University of Asia Pacific

Department of Computer Science & Engineering

Compiler Design Lab
CSE 430

Submitted to:

Baivab Das
Lecturer
CSE, University of Asia Pacific

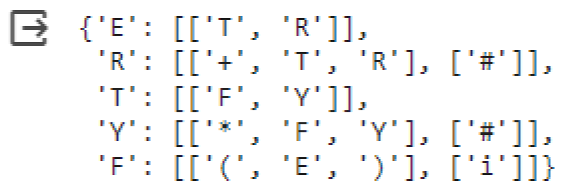
Submitted by:

Md. Farhad
20101073
Section: B1

Problem

FIRST & FOLLOW Function

Sample Input (Console Input/ File Input):



```
{'E': [['T', 'R']],
 'R': [['+', 'T', 'R'], ['#']],
 'T': [['F', 'Y']],
 'Y': [['*', 'F', 'Y'], ['#']],
 'F': [['(', 'E', ')'], ['i']]}
```

Code

import library

```
import re
```

take input from file

```
input = open("/content/drive/MyDrive/Uap/Compiler lab/farhad/input0.txt", "r")
input

productions = {}
first_dic = {}
follow_dic = {}
```

extract data from file which is grammar variable

```
for prod in input:
  l = re.split("( /->/\n/)*", prod)
  #print('l: ', l)
  m = []
  for i in l:
    if (i == "" or i == None or i == '\n' or i == " " or i == "-" or i == ">"):
#checking the splitation and enter is pressed or -> is found
      pass
    else:
      m.append(i)
  #print('m: ', m)

  left_prod = m.pop(0)
  right_prod = []
  t = []
  # taking input the values after |
  for j in m:
    if(j != '|'):
      t.append(j)
    else:
      right_prod.append(t)
      t = []

  right_prod.append(t)
  productions[left_prod] = right_prod
  print('productions= ', productions)
```

```
productions
```

create First function

```
def first_func(s, productions):
  first = set()
  # set() is used for storing multiple item into a single variable.
  #iterating in production dictionary
  for i in range(len(productions[s])):
    for j in range(len(productions[s][i])):
      c = productions[s][i][j]  #store all in c
      #if the variable is found then
      if(c.isupper()): #here, upper means any capital letter
        f = first_func(c, productions)
        #if no epsilon is present in f
        if('#' not in f):
          for k in f:
            first.add(k)
          break
        else:
          if(j == len(productions[s][i])-1):
            for k in f:
              first.add(k)
          else:
            f.remove('#')
            for k in f:
              first.add(k)
      else:
        first.add(c)
        break
  return first
```

create Follow function

```
def follow_func(s, productions, first):
  follow = set()

  if len(s)!=1 :
    return {}
  if(s == list(productions.keys())[0]): #in start we add $ intially
    follow.add('$')  #dollar sign is used for the non terminal same values
  # iterating in production dictionary
  for i in productions:
    for j in range(len(productions[i])):
      if(s in productions[i][j]):
        idx = productions[i][j].index(s) #here,idx is used to keep the keys or
index mapping of dictionary.

        #if index value of product and current value matches break
        if(idx == len(productions[i][j])-1):
          if(productions[i][j][idx] == i):
            break

          #else recursive function to find the follow of ith index value
          else:
            f = follow_func(i, productions, first)
            for x in f:
              follow.add(x) #add them in follow function

        #if there is not yet at the last index
        else:
          while(idx != len(productions[i][j]) - 1):
            idx += 1
            if(not productions[i][j][idx].isupper()):
              follow.add(productions[i][j][idx])
              break
            #calculating first of the rightmost empty variable
            else:
              f = first_func(productions[i][j][idx], productions)

              #if we find a non terminal value
              if('#' not in f):
                for x in f:
                  follow.add(x)
                break
              #else if there is a epsilon
              elif('#' in f and idx != len(productions[i][j])-1):
                f.remove('#')
                for k in f:
                  follow.add(k)

              elif('#' in f and idx == len(productions[i][j])-1):
                f.remove('#')
                for k in f:
                  follow.add(k)
                #recursive function to the add the follows
                f = follow_func(i, productions, first)
                for x in f:
                  follow.add(x)

  return follow
```

call first function

```
for s in productions.keys():
  first_dic[s] = first_func(s, productions)
```

print first

```
print("FIRST ")
for lhs, rhs in first_dic.items():
  print(lhs, "=" , rhs)
print("")
```

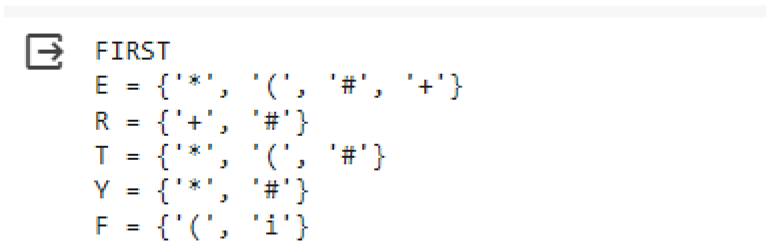
call Follow function

```
for lhs in productions:
  follow_dic[lhs] = set()

for s in productions.keys():
  follow_dic[s] = follow_func(s, productions, first_dic)
```

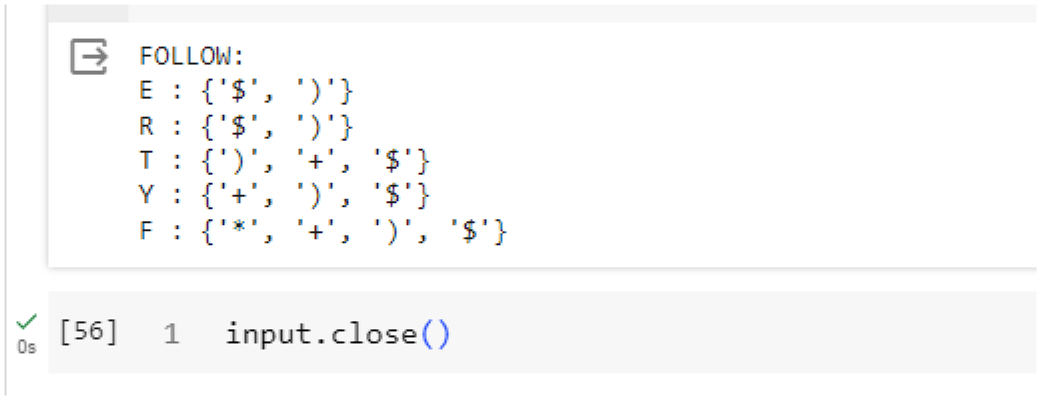
print Follow

```
print("FOLLOW:")
for lhs, rhs in follow_dic.items():
  print(lhs, ":" , rhs)

input.close()
```

Observed output:



```
FIRST
E = {'*', '(', '#', '+'}
R = {'+', '#'}
T = {'*', '(', '#'}
Y = {'*', '#'}
F = {'(', 'i'}
```



```
FOLLOW:
E : {'$', ')'}
R : {'$', ')'}
T : {')', '+', '$'}
Y : {'+', ')', '$'}
F : {'*', '+', ')', '$'}
```

```
[56]  input.close()
```