Experiment No. 5

Aim : Program to Compute first() and follow() set of the grammar.

Theory :
Automatic techniques for constructing parsers start with computing some basic functions for symbols in the grammar. These functions are useful in understanding both recursive descent and bottom-up LR parsers.

First(a) :
First(a) is the set of terminals that begin strings derived from a, which can include epsilon.

1. First(X) starts with the empty set.
2. if X is a terminal, First(X) is {X}.
3. if X -> epsilon is a production, add epsilon to First(X).
4. if X is a non-terminal and X -> Y1 Y2... Yk is a production, add First(Y1) to First(X).
5. for (i = 1; if Yi can derive epsilon; i++) add First(Yi+1) to First(X).

Follow(A) :
Follow(A) for nonterminal A is the set of terminals that can appear immediately to the right of A in some sentential form S -> aAxB... To compute Follow, apply these rules to all nonterminals in the grammar:

1. Add $ to Follow(S)
2. if A -> aBb then add First(b) - epsilon to Follow(B)
3. if A -> aB or A -> aBb where epsilon is in First(b), then add Follow(A) to Follow(B).

Program :

```
import re
ep = list()
fp = open("/content/grammer.txt","r") #read CFG from file cfg=dict() #create
dictionary to stored CFG
global non_terminal
def find_first(key):
 value=cfg[key] #find RHS of key(LHS)
 #print(key,value)
 if ('#' in value): #if key directly derived epsilon  value.remove('#')
 for item in value: #consider individual production rule  if item[0] in ep: #if that
variable produve epsilon  epsilon(item)
 #print("Epsilon called for ",item)
```

```
 else:
 if (item[0].islower()):
 #print(non_terminal,"-->",item[0])
 if item[0] not in temp:
 temp.append(item[0])
 else:
 find_first(item[0])

def epsilon(item):
 #print("From Epsilon ",item[0])
 find_first(item[0]) #find first of that variable
 length=len(item)
 i=1
 while(i<=length-1):
 if item[i] in ep:
 find_first(item[i])
 i=i+1
 if(i==length):
 #print(non_terminal,"-->#")
 if '#' not in temp:
 temp.append('#')
 break
 else:
 if (item[i].islower()):
 #print(non_terminal,"-->",item[i])
 if item[i] not in temp:
 temp.append(item[i])
 break
 else:
 find_first(item[i])
 break

def find_follow(key):

 for k,v in cfg.items():

 for item in v:

 if re.search(key,item): #search key in RHS
 index=item.find(key) #if found
 length=len(item)-1
 #print(key ,"is found at ",item, "at ",index)

 if (index==length):#if varaibel found at Right most side of  RHS
 temp1=follow[k] #then find follow[k]  for i in temp1:
```

```
 temp.append(i) #append it

 index=index+1 #find next symbol
 #print(index)
 for i in range(index,len(item)): #try all next  variable/terminal
 #print(i)
                                                    if(item[i].islower()): #if follow symbol is terminal
 temp.append(item[i]) #add it to follow()  break # stop
 else:
 temp1=first[item[i]] #find first[follow variable]  #print(temp1)
                                          for j in temp1:
 if(j!='#'):
 temp.append(j) #append it to follow  dictionary/temp

 if ('#' in temp1): #if first[follow item] contain  epsilon
 i=i+1 #check for next variable  else:
 break #else stop

 if(i==len(item)): #if we reach at right most side of RHS  temp1=follow[k] #find follow[LHS]
 for j in temp1:
 temp.append(j) #append result
```

```
for line in fp:
 line.strip() #remove begining and trailing white spaces  if re.search('\n',line):
 line=line[:line.find('\n')] #remove \n from line  split=line.split('->') #split line
(LHS and RHS)
 split=split[1].split('|') #split RHS based on |
 i=0
 for item in split:
 split[i]=item.strip() #remove begining and trailing white spaces  i=i+1
 #cfg[split[0]]=split
 cfg[line[0]]=split #store LHS as key and RHS as a values
```



```
print("\nGiven Context Free Grammar is =")

for key, value in cfg.items():
 print(key," ->",value) #print CFG
 if('#' in value): #if any variable/non terminal generate an epsilon  ep.append(key) #then stored
that varaible in list ep
```

```
temp=list()
first=dict()
for key,value in cfg.items():
 first[key]=[] #initialize value of key as list
 #print("FIRST[",key,"]") #find first of all variable(non-terminal)  non_terminal=key
 find_first(key)
 if key in ep:
 #print(non_terminal,"-->#")
 if '#' not in temp: #if varaible produce epsilon  temp.append('#') #add epsilon to
first[variable]  #print(temp)
 for item in temp:
 first[non_terminal].append(item) #add all results to first  #print(temp)
 temp.clear()

#print(first)
#print("Follow() are as follow->")
follow=dict()
flag=0
temp=list()

#print(first)
for key,value in cfg.items():
 follow[key]=[] #initialize value of key as list
 if flag==0:
 temp.append("$") #follow of start symbol is $
 flag=1
 find_follow(key)
 #print("follow[",key,"]")
 #print(temp)
 for k in temp:
 if( k not in follow[key]): # removed duplicate and add it to final  result
 follow[key].append(k)
 temp.clear()

#print(follow)
```



```
print(" Non Terminal First() Follow()")  print("-----------------------------------------------------")

for key, value in follow.items():
 print(" ",key," ",first[key]," ",value)  print("\n")
```

Output:

Given Context Free Grammar is =
S -> ['ABCDE']
A -> ['a', '#']
B -> ['b', '#']
C -> ['c']
D -> ['d', '#']
E -> ['e', '#']
Non Terminal First() Follow() ---------- S
['a', 'b', 'c'] ['$']

A ['a', '#'] ['b', 'c'] B ['b', '#'] ['c']

C ['c'] ['d', 'e', '$'] D ['d', '#'] ['e', '$'] E ['e', '#'] ['$']

Conclusion : Thus we have successfully implemented first and follow.