Theory of Computation

Batch: 02

Section: 2M

SHAIK RIYAZ  
VIGNAN UNIVERSITY  
231FA04885

SINDHU  
VIGNAN UNIVERSITY  
231FA04G93

RAVIKIRAN  
VIGNAN UNIVERSITY  
231FA04A08

GOWTHAM  
VIGNAN UNIVERSITY  
231FA04912

***Abstract***— This document explores the formal approach to regular expressions and finite automata. We investigate the expression by analyzing the possible set of strings generated by it, constructing an ε-NFA (epsilon Non-Deterministic Finite Automaton), and then converting it into a deterministic finite automaton (DFA). This document offers a comprehensive step-by-step breakdown of the theoretical and practical aspects of converting regular expressions to finite automata.

**INTRODUCTION**:

This document explores the formal approach to regular expressions and finite automata. We investigate the expression by analyzing the possible set of strings generated by it, constructing an ε-NFA (epsilon Non-Deterministic Finite Automaton), and then converting it into a deterministic finite automaton (DFA). This document offers a comprehensive step-by-step breakdown of the theoretical and practical aspects of converting regular expressions to finite automata.

**DESCRIPTION:**

The problem statement involves analyzing the given regular expression and understanding the behavior of finite automata models. First, the set of possible strings that match the expression was identified, highlighting patterns in valid string sequences. Next, an ε-NFA was constructed by creating states and transitions to accommodate each element of the expression, including loops for repetitions and final accepting states. Lastly, the conversion of the ε-NFA to DFA was completed by computing epsilon closures, defining states, transitions, and marking accepting states. This comprehensive approach demonstrates how formal language concepts can be applied to model and validate string patterns effectively.

**PROBLEM STATEMENT :**

Consider the expression 𝑏(𝑎 + 𝑏)∗ab

A. Write set of strings that can be obtained by the above expression

B. Construct e-NFA for the expression.

C. Convert the NFA from the above.

**CODE :**

**A. Write set of strings that can be obtained by the above expression**

n=input("Enter a string : ").lower()

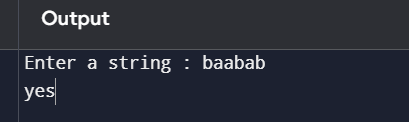
if n[0]=='b' and n[-2]=='a' and n[-1]=='b':

print("yes")

else:

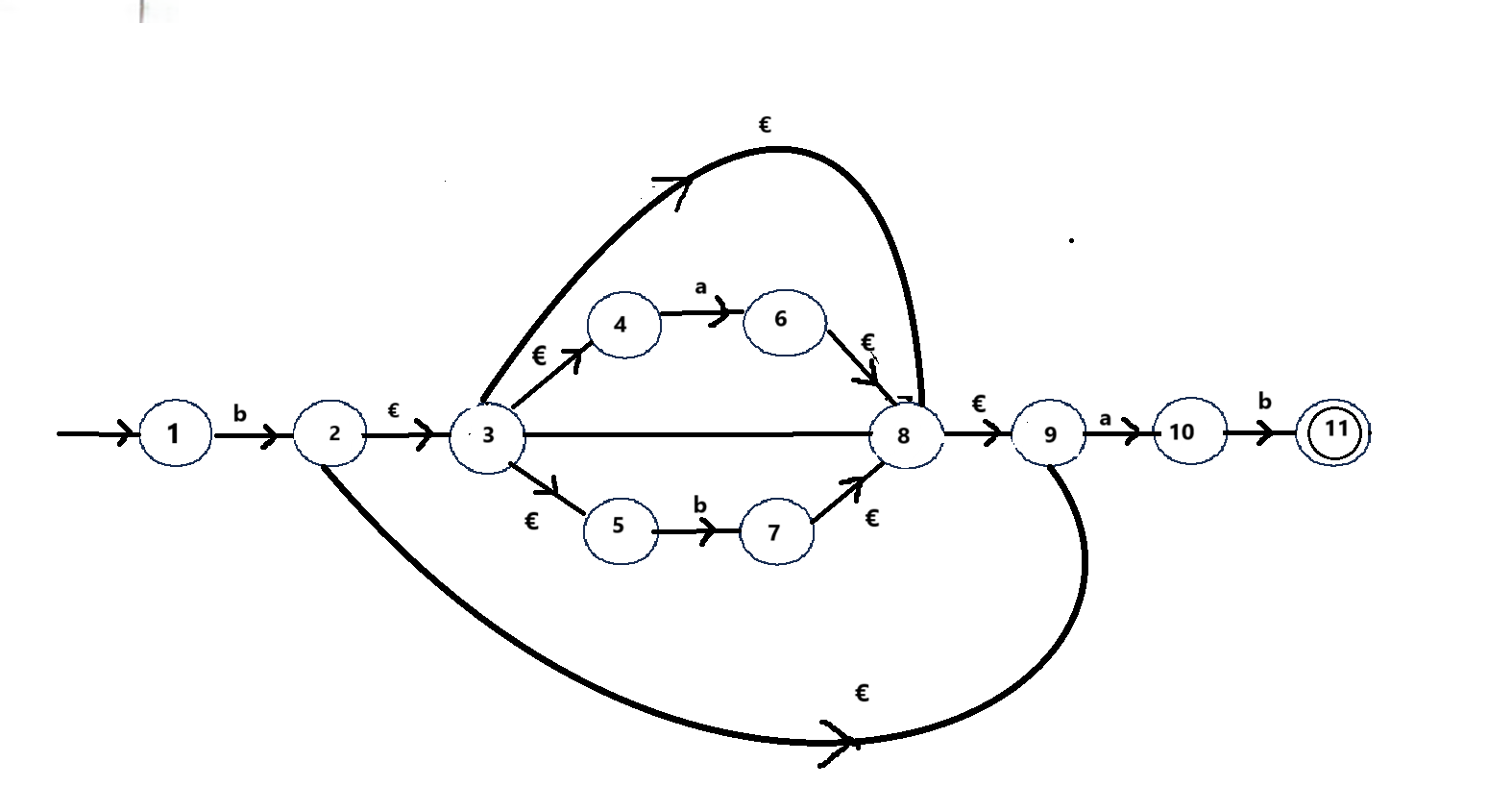
print("No")

**Output:**



**Construct e-NFA for the expression.**

* **Baabab**



**Code:**

**from graphviz import Digraph**

**from IPython.display import display**

**def construct\_e\_nfa():**

**dot = Digraph()**

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

**states = [str(i) for i in range(1, 12)]**

**for state in states:**

**shape = 'doublecircle' if state == '11' else 'circle'**

**fillcolor = 'lightblue' if state == '1' else 'lightgreen’**

**if state == '11' else 'white'**

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

**transitions = [**

**('1', '2', 'b'), ('2', '3', 'ε'), ('3', '4', 'ε'), ('3', '5', 'ε'), ('3', '8', 'ε'),**

**('4', '6', 'a'), ('5', '7', 'b'), ('6', '8', 'ε'), ('7', '8', 'ε'), ('8', '9', 'ε'),**

**('9', '10', 'a'), ('10', '11', 'b'), ('2', '9', 'ε'), ('8', '3', 'ε')**

**]**

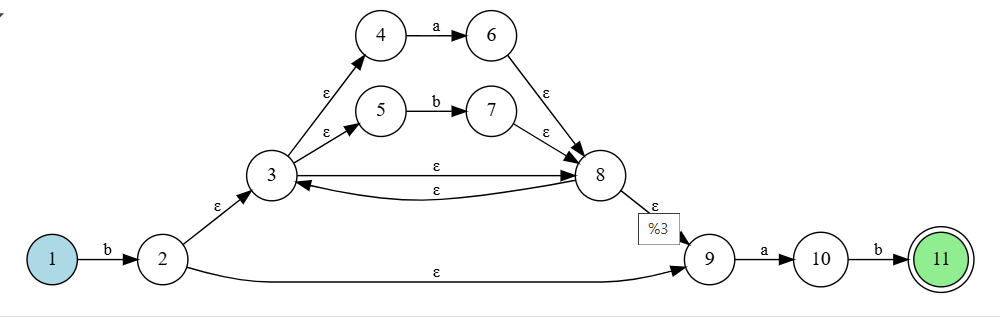
**for src, dst, label in transitions:**

**dot.edge(src, dst, label, fontsize='12')**

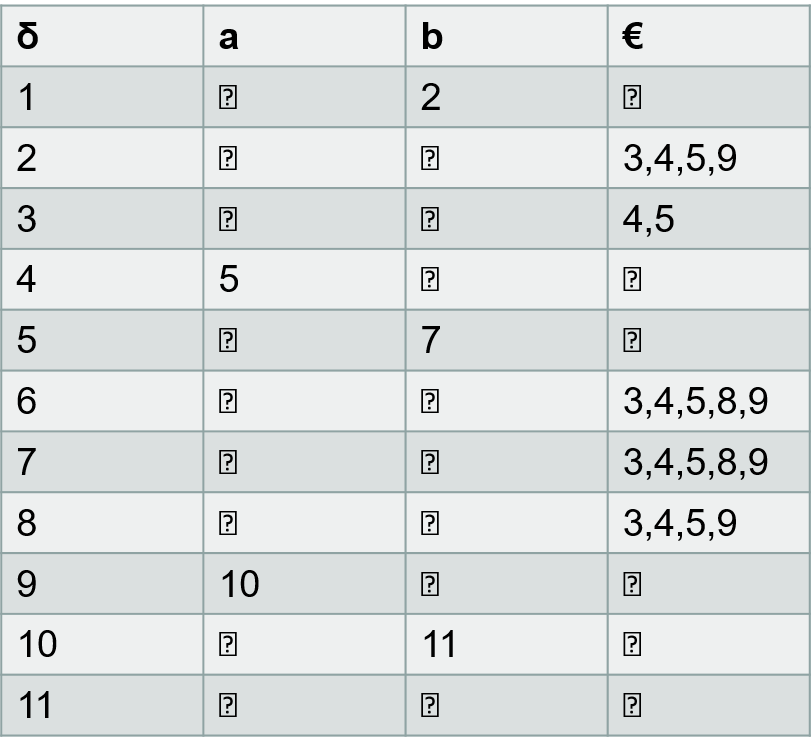
**return dot  
dot = construct\_e\_nfa()**

**display(dot)**

**Output:**

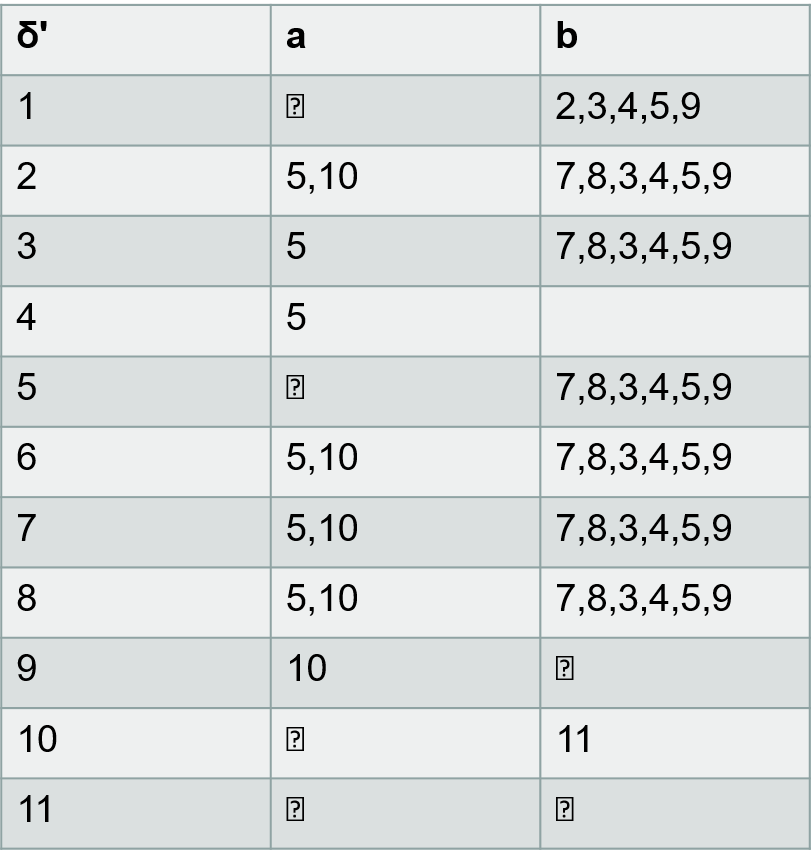


**C.Convert the NFA from the above.   
Tansition table and € closures for NFA :**



* **€ closure 1={1}**
* **€ closure 2={2,3,4,5,9}**
* **€ closure 3={4,5}**
* **€ closure 4={4}**
* **€ closure 5={5}**
* **€ closure 6={8,9,3,4,5}**
* **€ closure 7={8,9,3,4,5}**
* **€ closure 8={3,4,5,8,9}**
* **€ closure 9={9}**
* **€ closure 10={10}**
* **€ closure 11={11}**

**Conversion table for NFA :**



**Code:**

**from tabulate import tabulate**

**def construct\_transition\_table():**

**transitions = {**

**'1': {'a': 'Φ', 'b': '2,3,4,5,9'},**

**'2': {'a': '5,10', 'b': '7,8,3,4,5,9'},**

**'3': {'a': '5', 'b': '7,8,3,4,5,9'},**

**'4': {'a': '5', 'b': 'Φ’},**

**'5': {'a': 'Φ', 'b': '7,8,3,4,5,9’},**

**'6': {'a': '5,10', 'b': '7,8,3,4,5,9’},**

**'7': {'a': '5,10', 'b': '7,8,3,4,5,9'},**

**'8': {'a': '5,10', 'b': '7,8,3,4,5,9'},**

**'9': {'a': '10', 'b': 'Φ’},**

**'10': {'a': 'Φ', 'b': '11'},**

**'11': {'a': 'Φ', 'b': 'Φ'}**

**}**

**return transitions**

**def display\_transition\_table(transitions):**

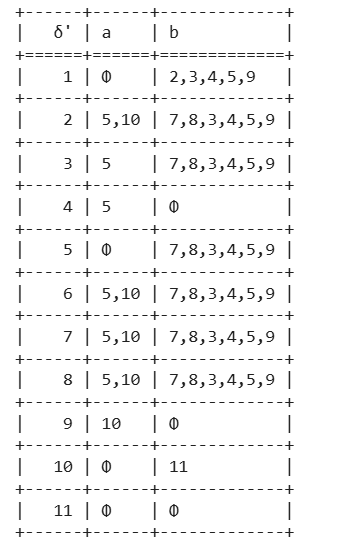
**table = [[state, trans['a'], trans['b']] for state, trans in transitions.items()]**

**print(tabulate(table, headers=["δ'", 'a', 'b'], tablefmt='grid’))**

**transitions = construct\_transition\_table()**

**display\_transition\_table(transitions)**

**output:**



CONCLUSION :

The conversion of **regular expressions** into **finite automata** is a crucial concept in **automata theory and formal language processing**. It provides an efficient and systematic approach to pattern recognition, making it widely applicable in **compiler design, lexical analysis, text search algorithms, and network security**.

While **finite automata** offer advantages such as **fast execution, structured representation, and ease of implementation**, they also come with challenges like **state explosion, limited expressiveness, and difficulty in handling nested patterns**. Despite these limitations, **finite automata remain a fundamental computational model** for recognizing regular languages and optimizing text-processing applications.