Ministry of Education, Culture and Research of Moldova
Technical University of Moldova
Faculty of Computers, Informatics and Microelectronics
Software Engineering and Automation Department

Laboratory Work
Formal Languages and Compiler Design

Student :Wu Xenia
Group: FAF-203

Professor: Cojuhari Irina
Vdovicenco Alexandr

Chisinau, 2021

Variant 24

Laboratory tasks:

1. For the formal grammar $G = (V_N, V_T, P, S)$ need to obtain 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 $V_N + 2$.
2. For each string build the non-inverted (derivation) tree and derivation table.
3. Convert regular grammar to Finite Automaton (FA).
4. Determine the grammar type by the Chomsky classification.
5. Write a program which converts regular grammar to Finite Automaton (FA).
6. Using Finite Automaton (FA) check if some input string is accepted by FA (meaning you could generate that string by traversing FA)
7. Bonus Point -> Using some graphic library plot FA graph

The given grammar:

$V_N = \{S, A, C, D\}$, $V_T = \{a, b\}$,

$P = \{$ 1. $S -> aA$;

2. $A -> bS$;
3. $A -> dD$;
4. $D -> bC$;
5. $C -> a$;
6. $C -> bA$;
7. $D -> aD.\}$

Ex.1

$S -> aA -> abS -> abaA -> abadD -> abadbC -> abadba$;

$S -> aA -> adD -> adbC -> adbbA -> adbbdD -> adbbdbC -> adbbdba$;

$S -> aA -> adD -> adaD -> adaaD -> adaabC -> adaaba$;

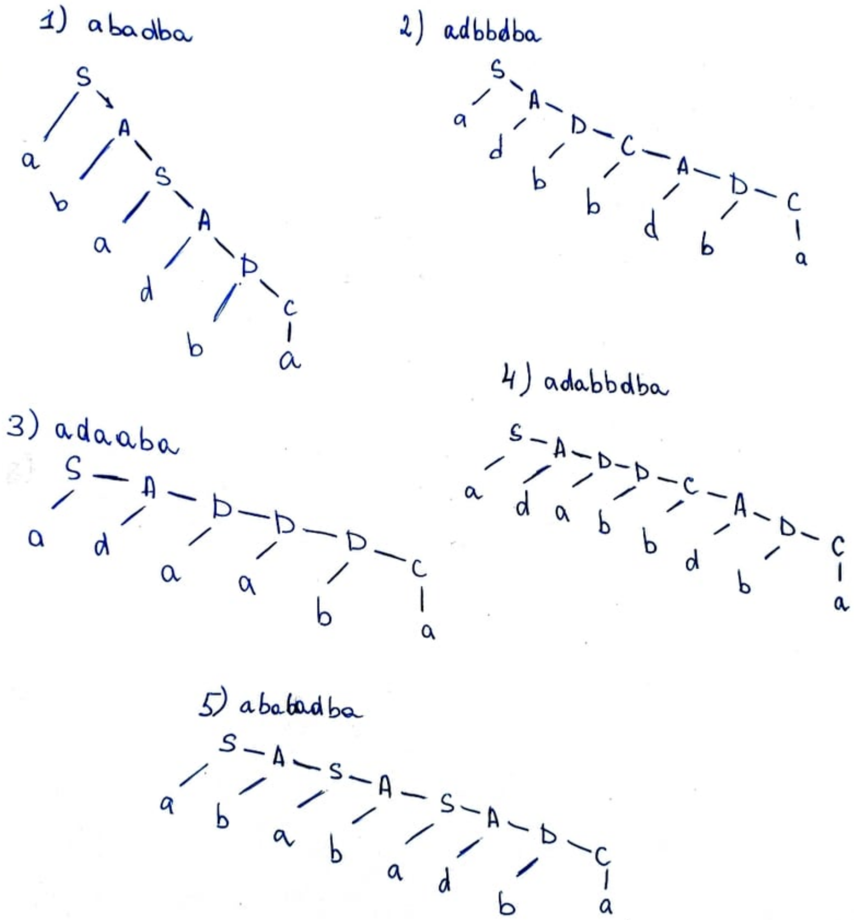
$S -> aA -> adD -> adaD -> adabC -> adabbA -> adabbdD -> adabbdbC -> adabbdba$;

$S -> aA -> abS -> abaA -> ababS -> ababaA -> ababadD -> ababadbC -> ababadba$.

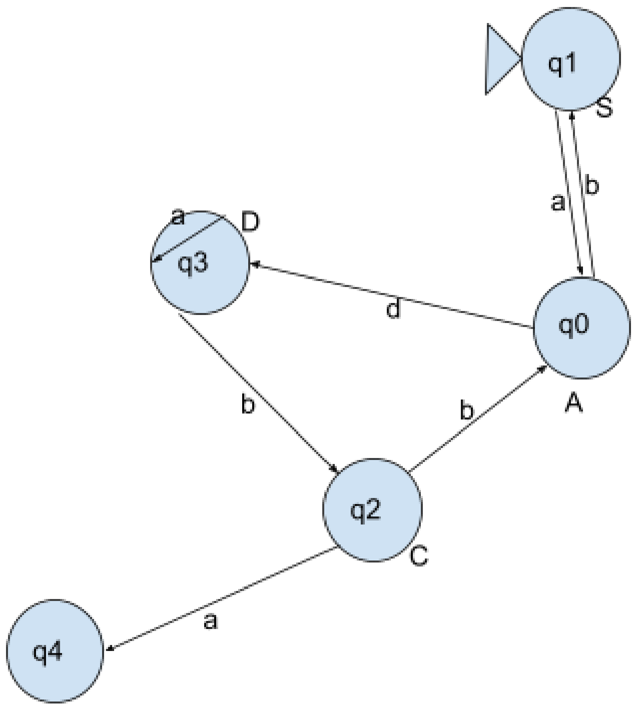
Ex.2

Production	Derivation
	S
S-> aA	aA
A->bS	abS
S->aA	abaA
A->dD	abadD
D->bC	abadbC
C->a	abadba
	S
S->aA	aA
A->dD	adD
D->bC	adbC
C->bA	adbbA
A->dD	adbbdD
D->bC	adbbdbC
C->a	adbbdba
	S
S->aA	aA
A->dD	adD
D->aD	adaD
D->aD	adaaD
D->bC	adaabC
C->a	adaaba

Production	Derivation
	S
S->aS	aA
A->dD	adD
D->aD	adaD
D->bC	adabC
C->bA	adabbA
A->dD	adabbdD
D->bC	adabbdbC
C->a	adabbdba
	S
S->aA	aA
A->bS	abS
S->aA	abaA
A->bS	ababS
S->aA	ababaA
A->dD	ababadD
D->bC	ababadbC
C->a	ababadba


1) abadba
2) adbbdba
3) adaaba
4) adabbdba
5) ababadba

Ex.3


q1
S
a
b
q3
D
d
q0
A
b
b
q2
C
a
q4

Ex.4

The type of the grammar by the Chomsky classification: Type 3, regular grammar.

Ex.5 -7

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
import pylab

#Ex. 1
# Convert RG to FA
vn_brut = input("Defineste elementele Vn (separandu-le prin
spatiu)\n")
vn = vn_brut.split(" ")
vn.append(".")
print("VN = {}".format(vn))

nr_rel = int(input ("Care este numarul de relatii?\n"))
matrice_graph = [[0 for x in range(len(vn))] for y in
range(len(vn))];

for k in range(nr_rel) :
    i=0
    j=0
    vertex_start, weight, vertex_terminal = input().split(" ")
    for n in range(len(vn)) :
        if (vn[n] == vertex_start ) : i=n
        if (vn[n] == vertex_terminal) : j=n
    matrice_graph[i][j] = weight
    print(matrice_graph)
#Ex. 2
#Check if there could exist such word in FA
cuvant = input ("Introdu cuvantul spre verificare:\n")
check_var = True
lungime_cuvant = len(cuvant)
h=0
valori_finale = [0 for i in range(len(vn))]
j = len(vn)-1
for i in range(j):
    if (matrice_graph[i][j]!='0') :
        valori_finale[h] = matrice_graph[i][j]
        h=h+1

for k in range(h):
    if (cuvant[lungime_cuvant-1] != valori_finale[k]):
```

```
        check_var = False
    else: check_var= True; break;

i=0
for k in range(lungime_cuvant):
    ch=cuvant[k]
    j=0
    if (check_var == True):
        if (matrice_graph[i][j]==ch):
            i=j
            j=0
            check_var = True
    else:
        if (j==len(vn)-1):
            check_var = False
        else: j+=1

if (check_var):
    print("Cuvantul corespunde FA")
else:
    print("Cuvantul nu corespunde FA")

#Ex. 3
#Plot the FA graph
G = nx.DiGraph()

G.add_edges_from([('q1\n{}'.format(vn[0]),
'q0\n{}'.format(vn[1])),('q2\n{}'.format(vn[2]),'q4\n{}'.format("X
")),('q3\n{}'.format(vn[3]),'q3\n{}'.format(vn[3]))], label="a")
G.add_edges_from([('q0\n{}'.format(vn[1]),'q1\n{}'.format(vn[0])),
('q2\n{}'.format(vn[2]),'q0\n{}'.format(vn[1])),('q3\n{}'.format(v
n[3]),'q2\n{}'.format(vn[2]))], label="b")
G.add_edges_from([('q0\n{}'.format(vn[1]),'q3\n{}'.format(vn[3]))]
, label="d")

edge_labels=dict([((u,v,),d['label'])
                for u,v,d in G.edges(data=True)])

node_labels = {node:node for node in G.nodes()}
edge_colors = ['black']
node_colors = ['#E2CDF1']
pos=nx.spring_layout(G)
nx.draw_networkx_labels(G, pos, labels=node_labels)
nx.draw_networkx_edge_labels(G,pos,edge_labels=edge_labels)
nx.draw(G,pos, node_color = node_colors,
node_size=1500,edge_color=edge_colors,edge_cmap=plt.cm.Reds)
pylab.show()
```