# EXPERIMENT - XI
# SHIFT-REDUCE PARSER

October 15, 2020

ADITHYA D RAJAGOPAL
ROLL NO : 9
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

## AIM

To construct a shift-reduce parser for a given language.

# THEORY

## Shift-Reduce Parser

Shift-reduce parsing attempts to construct a parse tree for an input string beginning at the leaves and working up towards the root. In other words, it is a process of reducing (opposite of deriving a symbol using a production rule) a string w to the start symbol of a grammar. At every (reduction) step, a particular substring matching the RHS of a production rule is replaced by the symbol on the LHS of the production.
A general form of shift-reduce parsing is LR (scanning from Left to right and using Right most derivation in reverse) parsing, which is used in a number of automatic parser generators like Yacc, Bison, etc. A handle of a string is a substring that matches the RHS of a production and whose reduction to the non-terminal (on the LHS of the production) represents one step along the reverse of a rightmost derivation toward reducing to the start symbol. The set of prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes. It is always possible to add terminal symbols to the end of a viable prefix to obtain a right-sentential form.

# ALGORITHM

**Algorithm 1** Algorithm to construct a Shift-reduce Parser

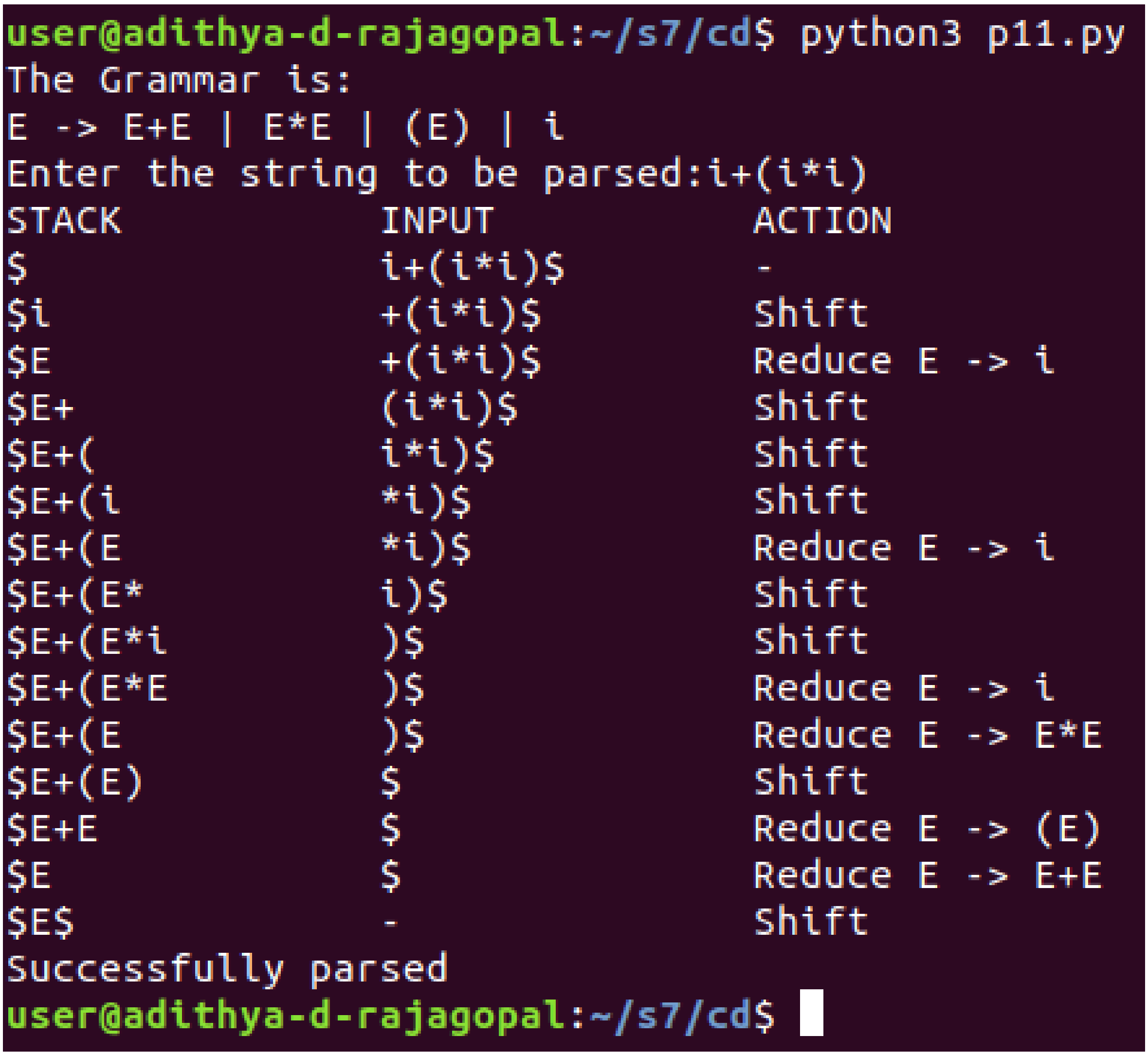
```
Loop forever:
    for top of stack symbol s and next input symbol a, case action of T[s,a]
        Shift x: (X is a STATE number)
            push a, then x on the top of the stack and
            advance ip to point to the next input symbol
        Reduce y: (y is a production number)
            Assume that the production is of the form A → $\beta$
            Pop 2 * |$\beta$| symbols of the stack.
            At this point the top of stack symbol should be a state number say s'.
            Push A, then goto of T[s',A] on the top of the stack.
            Output the production A → $\beta$
            accept:
                return - - - Successful parse
            default:
                error - - - Input string is not in language
```

## SOURCE CODE

```
def printStack():
        global Stack
        for i in Stack:
                print(i,end="")
        print("\t\t",end="")

def reduce():
        global Stack
        global handle
        global prevhandle
        if Stack[-1]=="i":
                Stack.pop()
                Stack.append("E")
                prevhandle=handle[0]
                return True
        if len(Stack)>=3:
                if Stack[-1]=="E" and Stack[-3]=="E":
                        op=Stack.pop()
                        op=Stack.pop()
                        if op=="+":
                                prevhandle=handle[1]
                        elif op=="*":
                                prevhandle=handle[2]
                        return True
                elif Stack[-1]==")" and Stack[-2]=="E" and Stack[-3]=="(":
                        op=Stack.pop()
                        op=Stack.pop(-2)
                        prevhandle=handle[3]
                        return True
        return False

def Shift_Reduce_Parser(str):
        global Stack
```

```
    global handle
    global prevhandle
    T=[’+’,’*’,’i’,’(’,’)’,’$’]
    Stack=[’$’]
    ip=0
    handle=[’i’,’E+E’,’E*E’,’(E)’]
    print("STACK\t\tINPUT\t\tACTION")
    print("$\t\t"+str+"\t-")
    while ip<len(str):
        Stack.append(str[ip])
        ip=ip+1
        printStack()
        if ip==len(str):
            print("-\t\tShift")
            break
        print(str[ip:],"\t\tShift")
        while(reduce()):
            printStack()
            print(str[ip:],end="\t\t")
            print("Reduce E -> "+prevhandle)
    if Stack[0]==’$’ and Stack[1]==’E’ and Stack[2]==’$’:
        return True
    return False

global Stack
global handle
global prevhandle
print("The Grammar is:")
print("E -> E+E | E*E | (E) | i")
s=input("Enter the string to be parsed:")
w=s+’$’
if(Shift_Reduce_Parser(w)):
    print("Successfully parsed")
else:
    print("Error in parsing")
```

## Sample Output



```
user@adithya-d-rajagopal:~/s7/cd$ python3 p11.py
The Grammar is:
E -> E+E | E*E | (E) | i
Enter the string to be parsed:i+(i*i)
STACK               INPUT               ACTION
$                   i+(i*i)$            -
$i                  +(i*i)$             Shift
$E                  +(i*i)$             Reduce E -> i
$E+                 (i*i)$              Shift
$E+(                i*i)$               Shift
$E+(i               *i)$                Shift
$E+(E               *i)$                Reduce E -> i
$E+(E*              i)$                 Shift
$E+(E*i             )$                  Shift
$E+(E*E             )$                  Reduce E -> i
$E+(E               )$                  Reduce E -> E*E
$E+(E)              $                   Shift
$E+E                $                   Reduce E -> (E)
$E                  $                   Reduce E -> E+E
$E$                 -                   Shift
Successfully parsed
user@adithya-d-rajagopal:~/s7/cd$
```

## RESULT

A program to construct a shift-reduce parser has been implemented using Python and the outputs have been verified.