Karachi Institute of Economics and Technology
College of Computing & Information Sciences

Accept and Reject Arithmetic Expression by Using CFG

Prepared by

M.Talib Farooq – 10270

Shaista Sharif – 10589

Asma Ghafoor – 10305

For

Automata 108983 – Thursday

Spring 2022

Project Advisor: Mr. Aziz Farooqi

17-May-22

CONTENTS

Page

Abstract

A grammar can be used to describe the possible hierarchical structure of a program.

Context-free grammars are more expressive than finite automata: if a language L is accepted by finite automata then L can be generated by a context-free grammar.

A context free grammar has 4 components: – A set of tokens, known as terminal symbols. – A set of no terminals. – A set of productions where each production consists of a no terminal, called the left side of the production, an arrow, and a sequence of tokens and/or no terminals, called the right side of the production. – A designation of one of the no terminals as the start symbol.

1. Project Description

- A 4-tuple G = < V, S, P > is a context-free grammar (CFG) if V and are finite sets sharing no elements between them, S V is the start symbol, and P is a finite set of productions of the form X ->, where X V, and (V)*.
- T describes a finite set of terminal symbols. S is the start symbol. In CFG, the start symbol is used to derive the string. You can derive the string by repeatedly replacing a non-terminal by the right-hand side of the production until all non-terminals have been replaced by terminal symbols.
- A context-free grammar is studied in the fields of theoretical computer science, compiler design, and linguistics. CFGs are used to describe programming languages and parser programs in compilers can be generated automatically from context-free grammar.

2. *3rd* party Libraries and purpose which you used in your project

- We used spilt function from library **re** also known as regular expression.
- Earlier we also used **rnd** as Random for generation of an arithmetic expression.

3. Project Meetings with your teacher (goals + achievements)

- First, we examined the problem, and how we can check the expression is valid or invalid by using CFG rules
- We understand this by looking into the libraries e.g., NLTK that can solve the same problem
- We also added **regex** which check the expression validity through regular expression but it under the scope of this project.
- We made use of python data type dictionary to define the rules for an arithmetic expression.

- The rules made by CFG method used to check for the input expression and returns if an expression is valid or invalid

4. Stack info = (Hardware & Software Requirement for your project)

- System should have python installed and should be run on Jupyter Notebook
- Modern Operating System:

  x86 64-bit CPU (Intel / AMD architecture)

  4 GB RAM.

  5 GB free disk space

5. Methodology (explanation of complex code logic or complex variable structure)

- Rules is a python dictionary
- Spilt is used by regex library **re**
- Used indexing for list to split a num into different variables
- Open and closing are validated by data structure stack (push and pop)
- Conversion is also used in this program for associability of python features.

6. Project Code

```
rules = {
   "S":[
      ["Exp"]
      ],
      "Exp":[
            ["BO","N","O","N","BC"]
            ],
      "BO":[
          ['(']
          ],
      "N":[
         '0','1','2','3','4','5','6','7','8','9',['N','N']
         ],
      "O": [
          ['+'],['-'],['*'],['/']
          ],
      "BC":[
          [')']
          ],
}
#(4+5) (12+10) (1+22) (55+698)
#final = input("Please enter an expression")
final = "(4+95)"
result = re.split(r"\+|\-|\*|\/|\(|\)", final)
#print(result)

num1 = result[1]
num2 = result[2]

# print(num1)
# print(num2)
```

```
def is_valid(myStr):

    opening = ['(']
    closing = [')']
    stack = []
    for i in myStr:
        if i in opening:
            stack.append(i)
        elif i in closing:
            pos = closing.index(i)
            if ((len(stack) > 0) and
                    (opening[pos] == stack[-1])):
                stack.pop()
            else:
                stack.append(i)
        else:
            pass
    if len(stack) == 0:
        return True
    else:
        return False

bracketsAccpet = is_valid(final)
# print(bracketsAccpet, "B")

num1Accept= False

if((len(num1)>=2)):
    x = [int(a) for a in str(num1)]
    print(x)
    count_num1=0
    for i in range(10):
        for j in range(len(x)):
            #print(rules['N'][i])
            #print('1'==rules['N'][i])
            if((str(x[j])==rules['N'][i])):
                count_num1=count_num1+1
                #print(count_num1)
                if(count_num1>=2):
                    num1Accept = True
```

```
else:
    for i in range(10):
        if(str(num1)==rules['N'][i]):
            num1Accept = True

#print(num1Accept)
num2Accept= False

if((len(num2)>=2)):
    y = [int(b) for b in str(num2)]
    #print(y)
    count_num2=0
    for k in range(10):
        for l in range(len(y)):
            #print(rules['N'][i])
            #print('1'==rules['N'][i])
            if((str(y[l])==rules['N'][k])):
                count_num2=count_num2+1
                #print(count_num2)
                if(count_num2>=2):
                    num2Accept = True

else:
    for i in range(10):
        if(str(num2)==rules['N'][i]):
            num2Accept = True

#print(num2Accept)

if(num1Accept and num2Accept and bracketsAccpet):
    print("Expression is Accpeted")
else:
    print("Expression is Rejected")
```

7. Three Test cases input + output

```
else:
    for i in range(10):
        if(str(num2)==rules['N'][i]):
            num2Accept = True

#print(num2Accept)

if(num1Accept and num2Accept and bracketsAccpet):
    print("Expression is Accpeted")
else:
    print("Expression is Rejected")
```

```
Please enter an expression(14+8)
[1, 4]
Expression is Accpeted
```

```
                count_num2=count_num2+1
                #print(count_num2)
                if(count_num2>=2):
                    num2Accept = True

else:
    for i in range(10):
        if(str(num2)==rules['N'][i]):
            num2Accept = True

#print(num2Accept)

if(num1Accept and num2Accept and bracketsAccpet):
    print("Expression is Accpeted")
else:
    print("Expression is Rejected")
```

```
Please enter an expression(4-+6)
Expression is Rejected
```

```
                if(count_num2>=2):
                    num2Accept = True

else:
    for i in range(10):
        if(str(num2)==rules['N'][i]):
            num2Accept = True

if(num1Accept and num2Accept and bracketsAccpet):
    print("Expression is Accpeted")
else:
    print("Expression is Rejected")
```

```
Please enter an expression(10/2)
[1, 0]
Expression is Accpeted
```

8. Conclusion

In future, we want to improve it by adding filing and making it more dynamic by adding UI to it for easy use.

3 → 2)

(4 + (3 − 2))

S → (Exp1)

Exp1 → NO Exp2

Exp2 → Exp3

Exp3 → NON

N → [0 ---- 9]

O → [+ − / *]

((5+4) − (3−1))

S → Exp1

Exp1 → Exp2 O Exp3

Exp2 → Exp4

Exp4 → NON

Exp3 → Exp5

Exp5 → NON

N → [0 ---- 9]

O → [+ − /*]

((3+1) / (1+4) − 3(5*3))

S → Exp1

Exp1 → Exp2 O Exp3 ON Exp4

Exp2 → Exp5

Exp5 → NON

Exp3 → Exp6

Exp6 → NON

Exp4 → Exp7

Exp7 → NON

N → [0 --- 9]

O → [+ − / *]

* ((2 * 6 − 1) * (2 − 1))

Exp1

S → Exp1

Exp1 → Exp2 O Exp3

Exp2 → Exp4

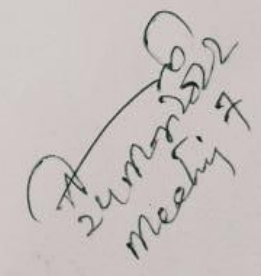
Exp4 → NONON

Exp3 → Exp5

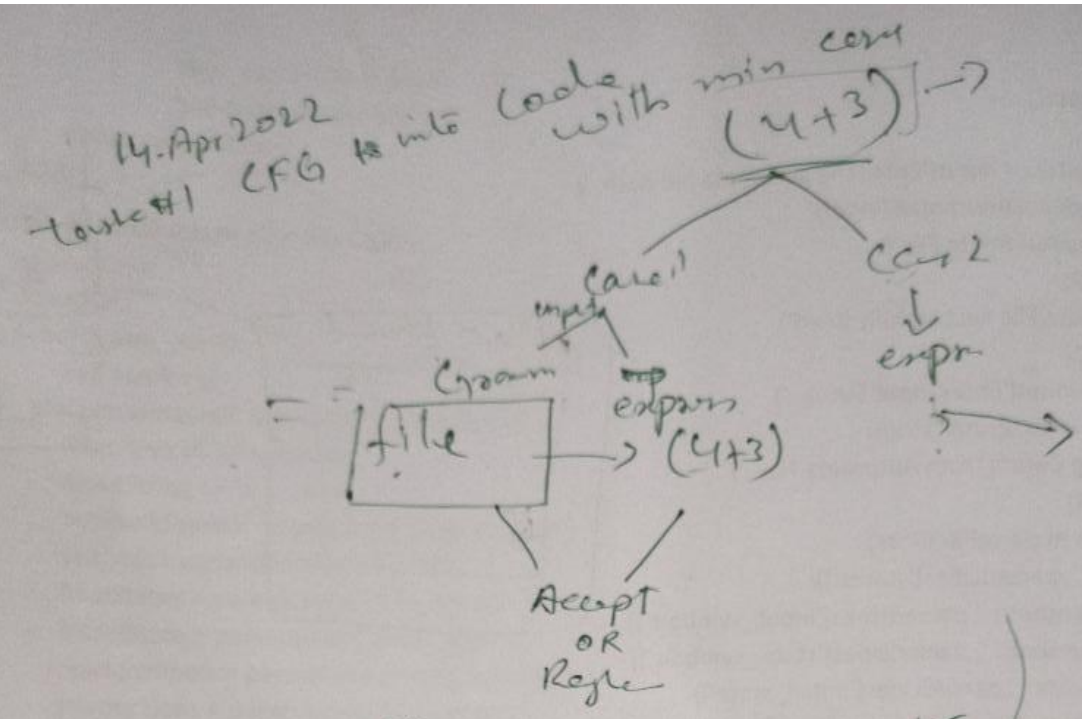
Exp5 → NON

N → [0 ---- 9]

O → [+ − /*].



14-Apr 2022
Task #1 CFG to into code with min cost



Task #2 & #3 CFG
PDA ?
CNF ?

$S \rightarrow (Exp_1)$

$Exp_1 \rightarrow NON$

$N \rightarrow [0 \cdots 9]$

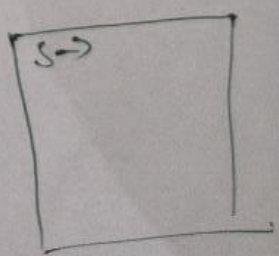
$O \rightarrow [+ - / \div]$

14 Apr 2022
Meeting #9

(4+3)

Task #2 Standard Example of CFG & code using Python

Task #1 Validate input (Expression) using our own code



(4+2−1)

(✓) × ↓

[(4+2)] (++) (22+21)

split

S → Exp → [BO−N−O−N−BC]

S → (4 + −

A A A R

Len 2

1 – Operator one only

2 – Number can be uncountable

(4+3)

S → Exp

Exp → (BO, N, O, N, BC)

BO → (

N → (0 — 9), (N,N)

O → (+, −, /, *)

BC →)

((

*Task #2 of 3 (4+2)

((4+3)-2)

$S \rightarrow (Exp_1)$

$Exp_1 \rightarrow Exp_2 OP$ ~~CFF~~ CFG → CNF

$Exp_2 \rightarrow (Exp_3)$

$Exp_3 \rightarrow NON$

$N \rightarrow [0 - \cdots - 9]$

$O \rightarrow [+ - / *]$

CFG → code

Task # 3 of 3

using Lib

nltk . CFG