Experiment No. 4

Aim:- Write a program to find first and follow sets for the given grammar. Program should accept the grammar from user and output the first and follow sets for each of the grammar symbol.

Theory:-

First(α) is set of terminals that begins strings derived from α. If α=>ε then ε is also in First(ε).
In predictive parsing when we have A-> α|β, if First(α) and First(β) are disjoint sets then we can select appropriate A-production by looking at the next input.
Follow(A), for any non terminal A, is set of terminals a that can appear immediately after A in some sentential form
If we have S => αAaβ for some α and β then a is in Follow(A)
If A can be the rightmost symbol in some sentential form, then $ is in Follow(A)

Finding First:-

First(x) for all grammar symbols X

Apply following rules:

1. If X is terminal, FIRST(X) = {X}.
2. If X → ε is a production, then add ε to FIRST(X).
3. If X is a non-terminal, and X→ Y1Y2… Yk is a production, and ε is in all of FIRST(Y1), …, FIRST(Yk), then add ε to FIRST(X).
4. If X is a non-terminal, and X→ Y1Y2… Yk is a production, then add a to FIRST(X) if for some i, a is in FIRST(Yi), and ε is in all of FIRST(Y1), …, FIRST(Yi-1).

Applying rules 1 and 2 is obvious. Applying rules 3 and 4 for FIRST(Y1Y2… Yk) can be done as follows:

Add all the non-ε symbols of FIRST(Y1) to FIRST(Y1 Y2 … Yk). If ε ∈ FIRST(Y1), add all the non-ε symbols of FIRST(Y2). If ε ∈ FIRST(Y1) and ε ∈ FIRST(Y2), add all the non-ε symbols of FIRST(Y3), and so on. Finally, add ε to FIRST(Y1 Y2 … Yk) if ε ∈ FIRST(Yi), for all $1 \leq i \leq k$.

Example:
Consider the following grammar.
E → E + T | T
T → T * F | F
F → (E) | id
Grammar after removing left recursion:
E → TX
X → +TX | ε
T → FY

Y → *FY | ε
F → (E) | id
For the above grammar, following the above rules, the FIRST sets could be computed as follows:
FIRST(E) = FIRST(T) = FIRST(F) = {(, id}
FIRST(X) = {+, ε}
FIRST(Y) = {*, ε}

Finding Follow:-
To compute Follow(A) for all nonterminals A, apply following rules until nothing can be added to any follow set:
1. Place $ in Follow(S) where S is the start symbol
2. If there is a production A-> αBβ then everything in First(β) except ε is in Follow(B).
3. If there is a production A->B or a production A->αBβ where First(β) contains ε, then everything in Follow(A) is in Follow(B)

Example:-
E → TX
X → +TX | ε
T → FY
Y → *FY | ε
F → (E) | id

FOLLOW(E) = {$,) }
FOLLOW(X) = {$,) }
FOLLOW(T) = {+, $,) }
FOLLOW(Y) = {+, $,) }
FOLLOW(F) = {*, +, $,) }

Input Specifications:
1. Students are supposed to take grammar productions(non-left recursive) as input.
2. They can implement the given program in any language like C, C++, Java or Python, etc.

```
# First and Follow
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=[]
```

```
rule_status=[]
rule_id=[]
first_list=[]
i=0
while i<no_of_rules :
    old_rules.append(re.split('->|\|',list_of_rules[i]))
    rule_status.append(False)
    rule_id.append(old_rules[i][0].strip())
    i=i+1
print(old_rules)
# Rule 1 and Rule 2
i=0
while i<len(old_rules) :
    j=1
    temp_list=[]
    value=True
    while j<len(old_rules[i]):
        temp=old_rules[i][j].strip()
        if temp != "abs" and not temp[0].islower():
            value=False
            break
        j+=1
    rule_status[i]=value
    j=1
    while j<len(old_rules[i]):
        temp=old_rules[i][j].strip()
        if  temp == "abs" :
            temp_list.append(temp)
        elif temp[0].islower() or temp.islower() :
             temp_list.append(temp[0])
        j+=1
    first_list.append(temp_list)
    i+=1
print(rule_status)
print(rule_id)
print(first_list)
print("-----------------------------------------------------------------
-------")

while False in rule_status:
    i=0
    while i<len(old_rules):
        if(rule_status[i]==True):
            i=i+1
            continue
```

```
        j=1
        temp_status=[False]*(len(old_rules[i])-1)
        m=1
        while m<(len(old_rules[i])):
            if old_rules[i][m].strip() == "abs" or
(old_rules[i][m].strip())[0].islower():
                temp_status[m-1]=True
            m+=1

        while j<len(old_rules[i]) :
            if old_rules[i][j].strip() == "abs" or
(old_rules[i][j].strip())[0].islower():
                j+=1
                continue
            temp=old_rules[i][j].strip()

            y = [False] * len(temp)
            k=0
            for a in temp:
                if a.islower() or rule_status[rule_id.index(a)] :
                    y[k]=True
                k+=1

            if False in y:
                break
            else:
                for a in temp:

                    if not a.islower() and not "abs" in
first_list[rule_id.index(a)]:
                        first_list[i] = list(set(first_list[i] +
first_list[rule_id.index(a)]))
                        break
                    elif not a.islower() and "abs" in
first_list[rule_id.index(a)] and temp[-1]!=a:
                        temp1=list(first_list[rule_id.index(a)])
                        temp1.remove("abs")
                        first_list[i] = list(set(first_list[i]
+temp1))

                    elif not a.islower() and "abs" in
first_list[rule_id.index(a)] and temp[-1]==a:
                        temp1=list(first_list[rule_id.index(a)])
                        first_list[i] = list(set(first_list[i]
+temp1))
```

```
                    elif a.islower():
                        if not a in first_list[i]:
                            first_list[i].append(a)
                        break
                    else:
                        None
                print(first_list)
                temp_status[j-1]=True
            j+=1
        if not False in temp_status:
            rule_status[i]=True
        i+=1

print("First : ")
print(first_list)
print("Follow :")
start=input("Enter the start symbol : ")
follow_list=[]

i=0
while i< no_of_rules:
    temp_list=[]
    if old_rules[i][0] == start :
        temp_list.append("$")
    follow_list.append(temp_list)
    i+=1
print(follow_list)
i=0
while i<no_of_rules:
    character=old_rules[i][0]
    j=0
    while j<no_of_rules:
        temp=''.join(old_rules[j][1:])
        if not character in temp:
            j=j+1
            continue
        follow=[]
        list_contain_char=[]
        for word in old_rules[j][1:]:
            if character in word:
                list_contain_char.append(word)
        for word in list_contain_char:
            start_index=word.index(character)+1
```

```
            while start_index<len(word):
                if word[start_index].islower():
                    if not word[start_index] in follow:
                         follow.append(word[start_index])
                    break
                elif start_index!=(len(word)-1) and not "abs" in
first_list[rule_id.index(word[start_index])]:
                    follow = list(set(follow +
first_list[rule_id.index(word[start_index])]))
                    break
                elif start_index!=(len(word)-1) and "abs" in
first_list[rule_id.index(word[start_index])]:

temp1=list(first_list[rule_id.index(word[start_index])])
                    temp1.remove("abs")
                    follow = list(set(follow + temp1))
                elif start_index==(len(word)-1) and not "abs" in
first_list[rule_id.index(word[start_index])]:
                    follow = list(set(follow +
first_list[rule_id.index(word[start_index])]))
                    break
                elif start_index==(len(word)-1) and "abs" in
first_list[rule_id.index(word[start_index])]:

temp1=list(first_list[rule_id.index(word[start_index])])
                    temp1.remove("abs")
                    follow = list(set(follow + temp1+follow_list[j]))

                start_index+=1
            if start_index==(len(word)):
                follow = list(set(follow + follow_list[j]))
            follow_list[i]=list(set(follow_list[i]+follow))
        j+=1
    i+=1
print(follow_list)
```

Output:

```
# root@coder:/coder/mnt/Rstar/SPCC_PY_CODES# python first_follow.py
# Enter the number of rules : 5
# Rule 1 : S->ABCD
# Rule 2 : A->a|abs
# Rule 3 : B->CD|b
```

```
# Rule 4 : C->c|abs
# Rule 5 : D->Aa|d|abs
# ['S->ABCD', 'A->a|abs', 'B->CD|b', 'C->c|abs', 'D->Aa|d|abs']
# [['S', 'ABCD'], ['A', 'a', 'abs'], ['B', 'CD', 'b'], ['C', 'c',
'abs'], ['D', 'Aa', 'd', 'abs']]
# [False, True, False, True, False]
# ['S', 'A', 'B', 'C', 'D']
# [[], ['a', 'abs'], ['b'], ['c', 'abs'], ['d', 'abs']]
# ----------------------------------------------------------------
--
# [[], ['a', 'abs'], ['b'], ['c', 'abs'], ['d', 'a', 'abs']]
# [[], ['a', 'abs'], ['abs', 'd', 'b', 'a', 'c'], ['c', 'abs'], ['d',
'a', 'abs']]
# [['abs', 'b', 'd', 'a', 'c'], ['a', 'abs'], ['abs', 'd', 'b', 'a',
'c'], ['c', 'abs'], ['d', 'a', 'abs']]
# First :
# [['abs', 'b', 'd', 'a', 'c'], ['a', 'abs'], ['abs', 'd', 'b', 'a',
'c'], ['c', 'abs'], ['d', 'a', 'abs']]
# Follow :
# Enter the start symbol : S
# [['$'], [], [], [], []]
# [['$'], ['$', 'd', 'b', 'a', 'c'], ['d', 'a', '$', 'c'], ['d', 'a',
'$', 'c'], ['d', 'a', '$', 'c']]
# root@coder:/coder/mnt/Rstar/SPCC_PY_CODES#
```

Post Lab Assignment:-

Q1. Which symbols are used as synchronizing tokens in predictive parsing?

A. Follow symbols B. First symbols C. First and Follow symbols D. None of these

Answer:

First and Follow symbols are used as synchronizing tokens in predictive parsing i.e. **Option C**.

Q2. Find the First and Follow sets for the grammar

E -> TP
P -> Eps | |TP
T -> FQ
Q -> Eps | FQ
F -> (E) R | iR
R -> Eps | *R

Answer:

	First	Follow
E -> TP	{ (, i }	{ $,) }
P -> Eps | |TP	{ Eps , | }	{ $,) }
T -> FQ	{ (, i }	{ | , $,) }
Q -> Eps | FQ	{Eps , (,i}	{ | , $,) }
F -> (E) R | iR	{ (, i }	{ | ,$, (,) , i }
R -> Eps | *R	{ Eps , * }	{ | , $, (,) , i }

Conclusion: From this experiment I understood about the computation of first and follow. This concept will be further used in next experiment