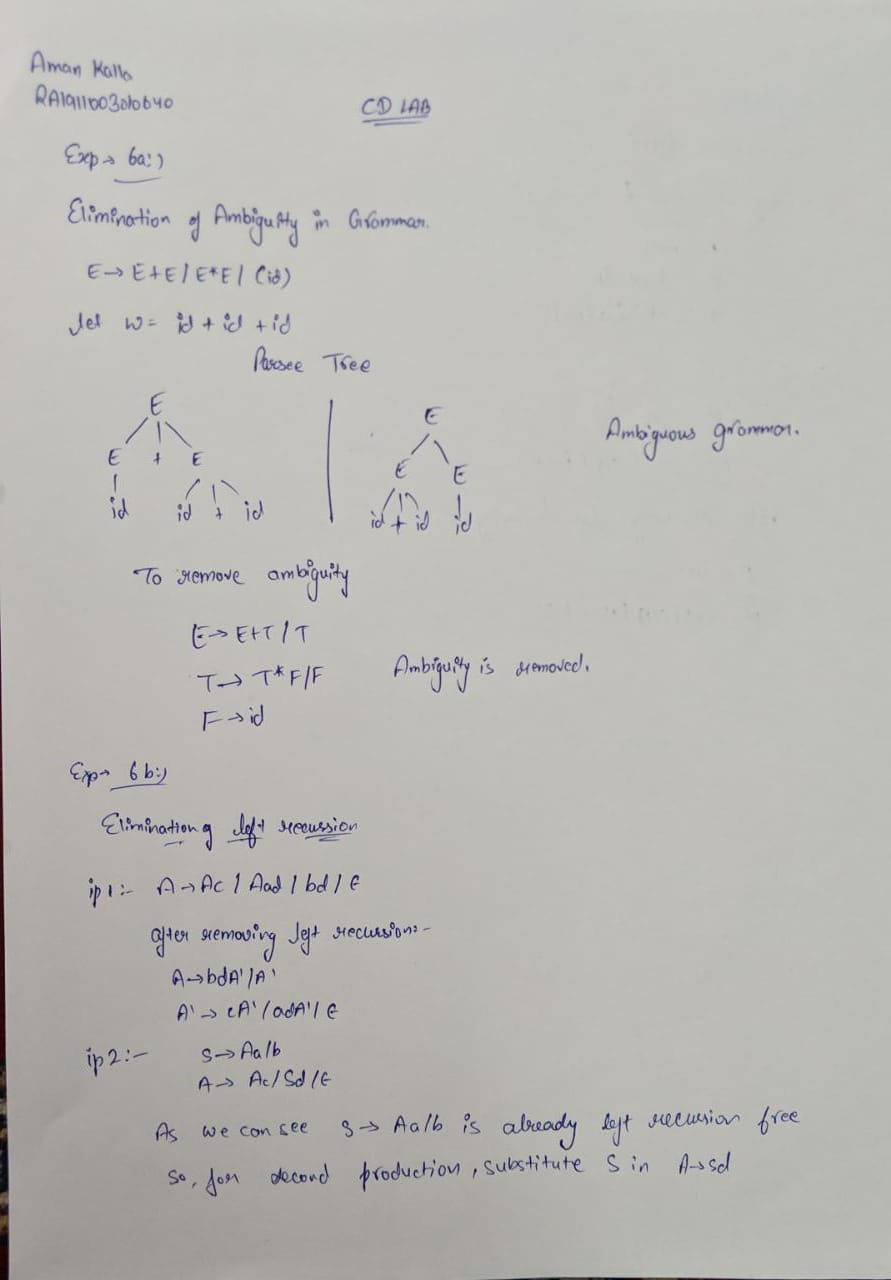
**CD EXP 6**

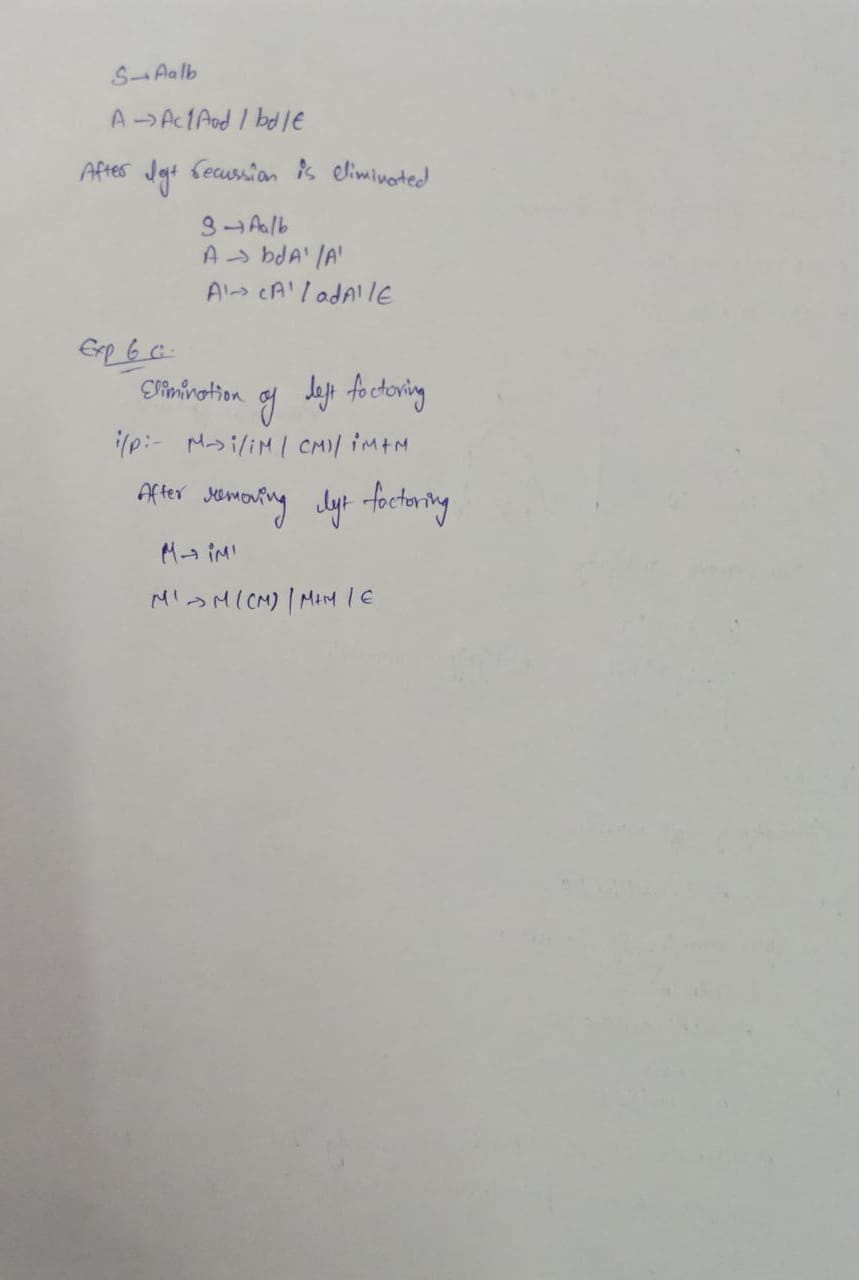
**LEFT RECURSION, LEFT FACTORING & AMBIGUITY**

**Aman Kalla**

**RA1911003010640**

**ELIMINATION OF AMBIGUITY**





**LEFT RECURSION:**

import sys

import re

import time

sys.setrecursionlimit(60)

def isTerm(ter):

if ter == '#':

return True

for t in terms:

if ter == t:

return True

return False

def isNTerm(nT):

for nt in nonterms:

if nT == nt:

return True

return False

def LeftCheck(nT, search, escape):

# print("NT:-"+nT+",Search:-"+search)

# time.sleep(3)

for prod in production\_dict[nT]:

if search == prod[0]:

leftcheck\_list[search].append(nT)

return True

else:

if isNTerm(prod[0]):

return LeftCheck(prod[0], search, escape+1)

else:

escape = 0

continue

return False

def RightCheck(nT, search):

# print("NT:-"+nT+",Search:-"+search)

for prod in production\_dict[nT]:

if search == prod[len(prod)-1]: # Direct

return True

return False

def Left\_toRight(nT):

bet = []

alpha = []

#expand on indirect relations

for nont in leftcheck\_list[nt]:

if nont != nT:

for i in range(0, len(production\_dict[nT])):

prod = production\_dict[nT][i]

if(nont == prod[0]):

prod\_temp = prod

remain = prod[1:]

for p in production\_dict[nont]:

if nT == p[0]:

final\_prod = p + remain

production\_dict[nT].pop(i)

production\_dict[nT].insert(i, final\_prod)

#remove Direct Left recursion

for prd in production\_dict[nT]:

if nT == prd[0]:

alpha.append(prd[1:])

else:

bet.append(prd)

gram\_dash = ""

for a in alpha:

gram\_dash = gram\_dash + a + nT + '\'' + "/"

gram\_dash = gram\_dash[0:-1]

gram = ""

for b in bet:

gram = gram+b+nT+'\''+"/"

gram = gram[0:-1]

print(nT+"->"+gram)

print(nT+"\'->"+gram\_dash+"/#")

def Prod\_print(nT):

full\_prod = ""

for pd in production\_dict[nT]:

full\_prod = full\_prod+pd+"/"

full\_prod = full\_prod[0:-1]

print(nT+"->"+full\_prod)

productions = []

n = input("Enter the number of Productions:-")

n = int(n)

print("\nRules:\nEpsilon is represented by # \nDo not use # or $ \n\n")

for i in range(n):

prod = input()

prod.strip()

productions.append(prod)

nonterms = []

terms = []

#Finding Non Terminals:-

for i in range(n):

nonterms.append(productions[i][0])

#Finding Terminals:-

for i in productions:

for j in range(3, len(i)):

check = True

for nt in nonterms:

if i[j] == nt or i[j] == '#' or i[j] == '/':

check = False

if check:

terms.append(i[j])

print("Non Terminals:-", nonterms)

print("Terminals:-", terms)

print("Productions:-", productions)

#Production Dict

production\_dict = {}

for nt in nonterms:

production\_dict[nt] = []

# split the productions into parts to simplify parsing

for production in productions:

nonterminal\_to\_production = production.split("->")

expanded = nonterminal\_to\_production[1].split(

"/") # assumption : single char terminals

for ex in expanded:

production\_dict[nonterminal\_to\_production[0]].append(ex)

print("production\_dict", production\_dict)

leftcheck\_dict = {}

leftcheck\_list = {} # Stores the non terminals which have indirect left recursion

for nt in nonterms:

leftcheck\_list[nt] = []

leftcheck\_dict[nt] = LeftCheck(nt, nt, 0)

rightcheck\_dict = {}

for nt in nonterms:

rightcheck\_dict[nt] = RightCheck(nt, nt)

print("\n")

for nt in nonterms:

if leftcheck\_dict[nt]:

print(nt+" contains Left Recursive Grammar")

if rightcheck\_dict[nt]:

print(nt+" contains Right Recursive Grammar")

print("\nLeft Eliminated Grammer:-")

for nt in nonterms:

if leftcheck\_dict[nt] == True:

Left\_toRight(nt)

else:

Prod\_print(nt)

**OUTPUT:**



**RESULT:** Hence, elimination of left recursion is successfully done.

**LEFT FACTORING:**

#include <iostream>

#include <string>

using namespace std;

int main()

{

int n,j,l,i,m;

int len[10] = {};

string a, b1, b2, flag;

char c;

cout<< "Enter the Parent Non-Terminal : ";

cin>> c;

a.push\_back(c);

b1 += a + "\'->";

b2 += a + "\'\'->";;

a += "->";

cout<< "Enter total number of productions : ";

cin>> n;

for (i = 0; i< n; i++)

{

cout<< "Enter the Production " <<i + 1 <<" : ";

cin>> flag;

len[i] = flag.size();

a += flag;

if (i != n - 1)

{

a += "|";

}

}

cout<< "The Production Rule is : " << a <<endl;

char x = a[3];

for (i = 0, m = 3; i< n; i++)

{

if (x != a[m])

{

while (a[m++] != '|');

}

else

{

if (a[m + 1] != '|')

{

b1 += "|" + a.substr(m + 1, len[i] - 1);

a.erase(m - 1, len[i] + 1);

}

else

{

b1 += "#";

a.insert(m + 1, 1, a[0]);

a.insert(m + 2, 1, '\'');

m += 4;

}

}

}

char y = b1[6];

for (i = 0, m = 6; i< n - 1; i++)

{

if (y == b1[m])

{

if (b1[m + 1] != '|')

{

flag.clear();

for (int s = m + 1; s < b1.length(); s++)

{

flag.push\_back(b1[s]);

}

b2 += "|" + flag;

b1.erase(m - 1, flag.length() + 2);

}

else

{

b1.insert(m + 1, 1, b1[0]);

b1.insert(m + 2, 2, '\'');

b2 += "#";

m += 5;

}

}

}

b2.erase(b2.size() - 1);

cout<< "After Left Factoring : " <<endl;

cout<< a <<endl;

cout<< b1 <<endl;

cout<< b2 <<endl;

return 0;

}

**OUTPUT**:



**RESULT**: Hence, the implementation of Left factoring is successfully done