|  |  |
| --- | --- |
|  | TECHNICAL UNIVERSITY OF MOLDOVA  Faculty of Computers, Informatics and Microelectronics  Department: Software Engineering and Automation  Study Program: Software Engineering |

**Formal Languages and Compiler Design**

**Laboratory Work Nr. 1**

**Regular Grammar to Finite Automaton**

**Executed by: Doni Daniela**

**Group: FAF 193**

**Verified by: Vdovicenco Alexandr**

**Content: Laboratory tasks**

1. For the formal grammar G=(VN, VT, P, S) need be obtained five strings, that belong to the language L(G), that is generated by this grammar. The length of strings must be no lesser than the number of characters from the alphabet VN+2.

2. For each string build the noninverted (derivation) tree and derivation table.

3.Determine the grammar type by the Chromsky classification.

4. Write a program which converts regular grammar to Finite Automaton (FA):

4.1 Convert regular grammar to Finite Automaton (FA).

4.2. Using Finite Automaton (FA) check is some string is accepted by FA (meaning you could generate that string by traversing FA)

4.3.BONUS: Using some graphic library plot FA graph

**Variant 10.**

VN = {S, B, L},

VT = {a, b, c } ,

P= {

1. S → aB

2. B → bB

3. B → cL

4. L → cL

5. L → aS

6. L → b }

**The Link for GitHub account:**

https://github.com/DoniDaniela/LFPC\_laboratories.git

**1. For the formal grammar G=(VN, VT, P, S) need be obtained five strings, that belong to the language L(G), that is generated by this grammar. The length of strings must be no lesser than the number of characters from the alphabet VN+2.**

Length of strings:

|w| ≥ |VN| + 2 then, |w| ≥ 3+2 = 5

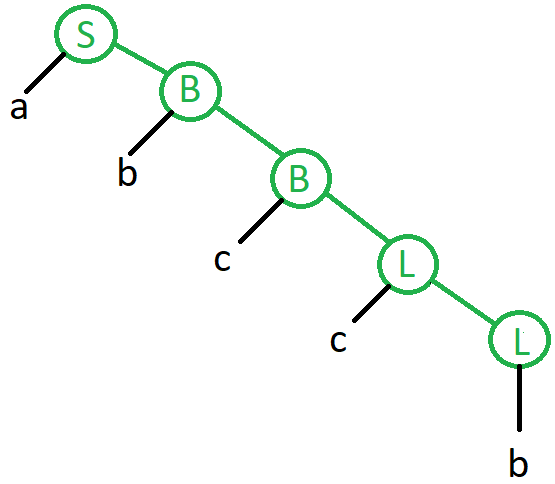
Five strings that belong to language L(G):

{abccb, abcca, abbca, acaab, accaa}

**2. For each string build the noninverted (derivation) tree and derivation table.**

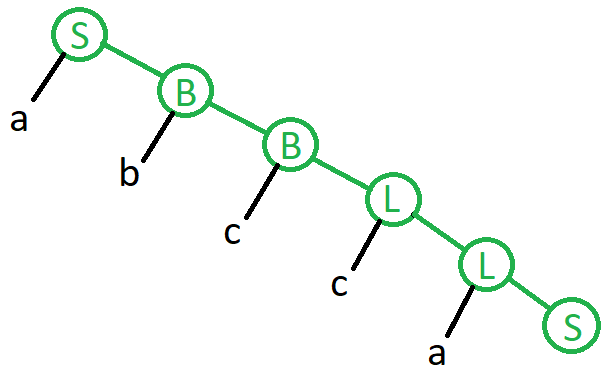
**String 1**: abccb

S->aB->abB->abcL->abccL->**abccb**

****

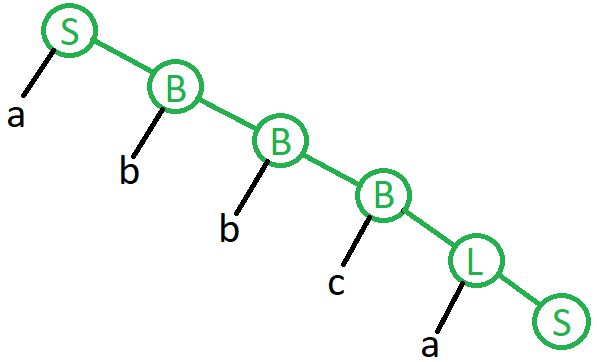
**String 2**: abcca

S->aB->abB->abcL->abccL->abccaS->**abcca**



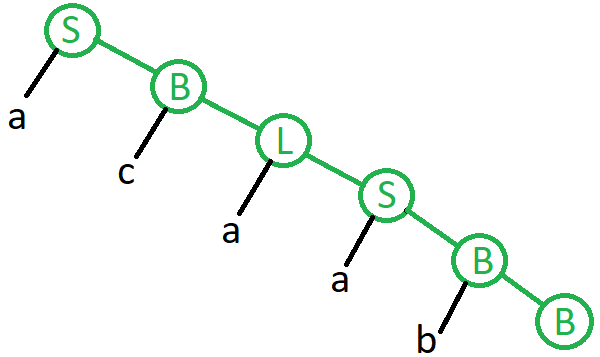
**String 3**: abbca

S->aB->abB->abbB->abbcL->abbcaS->**abbca**



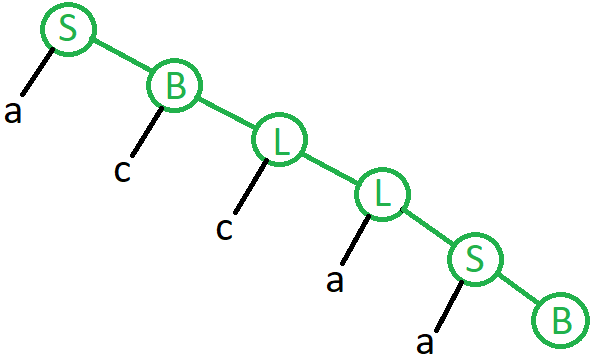
**String 4**: acaab

S->aB->acL->acaS->acaaB->acaabB->**acaab**



**String 5**: accaa

S->aB->acL->accL->accaS->accaaB->**accaa**



**3.Determine the grammar type by the Chromsky classification.**

Type 3- Regular grammar (least powerful)

Automaton- Finite State Automaton

**4. Write a program which converts regular grammar to Finite Automaton (FA):**

**4.1 Convert regular grammar to Finite Automaton (FA).**

**4.2. Using Finite Automaton (FA) check is some string is accepted by FA (meaning you could generate that string by traversing FA)**

**4.3.BONUS: Using some graphic library plot FA graph**

#include<iostream>

#include<string>

using namespace std;

int main(int argc, char const \*argv[]) {

char cur\_state, letter;

string sentence;

while (true) {

HERE:

cur\_state = 'S';

cout << "\n\nPunct initial: " << cur\_state << "\n\nIntrodu expresia: ";

cin >> sentence;

cout << "\nPath:\n";

for (char i : sentence) {

letter = i;

switch (cur\_state) {

case 'S':

if(letter == 'a') {

cur\_state = 'B';

break;

} else {

cout << "\nExpresia nu apartine productiilor";

goto HERE;

}

case 'B':

if (letter == 'b') {

cur\_state = 'B';

break;

} else if(letter == 'c') {

cur\_state = 'L';

break;

} else {

cout << "\nExpresia nu apartine productiilor";

goto HERE;

}

case 'L':

if (letter == 'b') {

cout << "terminal point reached" << "\nExpresia e corecta";

goto HERE;

}else if(letter == 'c') {

cur\_state = 'L';

break;

}else if(letter == 'a') {

cur\_state = 'S';

break;

} else {

cout << "\nExpresia nu apartine productiilor";

goto HERE;

}

}

printf("%c\n", cur\_state);

}

cout << "\nDestinatia finala - " << cur\_state << "\nExpresia e corecta";

}

}