EXP:3: REMOVING LEFT RECURSION AND LEFT FACTORING

AIM: To Design a code to remove the left recursions and left factors in the grammar.

LANGUAGE USED: Python 3

ALGORITHM/PROCEDURE: -

1. Write the given code in Python complier
2. We used the input given grammar and run by the given functions.
3. We use the transition functions and get the input from the class.
4. We print the grammar after removing left recursions and left factors.
5. After the step we print the final grammar after removing the left factors and left recursions.
6. We get the output after removing left recursion and left factors.

SOURCE CODE: -

Removing Left Recursion:

```
gram = {

        "S":["S0S1S","01"]

}



def removeDirectLR(gramA, A):

        """gramA is dictonary"""

        temp = gramA[A]

        tempCr = []

        tempInCr = []

        for i in temp:

                if i[0] == A:


#tempInCr.append(i[1:])
```

```
            tempInCr.append(i[1:]+[A+"'"])
        else:
            #tempCr.append(i)
            tempCr.append(i+[A+"'"])
    tempInCr.append(["e"])
    gramA[A] = tempCr
    gramA[A+"'"] = tempInCr
    return gramA



def checkForIndirect(gramA, a, ai):
    if ai not in gramA:
        return False
    if a == ai:
        return True
    for i in gramA[ai]:
        if i[0] == ai:
            return False
        if i[0] in gramA:
            return checkForIndirect(gramA, a, i[0])
    return False

def rep(gramA, A):
    temp = gramA[A]

newTemp = []
```

```
for i in temp:
        if checkForIndirect(gramA, A, i[0]):
            t = []
            for k in gramA[i[0]]:
                t=[]
                t+=k
                t+=i[1:]
                newTemp.append(t)

        else:
            newTemp.append(i)
    gramA[A] = newTemp
    return gramA

def rem(gram):
    c = 1
    conv = {}
    gramA = {}
    revconv = {}
    for j in gram:
        conv[j] = "A"+str(c)
        gramA["A"+str(c)] = []
        c+=1

    for i in gram:
        for j in gram[i]:
            temp = []
            for k in j:
                if k in conv:
                    temp.append(conv[k])
                else:
                    temp.append(k)
            gramA[conv[i]].append(temp)


    #print(gramA)
    for i in range(c-1,0,-1):
        ai = "A"+str(i)
        for j in range(0,i):
            aj = gramA[ai][0][0]
            if ai!=aj :
                if aj in gramA and checkForIndirect(gramA,ai,aj):
                    gramA = rep(gramA, ai)

    for i in range(1,c):
        ai = "A"+str(i)
        for j in gramA[ai]:
            if ai==j[0]:
                gramA = removeDirectLR(gramA, ai)
                break

    op = {}

for i in gramA:
```

```
            a = str(i)
            for j in conv:
                a = a.replace(conv[j],j)
            revconv[i] = a

        for i in gramA:
            l = []
            for j in gramA[i]:
                k = []
                for m in j:
                    if m in revconv:
                        k.append(m.replace(m,revconv[m]))
                    else:
                        k.append(m)
                l.append(k)
            op[revconv[i]] = l

        return op

result = rem(gram)


for i in result:
    print(f'{i}->{result[i]}')
```

Removing Left Factoring:

```
from itertools import takewhile
def groupby(ls):
    d = {}

    ls = [ y[0] for y in rules ]

    initial = list(set(ls))

    for y in initial:

        for i in rules:

            if i.startswith(y):

                if y not in d:

                    d[y] = []

                d[y].append(i)

    return d



def prefix(x):

    return len(set(x)) == 1




starting=""

rules=[]

common=[]

alphabetset=["A'","B'","C'","D'","E'","F'","G'","H'","I'","J'","K'","L'","M'","N'","O'","P'","Q'","R'","S'","T'","U'"
,"V'","W'","X'","Y'","Z'"]

s= "S->iE"

while(True):

    rules=[]



common=[]

    split=s.split("->")

    starting=split[0]

    for i in split[1].split("|"):
        rules.append(i)
#logic for taking commons out
    for k, l in groupby(rules).items():
        r = [l[0] for l in takewhile(prefix, zip(*l))]
        common.append(".join(r))
#end of taking commons
    for i in common:
        newalphabet=alphabetset.pop()
        print(starting+"->"+i+newalphabet)
        index=[]
        for k in rules:
            if(k.startswith(i)):
                index.append(k)
        print(newalphabet+"->",end="")
        for j in index[:-1]:
            stringtoprint=j.replace(i,"", 1)+"|"
            if stringtoprint=="|":
                print("\u03B5","|",end="")
            else:
                print(j.replace(i,"", 1)+"|",end="")
        stringtoprint=index[-1].replace(i,"", 1)+"|"
        if stringtoprint=="|":

 print("\u03B5","",end="")
        else:
            print(index[-1].replace(i,"", 1)+"",end="")
        print("")
    break
```

INPUT: -

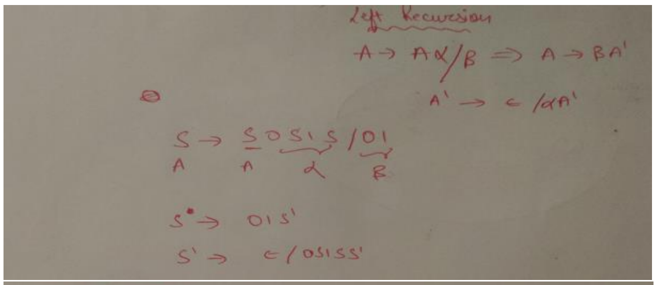
<u>For left recursion:</u>

"S":["S0S1S","01"]

<u>For left factoring</u>

"S->aSSbS/aSaSb/abb/b"

Space Tree Diagram:



Left Recursion

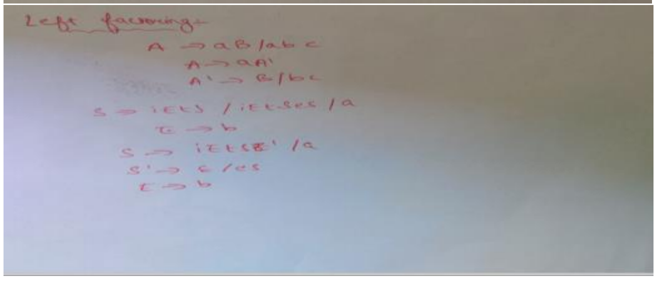
$A \to A\alpha / \beta \Rightarrow A \to \beta A'$

$A' \to \epsilon / \alpha A'$

$\underset{A}{S} \to \underset{A}{\underline{S}} \underbrace{0S1S}_{\alpha} / \underbrace{01}_{\beta}$

$S' \to 01S'$

$S' \to \epsilon / 0S1SS'$



Left factoring:

$A \to aB / abc$

$A \to aA'$

$A' \to B / bc$

$S \to iEtS / iEtSeS / a$

$E \to b$

$S \to iEtSS' / a$

$S' \to \epsilon / eS$

$E \to b$

OUTPUT: -

<u>For removing left recursion:</u>

```
S->[['0', '1', "S'"]]
S'->[['0', 'S', '1', 'S', "S'"], ['e']]
```

<u>For removing left factors:</u>

```
S->aSSbS/aSaSb/abb/bZ'
Z'->ε
```

RESULT: Therefore, we successfully implemented a code for removing left recursions and left factoring in the given grammar.