Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report #3

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 remove the left recursion from the given grammar.

Background

A grammar in the form G = (V, T, P, S) is said to have left recursion if it has the production rules of the form:

$$A \rightarrow A\,\alpha \mid \beta.$$

Here, the variable A in the left side occurs at the first position on the right side production. This is the condition for left recursion. Having a left recursion in a grammar leads to infinite recursion which can make it impossible for us to generate the given string. So, such left recursions should be removed.

Solution to the problem:

We can eliminate left recursion by replacing a pair of production with:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \epsilon$$

Here, A' is the newly added character symbol which was not present in the original grammar, so the productions of A' are also added to the grammar on removing the left recursion.

My attempt in implementing the program to remove left recursion in Python is discussed below. The program takes a grammar as input form the user, detects if there's the presence of left recursion and finally removes it and displays the updated grammar output to the user.

Source Code

```
# Lab 3 - Program to Remove Left Recursion from the given_grammar
# Sabin Thapa
# Roll no. 54
# CE 4th Year

given_grammar = []
def print_output(grammar):
  for i in grammar:
    statement = ''
    statement+=i[0]+'=>'
    for j in range(len(i[1])):
      if j:
        statement+='|'
      statement+=i[1][j]
    print(statement)
  print()
def take_grammar_input():
  given_grammar = []
  print('Note: \n 1. e represents epsilon. \n',
      '2. Represent in the form: A=>aB|cD|Aa \n',
      '3. Press enter after each production. \n')
   while(True):
    inp = input(f'Enter the Grammar: ')
    # Stop after user inputs nothing
    if inp == '':
      break
    given_grammar.append(inp)
  for i,grammar in enumerate(given_grammar):
    given_grammar[i]=grammar.split('=>')
    given_grammar[i][1]=given_grammar[i][1].split('|')
  i=0
  while i<len(given_grammar):
    j=0
    while j<len(given_grammar):
      if i!=j:
        if given_grammar[i][0] == given_grammar[j][0]:
          given_grammar[i][1].extend(given_grammar[j][1])
          given_grammar.pop(j)
      j+=1
```

```
        i+=1
    return given_grammar
def remove_left_recursion():
    i=0
    while i<len(given_grammar):
        r = []
        j = 0
        while j<len(given_grammar[i][1]):
            if len(given_grammar[i][0]) <= len(given_grammar[i][1][j]):
                if given_grammar[i][0] == given_grammar[i][1][j][:len(given_grammar[i][0])]:
                    if given_grammar[i][0] == given_grammar[i][1][j]:
                        given_grammar[i][1].pop(j)
                        j-=1
                    else:
                        r.append(given_grammar[i][1][j][len(given_grammar[i][0]):])
                        given_grammar[i][1].pop(j)
                        j-=1
            j+=1
        if r:
            for j in range(len(given_grammar[i][1])):
                given_grammar[i][1][j]+=given_grammar[i][0]+"'"
            if not given_grammar[i][1]:
                given_grammar[i][1].append(given_grammar[i][0]+"'")
            for j in range(len(r)):
                r[j]+=given_grammar[i][0]+"'"
            r.append('e')
            if (i+1)<len(given_grammar):
                given_grammar.insert(i+1,[given_grammar[i][0]+"'",r])
            else:
                given_grammar.append([given_grammar[i][0]+"'",r])
        i+=1
if __name__=='__main__':
    given_grammar = take_grammar_input()
    print('\nGiven Grammar:')
    print_output(given_grammar)

    remove_left_recursion()

    print('After removing Left Recursion:')
    print_output(given_grammar)
```

Explanation

When the user inputs a grammar that has a direct left recursion, the program checks for it and finds it and then it provides back the user with the grammar that has been freed from left recursion.

There are three functions with the following tasks:

1. **take_grammar_input()**

   This function takes the grammar as input from the user in the form:

   $$A => A\ \alpha \mid \beta$$

   The input is splitted into two parts from the => sign. The first character of both the parts is checked and if they match, there is a direct left recursion and is removed.

   Again the derivation of the right part is also splitted using the | symbol.

```
def take_grammar_input():
    given_grammar = []
    print('Note: \n 1. e represents epsilon. \n',
          '2. Represent in the form: A=>aB|cD|Aa \n',
          '3. Press enter after each production. \n')

    while(True):
        inp = input(f'Enter the Grammar: ')
        # Stop after user inputs nothing
        if inp == '':
            break
        given_grammar.append(inp)

    for i,grammar in enumerate(given_grammar):
        given_grammar[i]=grammar.split('=>')
        given_grammar[i][1]=given_grammar[i][1].split('|')
    i=0
    while i<len(given_grammar):
        j=0
        while j<len(given_grammar):
            if i!=j:
                if given_grammar[i][0] == given_grammar[j][0]:
                    given_grammar[i][1].extend(given_grammar[j][1])
                    given_grammar.pop(j)
            j+=1
        i+=1
    return given_grammar
```

2. **remove_left_recursion()**

   This function is the actual implementation to remove the left recursion from the given grammar. We modify the left recursive grammar by replacing it with a pair of production rules:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \epsilon$$

Here, ‘e’ as added as epsilon for the production of the added grammar rule A’.

```
def remove_left_recursion():
    i=0
    while i<len(given_grammar):
        r = []
        j = 0
        while j<len(given_grammar[i][1]):
            if len(given_grammar[i][0]) <= len(given_grammar[i][1][j]):
                if given_grammar[i][0] == given_grammar[i][1][j][:len(given_grammar[i][0])]:
                    if given_grammar[i][0] == given_grammar[i][1][j]:
                        given_grammar[i][1].pop(j)
                        j-=1
                    else:
                        r.append(given_grammar[i][1][j][len(given_grammar[i][0]):])
                        given_grammar[i][1].pop(j)
                        j-=1
            j+=1
        if r:
            for j in range(len(given_grammar[i][1])):
                given_grammar[i][1][j]+=given_grammar[i][0]+"'"
            if not given_grammar[i][1]:
                given_grammar[i][1].append(given_grammar[i][0]+"'")
            for j in range(len(r)):
                r[j]+=given_grammar[i][0]+"'"
            r.append('e')
            if (i+1)<len(given_grammar):
                given_grammar.insert(i+1,[given_grammar[i][0]+"'",r])
            else:
                given_grammar.append([given_grammar[i][0]+"'",r])
        i+=1
```

3. print_output()

This function displays the output after removing the left recursion to the user. The symbols ‘=>’ and ‘|’ are added accordingly.

```
def print_output(grammar):
    for i in grammar:
        statement = ''
        statement+=i[0]+'=>'
        for j in range(len(i[1])):
            if j:
                statement+='|'
            statement+=i[1][j]
        print(statement)
    print()
```

Output

```
sabinthapa pop-os ../Compiler Design/lab labs Compiler python3 lab3.py

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

Enter the Grammar: S=>A
Enter the Grammar: A=>aB|aC|Ad|Ae
Enter the Grammar: B=>bBc|f
Enter the Grammar: C=>g
Enter the Grammar:

Given Grammar:
S=>A
A=>aB|aC|Ad|Ae
B=>bBc|f
C=>g

After removing Left Recursion:
S=>A
A=>aBA'|aCA'
A'=>dA'|e A'|e
B=>bBc|f
C=>g
```

Here, the input grammar is left recursive, hence the program has detected it and returned a grammar that is free from left recursion.

```
sabinthapa pop-os ../Compiler Design/lab labs Compiler python3 lab3.py

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

Enter the Grammar: A=>AcB|cC|C
Enter the Grammar: B=>bB|id
Enter the Grammar: C=>CaB|BbB|B
Enter the Grammar:

Given Grammar:
A=>AcB|cC|C
B=>bB|id
C=>CaB|BbB|B

After removing Left Recursion:
A=>cCA'|CA'
A'=>cBA'|e
B=>bB|id
C=>BbBC'|BC'
C'=>aBC'|e
```

Similarly, in this case, the left recursion of the grammar has been successfully removed.

Conclusion

In this way, the program to remove left recursion from the given input grammar was implemented in Python. The source code and the outputs of the program are attached in the document above.