Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report #4

Compiler Design

[Course Code: COMP 409]

[For the partial fulfillment of 4th year/1st Semester in Computer Engineering]

Submitted by:

Sabin Thapa

Roll no. 54

CE 4th Year

Submitted to:

Mr. Sushil Nepal

Assistant Professor

Department of Computer Science and Engineering

Submission date Nov. 15, 2022

Task

Write a program to identify FIRST and FOLLOW from the given grammar.

Background

FIRST is a function that gives the set of terminals that begins the strings derived from the production rule. A symbol 'a' is in FIRST (α) if and only if α ⇒ aβ for some sequence β of grammar symbols.

Rules to find the FIRST() of a grammar:

To find the FIRST() of the grammar symbol, then we have to apply the following set of rules to the given grammar:

1. If X is a terminal, then FIRST(X) is {X}.
2. If X is a non-terminal and X =>aα is production, then add 'a' to the FIRST of X. If X => ε, then add null to the FIRST(X).
3. If X => YZ then if FIRST(Y)=ε, then FIRST(X) = { FIRST(Y)-ε} U FIRST(Z).
4. If X=>YZ, then if FIRST(X)=Y, then FIRST(Y)=teminal but null then FIRST(X)=FIRST(Y)=terminals.

A terminal symbol a is in FOLLOW(N) if and only if there is a derivation from the start symbol S of the grammar such that S ⇒ αNαβ, where α and β are a sequence of grammar symbols (possible empty). In other words, a terminal d is in FOLLOW(N) if d can follow N at some point in a derivation.

Rules to find the FOLLOW() of a grammar:

To find the follow() of the grammar symbol, then we have to apply the following set of rules to the given grammar:

1. $ is a FOLLOW of the start symbol 'S'.
2. If A=>αBβ, β!=ε, then FIRST(β) is in FOLLOW(B).
3. If A=>αB or A=>αBβ where FIRST(β)=ε, then everything in FOLLOW(A) is a FOLLOW(B).

Source Code

```python
from prettytable import PrettyTable as table
input_grammar = []

def evaluate_first(symbol):
    if symbol[0]==symbol[0].lower():
        return symbol[0]
    else:
        FIRST = ''
        for i in range(len(symbol)):
            for production in input_grammar:
                if production[0]==symbol[i]:
                    for x in production[1]:
                        if x[0]==symbol[0]:
                            FIRST+=evaluate_first(x[1:])
                        else:
                            FIRST+=evaluate_first(x)
            if 'e' in FIRST:
                continue
            else:
                break
        if FIRST[-1] != 'e':
            FIRST = FIRST.replace('e','')
        return FIRST

def evaluate_follow(symbol):
    # evaluate_follow of augmented input_grammar
    if symbol[0] == 'X':
        return '$'
    else:
        FOLLOW = ''
        for production in input_grammar:
            for prod in production[1]:
                if symbol[0] in prod:
                    if symbol[0] == prod[-1]:
                        FOLLOW += evaluate_follow(production[0]) if
production[0] != symbol[0] else ''
                    else:
                                                                FOLLOW +=
evaluate_first(prod[prod.index(symbol[0])+1:])
```

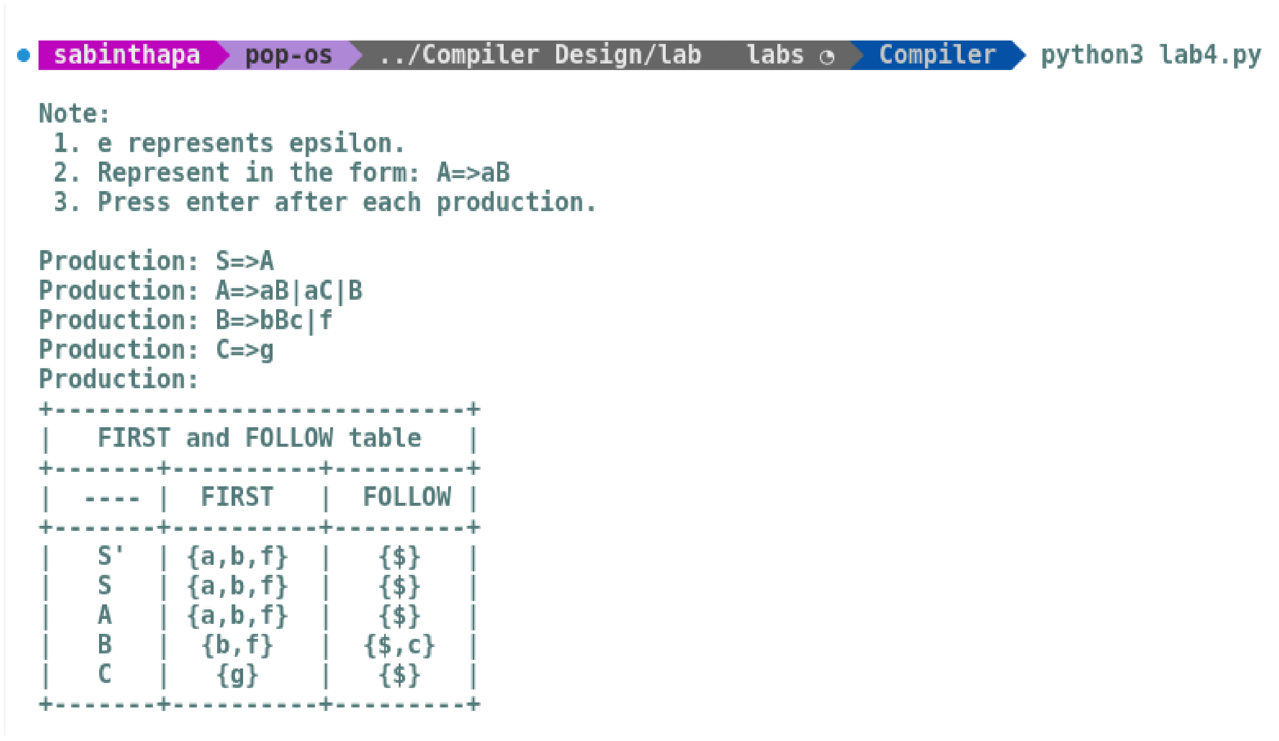
```
                    if 'e' in FOLLOW:
                        FOLLOW.replace('e','')
                        if symbol[0] != production[0]:
                            FOLLOW += evaluate_follow(production[0])
        return FOLLOW

def tabulate_results():
    output_table = []
    for production in input_grammar:

output_table.append([production[0],evaluate_first(production[0]),evaluate_fo
llow(production[0])])

    for i in range(len(output_table)):
        output_table[i][1] = "".join(dict.fromkeys(output_table[i][1]))
        output_table[i][2] = output_table[i][2].replace('e','')
        output_table[i][2] = "".join(dict.fromkeys(output_table[i][2]))
    return output_table

def display_table(tbl):
    output_table = table(['----','FIRST','FOLLOW'])
    output_table.title = 'FIRST and FOLLOW table'
    for i in range(len(tbl)):
        row=[]
        row.append(tbl[i][0] if not tbl[i][0]=='X' else tbl[i+1][0]+"'")
        for j in tbl[i][1:]:
            temp = '{'
            for k in j[:-1]:
                temp +=k+','
            temp+=j[-1]+'}'
            row.append(temp)
        output_table.add_row(row)
    print(output_table)

def take_grammar_input():
    input_grammar = []

    print('Note: \n 1. e represents epsilon. \n',
          '2. Represent in the form: A=>aB \n',
          '3. Press enter after each production. \n')
    while(True):
        inp = input(f'Production: ')
        if inp == '':
            break
        input_grammar.append(inp)

    for i,g in enumerate(input_grammar):
        input_grammar[i]=g.split('=>')
        input_grammar[i][1]=input_grammar[i][1].split('|')
    input_grammar.insert(0,['X',[input_grammar[0][0]+'$']])
    i=0
    while i<len(input_grammar):
        j=0
        while j<len(input_grammar):
            if i!=j:
                if input_grammar[i][0] == input_grammar[j][0]:
                    input_grammar[i][1].extend(input_grammar[j][1])
                    input_grammar.pop(j)
            j+=1
        i+=1
    return input_grammar
if __name__=='__main__':
    input_grammar = take_grammar_input()
    output_table = tabulate_results()
    display_table(output_table)
```

Explanation

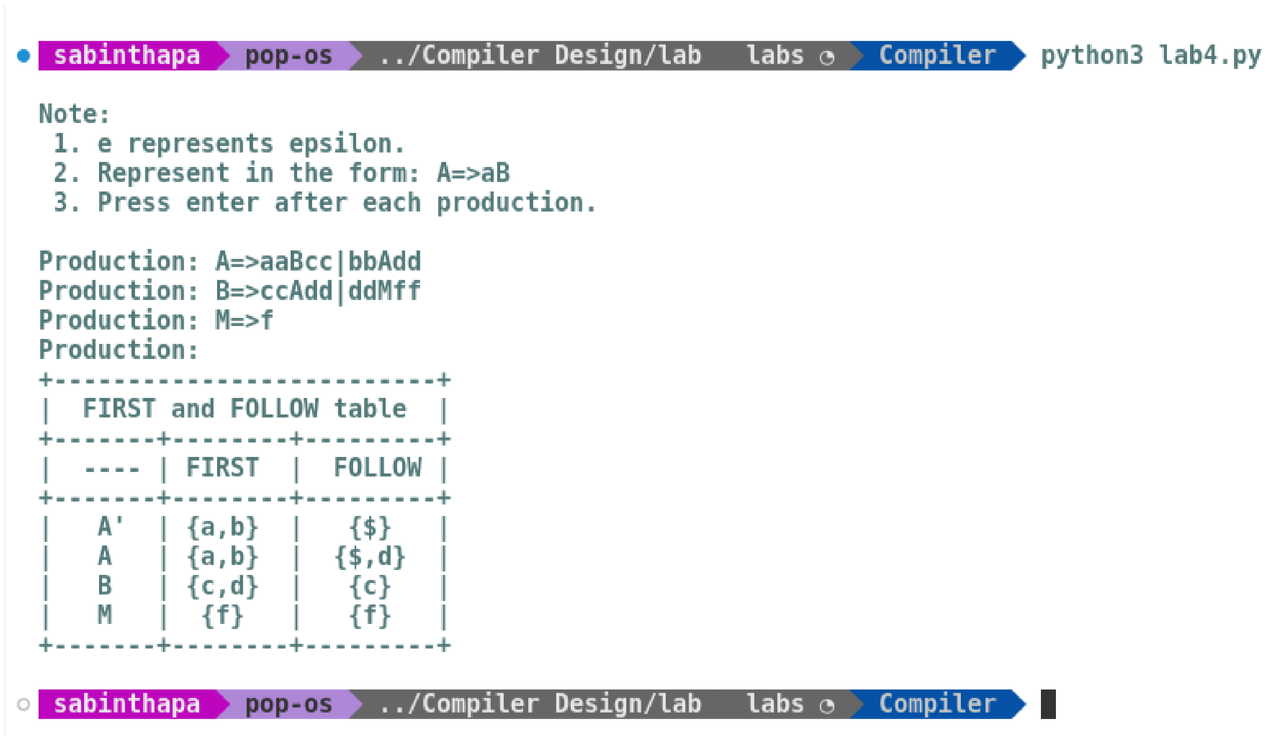
A grammar is taken as input from the user. Each of the production is accepted separately. The FIRST and FOLLOW sets are computed according to the rules as discussed above. The final results are displayed in a tabular form using the PrettyTable library of Python. The augmented production and its FIRST and FOLLOW sets are also computed. The follow of the starting symbol is considered to be a $ sign and epsilon is represented using the character ‘e’.

There are four main functions: evaluate_first() used to compute the FIRST sets, evaluate_follow() used to compute the FOLLOW sets, tabulate_results() and display_table() used to display the output to the user.

Outputs



```
● sabinthapa  pop-os  ../Compiler Design/lab  labs ◔  Compiler  python3 lab4.py

  Note:
   1. e represents epsilon.
   2. Represent in the form: A=>aB
   3. Press enter after each production.

  Production: S=>A
  Production: A=>aB|aC|B
  Production: B=>bBc|f
  Production: C=>g
  Production:
  +---------------------------+
  |   FIRST and FOLLOW table   |
  +-------+---------+---------+
  |  ---- |  FIRST   |  FOLLOW |
  +-------+---------+---------+
  |   S'  | {a,b,f}  |   {$}    |
  |   S   | {a,b,f}  |   {$}    |
  |   A   | {a,b,f}  |   {$}    |
  |   B   |  {b,f}   |  {$,c}   |
  |   C   |   {g}    |   {$}    |
  +-------+---------+---------+
```



```
● sabinthapa  pop-os  ../Compiler Design/lab  labs ◔  Compiler  python3 lab4.py

  Note:
   1. e represents epsilon.
   2. Represent in the form: A=>aB
   3. Press enter after each production.

  Production: A=>aaBcc|bbAdd
  Production: B=>ccAdd|ddMff
  Production: M=>f
  Production:
  +-------------------------+
  |  FIRST and FOLLOW table  |
  +-------+--------+---------+
  |  ---- | FIRST  |  FOLLOW |
  +-------+--------+---------+
  |   A'  | {a,b}  |   {$}   |
  |   A   | {a,b}  |  {$,d}  |
  |   B   | {c,d}  |   {c}   |
  |   M   |  {f}   |   {f}   |
  +-------+--------+---------+

○ sabinthapa  pop-os  ../Compiler Design/lab  labs ◔  Compiler 
```

As we can see from the outputs above, the FIRST and the FOLLOW sets for the given grammar are computed correctly and displayed in a tabular form.

Conclusion

In this way, the program to calculate the FIRST and FOLLOW sets from the given input grammar was implemented in Python. The source code and the outputs of the program are attached in the document above.