PES UNIVERSITY

100 feet Ring Road, BSK 3rd Stage, Bengaluru 560085

Design and Analysis of Algorithms

Project Report

λ-NFA to DFA conversion algorithm

Name : Chandan N Bhat
Email : chandanbhat9799@gmail.com
Code : https://github.com/chandan-n-bhat/nfa-to-dfa

λ-NFA to DFA conversion algorithm

Problem Statement:

In this project we aim on converting a non-deterministic finite automaton (λ-NFA) into deterministic finite automaton (DFA) using space-time trade off method.

Introduction:

For most of the problems in real world , the input from the user will always be in terms of NFA, this is because NFA allows transition to multiple states from a particular state for the same inputs ,allows λ-transition as well as many non-deterministic advantages ,thus making it easier for the user to enter the problem with less number of states .However for a computing machine it is always easy to if the problem is in the DFA form as there are unique transitions for each symbol and moreover the decisions needed by the computing machine are deterministic unlike non deterministic model and also removing the need for it to parallelly compute or run multiple threads to make the right decision. Thus, for most of the problem the approach of first obtaining an NFA and then converting to a DFA is simpler and appealing.

Implementation:

- The input for our program is the NFA transactions stored it in a dynamic array of structures. Structure contains the input variable, input state and output state which corresponds to each transaction of the NFA.
- Next for each state in the NFA we find its lambda closure. By doing the following steps:

    1. Creating a structure that has the state name and an array of size of number of states that has one (1) for states that come in closure and zero (0) for the ones that do not.
    2. To find the lambda closure we use a recursive function that recursively calls the function for each state connected to it by lambda.
    3. Then take the union of the produced array with the previously present array.

- Next to create the NFA table, we consider each row as a node of a structure called “nfa” that has an array of strings that stores the lambda closure of each output state for each input (Note: The output lambda closures are stored as strings). Our starting input state is the lambda closure of the start state.

Steps to find lambda closure of output for each input character for closure of a state:

1. Go through each state in the lambda closure of start state, see all the transactions from that state for that particular input and add its closure in output closure state.

2. Take the output closure and take that as initial state in the next row in the nfa table. Keep doing this until no new output closure is found.

- Then we find out all the final states by seeing what all output lambda closures contain the final states mentioned by the users (That was stored in an array) in the NFA.

- We then print the DFA table that contains the lambda closure of output states (which are the new states of the DFA) , that is basically the dynamic array of structures named ‘nfa’.(Note if the output state for a particular input alphabet is NULL SET we print ‘{ }’to denote that it is a null set)

- We also print the start state and the end state in the dfa.

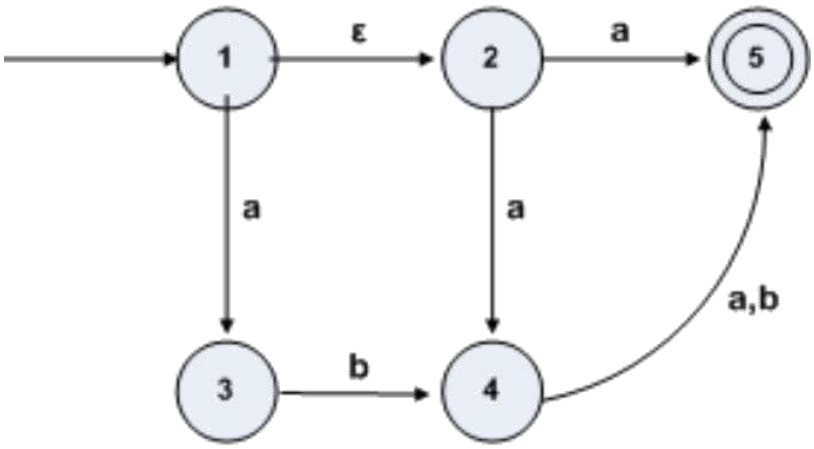
- Each of the output lambda closure acts as a new state in this DFA. So, we give a character as the name for each lambda closure state. This is done by taking the index of the lambda closure output and adding it to the character ‘A’, so if index is 1 the character assigned to the lambda closure is ‘B’ and so on. The NULL SETS in the above table are ignored while naming states.

- We print the DFA table, start state and end state of the dfa even after renaming the dfa states with characters as mentioned in the above point.

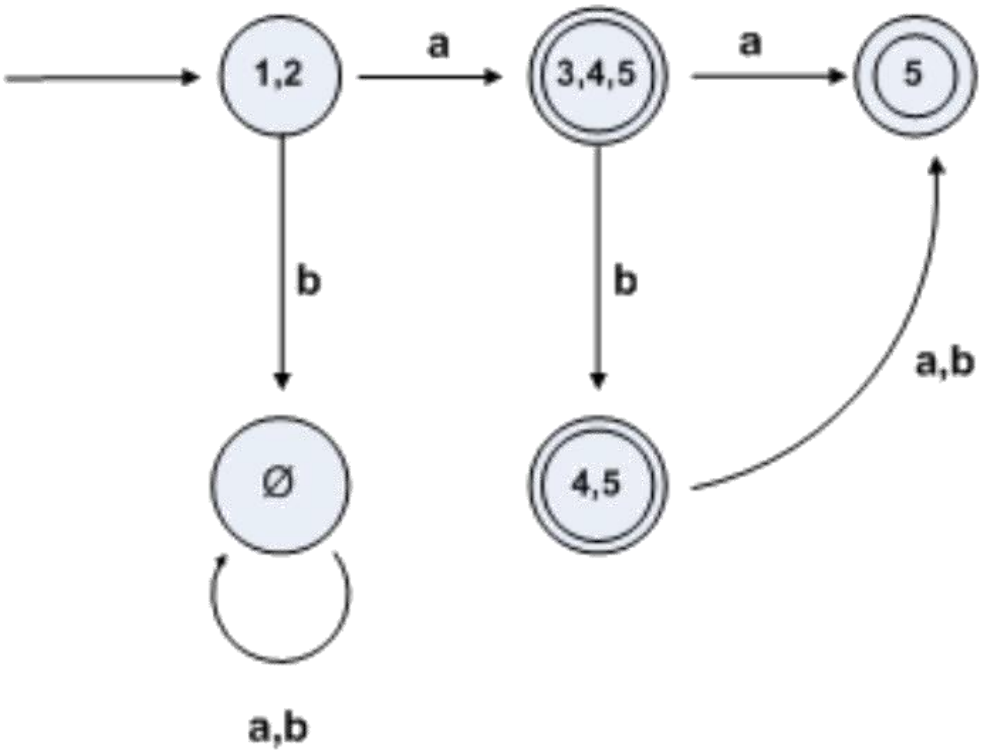
- We have implemented our Algorithm using C programming language

Input Output:

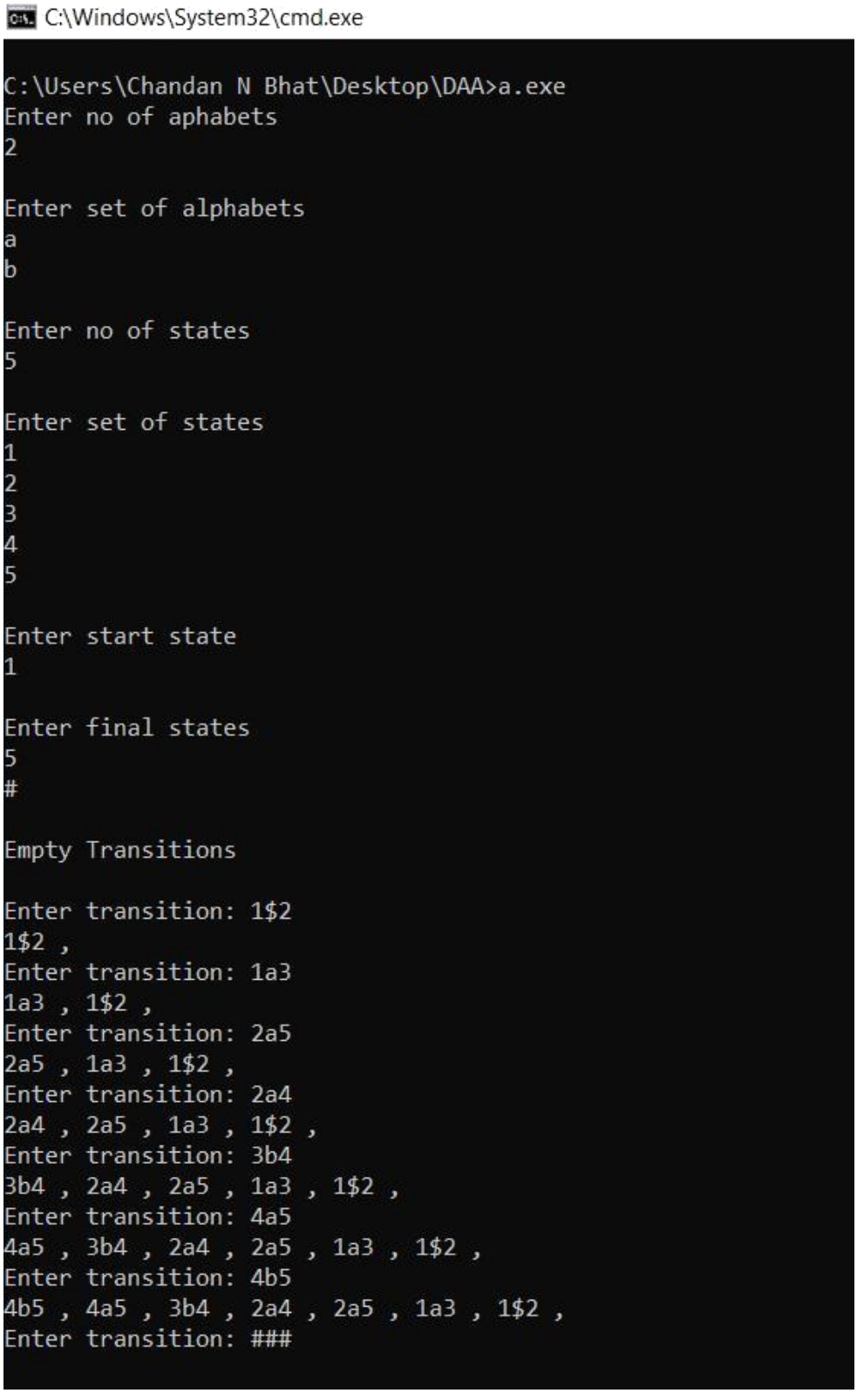
Input 1 modelled as graph:



The corresponding graph representation for our output:



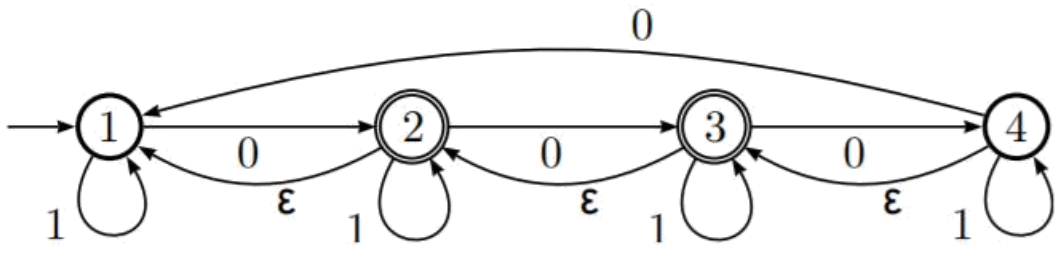
Corresponding inputs to our program:



Output for the above input:

```
State   a       b

12      345     {}
345     5       45
{}      {}      {}
5       {}      {}
45      5       5

Start state  : 12
Final states : 345  5  45


After renaming states


State   a       b

A       B       {}
B       D       E
{}      {}      {}
D       {}      {}
E       D       D

Start state  : A
Final states : B D E



C:\Users\Chandan N Bhat\Desktop\DAA>
```

Input 2:



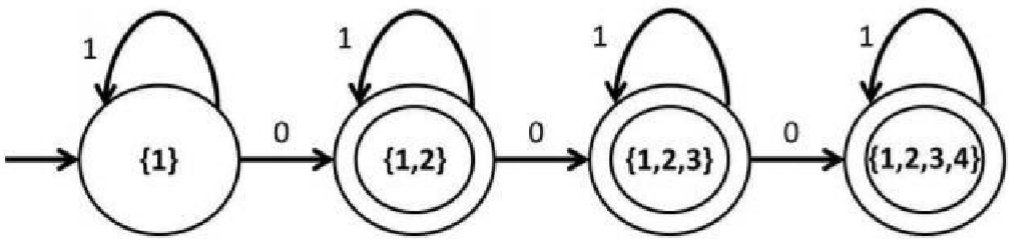
C:\Windows\System32\cmd.exe

```
C:\Users\Chandan N Bhat\Desktop\DAA>a.exe
Enter no of aphabets
2

Enter set of alphabets
1
0

Enter no of states
4

Enter set of states
1
2
3
4

Enter start state
1

Enter final states
2
3
#

Empty Transitions

Enter transition: 111
111 ,
Enter transition: 102
102 , 111 ,
Enter transition: 2$1
2$1 , 102 , 111 ,
Enter transition: 212
212 , 2$1 , 102 , 111 ,
Enter transition: 203
203 , 212 , 2$1 , 102 , 111 ,
Enter transition: 3$2
3$2 , 203 , 212 , 2$1 , 102 , 111 ,
Enter transition: 313
313 , 3$2 , 203 , 212 , 2$1 , 102 , 111 ,
Enter transition: 304
304 , 313 , 3$2 , 203 , 212 , 2$1 , 102 , 111 ,
Enter transition: 414
414 , 304 , 313 , 3$2 , 203 , 212 , 2$1 , 102 , 111 ,
Enter transition: 4$3
4$3 , 414 , 304 , 313 , 3$2 , 203 , 212 , 2$1 , 102 , 111 ,
Enter transition: 401
401 , 4$3 , 414 , 304 , 313 , 3$2 , 203 , 212 , 2$1 , 102 , 111 ,
Enter transition: ###
```

The corresponding Output:

C:\Windows\System32\cmd.exe

```
Enter transition: ###



State   1       0

1       1       12
12      12      123
123     123     1234
1234    1234    1234

Start state  : 1
Final states : 12  123  1234


After renaming states


State   1       0

A       A       B
B       B       C
C       C       D
D       D       D

Start state  : A
Final states : B C D



C:\Users\Chandan N Bhat\Desktop\DAA>
```



Conclusion:

Thus, our algorithm or program is capable: -

- Find the Lambda closure of all states.
- Convert NFA to DFA with single/multiple final states.
- Convert λ-NFA to DFA with single/multiple final states.

The entire implementation of our algorithm is using recursive approach like DFS and space-time trade off implementation.