Experiment V

**Aim**: Write a program to convert with ε-transition to NFA without ε-transition.

Algorithm

1. Start
2. Input the number of alphabets, number of states, number of transitions.
3. Input the set of alphabets
4. Input the start and final states.
5. Input the transactions.
6. Store each transaction in the transition table.
7. To find the ε - closure of each state, traverse through all the states.
8. Add the current state into the ε - closure of the state.
9. For each other state, if there exists a link from the current state to a new state and the alphabet is ε, add the new state into the ε - closure of the current state.
10. Repeat step 9 recursively until the condition returns false.
11. Display the ε - closures of all the states.
12. Display the NFA transition table.
13. Stop

Output

Enter the number of alphabets: 4

Enter the alphabet set: a b c e

Enter the number of states: 3

Enter the start state: 1

Enter the number of final states: 1

Enter the final states: 3

Enter no of transition: 5

Enter transitions

1 a 1

1 e 2

2 b 2

2 e 3

3 c 3

e-closure of states

e-closure(q1): {q1,q2,q3,}

e-closure(q2): {q2,q3,}

e-closure(q3): {q3,}

Equivalent NFA without epsilon

Start state: {q1,q2,q3,}

Alphabets: a b c e

States: {q1,q2,q3,} {q2,q3,} {q3,}

Final states: {q1,q2,q3,} {q2,q3,} {q3,}

NFA TRANSITION TABLE

{q1,q2,q3,} a {q1,q2,q3,}

{q1,q2,q3,} b {q2,q3,}

{q1,q2,q3,} c {q3,}

{q2,q3,} a {}

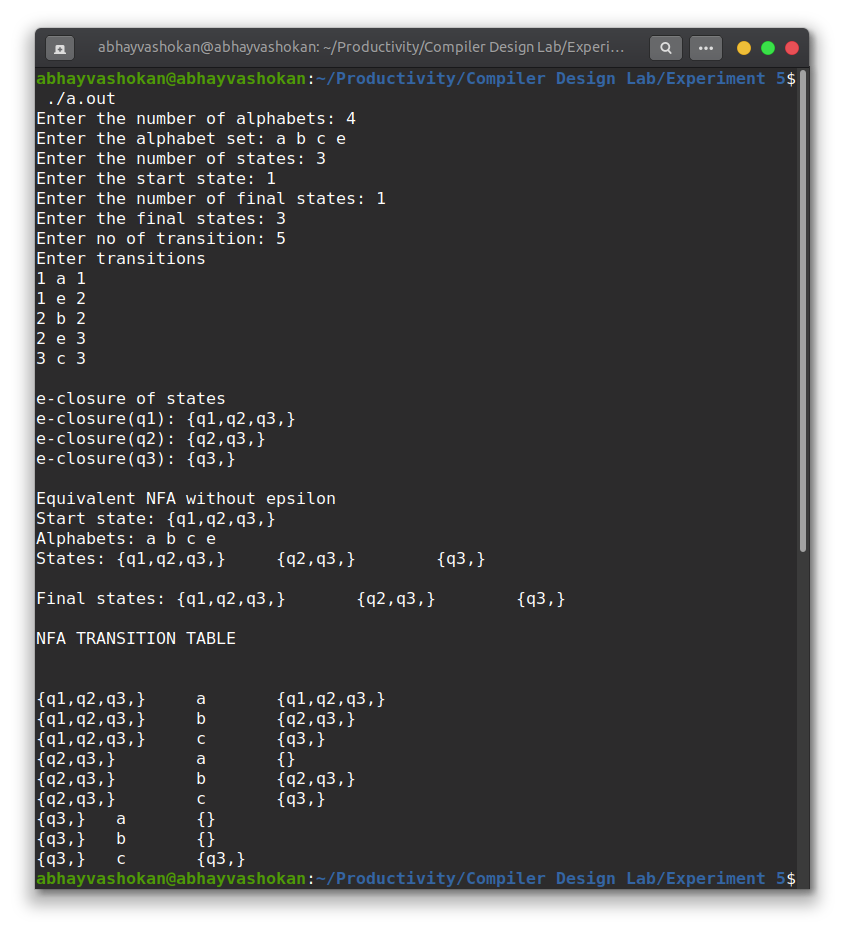
{q2,q3,} b {q2,q3,}

{q2,q3,} c {q3,}

{q3,} a {}

{q3,} b {}

{q3,} c {q3,}

Screenshot

Readme

1. Compile and run the program using the command

**gcc 2Abhay-P5.l && ./a.out**

2. Enter the number of alphabets

3. Enter the alphabet set

4. Enter the number of states

5. Enter the start state

6. Enter the number of final states

7. Enter the final states

8. Enter the number of transitions

9. Enter the transitions

10. The ε - closure of all states shall be obtained as output.

**Result**: Successfully implemented a program to convert an NFA to DFA.