EXPERIMENT NO 3

Aim : Program to remove left recursion for the given grammar. Program should accept the grammar from user, detect left recursion and eliminate it by generating a new non-terminal.

Code:

```
# Direct Recursion
import re
no_of_rules=int(input("Enter the number of rules : "))
list_of_rules=[]
i=0
while i<no_of_rules :
   list_of_rules.append(input("Rule "+str(i+1)+" : "))
   i=i+1

print(list_of_rules)
old_rules=[]
new_rules=[]
i=0
while i<no_of_rules :
   old_rules.append(re.split('->|\|',list_of_rules[i]))
   i=i+1
print(old_rules)

i=0
while i<len(old_rules) :
    length=len(old_rules[i])
    first=old_rules[i][0].strip()
    j=1
    left_detect=False
    left_detect_bt=[]
    left_detect_al=[]
    while j < length:
      temp=old_rules[i][j]
      if temp[0] is first:
         print("Left recursion detected at "+temp)
         left_detect=True
         left_detect_al.append(temp[1:])
      else:
         left_detect_bt.append(temp)
      j=j+1
    rule1=""
    rule1=str(first)+"->"
    if not left_detect :
      new_rules.append(rule1+".join(old_rules[i][1:]))
```

```
        i=i+1
        continue
    rule1=""
    rule1=str(first)+"->"
    k=0
    for k in range(len(left_detect_bt)):
        rule1+=left_detect_bt[k]+str(first)+"'|"
    new_rules.append(rule1)
    rule1=""
    rule1=str(first)+"'-> abs|"
    k=0
    for k in range(len(left_detect_al)):
        rule1+=left_detect_al[k]+str(first)+"'|"
    new_rules.append(rule1)

    i=i+1


print(new_rules)

# root@coder:/coder/mnt/Rstar/SPCC_PY_CODES# python3 left_recursion.py
# Enter the number of rules : 2
# Rule 1 : A->Aa|b
# Rule 2 : C->c
# ['A->Aa|b', 'C->c']
# [['A', 'Aa', 'b'], ['C', 'c']]
# Left recursion detected at Aa
# ["A->bA'|", "A'-> abs|aA'|", 'C->c']
# root@coder:/coder/mnt/Rstar/SPCC_PY_CODES# python3 left_recursion.py
# Enter the number of rules : 2
# Rule 1 : A->Aa|b
# Rule 2 : C->Ca|b
# ['A->Aa|b', 'C->Ca|b']
# [['A', 'Aa', 'b'], ['C', 'Ca', 'b']]
# Left recursion detected at Aa
# Left recursion detected at Ca
# ["A->bA'|", "A'-> abs|aA'|", "C->bC'|", "C'-> abs|aC'|"]
# root@coder:/coder/mnt/Rstar/SPCC_PY_CODES#
```

Post Lab Assignment:-

Q1. The following grammar can be used to describe traveling schemes:

TS → TS Time Time TS | Station

Station → Identifier

Time → Nat : Nat

Which of the following grammars is equivalent but no longer left-recursive? **(Tick Mark right answer(s))**

a) TS → TS (Time Time Station) ∗

Station → Identifier

Z

Time → Nat : Nat

b) TS → Station | Station Z

Z → Time Time TS | Time Time TS

Station → Identifier

Time → Nat : Nat

c) TS → Z Time Time TS | Station

Z → TS | #

Station → Identifier

Time → Nat : Nat

d) TS → Station Time Time Z

Z → Station | TS

Station → Identifier

Time → Nat : Nat

ANS:

b)

Q2. **Consider the grammar**

S → SX | SSb | XS | a

X → Xb | Sa | b

Eliminate left recursion and rewrite the grammar.

ANS:

If A → Aα1 | Aα2 | · · · | Aαm | β1 | β2| · · · | βn represents all the A-productions of the grammar, and no βi begins with A, then we can replace these A-productions by A → β1A′ | β2A′| · · · | βnA′ A′ → α1A′ | α2A′ | · · · | αmA′ | abs

Rule 1 : S → SX | SSb | XS | a

S → XSS′ | aS′

S′ → XS′ | SbS′| abs

Rule 2 : X → Xb | Sa | b

and the next obligation is to replace the production X → Sa with the productions

X → XSS′a | aS′a

We then eliminate immediate left recursion among X → XSS′a | aS′a | Xb | b

Eliminating immediate left recursion among X → XSS′a | Xb | b | aS′a yields

X → bX′ | aS′aX′

X′ → SS′aX′ | bX′ | abs

So the final result is

S → XSS′ | aS′

S′ → XS′ | SbS′| abs

X → bX′ | aS′aX′

X′ → SS′aX′ | bX′ | abs

Conclusion : From this experiment I got to know about how to remove left recursion from grammar.This concept will be used in compiler design