AUMKAR M GADEKAR
ROLL NO: 21
TE COMPUTERS

Experiment No.04

AIM :- Write a program to eliminate direct and indirect left recursion.

THEORY :-

Left Recursion :-
A grammar becomes left-recursive if it has any non-terminal ‘A’ whose derivation contains ‘A’ itself as the leftmost symbol. Left-recursive grammar is considered to be a problematic situation for top-down parsers. Top-down parsers start parsing from the Start symbol, which in itself is non-terminal. So, when the parser encounters the same non-terminal in its derivation, it becomes hard for it to judge when to stop parsing the left non-terminal and it goes into an infinite loop.

Example:
(1) A => Aα | β

(2) S => Aα | β
    A => Sd

(1) is an example of immediate left recursion, where A is any non-terminal symbol and αrepresents a string of non-terminals.
(2) is an example of indirect-left recursion.
A top-down parser will first parse the A, which in-turn will yield a string consisting of A itself and the parser may go into a loop forever.

Removal of Left Recursion
One way to remove left recursion is to use the following technique: The production
A => Aα | β
is converted into following productions
A => βA'
A'=> αA' | ε

This does not impact the strings derived from the grammar, but it removes immediate left recursion.

Second method is to use the following algorithm, which should eliminate all direct and indirect left recursions.

```
START
Arrange non-terminals in some order like A1, A2, A3,..., An
for each i from 1 to n
{
    for each j from 1 to i-1
    {
        replace each production of form Ai ⇒Ajγ withAi ⇒δ1γ |δ2γ|δ3γ|...|γ
        where Aj ⇒ δ1 | δ2|...| δn are current Aj productions
    }
}
eliminate immediate left-recursion
END
```

Example :

The production set

S => Aα | β

A => Sd

after applying the above algorithm, should become

S => Aα | β

A => Aαd | βd

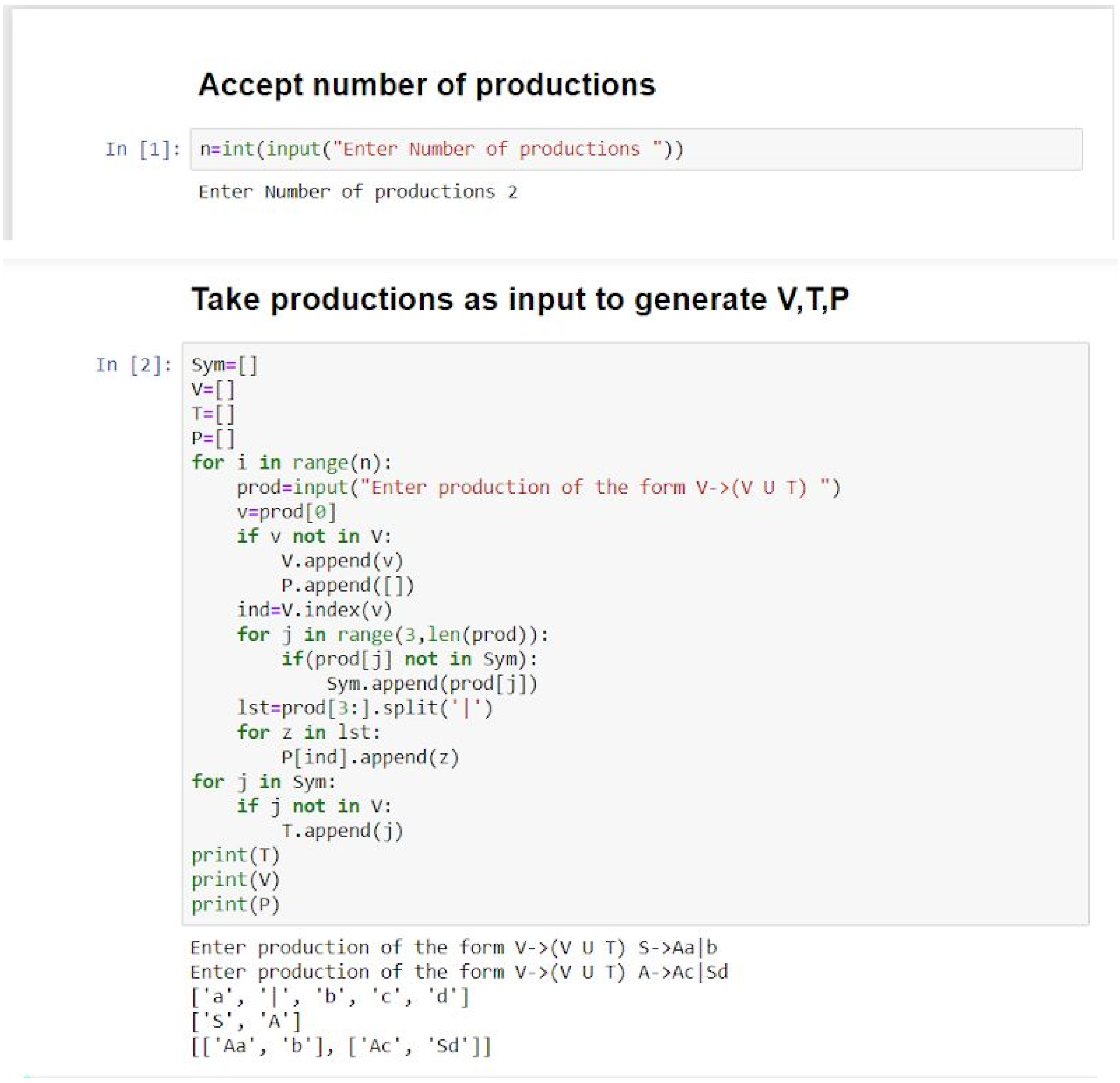
And then, remove immediate left recursion using the first technique.

A => βdA'

A' => αdA' | ε

Now none of the production has either direct or indirect left recursion.

IMPLEMENTATION IN PYTHON :

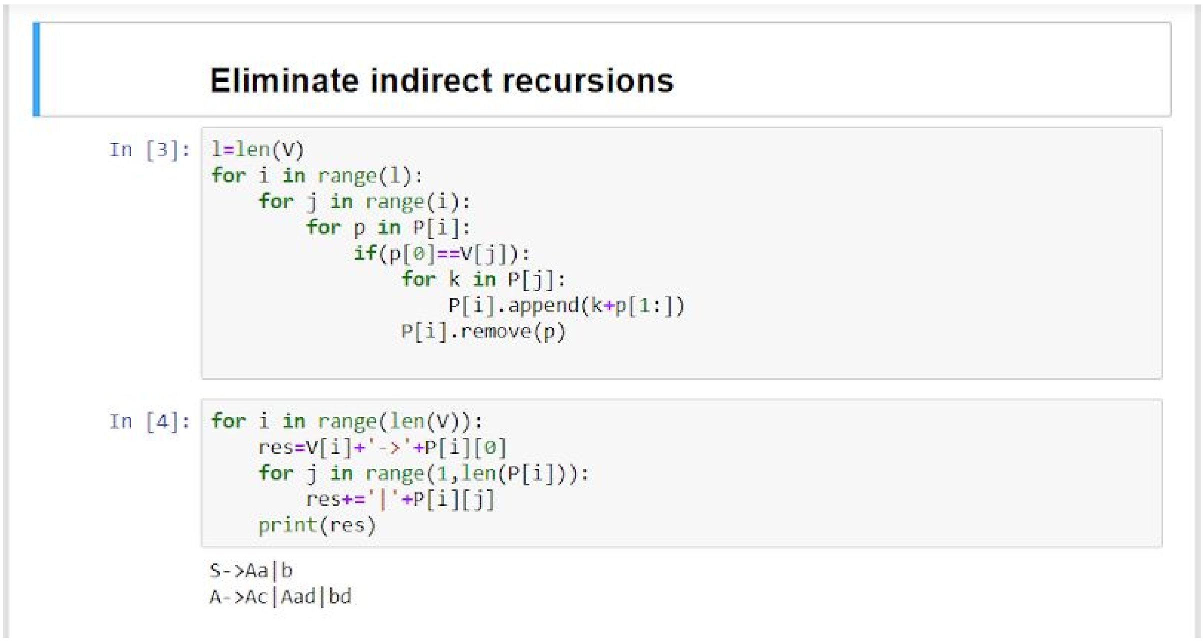


Accept number of productions

In [1]:
```
n=int(input("Enter Number of productions "))
```
```
Enter Number of productions 2
```

Take productions as input to generate V,T,P

In [2]:
```
Sym=[]
V=[]
T=[]
P=[]
for i in range(n):
    prod=input("Enter production of the form V->(V U T) ")
    v=prod[0]
    if v not in V:
        V.append(v)
        P.append([])
    ind=V.index(v)
    for j in range(3,len(prod)):
        if(prod[j] not in Sym):
            Sym.append(prod[j])
    lst=prod[3:].split('|')
    for z in lst:
        P[ind].append(z)
for j in Sym:
    if j not in V:
        T.append(j)
print(T)
print(V)
print(P)
```
```
Enter production of the form V->(V U T) S->Aa|b
Enter production of the form V->(V U T) A->Ac|Sd
['a', '|', 'b', 'c', 'd']
['S', 'A']
[['Aa', 'b'], ['Ac', 'Sd']]
```

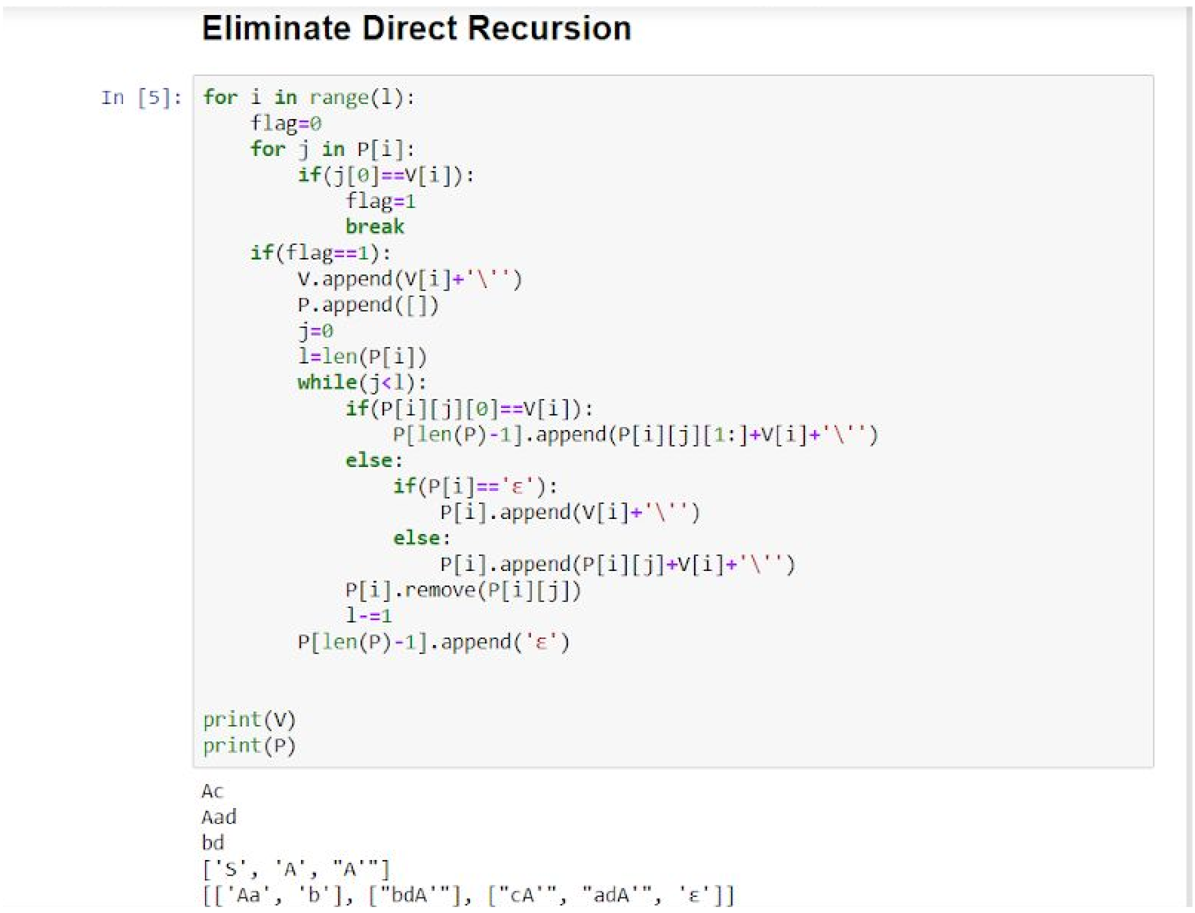


Eliminate indirect recursions

```
In [3]: l=len(V)
        for i in range(l):
            for j in range(i):
                for p in P[i]:
                    if(p[0]==V[j]):
                        for k in P[j]:
                            P[i].append(k+p[1:])
                        P[i].remove(p)
```

```
In [4]: for i in range(len(V)):
            res=V[i]+'->'+P[i][0]
            for j in range(1,len(P[i])):
                res+='|'+P[i][j]
            print(res)
```

```
S->Aa|b
A->Ac|Aad|bd
```

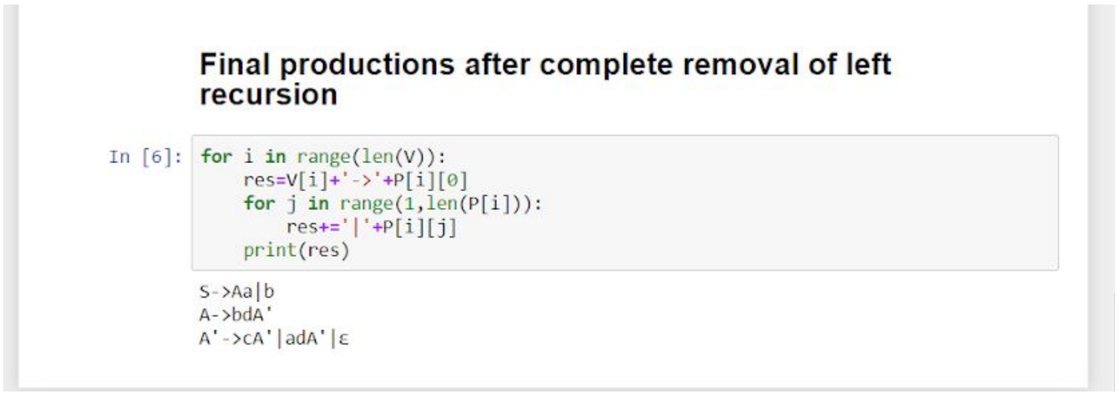


Eliminate Direct Recursion

```
In [5]: for i in range(l):
            flag=0
            for j in P[i]:
                if(j[0]==V[i]):
                    flag=1
                    break
            if(flag==1):
                V.append(V[i]+'\'')
                P.append([])
                j=0
                l=len(P[i])
                while(j<l):
                    if(P[i][j][0]==V[i]):
                        P[len(P)-1].append(P[i][j][1:]+V[i]+'\'')
                    else:
                        if(P[i]=='ε'):
                            P[i].append(V[i]+'\'')
                        else:
                            P[i].append(P[i][j]+V[i]+'\'')
                    P[i].remove(P[i][j])
                    l-=1
                P[len(P)-1].append('ε')


        print(V)
        print(P)
```

```
Ac
Aad
bd
['S', 'A', "A'"]
[['Aa', 'b'], ["bdA'"], ["cA'", "adA'", 'ε']]
```


Final productions after complete removal of left recursion

```
In [6]: for i in range(len(V)):
            res=V[i]+'->'+P[i][0]
            for j in range(1,len(P[i])):
                res+='|'+P[i][j]
            print(res)
```

```
S->Aa|b
A->bdA'
A'->cA'|adA'|ε
```

CONCLUSION:

Thus, we learnt how to eliminate indirect and direct left recursion in Grammar to make parsing easy.