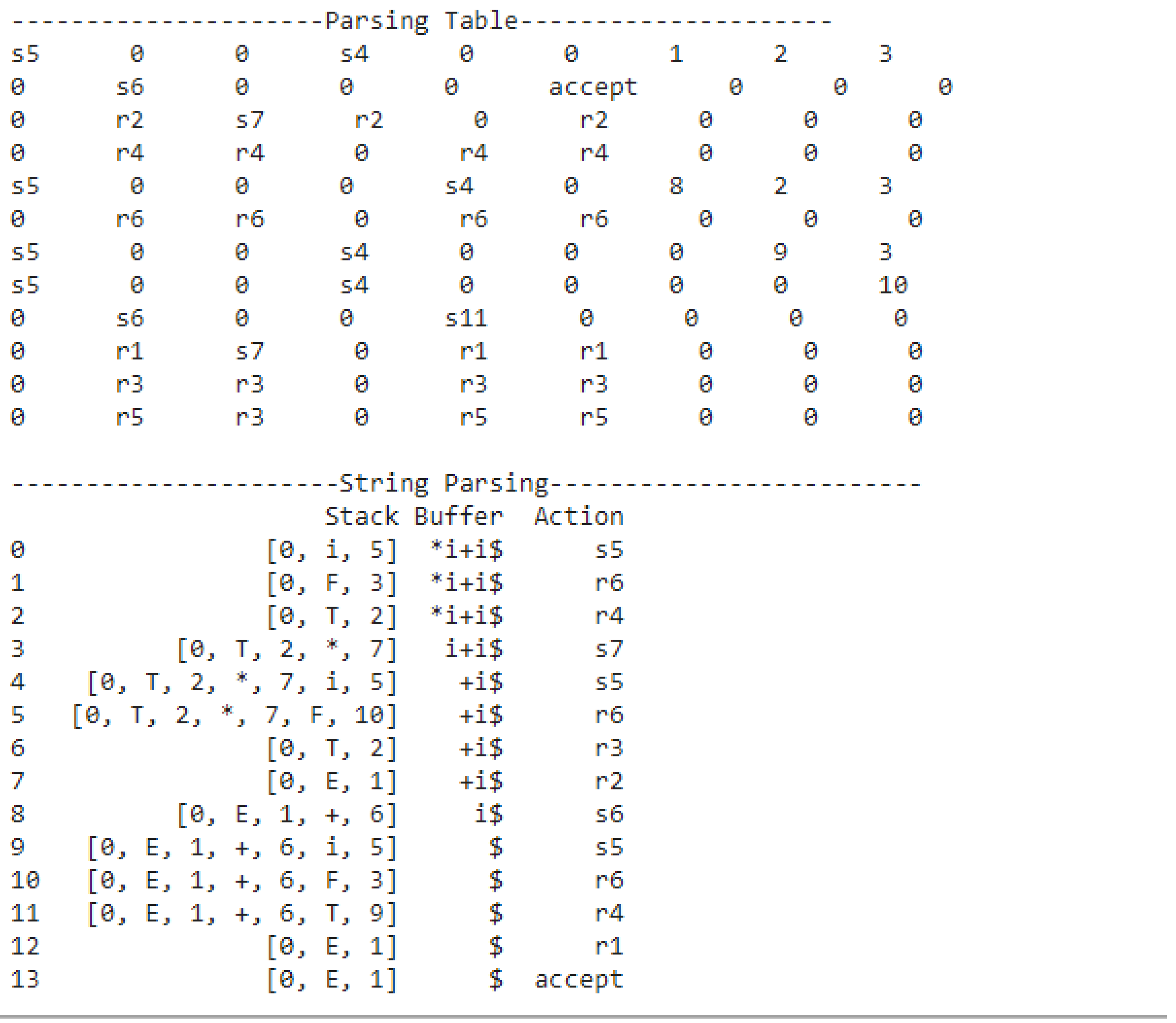
Reduce step : When the parser finds a complete grammar rule (RHS) and replaces it to (LHS), it is known as reduce-step. This occurs when the top of the stack contains a handle. To reduce, a POP function is performed on the stack which pops off the handle and replaces it with LHS non-terminal symbol.

Aim: Write a program to perform string validation for LR grammar.

Program:-

```
import pandas as pd

d={0:{'i':'s5','+':0,'*':0,'(':'s4',')':0,'$':0,'E':1,'T':2,'F':3},

1:{'i':0,'+':'s6','*':0,'(':0,')':0,'$':'accept','E':0,'T':0,'F':0},

2:{'i':0,'+':'r2','*':'s7','(':'r2',')':0,'$':'r2','E':0,'T':0,'F':0},

3:{'i':0,'+':'r4','*':'r4','(':0,')':'r4','$':'r4','E':0,'T':0,'F':0},

4:{'i':'s5','+':0,'*':0,'(':0,')':'s4','$':0,'E':8,'T':2,'F':3},

5:{'i':0,'+':'r6','*':'r6','(':0,')':'r6','$':'r6','E':0,'T':0,'F':0},

6:{'i':'s5','+':0,'*':0,'(':'s4',')':0,'$':0,'E':0,'T':9,'F':3},

7:{'i':'s5','+':0,'*':0,'(':'s4',')':0,'$':0,'E':0,'T':0,'F':10},

8:{'i':0,'+':'s6','*':0,'(':0,')':'s11','$':0,'E':0,'T':0,'F':0},

9:{'i':0,'+':'r1','*':'s7','(':0,')':'r1','$':'r1','E':0,'T':0,'F':0},

10:{'i':0,'+':'r3','*':'r3','(':0,')':'r3','$':'r3','E':0,'T':0,'F':0},

11:{'i':0,'+':'r5','*':'r3','(':0,')':'r5','$':'r5','E':0,'T':0,'F':0}}

g=[["E","E+T"],["E","T"],["T","T*F"],["T","F"],["F","(E)"],["F","i"]]

def push(stack,e):

   stack.append(e)

def remove(stack):

   stack.pop()

print("---------------------Parsing Table---------------------")

for i in d:

   for j in d[i]:
        print(d[i][j],end='      ')
    print()


ip='i*i+i$'


stack=[0]
z1=[]
z2=[]
z3=[]


j=0
while True:
    x=d[stack[-1]][ip[j]]
    # print(x,stack[-1],ip[j])
    if x=='accept':
        break
    if x[0]=='s':
        stack.append(ip[j])
        stack.append(int(x[1:]))
        ip=ip[1:]
    if x[0]=='r':
        indx=int(x[1:])
        l=len(g[indx-1][1])

        for i in range(2*l):
            remove(stack)
            push(stack,g[indx-1][0])

            push(stack,d[stack[-2]][stack[-1]])



        z1.append(stack.copy())

        z2.append(ip)

        z3.append(x)

        # print(stack,"\t\t\t\t",ip,"\t\t",x)

    # print(stack,"\t\t\t\t",ip,"\t\t\t",x)

     z1.append(stack.copy())

    z2.append(ip)

    z3.append(x)

    data={"Stack":z1,"Buffer":z2,"Action":z3}

    df = pd.DataFrame(data)

    print("\n----------------------String Parsing-------------------------")

    print(df)
```

Output:-



```
---------------------Parsing Table--------------------
s5      0       0       s4      0       0       1       2       3
0       s6      0       0       0       accept      0       0       0
0       r2      s7      r2      0       r2      0       0       0
0       r4      r4      0       r4      r4      0       0       0
s5      0       0       0       s4      0       8       2       3
0       r6      r6      0       r6      r6      0       0       0
s5      0       0       s4      0       0       0       9       3
s5      0       0       s4      0       0       0       0       10
0       s6      0       0       s11     0       0       0       0
0       r1      s7      0       r1      r1      0       0       0
0       r3      r3      0       r3      r3      0       0       0
0       r5      r3      0       r5      r5      0       0       0

----------------------String Parsing------------------------
                     Stack Buffer  Action
0                [0, i, 5]  *i+i$      s5
1                [0, F, 3]  *i+i$      r6
2                [0, T, 2]  *i+i$      r4
3          [0, T, 2, *, 7]   i+i$      s7
4    [0, T, 2, *, 7, i, 5]    +i$      s5
5   [0, T, 2, *, 7, F, 10]    +i$      r6
6                [0, T, 2]    +i$      r3
7                [0, E, 1]    +i$      r2
8          [0, E, 1, +, 6]     i$      s6
9    [0, E, 1, +, 6, i, 5]      $      s5
10   [0, E, 1, +, 6, F, 3]      $      r6
11   [0, E, 1, +, 6, T, 9]      $      r4
12               [0, E, 1]      $      r1
13               [0, E, 1]      $  accept
```