Compiler Design  
Lab Exercise 3

short line

**Aim: Conversion of a NFA to DFA.**

**Algorithm:**

1. Start
2. Get the input from the user.
3. Initialise a dictionary which would keep track of transitions and their states.
4. Feed in the states, the path taken and final state that the NFA would reach
5. Display the output
6. Compare with your table

**Code:(Language : Python3)**

import pandas as pd

nfa = {}

n = int(input("No. of states : "))

t = int(input("No. of transitions : "))

for i in range(n):

    state = input("state name : ")

    nfa[state] = {}

    for j in range(t):

        path = input("path : ")

        print("Enter end state from state {} travelling through path {} : ".format(state, path))

        reaching\_state = [x for x in input().split()]

        nfa[state][path] = reaching\_state

print("\nNFA Transitionsz :- \n")

print(nfa)

print("\nNFA Transition table :- ")

nfa\_table = pd.DataFrame(nfa)

print(nfa\_table.transpose())

print("Enter final state of NFA : ")

nfa\_final\_state = [x for x in input().split()]

new\_states\_list = []

#-------------------------------------------------

dfa = {}

keys\_list = list(

    list(nfa.keys())[0])

path\_list = list(nfa[keys\_list[0]].keys())

dfa[keys\_list[0]] = {}

for y in range(t):

    var = "".join(nfa[keys\_list[0]][

                      path\_list[y]])

    dfa[keys\_list[0]][path\_list[y]] = var

    if var not in keys\_list:

        new\_states\_list.append(var)

        keys\_list.append(var)

while len(new\_states\_list) != 0:

    dfa[new\_states\_list[0]] = {}

    for \_ in range(len(new\_states\_list[0])):

        for i in range(len(path\_list)):

            temp = []

            for j in range(len(new\_states\_list[0])):

                temp += nfa[new\_states\_list[0][j]][path\_list[i]]

            s = ""

            s = s.join(temp)

            if s not in keys\_list:

                new\_states\_list.append(s)

                keys\_list.append(s)

            dfa[new\_states\_list[0]][path\_list[i]] = s

    new\_states\_list.remove(new\_states\_list[0])

print("\nDFA Transitions:- \n")

print(dfa)

print("\DFA Transition table :- ")

dfa\_table = pd.DataFrame(dfa)

print(dfa\_table.transpose())

dfa\_states\_list = list(dfa.keys())

dfa\_final\_states = []

for x in dfa\_states\_list:

    for i in x:

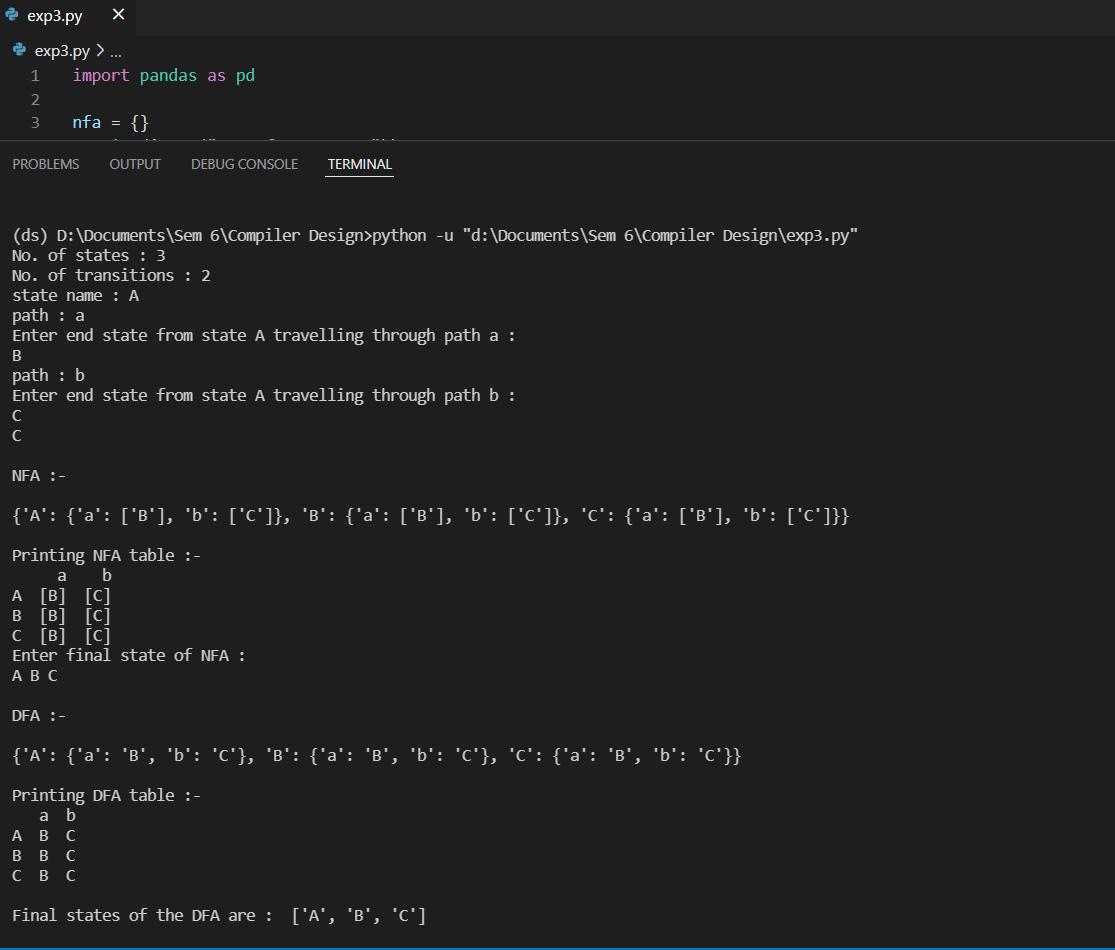
        if i in nfa\_final\_state:

            dfa\_final\_states.append(x)

            break

print("\nFinal states of the DFA are : ", dfa\_final\_states)

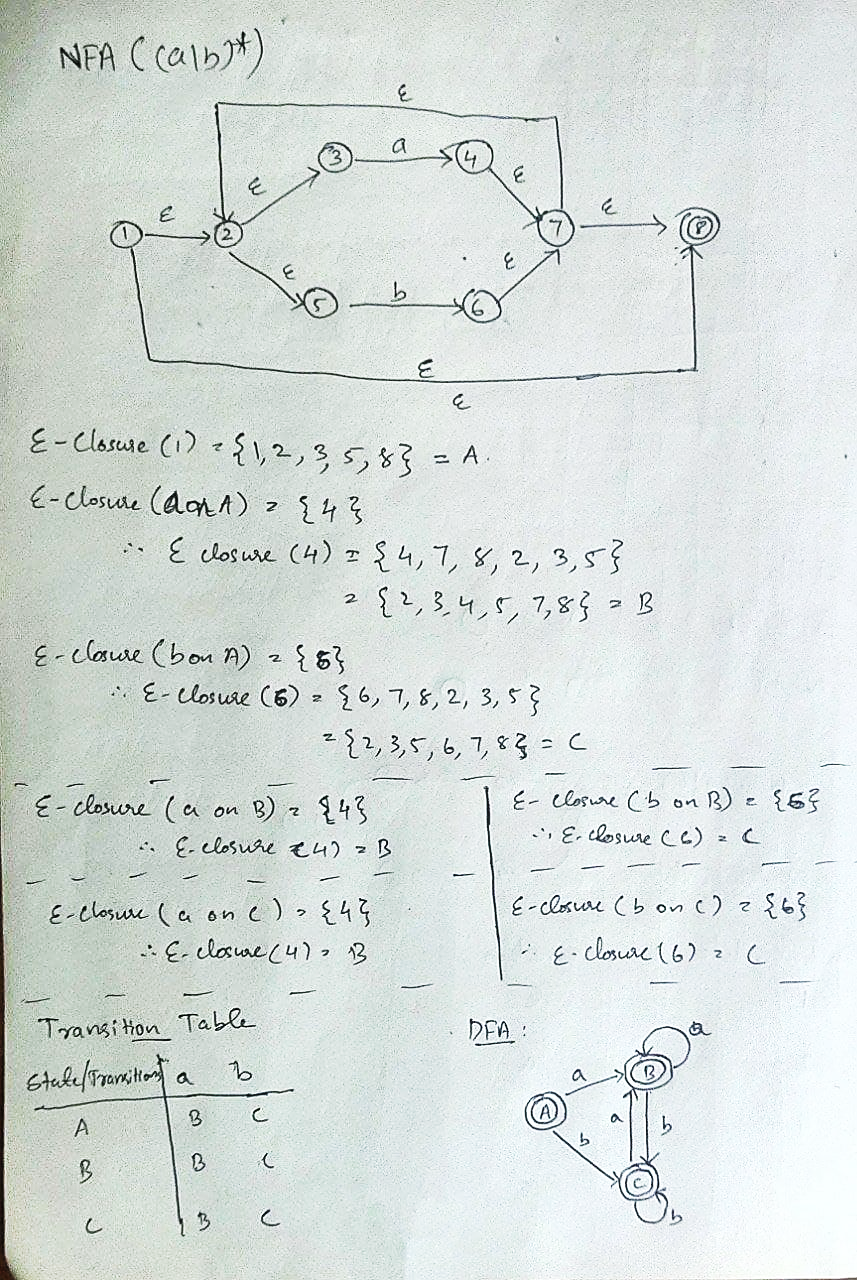
**Output:**



**Transition table:**

* First, we construct NFA Transition Table for (*a/b)\** according to ***Subset Construction*** method*.*
* Find the ε-Closure of each transition states on a and b. Record new states if found and repeat the process.
* After the construction of the NFA Transition table, we construct the DFA for (*a/b)\** and identify the final state/states in the DFA.

We compare the transition tables from code as well as handwritten and deduce that they are the same for the given expression.



**Conclusion:** Both the manual and output transition tables are the same hence the conversion of a NFA to DFA is verified.