NFA to DFA - Input: A valid description of NFA,Output: A valid description of DFA(Group-4)

Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri Campus, India
Kollam 690525
Kerala

Abhiram Prasad
AM.EN.U4AIE19001
abhiramprasad@am.students.amrita.edu

Nithin Sylesh
AM.EN.U4AIE19044
nithinsylesh@am.students.amrita.edu

Ritika R Prasad
AM.EN.U4AIE19053
ritikarprasad@am.students.amrita.edu

Vysakh S Nair
AM.EN.U4AIE19072
vysakhsnair@am.students.amrita.edu

Lakshmi G Pillai
AM.EN.U4AIE19074
lakshmigpillai@am.students.amrita.edu

Abstract- *The theory of computation is an area of computer science and mathematics concerned with determining whether and how efficiently problems may be addressed using an algorithm on a model of computation. Automata theory is the study of abstract machines and the computational problems that these abstract machines can solve in theoretical computer science. Our project's purpose is to use Python programming to transform non-deterministic finite state automata(NFA) to Deterministic finite state automata(DFA) with and without using epsilon. Automata as the name for these abstract machines.*

Keywords : DFA,NFA,Finite State machine, Subset Construction

I. Introduction

A regular language is one that may be stated using regular expressions, finite automata, or state machines, both deterministic and non-deterministic. Finite Automata are commonly used in computer science. Deterministic Finite Automata and Nondeterministic Finite Automata are two forms of finite automata.It is possible to transform any Nondeterministic Finite Automata (NFA) into one that accepts the same regular equivalence of Deterministic Finite Automata (DFA).The computation of the NFA is the process of converting an NFA to an equivalent DFA.The NFA has a smaller memory need, but its functioning is poorer since it must update all states every time it reads a character. However,

DFA's working efficiency is greater. As a result, an NFA is frequently converted into a DFA.

II.Background

The distinctions between an NFA and a DFA required to be known as the initial stage in building this method, or in other words, the NFA-to-DFA visualisation technique. An NFA, unlike a DFA, can be in several states at the same time, have undefined or transitions, and numerous transitions when only provided a single input.The power of both NFA and DFA is the same, and each NFA may be converted into a DFA.

a. *Differences between NFA and DFA*

An NFA is comparable to a DFA, with the exception of three major distinctions that are permitted in an NFA but not in a DFA.

1)The NFA allows for lambda transitions.

2)Transitions in NFA might be defined

3)After receiving a single input, NFA may transition to different states (it can be in several states at the same time).

If and only if two automatas accept the same language, they are comparable. As a consequence, it may be inferred that both automatas are equivalent if they accept the same language.It's also worth noting that the NFA-to-DFA method will always come to an end because an NFA's number of states is limited.

b.*Important factors during the conversion process*

During the conversion process, the algorithm should keep track of the following crucial factors:

1) Is there a need for a trap state?

2) What is the starting point?

3) What are the end results?

4) Are there any -transitions in the original NFA?

5) How will the algorithm recognise when it has completed its task?

6) Is there a -transition between the initial and final states?

III.Theory

3.1 *Finite State Machine*

Finite State Machines are machines that have a finite number of states. A finite state machine is a mathematical model of computation that is typically represented as a graph, with a finite number of nodes describing the system's various states and a finite number of arcs representing transitions that do or do not change the state.In response to certain inputs, the FSM can transition from one state to another; this transition is known as a transition. A set of states, the beginning state, and the inputs that trigger each transition describe an FSM. There are two forms of finite-state machines: deterministic and non-deterministic finite-state machines. Any non-deterministic finite-state machine can be built as a deterministic finite-state machine.

3.2 *Deterministic Finite Automata*

The term DFA is used to describe deterministic finite automata. The computation's uniqueness is referred to as deterministic. If the machine is given an input string, one symbol at a time then that finite automata is called deterministic finite automata.In DFA, there is just one path from one state to the next for specific input.The null move is not accepted by DFA, which means it cannot change state without any input character. Multiple final states can be found in DFA. DFA's are ideally suited to succinctly expressing any system that must retain an internal state specification.

3.3 *Non-Deterministic Finite Automata*

The term 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.Although not all NFAs are DFAs, all NFAs may be converted to DFAs.NFA is defined similarly to DFA, with the distinction that it contains numerous future states and transitions.

IV.METHODOLOGY

Subset construction algorithm

The method of converting a nondeterministic nite-state machine into a deterministic one is known as subset construction. Because nondeterministic machines are typically easier to describe than their deterministic equivalents, and the conversion of regular expressions to finite-state machines usually produces non deterministic machines, determinization is a key step in any implementation of finite-state machines.

Steps Involved in subset construction

1.Construct state diagram of NFA
2.Draw transition table of NFA diagram
3.Draw transition table of DFA from NFA transition table
4.Construct DFA state diagram

a)NFA TO DFA (without epsilon)

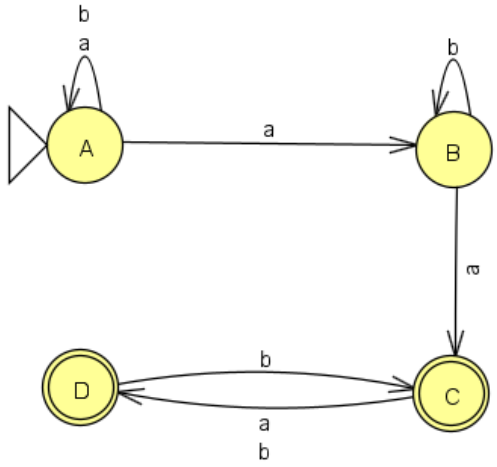

Fig1 shows the state transition diagram of NFA

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

Fig 2 shows the NFA transition table

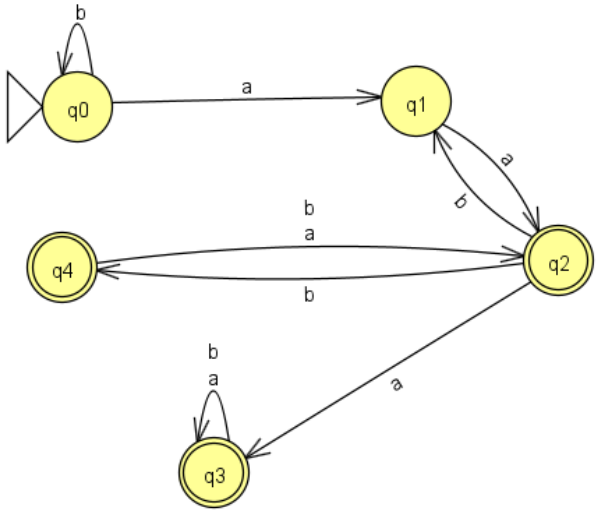

Fig 3 shows the state transition diagram of DFA

Previous State	Next 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}

Fig 4 shows the transition diagram of DFA

V. Implementation and Results

The Implementation was done in jupyter Notebook using python3.9.We were able to convert nfa to dfa with and without the use of the epsilon transition.

a.NFA to DFA using epsilon

Following are the steps involved in converting nfa to dfa using epsilon transition

Step 1 : Take ∈ closure for the beginning state of NFA as the beginning state of DFA.

Step 2 : Find the states that can be traversed from the present for each input symbol

Step 3 : If any new state is found, take it as the current state and repeat step 2.

Step 4 : Do repeat Step 2 and Step 3 until no new state is present in the DFA transition table.

Step 5 : Mark the states of DFA which contain the final state of NFA as final states of DFA.

```
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
```

Fig 1- shows the input for NFA -epsilon

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

set of states (Q): {a,b,c}
set of input symbols (Σ): {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
================================================================
```

Fig 2 - shows the output for NFA-epsilon

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

Ɛ-closure(a) = {a,b,c,d}

Ɛ-closure(b) = {b,c,d}

Ɛ-closure(c) = {c,d}

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

Fig 3 - shows the output for epsilon closure

```
================================================================
DFA
set of states (Q): {abcd,bccd,cd}
set of input symbols (Σ): {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
================================================================
```

Fig 4 - shows the output for DFA from NFA-epsilon

b.NFA to DFA without epsilon

Following are the steps involved in converting nfa to dfa without using epsilon transition.

Step 1: Create 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.

Step 2: Repeat the following steps until no more edges are missing.

Step 3:Every state of the Transition diagram in which the final state of NFA appears, make it the final state.

```
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
```

Fig 5- shows the input for NFA

```
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]
```

Fig 6 - shows the output for NFA

```
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']
```

Present State	0	Next State 1	Epsilon
a,b,c	b,c	a	b
b,c	---	b	c
c	c	c	-----

Fig 7 shows the transition diagram of nfa-∈

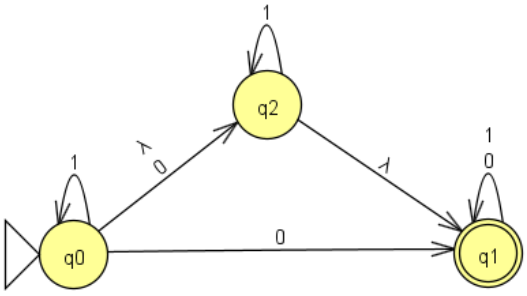

Fig 8 shows the state transition diagram for nfa-∈

Present State	0	Next 1
a,b,c	b,c	a,b,c
b,c	c	b,c
c	c	c

Fig 9 shows the state transition diagram of DFA

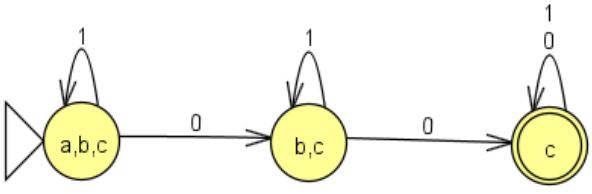

Fig 10 shows the transition diagram of dfa

VI. Conclusion

Understanding how the machines work is an important part of knowing how to build future technologies and how to custom-build machines and devices that are better suited for a particular job.It also enables us to determine what software improvements should be applied so that the programme can function more efficiently with the existing hardware.There are a large variety of possible applications which are benefited from FSA. 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

VII. Reference

[1] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Introduction to automata theory, languages and computation. 2007.

[2] M. J. Kearns and U. V. Vazirani. An introduction to computational learning theory. 1994.

[3] Andreas Kerren. Learning by generation in computer science education. Journal of Computer Science & Technology, 4(2):84 – 90, 8 2004.

[4] Andreas Kerren, Tomasz Müldner, and Elhadi Shakshuki. Novel algorithm explanation techniques for improving algorithm teaching. In Proceedings of the 2006 ACM Symposium on Software Visualization, SoftVis '06, pages 175–176, New York, NY, USA, 2006. ACM.

[5] Shiva Kintali. Review of elements of automata theory, by jacques sakarovitch, translator (from french) reuben thomas. pages 45–47, 2012

.

[6] John von Neumann. Collected works; vol. 5: Design of computers, theory of automata and numerical analysis. pages 290–326, 1963.

[7] Timothy M. White and Thomas P. Way. jfast: A java finite automata simulator. SIGCSE Bull., 38(1):384–388, March 2006.

[8] W. Patrick Merryman. Animating the conversion of non-deterministic finite state automata to deterministic finite state automata, master's thesis, montana state university, 2007.

[9] Susan H. Rodger, Anna O. Bilska, Kenneth H. Leider, Magdalena Procopiuc,Octavian Procopiuc, Jason R. Salemme, and Edwin Tsang. A collection of tools for making automata theory and formal languages come alive. SIGCSE Bull.,29(1):15–19, March 1997.

[10] Stephan Diehl, Andreas Kerren, and Torsten Weller. Visual exploration of generation algorithms for finite automata on the web. In Revised Papers from the 5th International Conference on Implementation and Application of Automata, CIAA '00, pages 327–328, London, UK, UK, 2001. Springer-Verlag.