GROUP-4

NFA to DFA -

Input: A valid description of NFA,Output: A valid description of DFA

TEAM MEMBERS

Abhiram Prasad
AM.EN.U4AIE19001

Nithin Sylesh
AM.EN.U4AIE19044

Ritika R Prasad
AM.EN.U4AIE19053

Vyshak S Nair
AM.EN.U4AIE19072

Lakshmi G Pillai
AM.EN.U4AIE19074

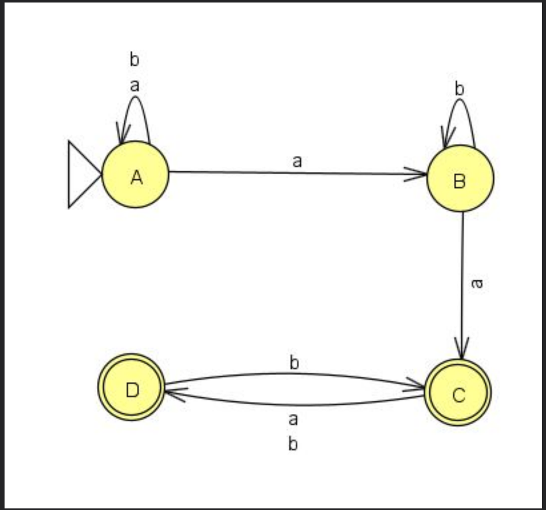
CONTENTS

INTRODUCTION

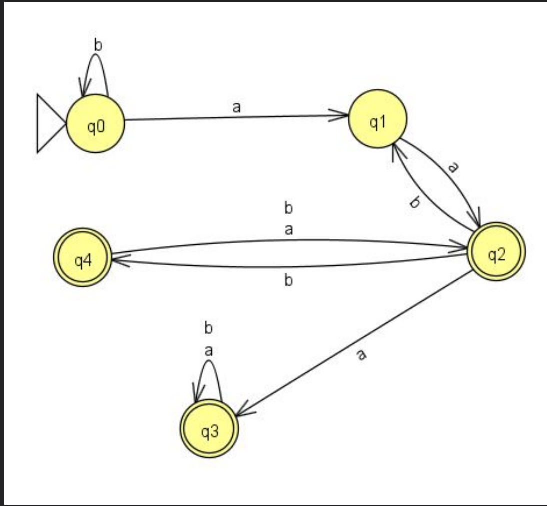
Our project's purpose is to use Python programming to transform NFA(non-deterministic finite automata) to DFA (Deterministic finite automata).NFA is given through input, it will be converted into DFA and DFA will be shown as output. We will further discuss the terms used and working of the project.

NFA

- NFA is an abbreviation for non-deterministic finite automata.
- For a given regular language, it is easier to design an NFA than a DFA.
- When there are multiple pathways for particular input from one state to the next, the finite automaton is referred to as NFA.



DFA



- **I**f the machine is given an input string, one symbol at a time then that finite automata is called deterministic finite automata
- **T**here is just one path from one state to the next for specific input.
- **M**ultiple final states can be found in DFA.

TRANSITION STATES

NFA

Present	Next	State
State	**a**	**b**
A[Initial State]	{A,B}	A
B	C	B
C[Final State]	D	D
D[Final State]	----	C

DFA

Previous	Next	State
State	**a**	**b**
A-Initial State	{A,B}	A
{A,B}	{A,B,C}	{A,B}
{A,B,C} - Final State	{A,B,C,D}	{A,B,D}
{A,B,C,D} - Final State	{A,B,C,D}	{A,B,C,D}
{A,B,D} - Final State	{A,B,C}	{A,B,C}

METHODOLOGY

1.NFA TO DFA

(without epsilon)

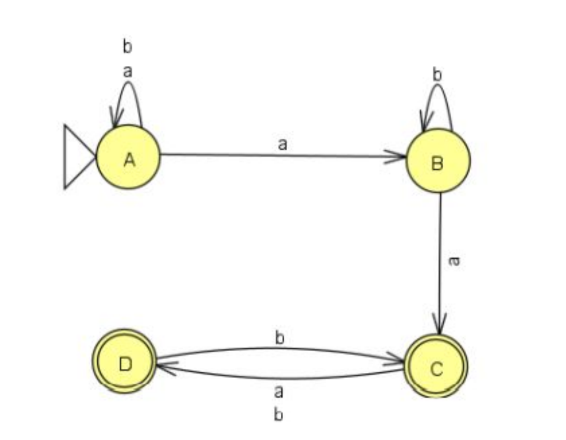
1 **C**onstruct state diagram of NFA

2 **D**raw transition table of NFA diagram

3 **D**raw a transition table of DFA from NFA

4 **C**onstruct DFA state diagram

a)NFA TO DFA (without epsilon)



```python
import pandas as pd

# Taking NFA input from User

nfa = {}
n = int(input("No. of states : "))              #Enter total no. of states
t = int(input("No. of transitions : "))         #Enter total no. of transitions/paths eg: a,b so input 2 for a,b,c input 3
for i in range(n):
    state = input("state name : ")              #Enter state name eg: A, B, C, q1, q2 ..etc
    nfa[state] = {}                             #Creating a nested dictionary
    for j in range(t):
        path = input("path : ")                 #Enter path eg : a or b in {a,b} 0 or 1 in {0,1}
        print("Enter end state from state {} travelling through path {} : ".format(state,path))
        reaching_state = [x for x in input().split()]  #Enter all the end states that
        nfa[state][path] = reaching_state       #Assigning the end states to the paths in dictionary

print("\nNFA :- \n")
print(nfa)                                      #Printing NFA
print("\nPrinting NFA 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(" ")]      # Enter final state/states of NFA
###################################################
```

```python
new_states_list = []                          #holds all the new states created in dfa
dfa = {}                                      #dfa dictionary/table or the output structure we needed
keys_list = list(list(nfa.keys())[0])                  #conatins all the states in nfa plus the states created in dfa are also appended further
path_list = list(nfa[keys_list[0]].keys())    #list of all the paths eg: [a,b] or [0,1]

###################################################
```

```
# Computing first row of DFA transition table

dfa[keys_list[0]] = {}                          #creating a nested dictionary in dfa
for y in range(t):
    var = "".join(nfa[keys_list[0]][path_list[y]])   #creating a single string from all the elements of the list which is a new state
    dfa[keys_list[0]][path_list[y]] = var            #assigning the state in DFA table
    if var not in keys_list:                         #if the state is newly created
        new_states_list.append(var)                  #then append it to the new_states_list
        keys_list.append(var)                        #as well as to the keys_list which contains all the states

###################################################

# Computing the other rows of DFA transition table

while len(new_states_list) != 0:                     #consition is true only if the new_states_list is not empty
    dfa[new_states_list[0]] = {}                     #taking the first element of the new_states_list and examining it
    for _ in range(len(new_states_list[0])):
        for i in range(len(path_list)):
            temp = []                                #creating a temporay list
            for j in range(len(new_states_list[0])):
                temp += nfa[new_states_list[0][j]][path_list[i]]  #taking the union of the states
            s = ""
            s = s.join(temp)                         #creating a single string(new state) from all the elements of the list
            if s not in keys_list:                   #if the state is newly created
                new_states_list.append(s)            #then append it to the new_states_list
                keys_list.append(s)                  #as well as to the keys_list which contains all the states
            dfa[new_states_list[0]][path_list[i]] = s   #assigning the new state in the DFA table
```

```python
    new_states_list.remove(new_states_list[0])      #Removing the first element in the new_states_list

print("\nDFA :- \n")
print(dfa)                                          #Printing the DFA created
print("\nPrinting DFA 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)      #Printing Final states of DFA
```

INPUT

```
No. of states : 4
No. of transitions : 2
state name : A
path : a
Enter end state from state A travelling through path a :
A B
path : b
Enter end state from state A travelling through path b :
A
state name : B
path : a
Enter end state from state B travelling through path a :
C
path : b
Enter end state from state B travelling through path b :
B
state name : C
path : a
Enter end state from state C travelling through path a :
D
path : b
Enter end state from state C travelling through path b :
D
state name : D
path : a
Enter end state from state D travelling through path a :

path : b
Enter end state from state D travelling through path b :
C
```

```
Enter final state of NFA :
C D
```

OUTPUT

```
NFA :-

{'A': {'a': ['A', 'B'], 'b': ['A']}, 'B': {'a': ['C'],
'b': ['B']}, 'C': {'a': ['D'], 'b': ['D']}, 'D': {'a': [],
'b': ['C']}}

Printing NFA table :-
        a     b
A  [A, B]   [A]
B     [C]   [B]
C     [D]   [D]
D      []   [C]
```

```
DFA :-

{'A': {'a': 'AB', 'b': 'A'}, 'AB': {'a': 'ABC', 'b': 'A
B'}, 'ABC': {'a': 'ABCD', 'b': 'ABD'}, 'ABCD': {'a': 'ABC
D', 'b': 'ABDC'}, 'ABD': {'a': 'ABC', 'b': 'ABC'}, 'ABDC':
{'a': 'ABCD', 'b': 'ABCD'}}

Printing DFA table :-
          a      b
A        AB      A
AB      ABC     AB
ABC    ABCD    ABD
ABCD   ABCD   ABDC
ABD     ABC    ABC
ABDC   ABCD   ABCD

Final states of the DFA are :  ['ABC', 'ABCD', 'ABD', 'ABD
C']
```

2.NFA TO DFA (with epsilon)

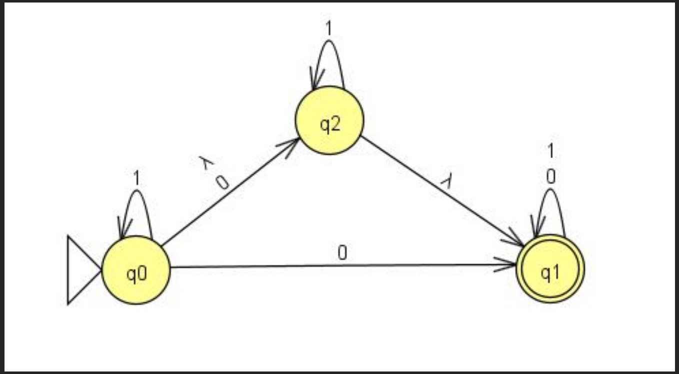
1. **C**reate a transition diagram with the initial state of NFA.

   For example, suppose {q0} as an initial state.

   Mark it as the initial state of DFA.

2. **R**epeat the following steps until no more edges are missing.

3. **E**very state of the Transition diagram in which the final state of NFA appears, make it the final state.



```python
def print_list(l,s='',e=''):
    #if type(l) in (list(),tuple(),set()):
    list(l).sort()
    print(*l,sep=s,end=e)

# find and return e-closure of given state
def closure(state):
    s=state
    for i in enfa_delta:
        for j in state:
            if i[0]==j and i[1]=='e': s+=[i[2]]
    s=list(set(s))
    s.sort()
    return s

# find and return next states given state and symbol
def delta(state,symbol):
    s = []
    for i in enfa_delta:
        if i[0]==state and i[1]==symbol: s+=[i[2]]
    s=list(set(s))
    s.sort()
    return s

def statesAcceptanceDFA(states,F):
    dfa_F = []
    for f in F:
        for s in states:
            if (f in s) and ((len(dfa_F)==0) or (s not in dfa_F)):
                dfa_F.append(s)
    return dfa_F
```

```python
custom = True


if custom:
    nQ = int(input("Enter the number of states :"))

    enfa_Q = []
    for i in range (nQ):
        enfa_Q += input ("Enter state " + str (i + 1) + " :")

    enfa_q0 = input ("Enter the input states :")

    enfa_F = input("Enter the acceptance states :").split(" ")

    nsig = int(input("Enter the number of symbols (excluding epsilon):"))

    dfa_sigma = []
    for i in range (nsig):
        dfa_sigma += [input ("Enter the symbol #" + str (i + 1) + " :")]

    ndel = int(input("Enter the number of transitions:"))
```

```python
print (" Enter the transitions in the following format: "
    "Input_State Symbol Output_State \n -> In the case of epsilon, write e in the symbol")
i = 0
enfa_delta = []
tran = []
while (i<ndel):
    tran = input ("Enter the transition #" + str (i + 1) + ":")
    tran = tran.split(" ")
    enfa_delta.append(tran)
    i += 1

print("=================================================================== ")
```

```python
print("eNFA")

enfa_sigma = dfa_sigma.copy()
enfa_sigma += 'e'

print()
print("set of states (Q): {",end='')
print_list(enfa_Q,s=',',e='}\n')

print("set of input symbols (Σ): {",end='')
print_list(enfa_sigma,s=',',e='}\n')

print("initial state (q0): {",end='')
print_list(enfa_q0,s=',',e='}\n')

print("final states (F): {",end='')
print_list(enfa_F,s=',',e='}\n')

print("Transition function (δ):\n\n\t",end='')

print_list(enfa_sigma,s='\t',e='\n\n')
for i in enfa_Q:
    print(i,end="\t")
    for j in enfa_sigma:
      d=delta(i,j)
      if d: print_list(d);
      else: print_list('-');
      print("\t",end='');
    print()

print("================================================================ ")
print("Epsilon Closures of all states:\n")
```

```python
for i in enfa_Q:
  print('ε-closure({}) = {{'.format(i),end='')
  print_list(closure([i]),s=',',e='}\n\n')

print("================================================================ ")
print("DFA")

dfa_q0 = closure([enfa_q0])
dfa_Q = [dfa_q0]
c = 0
k = 0
dfa_delta = []

while(True):
    for i in dfa_sigma:
        flag = True
        q1 = []

        for j in dfa_Q[k]:
            q1 += delta(j,i)
            q1 = list(set(q1))
        q1 = closure(q1)

        dfa_delta += [[dfa_Q[k],i,q1]]
        for j in dfa_Q:
            if j==q1: flag=False;
        if flag:
            dfa_Q += [q1]
            c += 1
    k += 1
    if k>c: break;
```

```python
dfa_F = statesAcceptanceDFA(dfa_Q,enfa_F) # Set of acceptance states in DFA

print("set of states (Q): {",end='')
for i in range(len(dfa_Q)-1): print_list(dfa_Q[i],e=',');
print_list(dfa_Q[len(dfa_Q)-1],e='}\n')

print("set of input symbols (Σ): {",end='')
print_list(dfa_sigma,s=',',e='}\n')

print("initial state (q0): {",end='')
print_list(dfa_q0,e='}\n')

print("final states (F): {",end='')
for i in range(len(dfa_F)-1): print_list(dfa_F[i],e=',');
print_list(dfa_F[len(dfa_F)-1],e='}\n')

print("Transition function (δ):\n\n\t",end='')

qi=0
i=0

print_list(dfa_sigma,s='\t',e='\n\n')
while(i<(len(dfa_Q)*len(dfa_sigma))):
    print_list(dfa_Q[qi], e = "\t")

    while (i<(len(dfa_sigma)*(qi+1))):
        print_list(dfa_delta[i][2],e="\t")
        i += 1
    print()
    qi += 1
print("================================================================== ")
```

OUTPUT

INPUT

```
Enter the number of states :3
Enter state 1 :a
Enter state 2 :b
Enter state 3 :c
Enter the input states :a
Enter the acceptance states :c
Enter the number of symbols (excluding epsilon):2
Enter the symbol #1 :0
Enter the symbol #2 :1
Enter the number of transitions:9
 Enter the transitions in the following format: Input_State Symbol Output_State
 -> In the case of epsilon, write e in the symbol
Enter the transition #1:a 0 bc
Enter the transition #2:a 1 a
Enter the transition #3:a e b
Enter the transition #4:b 0 d
Enter the transition #5:b 1 b
Enter the transition #6:b e c
Enter the transition #7:c 0 c
Enter the transition #8:c 1 c
Enter the transition #9:c e d
```

```
==========================================================================
eNFA

set of states (Q): {a,b,c}
set of input symbols (∑): {0,1,e}
initial state (q0): {a}
final states (F): {c}
Transition function (δ):

           0          1          e

a          bc         a          b
b          d          b          c
c          c          c          d
==========================================================================
```

```
==========================================================================
Epsilon Closures of all states:

Ɛ-closure(a) = {a,b,c,d}

Ɛ-closure(b) = {b,c,d}

Ɛ-closure(c) = {c,d}

==========================================================================
```

```
==========================================================================
DFA
set of states (Q): {abcd,bccd,cd}
set of input symbols (∑): {0,1}
initial state (q0): {abcd}
final states (F): {abcd,bccd,cd}
Transition function (δ):

           0          1

abcd       bccd       abcd
bccd       cd         cd
cd         cd         cd
==========================================================================
```

CONCLUSION

A good understanding of FSA may help us in numerous fields. From this point of view, the subset construction algorithm attempts to provide conversion of nfa to dfa with and without using the epsilon transition.The educational aim of this work is to provide a practical experience in developing an NFA to DFA conversion using python

THANK you