EXPERIMENT - IX
FIRST AND FOLLOW

September 30, 2020

ADITHYA D RAJAGOPAL
ROLL NO : 9
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

AIM

To write a program to simulate the FIRST and FOLLOW for any given grammar.

THEORY

Each time a predictive parser makes a decision, it needs to determine which production rule to apply to the leftmost non-terminal in an intermediate form, based on the next terminal (i.e. the look-ahead symbol).This is the significance of the FIRST sets: they tell you when a non-terminal can produce the look-ahead symbol as the beginning of a statement, so that it can be matched away and reduce the input. FOLLOW covers the possibility that the leftmost non-terminal can disappear, so that the look-ahead symbol is not actually a part of what we're presently expanding, but rather the beginning of the next construct. This is the significance of the FOLLOW sets: they tell you when a non-terminal can hand you the look-ahead symbol at the beginning of a statement by disappearing. Choosing productions that give epsilon doesn't reduce the input string, but you still have to make a rule for when the parser needs to take them, and the appropriate conditions are found from the FOLLOW set of the troublesome non-terminal.

FIRST(X) for all Grammar Symbols X

Apply the following rules:

1. If X is terminal, FIRST(X)=X.
2. If X $\rightarrow \epsilon$ is a production then add ϵ to FIRST(X).
3. If X is a non-terminal, and X $\rightarrow Y_1\ Y_2\ Y_k$ is a production, and ϵ is in all of FIRST(Y_1), FIRST(Y_2), ..., FIRST(Y_k), then add ϵ to FIRST(X).
4. If X is a non-terminal, and X $\rightarrow Y_1\ Y_2\ Y_k$ is a production, then add a to FIRST(X) if for some i, a is in FIRST(Y_k) and is in all of FIRST(Y_1), FIRST(Y_2), ..., FIRST(Y_{k-1}).

FOLLOW(A) for all Non-terminal A

Apply the following rules:

1. If \$ is the input end-marker, and S is the start symbol, \$ $\in$ FOLLOW(S).
2. If there is a production, A $\rightarrow \alpha$Bβ, then FIRST(β) - $\epsilon \subseteq$ FOLLOW(B).
3. If there is a production, A $\rightarrow \alpha$B or A $\rightarrow \alpha$Bβ, where $\epsilon \in$ FIRST(β) then FOLLOW(A) $\subseteq$ FOLLOW(B).

ALGORITHM

Algorithm 1 Algorithm to find FIRST(X) for all grammar symbols X

Start
if X is a terminal **then**
    FIRST(X) = X.
end if
if X $\rightarrow \epsilon$ is a production **then**
    Add ϵ to FIRST(X).
end if
if X is a non-terminal and X $\rightarrow Y_1\ Y_2 \ldots Y_k$ is a production **then**
    if ϵ is in all of FIRST(Y_1),FIRST(Y_2),...,FIRST(Y_k) **then**
        Add ϵ to FIRST(X).
    end if
    if $\exists$i, a $\in$ FIRST(Y_i) and ϵ is in all of FIRST(Y_1),FIRST(Y_2),...,FIRST(Y_k) **then**
        Add a to FIRST(X).
    end if
end if
Stop

Algorithm 2 Algorithm to find FOLLOW(A) for all non-terminals A

Start
if \$ is the input end-marker and S is the start symbol **then**
    Add \$ to FOLLOW(S).
end if
if A $\rightarrow \alpha$Bβ is a production **then**
    FIRST(β)-$\epsilon \subseteq$ FOLLOW(B).
end if
if there is a production, A $\rightarrow \alpha$B or A $\rightarrow \alpha$Bβ, where $\epsilon \in$ FIRST(β) **then**
    FOLLOW(A) $\subseteq$ FOLLOW(B)
end if
Stop

SOURCE CODE

```
def FIRST(X):
        global production
        global T
        global N
        first=[]
        if X in T:
                first.append(X)
        if X in N:
                if '#' in production[X]:
                        first.append('#')
                for pr in production[X]:
                        for p in pr:
                                f=FIRST(p)
                                for i in f:
                                        if i not in first:
                                                first.append(i)
                                if '#' not in f:
                                        break
                        flag=0
                        for p in pr:
                                if '#' not in FIRST(p):
                                        flag=1
                                        break
                        if flag==0:
                                if '#' not in first:
                                        first.append('#')
        return first

def FOLLOW():
        global production
        global follow
        global first
        global N
```

```
        for X in N:
                for pr in production[X]:
                        if pr[-1] in N and '$' not in follow[pr[-1]]:
                                follow[pr[-1]].append('$')
                        for i in range(len(pr)-1):
                                if pr[i] in N:
                                        f=first[pr[i+1]]
                                        for x in f:
                                                if (x not in follow[pr[i]]
                                                and x!='#'):
                                                        follow[pr[i]].append(x)
        for X in N:
                for pr in production[X]:
                        for p in pr:
                                if ((p==pr[-1]
                                or '#' in first[pr[pr.index(p)+1]])
                                and p in N):
                                        f=follow[X]
                                        for x in f:
                                                if x not in follow[p]:
                                                        follow[p].append(x)

global production
global T
global N
global follow
global first
production={}
n=int(input("Enter the number of productions:"))
print("Enter the productions:\t\t(Use \# for epsilon)")
T=[]
N=[]
s=[]
first={}
follow={}
```

```
for _ in range(n):
        pr=input().split(" -> ")
        if pr[0] in production.keys():
                production[pr[0]].append(pr[1])
        else:
                production[pr[0]]=[pr[1]]
        if pr[0] not in N:
                N.append(pr[0])
        if pr[0] not in s:
                s.append(pr[0])
        if pr[1]=='#':
                continue
        for i in pr[1]:
                if i not in s:
                        s.append(i)

for i in s:
        if i not in N:
                T.append(i)
for i in s:
        first[i]=FIRST(i)
        if i in N:
                follow[i]=[]
follow[N[0]].append('$')
print("Set of Terminals : ",end="")
for i in range(len(T)-1):
        print(T[i],end=",")
print(T[-1])
print("Set of Non-Terminals : ",end="")
for i in range(len(N)-1):
        print(N[i],end=",")
print(N[-1])
print("FIRST")
for i in s:
        print("\t",i,end=" : ")
```

```
        for f in first[i]:
                print(f,end=" ")
        print()
print("FOLLOW")
FOLLOW()
for i in follow.keys():
        print("\t",i,end=" : ")
        for f in follow[i]:
                print(f,end=" ")
        print()
```

SAMPLE OUTPUT

```
user@adithya-d-rajagopal:~/s7/cd$ python3 p9.py
Enter the number of productions:8
Enter the productions:          (Use \# for epsilon)
E -> TR
F -> (E)
F -> i
R -> #
R -> +TR
T -> FY
Y -> #
Y -> *FY
Set of Terminals : (,),i,+,*
Set of Non-Terminals : E,F,R,T,Y
FIRST
         E : ( i
         T : ( i
         R : # +
         F : ( i
         ( : (
         ) : )
         i : i
         + : +
         Y : # *
         * : *
FOLLOW
         E : $ )
         T : + $ )
         R : $ )
         F : * + $ )
         Y : $ + )
user@adithya-d-rajagopal:~/s7/cd$
```

RESULT

A program to simulate the FIRST and FOLLOW for any given grammar has been implemented using Python and the outputs have been verified.