**Name- DEVANGI PURKAYASTHA**

**REG NO- RA1811003010689**

**CSE D2**

**DATE- 26/02/2021**

AIM:- Elimination of Left Recursion and Left Factoring

CODE-

1. Elimination of Left Recursion:

gram = {

    "E":["E+T","T"],

    "T":["T\*F","F"],

    "F":["(E)","i"]

}

def removeDirectLR(gramA, A):

    temp = gramA[A]

    tempCr = []

    tempInCr = []

    for i in temp:

        if i[0] == A:

            tempInCr.append(i[1:]+[A+"'"])

        else:

            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)

    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

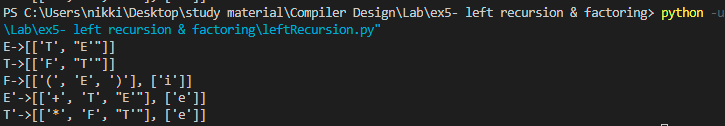
    return op

result = rem(gram)

for i in result:

    print(f'{i}->{result[i]}')

OUTPUT-



1. 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->iEtS|iEtSeS|a"

while(True):

    rules=[]

    common=[]

    split=s.split("->")

    starting=split[0]

    for i in split[1].split("|"):

        rules.append(i)

    for k, l in groupby(rules).items():

        r = [l[0] for l in takewhile(prefix, zip(\*l))]

        common.append(''.join(r))

    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="")

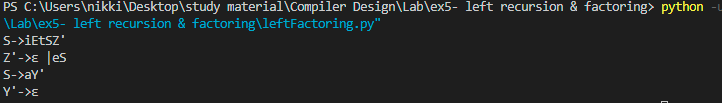
        else:

            print(index[-1].replace(i,"", 1)+"",end="")

        print("")

    break

OUTPUT-



ALGORITHM:-

Left Recursion:  
Grammar of the form,

S --> S / a / b It is called left recursive where S is any non Terminal and a, and b are any set of terminals.

Problem with Left Recursion:  
If a left recursion is present in any grammar then, during parsing in the the syntax analysis part of compilation there is a chance that the grammar will create infinite loop. This is because at every time of production of grammar S will produce another S without checking any condition.

Algorithm to Remove Left Recursion with an example:  
Suppose we have a grammar which contains left recursion:

S-->S a / S b / c / d

1. Check if the given grammar contains left recursion, if present then separate the production and start working on it.  
   In our example,

S-->S a/ S b /c / d

1. Introduce a new nonterminal and write it at the last of every terminal. We produce a new nonterminal S’and write new production as,

S-->cS' / dS'

1. Write newly produced nonterminal in LHS and in RHS it can either produce or it can produce new production in which the terminals or non terminals which followed the previous LHS will be replaced by new nonterminal at last.

S'-->? / aS' / bS'

So after conversion the new equivalent production is

S-->cS' / dS'

S'-->? / aS' / bS'

RESULT-

The given program has been successfully executed.