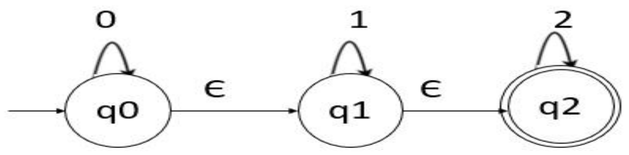
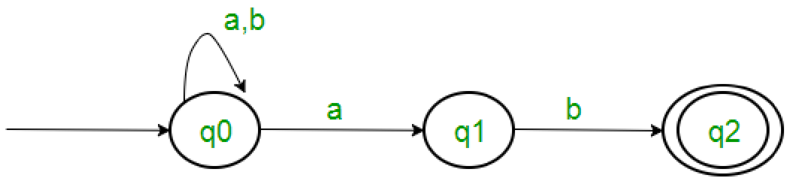
SR 22

St. Peter’s Engineering College (Autonomous) Dullapally (P), Medchal, Hyderabad – 500100. QUESTION BANK					Dept.	:	CSE(AIML)
					Academic Year 2024-25		
Subject Code	:	AS22-66PC02	Subject	:	AUTOMATA THEORY & COMPILER DESIGN		
Class/Section	:	B. Tech.	Year	:	II	Semester	: II

BLOOMS LEVEL					
Remember	L1	Understand	L2	Apply	L3
Analyze	L4	Evaluate	L5	Create	L6

*****

Q. No	Question (s)	Marks	BL	CO
	UNIT – I			
1	Define Finite Automata.	1M	L1	C222.1
	Define Transition Function in FA.	1M	L1	C222.1
	Define Computation.	1M	L1	C222.1
	Define NFA-Ɛ.	1M	L1	C222.1
	Define Theory of Computation.	1M	L1	C222.1
2	Discuss about DFA with example.	3M	L2	C222.1
	Discuss about NFA with example.	3M	L2	C222.1
	Design a DFA with Dead States.	3M	L6	C222.1
	What is the computational problem and give some of the examples?	3M	L6	C222.1
	Design a NFA-Ɛ with example.	3M	L6	C222.1
3	Design a DFA which accepts set of all strings containing 1100 as a substring with in an alphabet ∑ = {0, 1}.	5M	L6	C222.1
	Design an NFA which accepts strings which ends with 00|11 within an alphabet ∑ = {0, 1}.	5M	L6	C222.1
	Design an NFA which accepts strings with 1100 or 1010 as a substring with in an alphabet ∑ = {0, 1}.	5M	L6	C222.1
	Design a DFA which accepts set of all strings containing even number 0’s and 1’s within an alphabet ∑ = {0, 1}.	5M	L6	C222.1
	Design an DFA which accepts strings which ends with 00|11 within an alphabet ∑ = {0, 1}.	5M	L6	C222.1
4	Convert the following NFA-Ɛ in to NFA,	10M	L5	C222.1

				
	a) Design a DFA which accepts set of all strings that starts with 1 and ends with 0 within an alphabet ∑ = {0, 1}. b) Design an NFA with Multiple Final States.	**10M**	L6	C222.1
	Convert the following NFA in to DFA, 	**10M**	L5	C222.1

Answers

1.

Define Finite Automata.

Finite Automata, finite automaton, or simply a state machine, is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time.

Define Transition Function in FA.

The transition function in a finite automaton specifies how the automaton moves from one state to another based on an input symbol.

Define Computation.

Computation is the movement and alteration which occurs during the transition of data or the processing of data based on a set of operations. The theory of computation includes the fundamental mathematical properties of computer hardware, software and their applications.

Define NFA-Ɛ.

NFA-Ɛ is a special type of transition where the automaton can move from one state to another without reading any input symbol (i.e., on the empty string).

Define Theory of Computation.

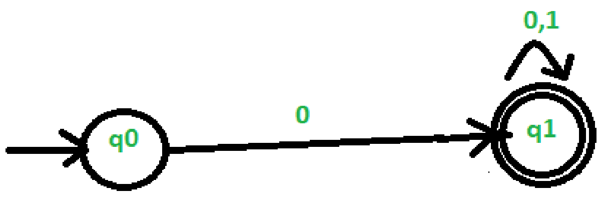
In theoretical computer science, the theory of computation is the branch that deals with whether and how efficiently problems can be solved on a model of computation, using an algorithm. The field is divided into three major branches: automata theory, computability theory and computational complexity theory.

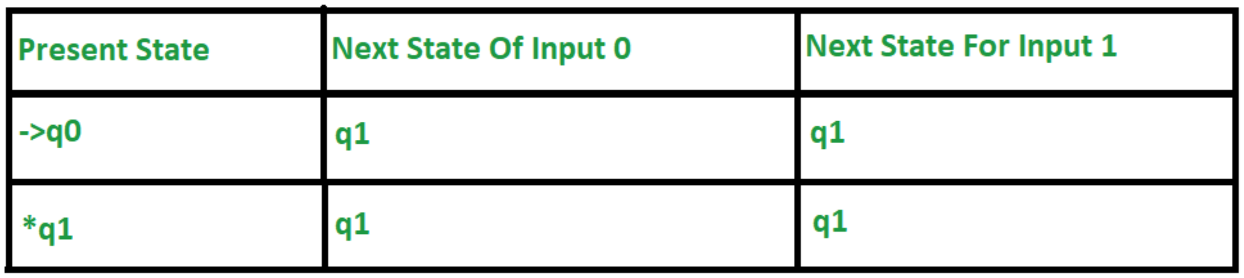
Discuss about DFA with example.

A Deterministic Finite Automaton (DFA) is defined as a 5-tuple $(Q, \Sigma, \delta, s, F)$ consisting of

- A finite set Q (the set of states)
- A finite set of symbols Σ (the input alphabet)
- A transition function $\delta: Q \times \Sigma \to Q$ mapping the current state $q \in Q$ and input symbol $a \in \Sigma$ to a new state $\delta(q, a) \in Q$
- An initial state $s \in Q$ (the start state)
- A set of accepting states F (the final states)

A DFA is a mathematical model of a simple computational device that reads a string of symbols over the input alphabet Σ, and either accepts or reject the input string. We would like to turn this mathematical definition into a working program, so that we can run DFAs on our computer





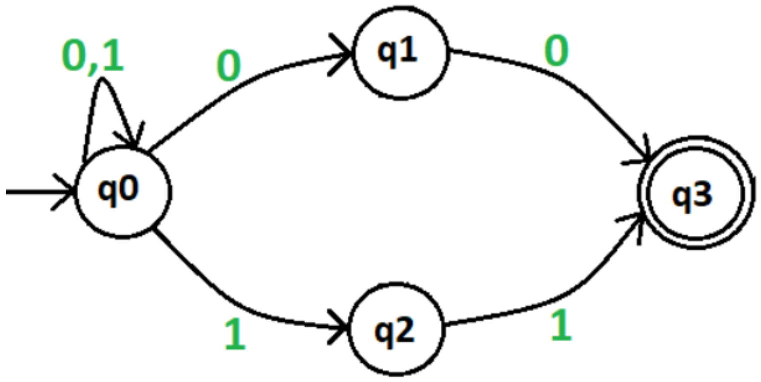
Present State	Next State Of Input 0	Next State For Input 1
->q0	q1	q1
*q1	q1	q1

Discuss about NFA with example.

- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition.

Q: finite set of states
∑: finite set of the input symbol
q0: initial state
F: **final** state
δ: Transition function

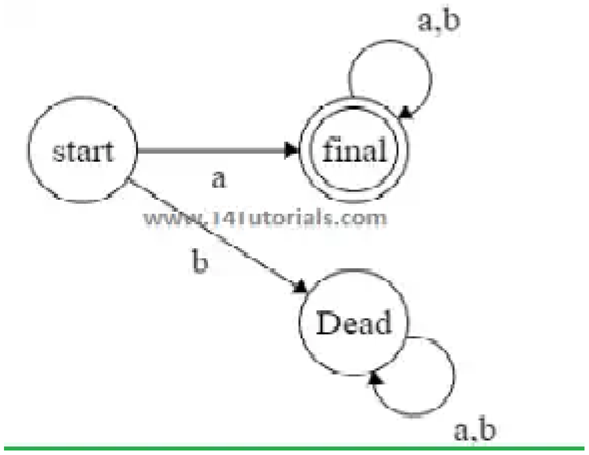
$\delta: Q \times \sum \to 2^Q$



Present State	Next State Of Input 0	Next State For Input 1
->q0	q0, q1	q0, q2
q1	q3	Nill
q2	Nill	q3
*q3	Nill	Nill

Design a DFA with Dead States.

We need to start a machine (DFA) to read a string. If machine reached successfully till its final string accepting state, then we say that string is accepted by our machine. But if we reach on a state where machine can't move further to its final state, then this state is called dead state. Dead state is also known as dummy state.



What is the computational problem and give some of the examples?

A computational problem is a problem that may be solved by an algorithm.

- Computing the greatest common divisor of a pair of integers.
- Computing the least common multiple of a pair of integers.
- Finding the shortest path between a pair of nodes in a finite graph.
- Determining whether a propositional formula is a tautology.

For example, the problem of factoring

"Given a positive integer n, find a nontrivial prime factor of n."
is a computational problem. A computational problem can be viewed as a set of instances or cases together with a, possibly empty, set of solutions for every instance/case. For example, in the factoring problem, the instances are the integers n, and solutions are prime numbers p that are the nontrivial prime factors of n.

Prove that, If |w|=n, then the number substrings are $\sum_0^n +1$

For example, w = cse

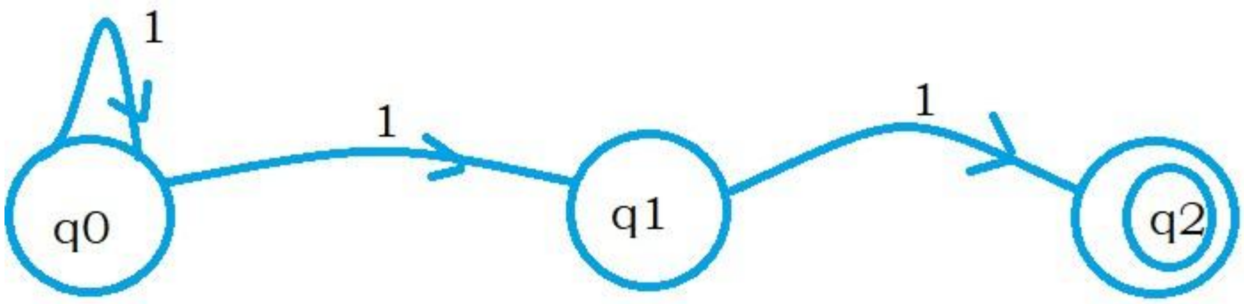
| cse | = 3

∑0 = {ε}
∑1 = {c, s. e}
∑2 = {cs, se. ce}
∑3 = {cse}

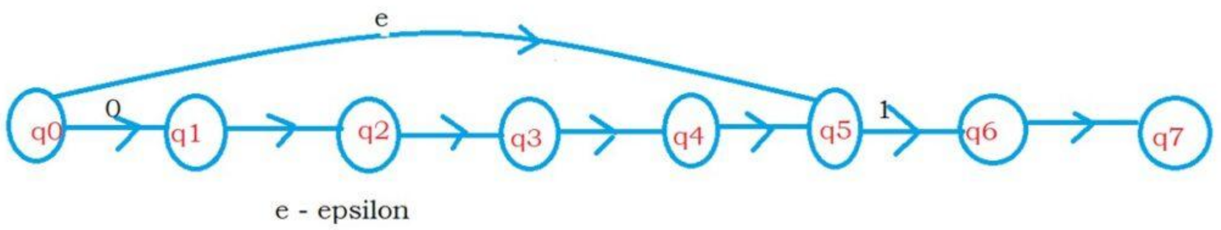
Substrings are {ε, c, s, e, cs, se, ce, cse}.

Design a NFA-Ɛ with example.

M={Q,Σ,q0,F,δ}
Q is a finite set of states.
Σ is a finite set of input symbols.
q0 is the initial state.
F is a finite set of final states. In our example, we have one final state qf.
δ is the transition function. We represent transition function as δ: QXΣ->Q.
Condition 1: One input symbol causes to move more than one state.



After seeing input symbol one, we move to states q0 and q1.
One machine took the next input symbol and applied it on state q0.
Another machine takes the next input symbol and applies it on state q1.
Condition 2: It is not compulsory that all the states have to consume all the input symbols.
NFA is doing parallel execution.
The same way NFA with epsilon moves is also the same as NFA but with one extra condition.
Condition 3: Epsilon Moves
We are moving on states without receiving any input symbol.
The below diagram shows the NFA with epsilon moves.

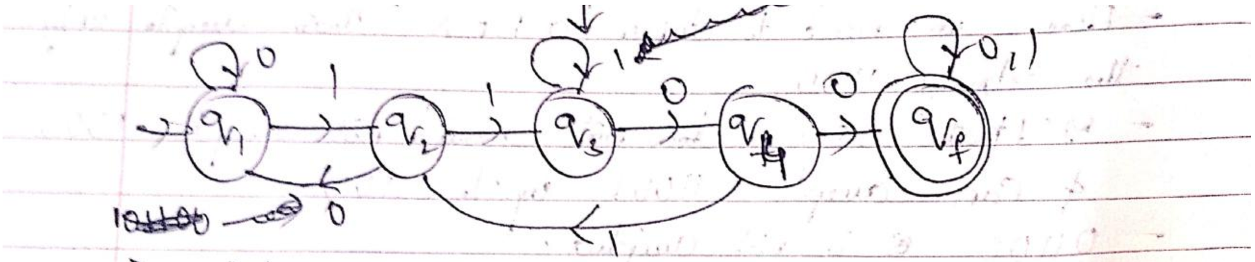


3.
Design a DFA which accepts set of all strings containing 1100 as a substring with in an alphabet ∑ = {0, 1}.

A Deterministic Finite Automaton (DFA) is defined as a 5-tuple $(Q, \Sigma, \delta, s, F)$ consisting of

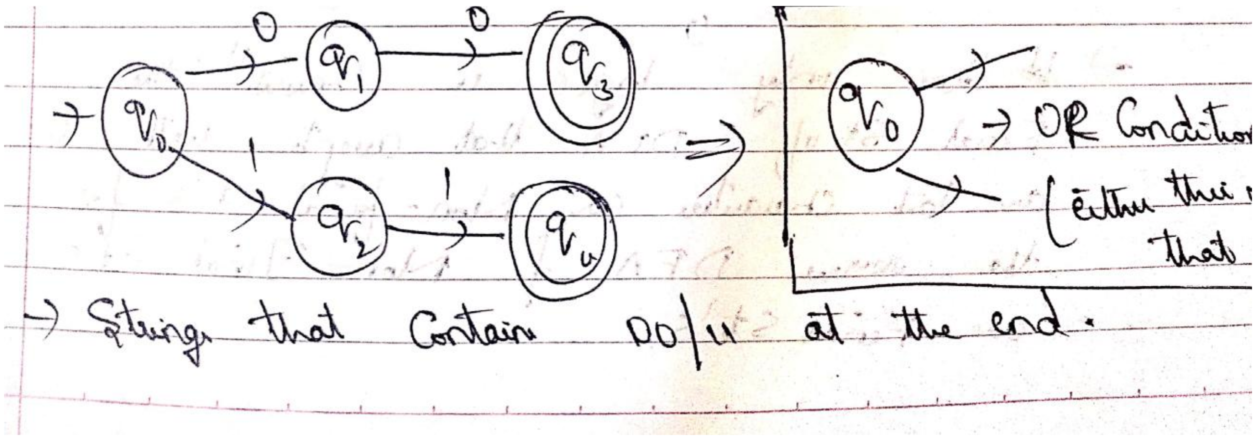
- A finite set Q (the set of states)
- A finite set of symbols Σ (the input alphabet)
- A transition function $\delta: Q \times \Sigma \rightarrow Q$ mapping the current state $q \in Q$ and input symbol $a \in \Sigma$ to a new state $\delta(q, a) \in Q$
- An initial state $s \in Q$ (the start state)
- A set of accepting states F (the final states)

A DFA is a mathematical model of a simple computational device that reads a string of symbols over the input alphabet Σ, and either accepts or reject the input string. We would like to turn this mathematical definition into a working program, so that we can run DFAs on our computer.

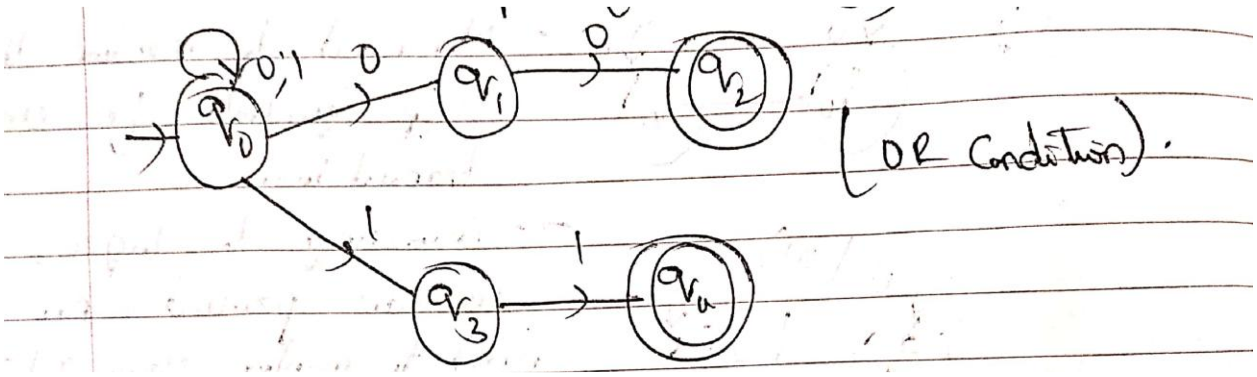


→ For '0', the m/c should be move to next st, beca-
-use here the string is 1100 & ~~to~~ the m/c
should move only from '1'.
→ So, for all 0's in the beginning or for
a single 0 the m/c has a self move
from q_1 to q_1
→ From '1' the m/c moves from q_1 to q_2.

Design an NFA which accepts strings which ends with 00|11 within an alphabet ∑ = {0, 1}.



- NFA with OR Condition



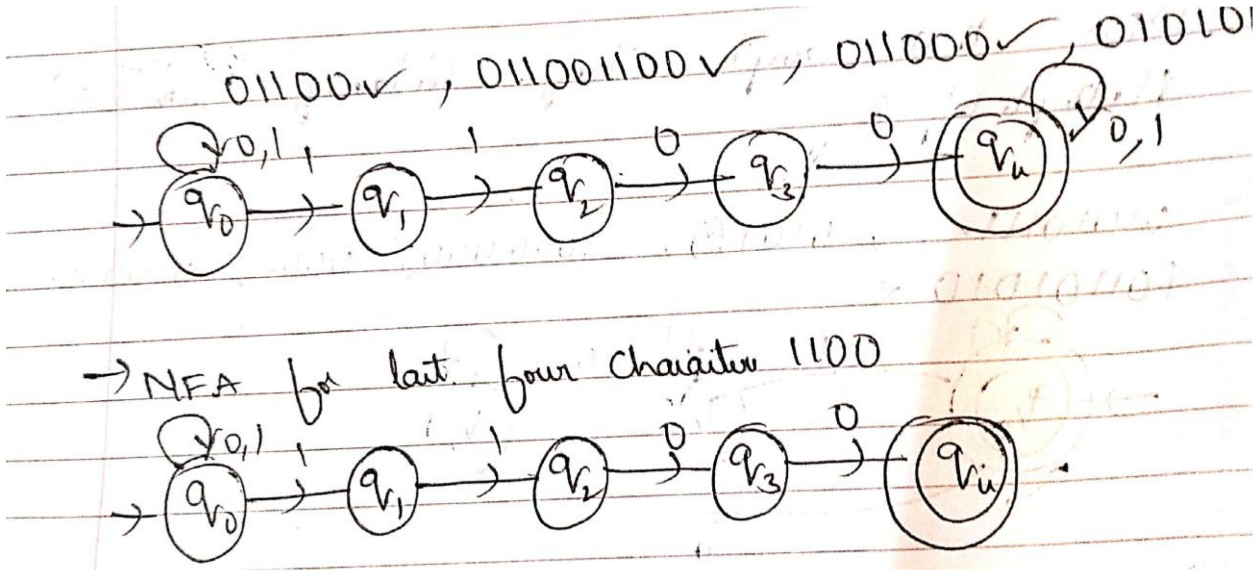
- The self-move with input symbols o,1 of q0 can accept any number 0's and 1's.
- The machine moving from q0 to q1 with input symbol 0 and q1 to q2 with input symbol 0 accepts the strings that end with 00.
- The machine moving from q0 to q3 with input symbol 1 and q3 to q4 with input symbol 1 accepts the strings that end with 11.

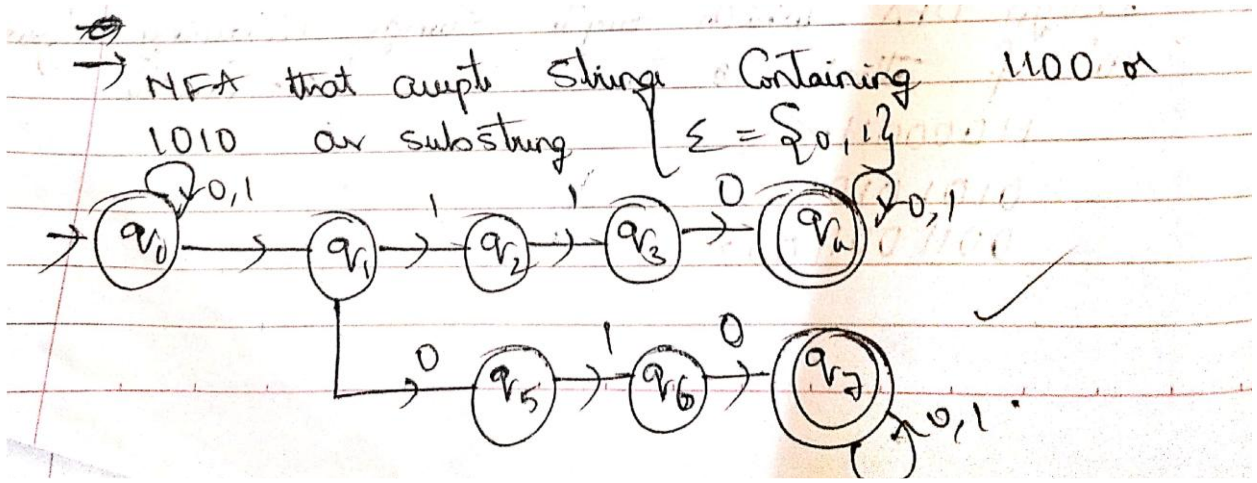
Design an NFA which accepts strings with 1100 or 1010 as a substring with in an alphabet ∑ = {0, 1}.

- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition.

Q: finite set of states
∑: finite set of the input symbol
q0: initial state
F: **final** state
δ: Transition function

$\delta : Q \times \sum \rightarrow 2^Q$


01100✓, 011001100✓, 011000✓, 01010
0,1
1
1
0
0
0,1
q_0 q_1 q_2 q_3 q_4
→ NFA for last four character 1100
0,1
1
1
0
0
q_0 q_1 q_2 q_3 q_4

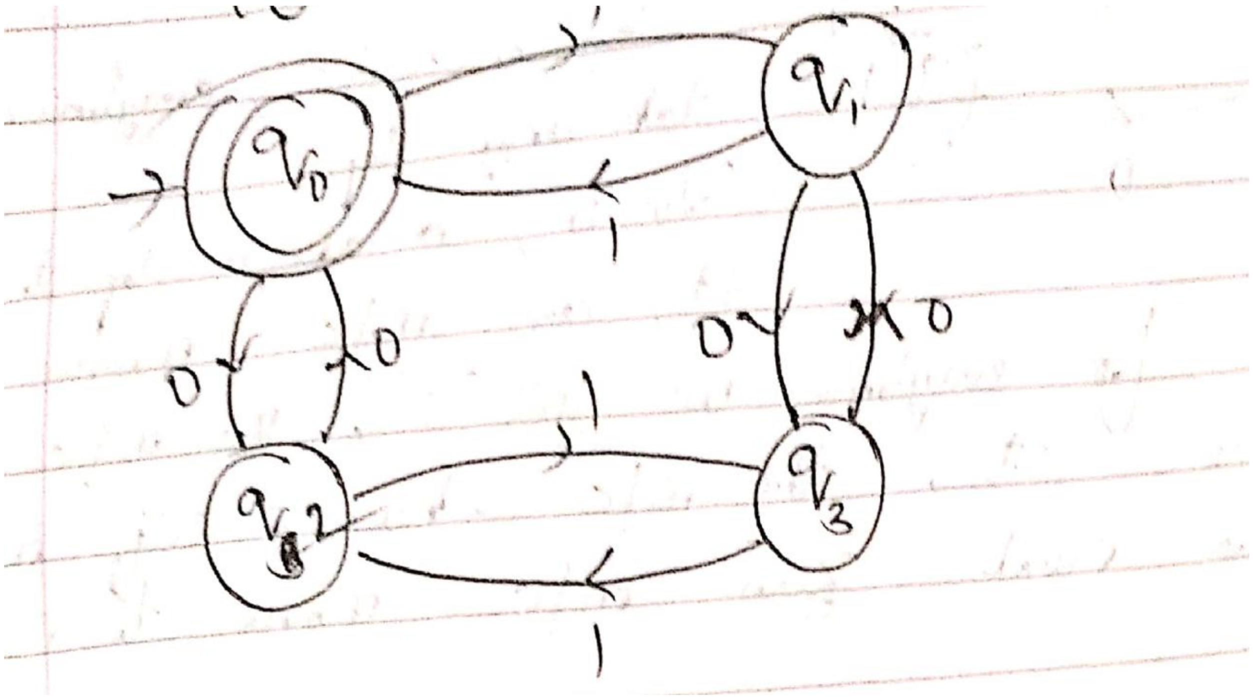

→ NFA that accepts strings containing 1100 or 1010 as substring { Σ = {0,1}
0,1
1
1
0
0,1
0
1
0
0,1
q_0 q_1 q_2 q_3 q_4 q_5 q_6 q_7

Design a DFA which accepts set of all strings containing even number 0's and 1's within an alphabet $\sum = \{0, 1\}$.

A Deterministic Finite Automaton (DFA) is defined as a 5-tuple $(Q, \Sigma, \delta, s, F)$ consisting of

- A finite set Q (the set of states)
- A finite set of symbols Σ (the input alphabet)
- A transition function $\delta: Q \times \Sigma \rightarrow Q$ mapping the current state $q \in Q$ and input symbol $a \in \Sigma$ to a new state $\delta(q, a) \in Q$
- An initial state $s \in Q$ (the start state)
- A set of accepting states F (the final states)

A DFA is a mathematical model of a simple computational device that reads a string of symbols over the input alphabet Σ, and either accepts or reject the input string. We would like to turn this mathematical definition into a working program, so that we can run DFAs on our computer.



Design an DFA which accepts strings which ends with 00|11 within an alphabet ∑ = {0, 1}.

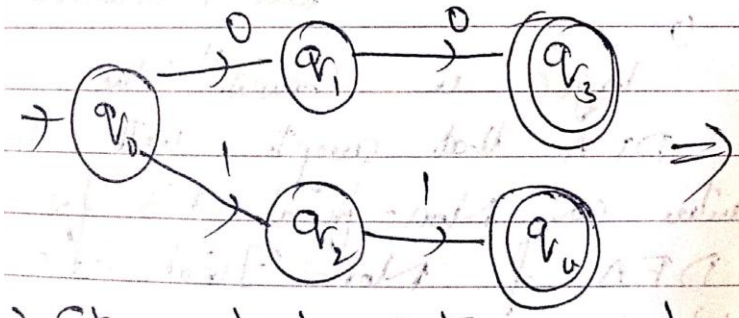
A Deterministic Finite Automaton (DFA) is defined as a 5-tuple $(Q, \Sigma, \delta, s, F)$ consisting of

- A finite set Q (the set of states)
- A finite set of symbols Σ (the input alphabet)
- A transition function $\delta: Q \times \Sigma \rightarrow Q$ mapping the current state $q \in Q$ and input symbol $a \in \Sigma$ to a new state $\delta(q, a) \in Q$
- An initial state $s \in Q$ (the start state)
- A set of accepting states F (the final states)

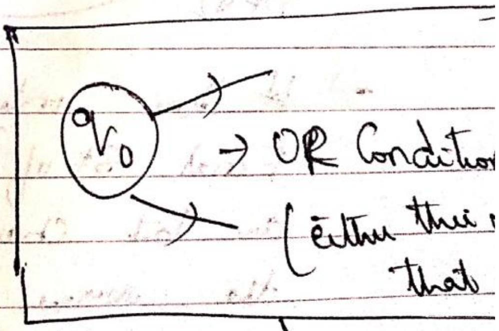
A DFA is a mathematical model of a simple computational device that reads a string of symbols over the input alphabet Σ, and either accepts or reject the input string. We would like to turn this mathematical definition into a working program, so that we can run DFAs on our computer.

Σ = {0,1}

Eg: 010111 ✓ , 00000 ✓, 101010 ✗

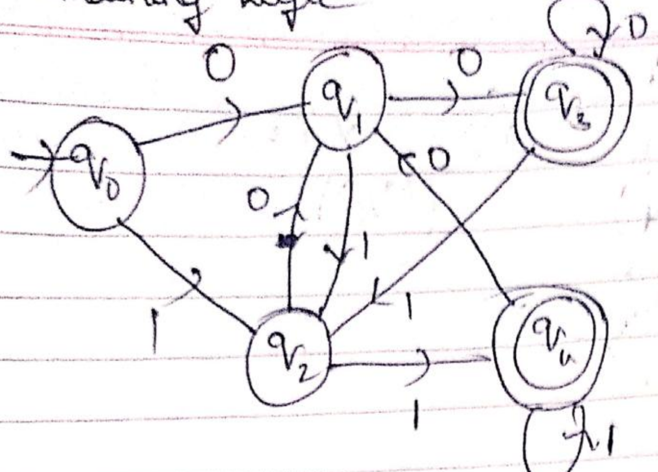


⇒



→ Strings that contain 00/11 at the end.

→ Remaining logic



$q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_4 \Rightarrow$ 011 ✓

$q_0 \xrightarrow{} q_1 \xrightarrow{} q_2 \xrightarrow{} q_4 \xrightarrow{} q_4 \Rightarrow$ 0111 ✓

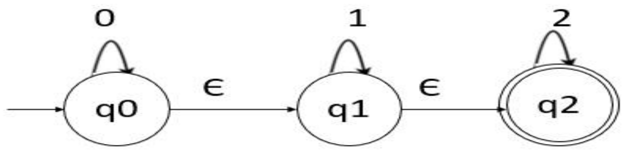
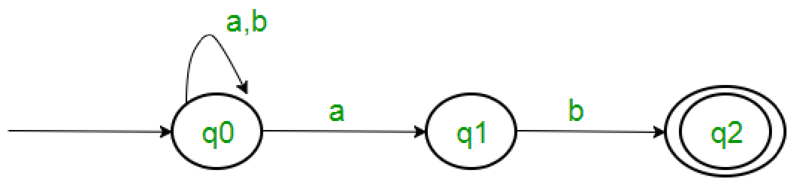
$q_0 \xrightarrow{1} q_2 \xrightarrow{0} q_1 \xrightarrow{0} q_3 \Rightarrow$ 100 ✓

$q_0 \xrightarrow{} q_2 \xrightarrow{} q_1 \xrightarrow{} q_3 \xrightarrow{} q_3 \Rightarrow$ 1000 ✓

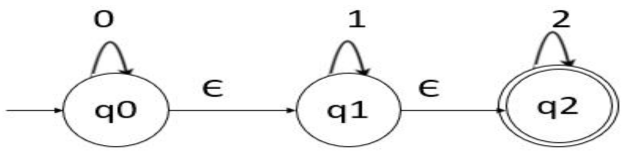
$q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_3 \xrightarrow{1} q_2 \xrightarrow{1} q_4 \Rightarrow$ 0011 ✓

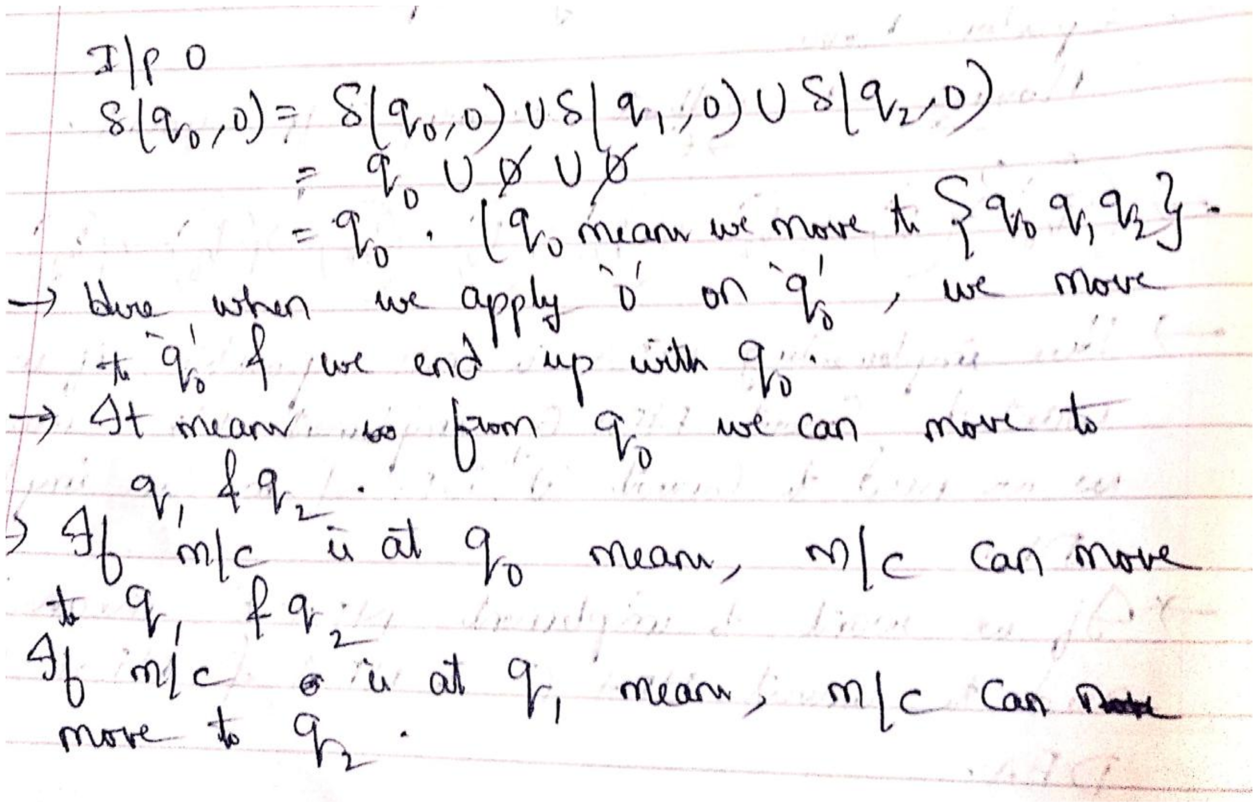
$q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_3 \xrightarrow{1} q_2 \xrightarrow{1} q_4 \xrightarrow{1} q_4 \Rightarrow$ 00111 ✓

4.

Convert the following NFA-Ɛ in to NFA, 
a) Design a DFA which accepts set of all strings that starts with 1 and ends with 0 within an alphabet ∑ = {0, 1}.
b) Design an NFA with Multiple Final States.
Convert the following NFA in to DFA, 

Convert the following NFA-Ɛ in to NFA,





I/p 0

$\delta(q_0,0) = \delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0)$

$= q_0 \cup \emptyset \cup \emptyset$

$= q_0$. (q_0 means we move to $\{q_0\, q_1\, q_2\}$.

→ Here when we apply '0' on 'q_0', we move to 'q_0' & we end up with q_0.

→ It means, ~~so~~ from q_0 we can move to q_1 & q_2.

→ If m/c is at q_0 means, m/c can move to q_1 & q_2

If m/c is at q_1 means, m/c can move to q_2.

Expression

ϵ-Closure $(q_0) = \{q_0 q_1 q_2\}$

" $(q_1) = \{q_1, q_2\}$

" $(q_2) = \{q_2\}$.

$\delta \rightarrow$ Transition fun of NFA-ϵ, $\delta' \rightarrow$ Transition fun of NFA.

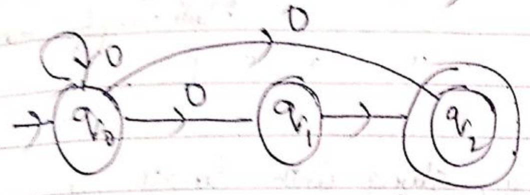
$\delta'(q_0, 0) = \epsilon$-Closure$(\delta(\epsilon$-Closure$(q_0), 0)$

$\vdash \epsilon$-Closure$(\delta(q_0 q_1 q_2), 0)$

$\vdash \epsilon$-Closure$(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$

$\vdash \epsilon$-Closure$(q_0 \cup \emptyset \cup \emptyset)$

$\vdash \epsilon$-closure$(q_0) = \{q_0, q_1, q_2\}$



$\delta'(q_0, 1) = \epsilon$-Closure$(\delta(\epsilon$-Closure$(q_0), 1))$

$\vdash \epsilon$-Closure$(\delta(q_0 q_1 q_2)\}, 1))$

$\vdash \epsilon$-Clos$(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1))$

$\vdash \epsilon$-Clos$(\emptyset \cup q_1, \emptyset \emptyset)$

$\vdash \epsilon$-Clos$(q_1) = \{q_1, q_2\}$

|| ly $\delta'(q_0, 2) = \{q_2\}$

$\delta'(q_1, 0) = \{\emptyset\}$

$\delta'(q_1, 1) = \{q_1, q_2\}$

$\delta'(q_1, 2) = \{q_2\}$

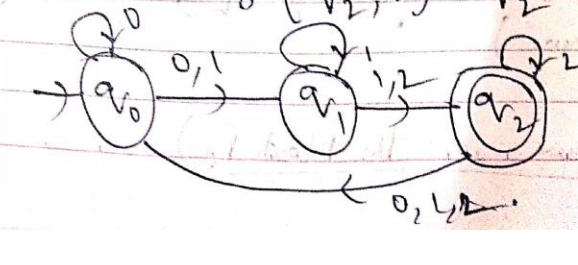
$\delta'(q_2, 0) = \emptyset$

$\delta'(q_2, 1) = \emptyset$

$\delta'(q_2, 2) = q_2$.

Transition Table - NFA (δ')

	0	1	2
q_0	$q_0 q_1 q_2$	$q_1 q_2$	q_2
q_1	$\emptyset$	$q_1 q_2$	q_2
q_2	$\emptyset$	$\emptyset$	q_2.

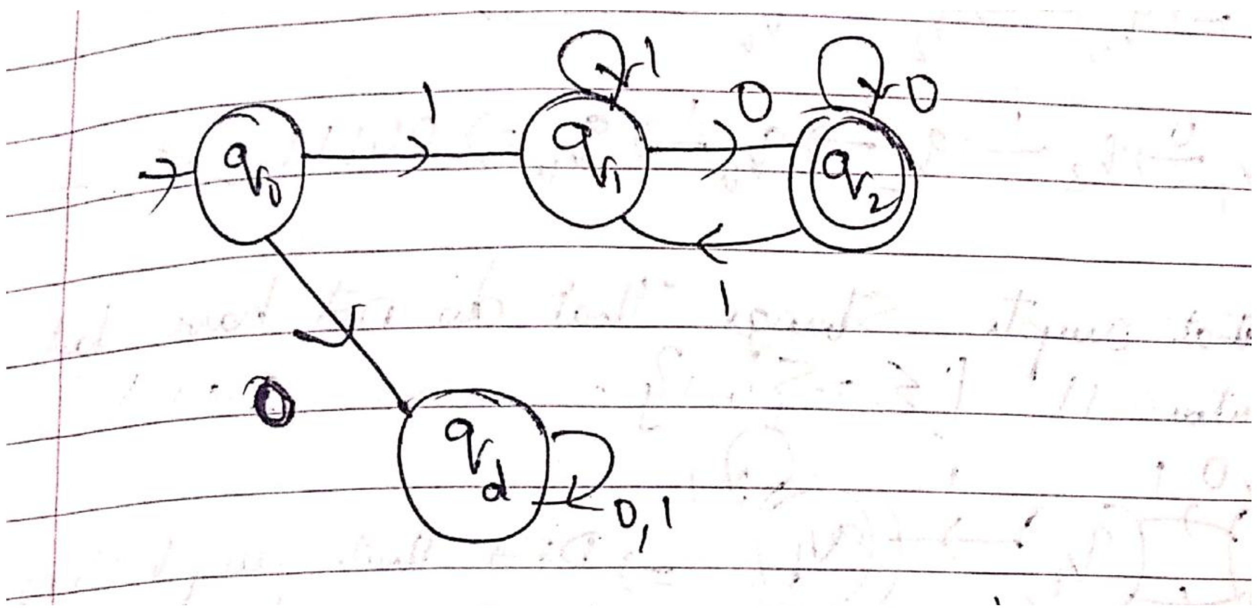


a) Design a DFA which accepts set of all strings that starts with 1 and ends with 0 within an alphabet ∑ = {0, 1}.

A Deterministic Finite Automaton (DFA) is defined as a 5-tuple $(Q, \Sigma, \delta, s, F)$ consisting of

- A finite set Q (the set of states)
- A finite set of symbols Σ (the input alphabet)
- A transition function $\delta: Q \times \Sigma \to Q$ mapping the current state $q \in Q$ and input symbol $a \in \Sigma$ to a new state $\delta(q, a) \in Q$
- An initial state $s \in Q$ (the start state)
- A set of accepting states F (the final states)

A DFA is a mathematical model of a simple computational device that reads a string of symbols over the input alphabet Σ, and either accepts or reject the input string. We would like to turn this mathematical definition into a working program, so that we can run DFAs on our computer.

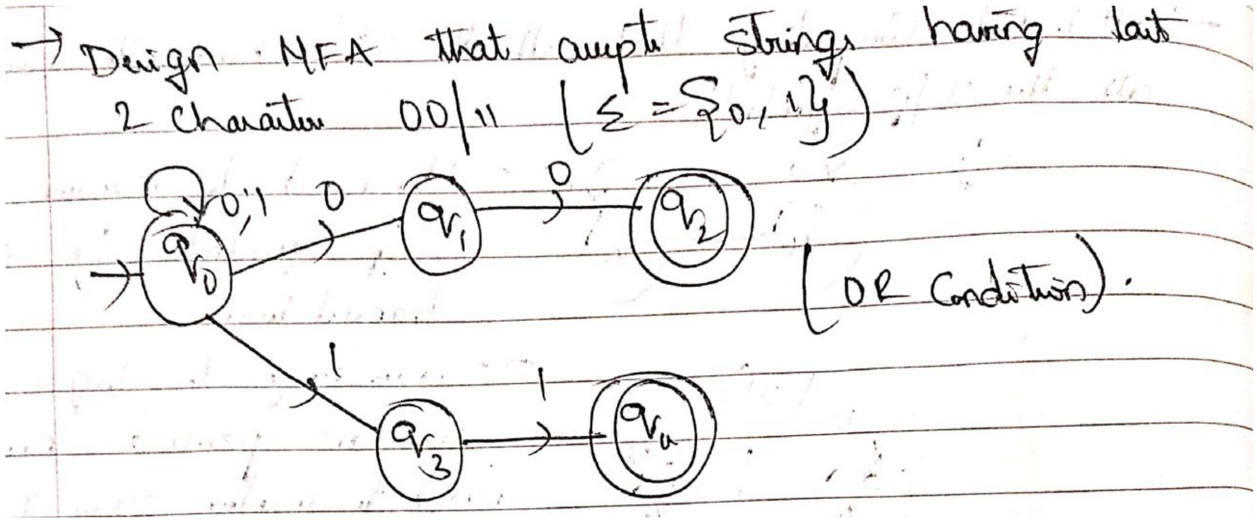


b) Design an NFA with Multiple Final States.

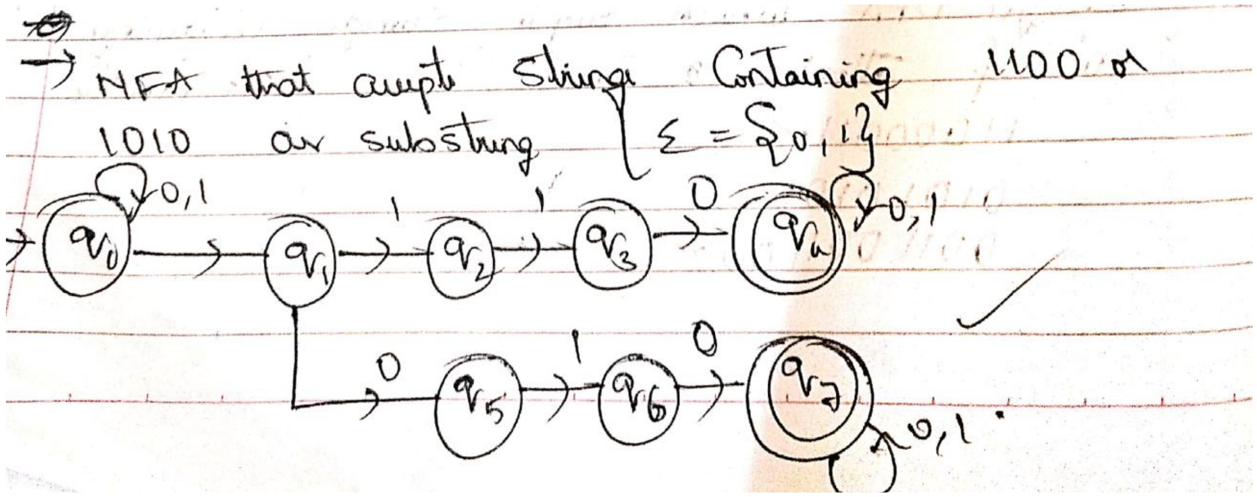
- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition.

Q: finite set of states
∑: finite set of the input symbol
q0: initial state
F: **final** state
δ: Transition function

$\delta: Q \times \Sigma \to 2^Q$

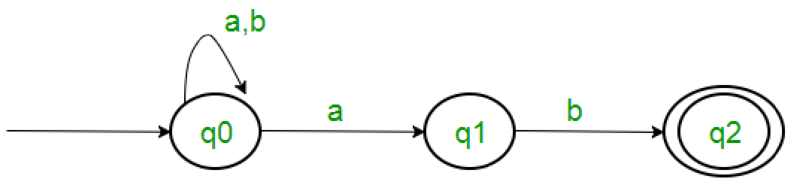


→ Design NFA that accepts strings having last 2 character 00/11 $(\Sigma = \{0,1\})$



→ NFA that accepts strings Containing 1100 or 1010 as substring $(\Sigma = \{0,1\})$

Convert the following NFA in to DFA,



An NFA can have zero, one or more than one move from a given state on a given input symbol. An NFA can also have NULL moves (moves without input symbol). On the other hand, DFA has one and only one move from a given state on a given input symbol.

Steps for converting NFA to DFA:

- **Step1: Convert the given NFA to its equivalent transition table**

  To convert the NFA to its equivalent transition table, we need to list all the states, input symbols, and the transition rules.
- The transition rules are represented in the form of a matrix, where the rows represent the current state, the columns represent the input symbol, and the cells represent the next state.
- **Step2: Create the DFA's start state**

  The DFA's start state is the set of all possible startin
- g states in the NFA. This set is called the "epsilon closure" of the NFA's start state.
- The epsilon closure is the set of all states that can be reached from the start state by following epsilon (?) transitions.

Step3: Create the DFA's transition table

- The DFA's transition table is similar to the NFA's transition table, but instead of individual states, the rows and columns represent sets of states.
- For each input symbol, the corresponding cell in the transition table contains the epsilon closure of the set of states obtained by following the transition rules in the NFA's transition table.

Step4: Create the DFA's final states

The DFA's final states are the sets of states that contain at least one final state from the NFA.

Step5: Simplify the DFA

The DFA obtained in the previous steps may contain unnecessary states and transitions. To simplify the DFA, we can use the following techniques:

- **Remove unreachable states:** States that cannot be reached from the start state can be removed from the DFA.
- **Remove dead states:** States that cannot lead to a final state can be removed from the DFA.
- **Merge equivalent states:** States that have the same transition rules for all input symbols can be merged into a single state.

Step6: Repeat steps 3-5 until no further simplification is possible

After simplifying the DFA, we repeat steps 3-5 until no further simplification is possible.
The final DFA obtained is the minimized DFA equivalent to the given NFA.

Example: Consider the following NFA shown in Figure 1.

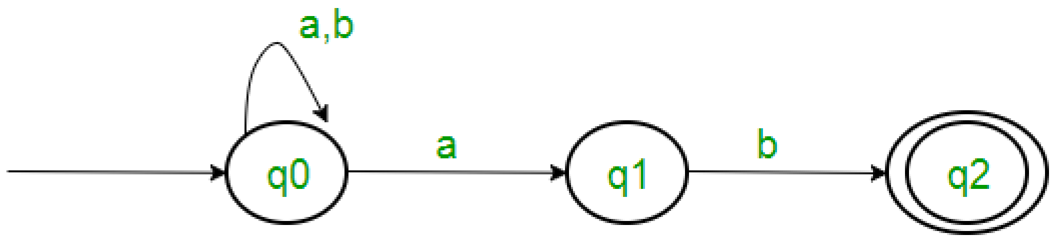
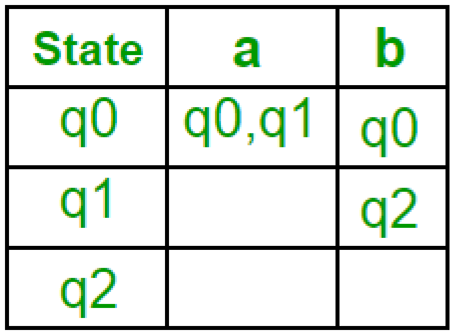


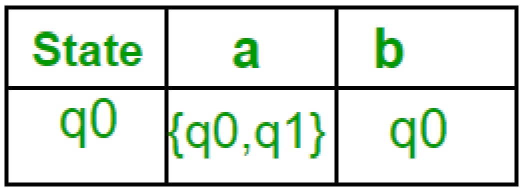
Figure 1

Following are the various parameters for NFA. Q = { q0, q1, q2 } ? = (a, b) F = { q2 } ? (Transition Function of NFA)



State	a	b
q0	q0,q1	q0
q1		q2
q2		

Step 1: Q’ = ? Step 2: Q’ = {q0} Step 3: For each state in Q’, find the states for each input symbol. Currently, state in Q’ is q0, find moves from q0 on input symbol a and b using transition function of NFA and update the transition table of DFA. ?’ (Transition Function of DFA)



State	a	b
q0	{q0,q1}	q0

Now { q0, q1 } will be considered as a single state. As its entry is not in Q’, add it to Q’. So Q’ = { q0, { q0, q1 } } Now, moves from state { q0, q1 } on different input symbols are not present in transition table of DFA, we will calculate it like: ?’ ({ q0, q1 }, a) = ? (q0, a) ? ? (q1, a) = { q0, q1 } ?’ ({ q0, q1 }, b) = ? (q0, b) ? ? (q1, b) = { q0, q2 } Now we will update the transition table of DFA. ?’ (Transition Function of DFA)

State	a	B
q0	{q0,q1}	q0
{q0,q1}	{q0,q1}	{q0,q2}

Now { q0, q2 } will be considered as a single state. As its entry is not in Q’, add it to Q’. So Q’ = { q0, { q0, q1 }, { q0, q2 } } Now, moves from state {q0, q2} on different input symbols are not present in transition table of DFA, we will calculate it like: ?’ ({ q0, q2 }, a) = ? (q0, a) ? ? (q2, a) = { q0, q1 } ?’ ({ q0, q2 }, b) = ? (q0, b) ? ? (q2, b) = { q0 } Now we will update the transition table of DFA. ?’ (Transition Function of DFA)

State	a	B
q0	{q0,q1}	q0
{q0,q1}	{q0,q1}	{q0,q2}
{q0,q2}	{q0,q1}	q0

As there is no new state generated, we are done with the conversion. Final state of DFA will be state which has q2 as its component i.e., { q0, q2 } Following are the various parameters for DFA. Q’ = { q0, { q0, q1 }, { q0, q2 } } ? = (a, b) F = { { q0, q2 } } and transition function ?’ as shown above. The final DFA for above NFA has been shown in Figure 2.

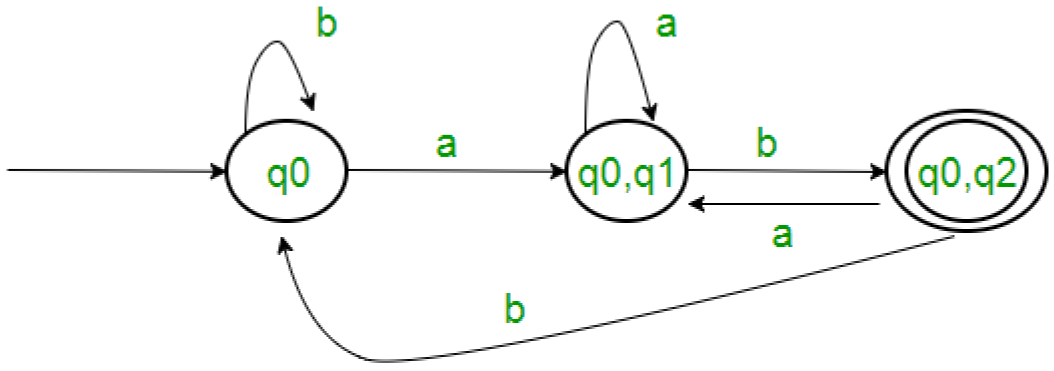


Figure 2