|  |  |
| --- | --- |
| Name: Varun Khadayate | Subject: Compiler Design |
| Roll No: A016 | Date of Submission: 3rd September 2021 |

# Aim

To find the First and Follow of the given grammar

# Program logic

## First Function-

|  |
| --- |
| First(α) is a set of terminal symbols that begin in strings derived from α. |

### Example-

Consider the production rule-

A → abc / def / ghi

Then, we have-

First(A) = { a , d , g }

### Rules For Calculating First Function-

#### Rule-01:

 For a production rule X → ∈,

First(X) = { ∈ }

#### Rule-02:

 For any terminal symbol ‘a’,

First(a) = { a }

#### Rule-03:

 For a production rule X → Y1Y2Y3,

##### Calculating First(X)

* If ∈ ∉ First(Y1), then First(X) = First(Y1)
* If ∈ ∈ First(Y1), then First(X) = { First(Y1) – ∈ } ∪ First(Y2Y3)

##### Calculating First(Y2Y3)

* If ∈ ∉ First(Y2), then First(Y2Y3) = First(Y2)
* If ∈ ∈ First(Y2), then First(Y2Y3) = { First(Y2) – ∈ } ∪ First(Y3)

Similarly, we can make expansion for any production rule X → Y1Y2Y3…..Yn.

## Follow Function-

|  |
| --- |
| Follow(α) is a set of terminal symbols that appear immediately to the right of α. |

## Rules For Calculating Follow Function-

### Rule-01:

 For the start symbol S, place $ in Follow(S).

### Rule-02:

 For any production rule A → αB,

Follow(B) = Follow(A)

### Rule-03:

 For any production rule A → αBβ,

* If ∈ ∉ First(β), then Follow(B) = First(β)
* If ∈ ∈ First(β), then Follow(B) = { First(β) – ∈ } ∪ Follow(A)

Important Notes-

 Note-01:

* ∈ may appear in the first function of a non-terminal.
* ∈ will never appear in the follow function of a non-terminal.

Note-02:

* Before calculating the first and follow functions, eliminate**Left Recursion**from the grammar, if present.

Note-03:

* We calculate the follow function of a non-terminal by looking where it is present on the RHS of a production rule.

# PRACTICE PROBLEMS BASED ON CALCULATING FIRST AND FOLLOW-

## Problem-01:

Calculate the first and follow functions for the given grammar-

S → aBDh

B → cC

C → bC / ∈

D → EF

E → g / ∈

F → f / ∈

## **Solution-**

The first and follow functions are as follows-

### **First Functions-**

* First(S) = { a }
* First(B) = { c }
* First(C) = { b , ∈ }
* First(D) = { First(E) – ∈ } ∪ First(F) = { g , f , ∈ }
* First(E) = { g , ∈ }
* First(F) = { f , ∈ }

### Follow Functions-

* Follow(S) = { $ }
* Follow(B) = { First(D) – ∈ } ∪ First(h) = { g , f , h }
* Follow(C) = Follow(B) = { g , f , h }
* Follow(D) = First(h) = { h }
* Follow(E) = { First(F) – ∈ } ∪ Follow(D) = { f , h }
* Follow(F) = Follow(D) = { h }

# Lab Assignment

## What is First and Follow?

First(α) is a set of terminal symbols that begin in strings derived from α.

Follow(α) is a set of terminal symbols that appear immediately to the right of α.

## Specify rules for first and follow?

### Rules For Calculating First Function-

#### Rule-01:

 For a production rule X → ∈,

First(X) = { ∈ }

#### Rule-02:

 For any terminal symbol ‘a’,

First(a) = { a }

#### Rule-03:

 For a production rule X → Y1Y2Y3,

##### Calculating First(X)

* If ∈ ∉ First(Y1), then First(X) = First(Y1)
* If ∈ ∈ First(Y1), then First(X) = { First(Y1) – ∈ } ∪ First(Y2Y3)

##### Calculating First(Y2Y3)

* If ∈ ∉ First(Y2), then First(Y2Y3) = First(Y2)
* If ∈ ∈ First(Y2), then First(Y2Y3) = { First(Y2) – ∈ } ∪ First(Y3)

Similarly, we can make expansion for any production rule X → Y1Y2Y3…..Yn.

## Rules For Calculating Follow Function-

### Rule-01:

 For the start symbol S, place $ in Follow(S).

### Rule-02:

 For any production rule A → αB,

Follow(B) = Follow(A)

### Rule-03:

 For any production rule A → αBβ,

* If ∈ ∉ First(β), then Follow(B) = First(β)
* If ∈ ∈ First(β), then Follow(B) = { First(β) – ∈ } ∪ Follow(A)

## Define algorithm for first and follow?

### Algorithm for calculating First set

* if α is a terminal, then FIRST(α) = { α }.
* if α is a non-terminal and α → ℇ is a production, then FIRST(α) = { ℇ }.
* if α is a non-terminal and α → 𝜸1 𝜸2 𝜸3 … 𝜸n and any FIRST(𝜸) contains t then t is in FIRST(α).

### Algorithm for calculating Follow set:

* if α is a start symbol, then FOLLOW() = $
* if α is a non-terminal and has a production α → AB, then FIRST(B) is in FOLLOW(A) except ℇ.
* if α is a non-terminal and has a production α → AB, where B ℇ, then FOLLOW(A) is in FOLLOW(α).

# Lab Assignment Program

Write a program to implement first and follow from given grammar.

## Code

gram = {

    "S":["aBDh"],

    "B":["cC"],

    "C":["bC","e"],

    "D":["EF"],

    "E":["g","e"],

    "F":["f","e"]

}

def removeDirectLR(gramA, A):

    """gramA is dictonary"""

    temp = gramA[A]

    tempCr = []

    tempInCr = []

    for i in temp:

        if i[0] == A:

            tempInCr.append(i[1:]+[A+"'"])

        else:

            tempCr.append(i+[A+"'"])

    tempInCr.append(["e"])

    gramA[A] = tempCr

    gramA[A+"'"] = tempInCr

    return gramA

def checkForIndirect(gramA, a, ai):

    if ai not in gramA:

        return False

    if a == ai:

        return True

    for i in gramA[ai]:

        if i[0] == ai:

            return False

        if i[0] in gramA:

            return checkForIndirect(gramA, a, i[0])

    return False

def rep(gramA, A):

    temp = gramA[A]

    newTemp = []

    for i in temp:

        if checkForIndirect(gramA, A, i[0]):

            t = []

            for k in gramA[i[0]]:

                t=[]

                t+=k

                t+=i[1:]

                newTemp.append(t)

        else:

            newTemp.append(i)

    gramA[A] = newTemp

    return gramA

def rem(gram):

    c = 1

    conv = {}

    gramA = {}

    revconv = {}

    for j in gram:

        conv[j] = "A"+str(c)

        gramA["A"+str(c)] = []

        c+=1

    for i in gram:

        for j in gram[i]:

            temp = []

            for k in j:

                if k in conv:

                    temp.append(conv[k])

                else:

                    temp.append(k)

            gramA[conv[i]].append(temp)

    for i in range(c-1,0,-1):

        ai = "A"+str(i)

        for j in range(0,i):

            aj = gramA[ai][0][0]

            if ai!=aj :

                if aj in gramA and checkForIndirect(gramA,ai,aj):

                    gramA = rep(gramA, ai)

    for i in range(1,c):

        ai = "A"+str(i)

        for j in gramA[ai]:

            if ai==j[0]:

                gramA = removeDirectLR(gramA, ai)

                break

    op = {}

    for i in gramA:

        a = str(i)

        for j in conv:

            a = a.replace(conv[j],j)

        revconv[i] = a

    for i in gramA:

        l = []

        for j in gramA[i]:

            k = []

            for m in j:

                if m in revconv:

                    k.append(m.replace(m,revconv[m]))

                else:

                    k.append(m)

            l.append(k)

        op[revconv[i]] = l

    return op

result = rem(gram)

def first(gram, term):

    a = []

    if term not in gram:

        return [term]

    for i in gram[term]:

        if i[0] not in gram:

            a.append(i[0])

        elif i[0] in gram:

            a += first(gram, i[0])

    return a

firsts = {}

for i in result:

    firsts[i] = first(result,i)

    print(f'First of ({i}):',firsts[i])

def follow(gram, term):

    a = []

    for rule in gram:

        for i in gram[rule]:

            if term in i:

                temp = i

                indx = i.index(term)

                if indx+1!=len(i):

                    if i[-1] in firsts:

                        a+=firsts[i[-1]]

                    else:

                        a+=[i[-1]]

                else:

                    a+=["e"]

                if rule != term and "e" in a:

                    a+= follow(gram,rule)

    return a

follows = {}

print("\n")

for i in result:

    follows[i] = list(set(follow(result,i)))

    if "e" in follows[i]:

        follows[i].pop(follows[i].index("e"))

    follows[i]+=["$"]

    print(f'Follow of ({i}):',follows[i])

## Output

Text

Description automatically generated

# Conclusion

Hence, we were able to implement first and follow of the given grammar.