THEORY OF COMPUTATION

BATCH-18;

MN KARTHIK-231FA04920

G SAI MANINDRA-231FA04A17

KVS ADITYA-231FA04936

S SAHITHI-231FA04951

QUESTION:

consider the following languages over the alphabet{0,1}

a) the set of all strings ends with 00.

b) with three consecutive 0"s.

c) with 011 as a substring.

d) either begin or ends with 01.

design DFA and NFA for the above.

covert NFA to DFA for the obtained NFA given in question C.

write regular expressions for the above automata

ABSTRACT:

Finite Automata are fundamental models of computation used to recognize patterns within input strings over a given alphabet. This paper discusses the design and implementation of Deterministic Finite Automata (DFA) and Nondeterministic Finite Automata (NFA) for specific languages over the binary alphabet {0,1}. Additionally, it covers the conversion of an NFA to its equivalent DFA and derives regular expressions corresponding to the given languages.

INTRODUCTION:

Finite Automata serve as essential tools in the fields of theoretical computer science and formal language processing. DFAs and NFAs are widely utilized in lexical analysis, text searching, and pattern recognition. This study focuses on designing automata for specific language constraints and converting an NFA into a DFA using the subset construction method. Furthermore, it provides corresponding regular expressions to describe these languages concisely.

PROBLEM STATEMENT:

Given the alphabet {0,1}, design both DFA and NFA for the following languages:

1. The set of all strings that end with "00".
2. The set of all strings containing three consecutive "0"s.
3. The set of all strings containing "011" as a substring.
4. The set of all strings that either begin or end with "01".

Additionally, convert the NFA obtained in (c) into an equivalent DFA and provide the corresponding regular expressions for all the defined languages

**a)The set of all strings that end with "00".   
CODE:  
NFA:**

from graphviz import Digraph  
from IPython.display import display  
def construct\_e\_nfa():  
 dot = Digraph()  
 dot.attr(rankdir='LR', size='8')  
 states = ['q0', 'q1', 'q2', ‘q2]  
for state in states:  
 shape = 'doublecircle' if state == 'q2' else 'circle’   
fillcolor = 'lightblue' if state == 'q0' else 'lightgreen' dot.node(state, state, shape=shape, style='filled', fillcolor=fillcolor)   
 dot.edge('q0', 'q0', label='0’)   
 dot.edge('q0', ‘q0', label='1’)   
 dot.edge(‘q0', ‘q1', label=‘0')  
 dot.edge('q1', 'q2', label='0’)   
 display(dot)  
construct\_e\_nfa()

OUTPUT:

A diagram of a green circle with black arrows

AI-generated content may be incorrect.

**DFA:**

from graphviz import Digraph

from IPython.display import display

def construct\_dfa():

dot = Digraph()

dot.attr(rankdir='LR', size='8')

states = ['q0', 'q1', 'q2’]

for state in states:

shape = 'doublecircle' if state == 'q2' else 'circle' fillcolor = ‘l ‘ “lightblue’ dot.node(state, state, shape=shape, style='filled', fillcolor=fillcolor)

dot.edge('q0', 'q1', label='0')

dot.edge('q0', 'q0', label='1')

dot.edge('q1', 'q2', label='0')

dot.edge('q1', 'q0', label='1')

dot.edge('q2', 'q2', label=‘0 )

dot.edge('q2', 'q0', label='1')

display(dot)

construct\_dfa()

OUTPUT:

A diagram of a diagram

AI-generated content may be incorrect.

**b)The set of all strings containing three consecutive "0"s  
CODE:  
NFA.**

from graphviz import Digraph  
from IPython.display import display  
def construct\_e\_nfa():  
 dot = Digraph()  
 dot.attr(rankdir='LR', size='8')  
 states = ['q0', 'q1', 'q2', 'q3']  
 for state in states:  
 shape = 'doublecircle' if state == 'q3' else 'circle' # q3 is the accepting state  
 fillcolor = 'lightblue' if state == 'q0' else 'lightgreen' # Color choices  
 dot.node(state, state, shape=shape, style='filled', fillcolor=fillcolor   
 dot.edge('q0', 'q0', label='1’)   
 dot.edge('q0', ‘q0', label='0')  
 dot.edge(‘q0', ‘q1', label='0’)   
 dot.edge(‘q1', 'q2', label=‘0’)   
 dot.edge('q2', 'q3', label='0')   
 dot.edge('q3', 'q3', label='0’)   
 dot.edge('q3', 'q3', label='1')  
 display(dot)  
 construct\_e\_nfa()

OUTPUT:

A diagram of a number of circles

AI-generated content may be incorrect.

**DFA**:

from graphviz import Digraph

from IPython.display import display

def construct\_dfa():

dot = Digraph()

dot.attr(rankdir='LR', size='8')

states = ['q0', 'q1', 'q2', 'q3']

for state in states:

shape = 'doublecircle' if state == 'q3' else 'circle'

fillcolor = 'lightblue' if state == 'q0' else 'lightgreen'

dot.node(state, state, shape=shape, style='filled', fillcolor=fillcolor)

dot.edge('q0', 'q0', label='1’)

dot.edge('q0', 'q1', label='0')

dot.edge('q1', 'q0', label='1’)

dot.edge('q1', 'q2', label='0’)

dot.edge('q2', 'q0', label='1')

dot.edge('q2', 'q3', label='0')

dot.edge('q3', 'q3', label='0’)

dot.edge('q3', 'q3', label='1’)

display(dot)

construct\_dfa()

OUTPUT:

A diagram of a diagram

AI-generated content may be incorrect.

**c)The set of all strings containing "011" as a substring.  
CODE:  
NFA:**

from graphviz import Digraph  
from IPython.display import display  
def construct\_nfa():  
 dot = Digraph()  
 dot.attr(rankdir='LR', size='8')  
 states = ['q0', 'q1', 'q2', 'q3']  
 for state in states:  
 shape = 'doublecircle' if state == 'q3' else 'circle’   
 fillcolor = 'lightblue' if state == 'q0' else 'lightgreen'  
 dot.node(state, state, shape=shape, style='filled', fillcolor=fillcolor)  
 dot.edge('q0', ‘q0', label='0')   
 dot.edge(‘q0’, ‘q0', label='0’)   
 dot.edge(‘q0', ‘q1', label=‘0')   
 dot.edge(‘q1', ‘q2', label='1')   
 dot.edge(‘q2', 'q3', label=‘1’)   
 dot.edge('q3', 'q3', label='1’)   
 dot.edge('q3', 'q3', label=‘0’)  
display(dot)  
construct\_nfa()

OUTPUT:

A diagram of a diagram

AI-generated content may be incorrect.

**DFA:**

from graphviz import Digraph

from IPython.display import display

def construct\_dfa():

dot = Digraph()

dot.attr(rankdir='LR', size='8')

states = ['q0', 'q1', 'q2', 'q3’]

for state in states:

shape = 'doublecircle' if state == 'q3' else 'circle’

fillcolor = 'lightblue' if state == 'q0' else 'lightgreen dot.node(state, state, shape=shape, style='filled', fillcolor=fillcolor)

dot.edge('q0', 'q0', label='1’)

dot.edge('q0', 'q1', label='0’)

dot.edge('q1', 'q2', label='1')

dot.edge('q1', 'q0', label='0')

dot.edge('q2', 'q3', label='1’)

dot.edge('q2', 'q0', label='0')

dot.edge('q3', 'q3', label='0’)

dot.edge('q3', 'q3', label='1')

display(dot)

construct\_dfa()

OUTPUT:

A diagram of a diagram

AI-generated content may be incorrect.

**d)The set of all strings that either begin or end with "01".  
CODE:  
NFA:**from graphviz import Digraph  
from IPython.display import display  
def construct\_nfa():  
 dot = Digraph()  
 dot.attr(rankdir='LR', size='8')  
 states = ['q0', 'q1', ‘q2’]  
 for state in states:  
 shape = 'doublecircle' if state in ['q1', 'q2'] else 'circle'  
 fillcolor = 'lightblue' if state == 'q0' else 'lightgreen'  
 dot.node(state, state, shape=shape, style='filled', fillcolor=fillcolor)   
 dot.edge('q0', 'q1', label='0’)   
 dot.edge('q1', ‘q2', label=‘0’)   
 dot.edge('q2', 'q2', label='1')  
 dot.edge('q2', 'q2', label='0')  
 display(dot)  
 construct\_nfa()

OUTPUT:

A diagram of a circle with arrows and a green circle

AI-generated content may be incorrect.

**DFA**:

from graphviz import Digraph

from IPython.display import display

def construct\_dfa():

dot = Digraph()

dot.attr(rankdir='LR', size='8’)

states = ['q0', 'q1', 'q2']

for state in states:

shape = 'doublecircle' if state in ['q1', 'q2'] else 'circle'

fillcolor = 'lightblue' if state == 'q0' else 'lightgreen'

dot.node(state, state, shape=shape, style='filled', fillcolor=fillcolor)

dot.edge('q0', 'q1', label='0')

dot.edge('q0', 'q0', label='1’)

dot.edge('q1', 'q2', label='1’)

dot.edge('q1', ‘q0', label='0')

dot.edge('q2', 'q2', label='1')

dot.edge('q2', 'q2', label='0')

display(dot)

construct\_dfa()

OUTPUT:

A diagram of a number one

AI-generated content may be incorrect.

CONCLUSION:

In this study, we designed DFA and NFA for various string constraints over the binary alphabet. We demonstrated the conversion of an NFA to DFA and provided corresponding regular expressions. This work highlights the significance of automata in formal language processing, pattern matching, and compiler design. Additionally, the provided source code offers a practical implementation of the discussed automata.

