EXP 4 : ELIMINATION OF LEFT RECURSION AND LEFT FACTORING

AIM:

* To eliminate left recursion and left factoring using python program.

ALGORITHM:

For Left factoring:

1. Create a function to check the longest common subsequence

2. Create a function left factoring with the grammar as input

3. Where grammar is a dictionary

4. Loop through all the keys in the dictionary

5. Then check the longest common subsequence for all the keys

6. Then store the largest one

7. That will be the left factor for the given input grammar

8. Then define a main function for all the input to be taken

For left recursion:

1. Create a function called left recursion with grammar as input

2. Where grammar is a dictionary

3. Loop through the keys of the grammar

4. If the first prod of grammar is the key then recursion is found

5. We then raise a flag

6. Signalling we found a left recursion

7. We also increase the position by 1

8. Then since the flag is raised we enter another loop

9. For updating the grammar according to the left recursion

10. Then again we can define a main function to take all the input

PROGRAM:

LEFT FACTORING:

#Elimination of left factoring  
*import* itertools  
*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)  
  
#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*

LEFT RECURSION:

#Elimination of Left Recursion  
gram = {  
 "E":["E+T","T"],  
 "T":["T\*F","F"],  
 "F":["(E)","i"]  
}  
  
*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]}')

OUTPUT:

C:\Users\hp\AppData\Local\Programs\Python\Python39\python.exe "F:/Python/DAA/Compiler Design/leftfactoring.py"

S->aZ'

Z'->ε

S->iEtSY'

Y'->ε |eS

Process finished with exit code 0

C:\Users\hp\AppData\Local\Programs\Python\Python39\python.exe "F:/Python/DAA/Compiler Design/leftrecursion.py"

E->[['T', "E'"]]

T->[['F', "T'"]]

F->[['(', 'E', ')'], ['i']]

E'->[['+', 'T', "E'"], ['e']]

T'->[['\*', 'F', "T'"], ['e']]

Process finished with exit code 0

RESULT:

* The elimination of left recursion and left factoring using python program is successfully completed.