MODULE 1

FINITE AUTOMATA

1. The Central Concepts of Automata Theory

Symbol: Symbol is an entity or user defined entity. It can be a single digit, a single alphabet, or a single symbol.

Ex: 1, a, b, #

Alphabet: Alphabet is a finite set of symbols. It is denoted by ∑.

Ex: ∑ = {a, b}

∑ = {0, 1, 2, +}

∑ = {#, β, Δ}

String: It is a finite sequence of symbols over an alphabet ∑. The string is denoted by w.

Ex: Let ∑ = {#, β, Δ} ,

the valid set of strings w1 = # , w2 = β , w3 = Δ

w4 = #β , w5 = #Δ , w6 = βΔ,

w7 = #βΔ, w8 = ### etc

Length of string: number of symbols in the string. It is represented as |w|.

Ex: if string w = # , then |w| = 1

Ex: if string w = #βΔ , then |w| = 3

Empty String: A string with zero symbols. It is represented as ε or **Λ**.

Ex: if string w = ε , then |w| = 0

Prefix of a String: It is any number of leading symbols of the string.

Ex: if string w = #βΔ, then

prefixes are ε , # , #β , #βΔ

proper prefixes are ε , # , #β

Suffix of a String: It is any number of trailing symbols of the string.

Ex: if string w = #βΔ, then

suffixes are #βΔ , βΔ , Δ , ε

proper suffixes are βΔ , Δ , ε

Concatenation of Strings: If w1 and w2 are any two strings, then their concatenation is represented as "w1.w2". Concatenation is represented with "."

Ex: if w1 = #β, w2 = Δ, then w1.w2 = #β Δ

Language: Let the finite automata M be defined over the alphabet ∑. The formal language of M is a set comprising of valid strings over the alphabet ∑. It is represented as L.

Ex: Let M be the finite automata with all strings over ∑ = {#, β, Δ}

then L = { ε , #, β , Δ , #β, #Δ, βΔ, #βΔ, ###   . . . . . }

Ex: Let M be the finite automata with all strings of length two over ∑ = {#, β, Δ}

then L = { #β, #Δ, βΔ }

Grammar: An "automata grammar" typically refers to a formal grammar used to define the language accepted by an automaton. This grammar is a set of rules that specifies the structure of valid strings or sequences of symbols that the automaton can recognize or generate.

Operations on Languages:

a) Union – Let L1 and L2 be two languages. Then their union is a language comprising of all strings from both L1 and L2. It represented as L1 U L2.

Ex: Let L1 = { #β , #Δ } , L2 = { Δ , #βΔ }

L1 U L2 = { #β , #Δ , Δ , #βΔ }

b) Concatenation – Let L1 and L2 be two languages. Then their concatenation is a language comprising of all strings from L1 concatenated with L2. It represented as L1.L2 = { X.Y | X ϵ L1 , Y ϵ L2 }

Ex: Let L1 = { #β , #Δ } , L2 = { Δ , #βΔ }

L1.L2 = { #βΔ , #β#βΔ , #ΔΔ , #Δ#βΔ }

c) Closure Operations

i) Kleene Closure (Σ^*) – Set of all possible combination of strings including ε

Ex: Let ∑ = {# , β , Δ}

then Σ^* = { ε , # , β , Δ , #β , #Δ , βΔ , #βΔ , ###   . . . . . }

ii) Positive Closure (Σ^+) – Set of all possible combination of strings excluding ε

Ex: Let ∑ = {#, β, Δ}

then Σ^+ = { #, β , Δ , #β , #Δ , βΔ , #βΔ , ###   . . . . . }

2. Problems on Automata Languages

Problem 1: Determine the language over ∑ = {#, β, Δ} comprising of all strings that contain at least one Δ.

Answer : L = { Δ , #Δ , βΔ , ##Δ , #Δ# , Δ## , ΔΔ# , Δ#Δ , ΔΔΔ , #βΔ , #Δβ , ... }

Problem 2: Determine the language over ∑ = {#, β, Δ} comprising of all strings in which the 2nd symbol is Δ.

Answer : L = { #Δ , βΔ , ΔΔ , #Δ# , ΔΔ# , ΔΔΔ , #Δβ , ... }

3. Introduction to Automata

An automaton is a “system where energy, materials and information are transformed, transmitted and used for performing some functions without direct participation of man”. Automaton is a discrete machine. Figure 1.1 shows the model of discrete automaton.

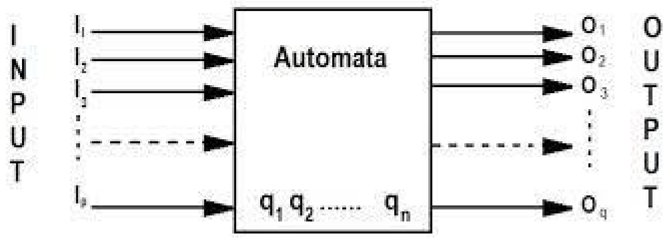


Figure 1.1 Model of Discrete Automaton

The characteristics of discrete automaton are:

(i) **Input** – At each of the discrete instants of time t1, t2, . . . tm , the input values I1 , I2 . . . . Ip are applied as input to model. Each input symbol takes a value from the input alphabet ∑.

iii) **Output** – O1 , O2 , ... Oq are the outputs of the model; each output symbol takes a value from the output alphabet O.

(iii) **States** – At any instant of time the automaton can be in one of the states q1, q2 , . . . , qn.

(iv) **State relation** – At any instant of time, the next state of the automaton is determined by the present state and the present input.

(v) **Output relation** – The output is related to either state only or to both the input and the state. It should be noted that at any instant of time the automaton is in some state. On ‘reading’ an input symbol, the automaton moves to a next state which is given by the state relation.

4. Introduction to Finite Automata

FA is a mathematical model of a system with discrete input and output. Finite automata (FA), often referred to as Finite State Machines (FSMs), represent a fundamental concept in theoretical computer science and are essential in understanding computation and formal language theory. They serve as abstract mathematical models of computation with a finite set of states and a well-defined set of transitions between these states based on input symbols.

FA serve as fundamental models of computation with a finite set of states and transitions between these states based on input symbols. These models help computer scientists and researchers understand the nature of computation, formal languages, and the fundamental limits of what can be efficiently computed within the realm of regular languages.

FA finds **applications** in various domains of computer science, including:

a) Lexical analysis in compilers: They are used to recognize and tokenize strings based on specific patterns or regular expressions.

b) Modeling and understanding regular languages and their properties.

c) Pattern matching:

- ✓ Automata aid in searching for patterns within text or sequences efficiently
- ✓ for scanning large bodies of text, such as web pages to find occurrence of words or other pattens

d) Network protocols:

- ✓ Finite automata are employed in protocol specifications and network traffic analysis.
- ✓ for verifying systems of all types that have a finite number of distinct states such as communication protocols

e) Software: for designed and checking behavior of digital circuit;

f) Hardware: to implement switches

At its core, a finite automaton consists of:

a) **States**: These are distinct configurations or conditions in which the automaton can exist at any given point during its operation.

b) **Transitions**: These depict the movement between states based on input symbols. Transitions are governed by a set of rules or a transition function.

c) **Accepting States** (optional): In certain types of finite automata, there might be specific states that are designated as accepting or final states, indicating successful recognition of an input string.

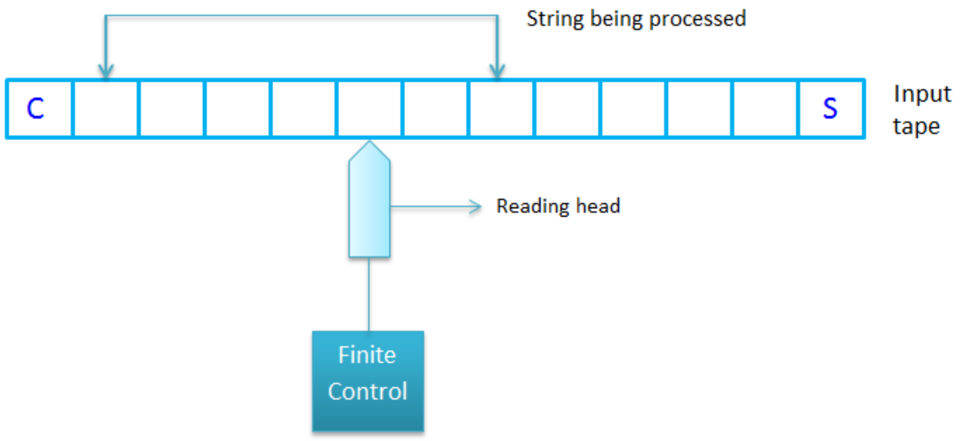


Figure 1.2 Block diagram of the Finite Automaton

A FA operates by transitioning between different states in response to the input symbols it reads. It follows a set of rules defined by its structure and transition function. Figure 1.2 shows the block diagram of the finite automaton.

Finite Automata can recognize and accept strings that belong to the languages they are designed for (e.g., regular languages for DFAs and NFAs). The step-by-step explanation of how a Finite Automaton operates:

a) **States**: The FA starts in a designated initial state from a set of finite states. Each state represents a particular configuration or condition of the automaton at a given moment.

b) **Transition Function**: The FA has a transition function that defines the rules for transitioning between states based on the input symbols it receives. This function specifies the next state the FA moves to when it reads a particular input symbol while being in a certain state. <u>The transition function can be represented in the form of a transition table or a transition diagram or a transition relation.</u>

c) **Reading Input**: As the FA processes an input string symbol by symbol, it transitions between states according to the transition function. For each symbol it reads from the input, the automaton moves from its current state to a new state as determined by the transition function.

d) **Acceptance (for DFAs):** In the case of a Deterministic Finite Automaton (DFA), if the input string is completely processed and the automaton ends up in one of its designated accepting (or final) states, the input string is accepted by the automaton. Otherwise, if the automaton ends up in a non-accepting state or cannot transition further, the input string is rejected.

e) **Behavior (for NFAs):** For a Nondeterministic Finite Automaton (NFA), the process is similar, but with more flexibility. An NFA might have multiple possible transitions for a given state and input symbol. It can be in multiple states simultaneously, and it accepts an input string if there exists at least one path that leads to an accepting state.

f) **Completion of Input:** Once the entire input string is processed, the FA halts, and its final state or states determine whether the input string is accepted or rejected based on the language it recognizes.

5. Types of Finite Automata

Finite automata are classified into two main types:

a) **Deterministic Finite Automata (DFA):** These machines accept or reject strings of symbols by transitioning between states according to a single, unique transition for each input symbol. DFAs recognize regular languages and are defined by a precise set of rules for each state and input symbol combination.

b) **Nondeterministic Finite Automata (NFA):** Unlike DFAs, NFAs can have multiple possible transitions for a given state and input symbol. They are more flexible in their behavior, allowing transitions to multiple states simultaneously or the option to "guess" the correct path. NFAs recognize the same class of languages as DFAs, but their design and operation are more versatile.

6. Mathematical Model of DFA

A DFA can be represented mathematically as a 5-tuple:

M = (Q , Σ , δ , q0 , F)

Where Q: finite nonempty set of states

Σ: finite nonempty set of the input symbols called input alphabet

q0: initial state q0 ∈ Q

F: set of final states F ⊆ Q

δ: Transition function **Q X Σ → Q**

<u>Every transition in a DFA has only one possible next state.</u>

Example DFA: Let the DFA be M = (Q , Σ , δ , q4 , F)

where Q = { q4 , q7 } , Σ = { +, X } , q4 is initiate state , F = {q4}

δ is defined as

<u>Transition Relation</u>

δ (q4 , +) = q7

δ (q4 , X) = q4

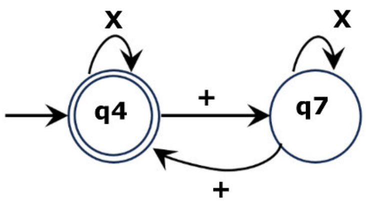
δ (q7 , +) = q4

δ (q7 , X) = q7

<u>Transition Table</u>

Present State/Σ	**Next State**	
	+	X
➔ q4	q7	q4
q7	q4	q7

<u>Transition Diagram</u>



7. Mathematical Model of NFA

A NFA can be represented mathematically as a 5-tuple:

M = (Q , Σ , δ , q0 , F)

Where Q: finite nonempty set of states

Σ: finite nonempty set of the input symbols called input alphabet

q0: initial state q0 ∈ Q

F: set of final states F ⊆ Q

δ: Transition function **Q X Σ → 2^{Q}**

<u>A transition in an NFA can have more than one possible next state.</u>

Example NFA: Let the NFA be M = (Q , Σ , δ , q4 , F)

where Q = { q4 , q7 } , Σ = { +, X } , q4 is initiate state , F = {q4}

δ is defined as

Transition Relation

δ (q4 , +) = q7

δ (q4 , X) = q4

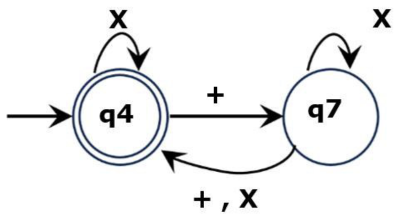
δ (q7 , +) = q4

δ (q7 , X) = q4 , q7

Transition Table

Present State/Σ	**Next State**	
	+	X
→ $\bigcirc$q4	q7	q4
q7	q4	q4 , q7

Transition Diagram

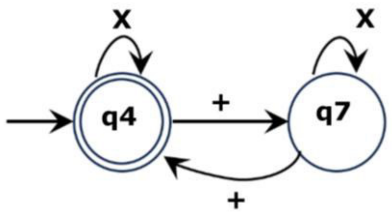


8. Acceptability of Strings

Let the DFA be M = (Q , Σ , δ , q0 , F). A string X is said to be accepted by M if δ (q0, X) = q where q ∈ F.

A string is said to be accepted by NFA if there exists at least one completed path that ends with a final state. Let the MFA be M = (Q , Σ , δ , q0 , F). A string X is said to be accepted by M if δ (q0, X) contains some final state.

Example Problem 1: For the following DFA, determine the acceptability of the string XX+X.



Answer: Let the given DFA be M = M = (Q , Σ , δ , q4 , F) where Q = { q4 , q7 } ,

Σ = { + , X } , q4 is initiate state , F = { q4 }

δ is defined as

Transition Relation

δ (q4 , +) = q7

δ (q4 , X) = q4

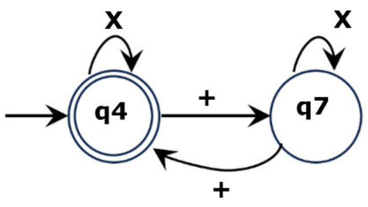
δ (q7 , +) = q4

δ (q7 , X) = q7

Transition Table

Present State/Σ	Next State	
	+	X
→ q4	q7	q4
q7	q4	q7

Transition Diagram



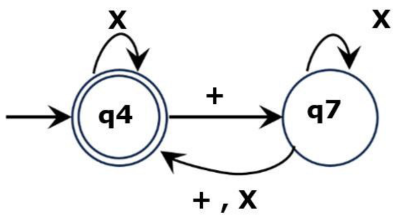
Acceptability of the string "XX+X"

δ (q4 , XX+X) = δ (q4 , X+X) = δ (q4 , +X) = δ (q7 , X) = q7

$\uparrow \qquad \uparrow \qquad \uparrow \qquad \uparrow$

As q7 is not a final state, string "XX+X" is not accepted by the given FA.

Example Problem 2: For the following NFA, determine the acceptability of the string XX+X.



Answer: Let the given NFA be M = (Q , Σ , δ , q4 , F) where Q = { q4 , q7 } ,

Σ = { + , X } , q4 is initiate state , F = { q4 }

δ is defined as

Transition Relation

δ (q4 , +) = q7

δ (q4 , X) = q4

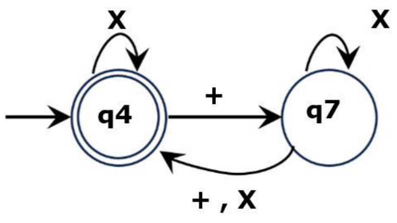
δ (q7 , +) = q4

δ (q7 , X) = q4 , q7

Transition Table

Present State/Σ	Next State	
	+	X
→ q4	q7	q4
q7	q4	q4 , q7

Transition Diagram



Acceptability of the string "XX+X"

$\delta (q4 , \underset{\uparrow}{X}X{+}X) = \delta (q4 , \underset{\uparrow}{X}{+}X) = \delta (q4 , \underset{\uparrow}{+}X) = \delta (q7 , \underset{\uparrow}{X}) = q4$

As q4 is a final state, string “XX+X” is accepted by the given FA.

9. Problems on Design of FA

Problem 1: Design a DFA that recognizes the language over { v , p } containing strings that start with 'v' and have an odd length.

Answer: Let the FA that recognizes the language over { v , p } containing strings that start with 'v' and have an odd length be

M = (Q , Σ , δ , q0 , F) where Q = { q4 , q5 , q6 } ,

Σ = { v , p } , q4 is initiate state , F = { q5 }

δ is defined as

Transition Relation

δ (q4 , v) = q5

δ (q5 , v) = q6

δ (q5 , p) = q6

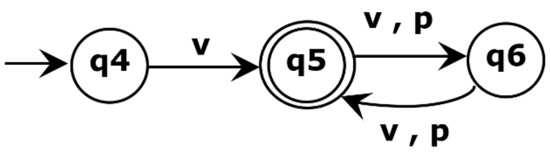
δ (q6 , v) = q5

δ (q6 , p) = q5

Transition Table

Present State/Σ	**Next State**	
	v	p
→ q4	q5	
q5	q6	q6
q6	q5	q5

Transition Diagram



Acceptability of the string “vpvpp”

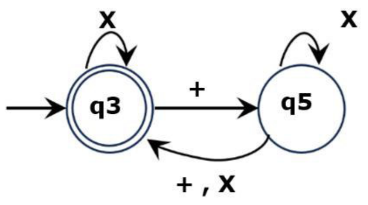
$\delta (q4 , \underset{\uparrow}{v}pvpp) = \delta (q5 , \underset{\uparrow}{p}vpp) = \delta (q6 , \underset{\uparrow}{v}pp) = \delta (q5 , \underset{\uparrow}{p}p) = \delta (q6 , \underset{\uparrow}{p}) = q5$

As q5 is a final state, string “vpvpp” is accepted by the given FA.

10. Equivalence of NFA and DFA

Note: δ' ([q0 , q1 , ... , qn] , a) = δ (q0 , a) U δ (q1 , a) U . . . . U δ (qn , a)

Example Problem: Convert the following NFA to equivalent DFA.



Answer: Let the given NFA be M = (Q , Σ , δ , q3 , F) where Q = { q3 , q5 } ,

Σ = { + , X } , q3 is initiate state , F = { q3 }

δ is defined as

Transition Relation

δ (q3 , +) = q5

δ (q3 , X) = q3

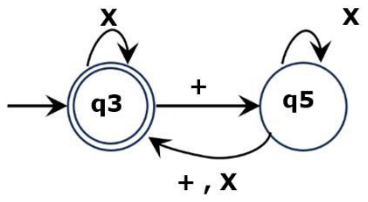
δ (q5 , +) = q3

δ (q5 , X) = q3 , q5

Transition Table

Present State/Σ	**Next State**	
	+	X
→ (q3)	q5	q3
q5	q3	q3 , q5

Transition Diagram



Let the equivalent DFA be M′ = (Q′ , Σ , δ′ , q3 , F′)

where Q′ = { [q3] , [q5] , [q3 , q5] } , Σ = { + , X } , [q3] is initiate state ,
F′ = { [q3] , [q3 , q5] }

δ′ is defined as

Transition Relation

δ′ (q3 , +) = [q5]

δ′ (q3 , X) = [q3]

δ′ (q5 , +) = [q3]

δ′ (q5 , X) = [q3 , q5]

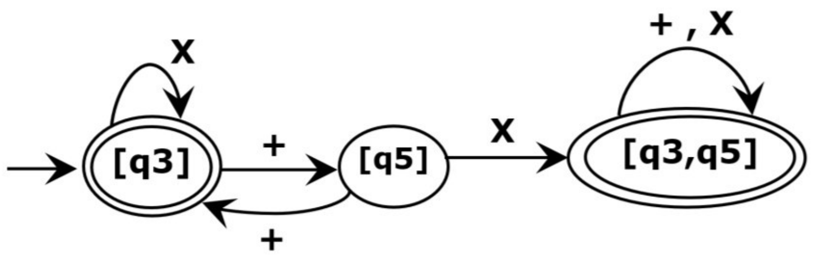
δ′ ([q3 , q5] , +) = [q3 , q5]

δ′ ([q3 , q5] , X) = [q3 , q5]

Transition Table

Present State/Σ	**Next State**	
	+	X
→ ([q3])	[q5]	[q3]
[q5]	[q3]	[q3 , q5]
([q3,q5])	[q3 , q5]	[q3 , q5]

Transition Diagram



11. NFA with Ɛ-transitions (NFA-Ɛ)

It is an NFA including transitions on the empty input symbol Ɛ.

An NFA-Ɛ can be represented mathematically as a 5-tuple:

M = (Q , Σ , δ , q0 , F)

Where Q: finite nonempty set of states

Σ: finite nonempty set of the input symbols called input alphabet

q0: initial state $q0 \in Q$

F: set of final states $F \subseteq Q$

δ: Transition function **Q X { Σ x Ɛ } → 2^Q**

Example NFA-Ɛ: Let the NFA-Ɛ be M = (Q , Σ , δ , q4 , F)

where Q = { q4 , q7 } , Σ = { + , X } , q4 is initiate state , F = { q4 }

δ is defined as

Transition Relation

δ (q4 , Ɛ) = q7

δ (q4 , X) = q4

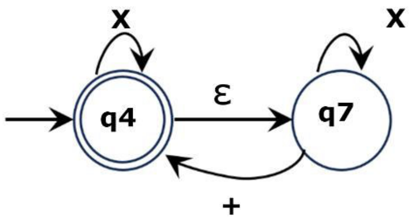
δ (q7 , +) = q4

δ (q7 , X) = q7

Transition Table

Present State/Σ	**Next State**		
	+	X	Ɛ
→ $\bigcirc$ q4		q4	q7
q7	q4	q7	

Transition Diagram



12. Conversion of NFA-Ɛ to NFA

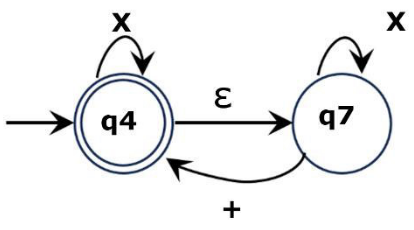
Note: Ɛ-closure(q) = set of all states p such that there is a path from q to p with label Ɛ.

Ɛ-closure(q) includes "q" itself.

$$\hat{\delta}(q, \varepsilon) = \varepsilon - closure(q)$$

$$\hat{\delta}(q, \mathrm{a}) = \varepsilon - closure\Big(\delta(\hat{\delta}\,(q\,, \varepsilon\,), a)\Big)$$

Example Problem: Convert the following NFA-ɛ to equivalent NFA without ɛ-transitions.



Answer: Let the NFA-Ɛ be M = (Q , Σ , δ , q4 , F)

where Q = { q4 , q7 } , Σ = { + , X } , q4 is initiate state , F = { q4 }

δ is defined as

Transition Relation

δ (q4 , Ɛ) = q7

δ (q4 , X) = q4

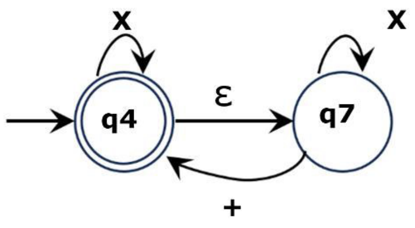
δ (q7 , +) = q4

δ (q7 , X) = q7

Transition Table

Present State/Σ	Next State		
	+	X	Ɛ
→ $\circ$q4		q4	q7
q7	q4	q7	

Transition Diagram



Step 1: Find Ɛ-closure(q) for all states q ∈ Q

Ɛ-closure(q4) = $\hat{\delta}(q4\,, \varepsilon)$ = { q4 , q7 }

Ɛ-closure(q7) = $\hat{\delta}(q7\,, \varepsilon)$ = { q7 }

Step 2: Find the mapping function $\hat{\delta}(q)$ for all states q ∈ Q

$$\hat{\delta}(q4\,, +) = \varepsilon - closure\left(\delta\big(\hat{\delta}(q4, \varepsilon), +\big)\right)$$

$$= \varepsilon - closure\big(\delta(\varepsilon - \mathrm{closure}(q4), +)\big)$$

$$= \varepsilon - closure\big(\delta(\{q4\,,q7\},+)\big)$$

$$= \varepsilon - closure\big(\delta(q4,+) \;\cup\; \delta(q7,+)\big)$$

$$= \varepsilon - closure(\,\{\,\emptyset\,\} \cup \{\,q4\,\})$$

$$= \varepsilon - closure(\,\{\,q4\,\})$$

$$= \{\,q4\,,q7\,\}$$

$$\hat{\delta}(q4\,,X) = \varepsilon - closure\left(\delta\big(\hat{\delta}(q4,\varepsilon),X\big)\right)$$

$$= \varepsilon - closure\big(\delta(\varepsilon - \text{closure}(q4),X)\big)$$

$$= \varepsilon - closure\big(\delta(\{q4\,,q7\},X)\big)$$

$$= \varepsilon - closure\big(\delta(q4,X) \;\cup\; \delta(q7,X)\big)$$

$$= \varepsilon - closure(\,\{\,q4\,\} \cup \{\,q7\,\})$$

$$= \varepsilon - closure(\,\{\,q4\;\,,q7\,\})$$

$$= \varepsilon - closure(\,q4) \;\cup\; \varepsilon - closure(\,q7)$$

$$= \{\,q4\,,q7\,\} \cup \{\,q7\,\}$$

$$= \{\,q4\,,q7\,\}$$

$$\hat{\delta}(q7\,,+) = \varepsilon - closure\left(\delta\big(\hat{\delta}(q7,\varepsilon),+\big)\right)$$

$$= \varepsilon - closure\big(\delta(\varepsilon - \text{closure}(q7),+)\big)$$

$$= \varepsilon - closure\big(\delta(\{q7\}\,,+)\big)$$

$$= \varepsilon - closure(\,\{\,q4\,\})$$

$$= \{\,q4\,,q7\,\}$$

$$\hat{\delta}(q7\,,X) = \varepsilon - closure\left(\delta\big(\hat{\delta}(q7,\varepsilon),X\big)\right)$$

$$= \varepsilon - closure\big(\delta(\varepsilon - \text{closure}(q7),X)\big)$$

$$= \varepsilon - closure\big(\delta(\{q7\}\,,X)\big)$$

$$= \varepsilon - closure(\,\{\,q7\,\})$$

$$= \{\,q7\,\}$$

Let the NFA without ε-transitions be M′ = (Q , Σ , δ′ , q4 , F′)

where Q = { q4 , q7 } , Σ = { + , X } , q4 is initiate state ,

$F' = F \cup \{ q \text{ , if } \varepsilon\text{-closure}(q) \in F \} = \{ q4 \} \cup \{ \Phi \} = \{ q4 \}$

δ' is defined as

Transition Relation

$\hat{\delta}(q4, +) == \{ q4, q7 \}$

$\hat{\delta}(q4, X) == \{ q4, q7 \}$

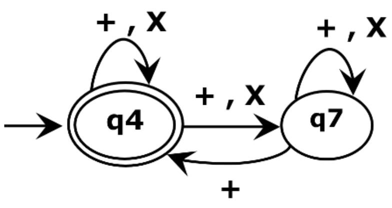
$\hat{\delta}(q7, +) == \{ q4, q7 \}$

$\hat{\delta}(q7, X) == \{ q7 \}$

Transition Table

Present State/Σ	**Next State**	
	+	X
➔ $\circ$ q4	q4, q7	q4, q7
q7	q4, q7	q7

Transition Diagram



13. FA with Output

Finite Automata with outputs are extensions of traditional Finite Automata (FA) where transitions not only depend on the current state and input symbol but also produce outputs during state transitions. These outputs could be associated with transitions or states themselves. FA with output is of two types - Mealy and Moore machines.

a) Moore Machine: The output is associated with each state rather than with transitions. Upon entering a state, the machine produces an output determined by that state.

b) Mealy Machine: Outputs are associated with transitions, meaning the output depends on both the current state and the input symbol. Outputs are produced when a transition occurs from one state to another due to an input symbol.

Moore Machine:

In Moore machine, output function *Z(t)* depends only on the present state *q(t)* and is independent of the current input. "*t*" is a discrete instant of time.

$$Z(t) = \lambda \ (q(t))$$

A Moore machine is represented mathematically using a 6-tuple:

$$M = (Q , \Sigma , \Delta , \delta , \lambda , q0)$$

Where Q: finite nonempty set of states

Σ: finite nonempty set of the input symbols called input alphabet

Δ: finite nonempty set of the output symbols called output alphabet

q0: initial state $q0 \in Q$

δ: Transition function **$Q \times \Sigma \rightarrow Q$**

λ: output function **Q → Δ**

<u>For a Moore machine if the input string is of length "n", the output string is of length "n+1".</u>

Example Moore Machine:

Let the Moore Machine be M = (Q , Σ , Δ , δ , λ , q4)

where Q = { q4 , q7 } , Σ = { + , X } , Δ = { P , M } , q4 is initiate state

δ is defined as

<u>Transition Relation</u>

δ (q4 , +) = q7

δ (q4 , X) = q4

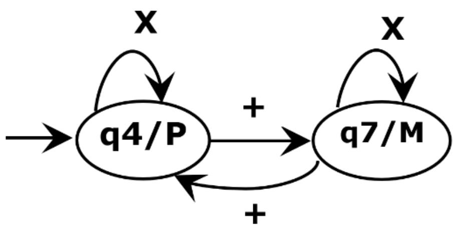
δ (q7 , +) = q4

δ (q7 , X) = q7

<u>Transition Table</u>

Present State/Σ	**Next State δ**		**Output λ**
	+	X	
→ q4	q7	q4	P
q7	q4	q7	M

<u>Transition Diagram</u>



<u>Output Function λ</u>

λ (q4) = P

λ (q7) = M

Problem: For the above Moore Machine, determine the output for the input string "XX+X".

δ (q4 , XX+X) = δ (q4 , X+X) = δ (q4 , +X) = δ (q7 , X) = q7

↑ ↑ ↑ ↑

λ (q4) -------→ λ (q4) -------→ λ (q4) -------→ λ (q7) --→ λ (q7)

Output **P** **P** **P** **M** **M**

Output for the given input string "XX+X" is "PPPMM"

Mealy Machine:

In Mealy machine, output function *Z(t)* depends on the present state *q(t)* and the current input *x(t)*.

Z(t) = λ (q(t) , x(t))

A Mealy machine is represented mathematically using a 6-tuple:

M = (Q , Σ , Δ , δ , λ , q0)

Where Q: finite nonempty set of states

Σ: finite nonempty set of the input symbols called input alphabet

Δ: finite nonempty set of the output symbols called output alphabet

q0: initial state   q0 ∈ Q

δ: Transition function   **Q X Σ → Q**

λ: output function **Q X Σ → Δ**

For a Mealy machine if the input string is of length "n", the output string is of length "n".

Example Mealy Machine:

Let the Mealy Machine be M = (Q , Σ , Δ , δ , λ , q4)

where Q = { q4 , q7 } , Σ = { + , X } , Δ = { P , M } , q4 is initiate state

δ is defined as

Transition Relation

δ (q4 , +) = q7

δ (q4 , X) = q4

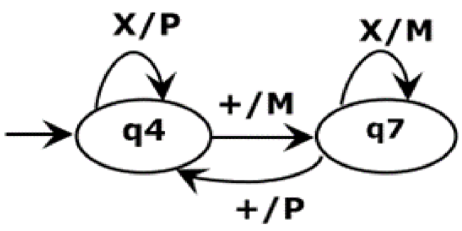
δ (q7 , +) = q4

δ (q7 , X) = q7

Transition Table

Present State/Σ	+		X	
	Next State δ	**Output λ**	**Next State** δ	**Output λ**
→ q4	q7	M	q4	P
q7	q4	P	q7	M

Transition Diagram



Output Function λ

λ (q4 , +) = M

λ (q4 , X) = P

λ (q7 , +) = P

λ (q7 , X) = M

Problem: For the above Mealy Machine, determine the output for the input string "XX+X".

δ (q4 , XX+X) = δ (q4 , X+X) = δ (q4 , +X) = δ (q7 , X) = q7

↑ ↑ ↑ ↑

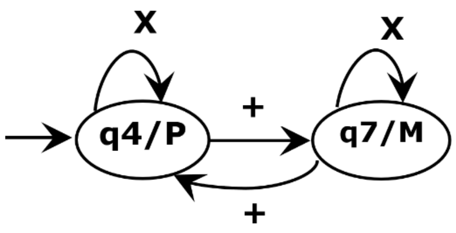
λ (q4 , X) ---→ λ (q4 , X) ---→ λ (q4 , +) --→ λ (q7 , X)

Output **P** **P** **M** **M**

Output for the given input string "XX+X" is "PPMM"

14. Converting Moore Machine to Mealy Machine

Problem: Convert the following Moore Machine to Mealy Machine.



Answer:

Let the given Moore Machine be M = (Q , Σ , Δ , δ , λ , q4)

where Q = { q4 , q7 } , Σ = { + , X } , Δ = { P , M } , q4 is initiate state

δ is defined as

Transition Relation

δ (q4 , +) = q7

δ (q4 , X) = q4

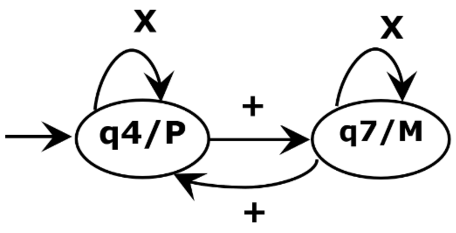
δ (q7 , +) = q4

δ (q7 , X) = q7

Transition Table

Present State/Σ	**Next State δ** +	X	**Output λ**
➔ q4	q7	q4	P
q7	q4	q7	M

Transition Diagram



Output Function λ

λ (q4) = P

λ (q7) = M

Let the equivalent Mealy Machine be M′ = (Q , Σ , Δ , δ , λ′ , q4)

where Q = { q4 , q7 } , Σ = { + , X } , Δ = { P , M } , q4 is initiate state

δ is defined as

Transition Relation

δ (q4 , +) = q7

δ (q4 , X) = q4

δ (q7 , +) = q4

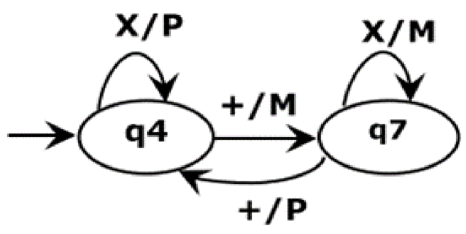
δ (q7 , X) = q7

Transition Table

Present State/Σ	+		X	
	Next State δ	**Output λ**	**Next State** δ	**Output λ**
➔ q4	q7	M	q4	P
q7	q4	P	q7	M

$\boxed{= \lambda (q4)}$ $\boxed{= \lambda (q7)}$

Transition Diagram



Output Function λ'

λ (q4 , +) = M

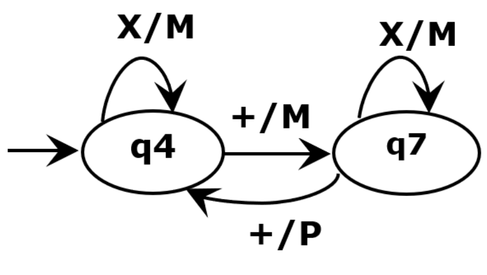
λ (q4 , X) = P

λ (q7 , +) = P

λ (q7 , X) = M

15. Converting Mealy Machine to Moore Machine

Problem: Convert the following Mealy Machine to Moore Machine.



Answer:

Let the given Mealy Machine be M = (Q , Σ , Δ , δ , λ , q4)

where Q = { q4 , q7 } , Σ = { + , X } , Δ = { P , M } , q4 is initiate state

δ is defined as

Transition Relation

δ (q4 , +) = q7

δ (q4 , X) = q4

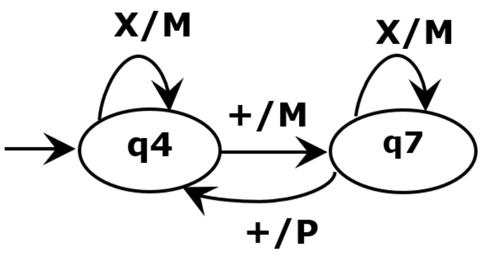
δ (q7 , +) = q4

δ (q7 , X) = q7

Transition Table

Present State/Σ	+		X	
	Next State δ	**Output λ**	**Next State** δ	**Output λ**
➔ q4	q7	M	q4	M
q7	q4	P	q7	M

Transition Diagram



Output Function λ

λ (q4 , +) = M

λ (q4 , X) = M

λ (q7 , +) = P

λ (q7 , X) = M

Let the equivalent Moore Machine be M′ = (Q′ , Σ , Δ , δ′ , λ′ , q4′)

δ′ is defined as

Transition Table

Present State/Σ	+		X	
	Next State δ	**Output λ**	**Next State** δ	**Output λ**
➔ q4M	q7	M	q4M	M
q4P	q7	M	q4M	M
q7	q4P	P	q7	M

Present State/Σ	+		X	
	Next State δ	**Output λ**	**Next State** δ	**Output λ**
➔ q4′	q7	M	q4M	M
q4M	q7	M	q4M	M
q4P	q7	M	q4M	M
q7	q4P	P	q7	M

Present State/Σ	**Next State δ**		**Output λ**
	+	X	
➔ q4′	q7	q4M	ε
q4M	q7	q4M	M
q4P	q7	q4M	P
q7	q4P	q7	M

<u>Transition Relation</u>

δ (q4′ , +) = q7

δ (q4′ , X) = q4M

δ (q4M , +) = q7

δ (q4M , X) = q4M

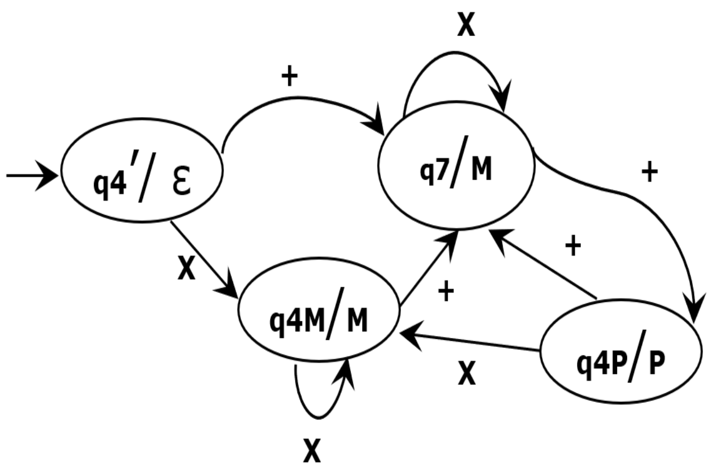
δ (q4P , +) = q7

δ (q4P , X) = q4M

δ (q7 , +) = q4P

δ (q7 , X) = q7

<u>Transition Diagram</u>



where Q = { q4′ , q4M , q4P, q7 } , Σ = { + , X } , Δ = { P , M } , q4′ is initiate state

<u>Output Function λ′</u>

λ (q4′) = ε

λ (q4M) = M

λ (q4P) = P

λ (q7) = M

Module-II

Regular Expressions, Grammar and Languages

Finite automata can only recognize regular languages, which are a particularly restricted category of languages.

Regular Expressions

Regular languages are denoted by regular expressions.

An expression is regular if:

- ϕ is a regular expression for regular language ϕ.
- ε is a regular expression for regular language $\{\varepsilon\}$.
- If $a \in \Sigma$ (Σ represents the input alphabet), a is regular expression with language $\{a\}$.
- If a and b are regular expression, $a + b$ is also a regular expression with language $\{a, b\}$.
- If a and b are regular expression, ab (concatenation of a and b) is also regular.
- If a is regular expression, a^* (0 or more times a) is also regular.

Identities for regular expression –

The regular expression has multiple identities. Consider the regular expressions p, q, and r.

- $\emptyset + r = r$
- $\emptyset.r = r.\emptyset = \emptyset$
- $\in.r = r.\in = r$
- $\in^* = \in$ and $\emptyset^* = \in$
- $r + r = r$
- $r^*.r^* = r^*$
- $r.r^* = r^*.r = r^+$.
- $(r^*)^* = r^*$
- $\in + r.r^* = r^* = \in + r.r^*$
- $(p.q)^*.p = p.(q.p)^*$
- $(p + q)^* = (p^*.q^*)^* = (p^* + q^*)^*$
- $(p + q).r = p.r + q.r$ and $r.(p+q) = r.p + r.q$

Regular Expression

Regular languages are denoted by Regular Expressions.

An expression isregular if:

- ϕ is a regular expression for regular language ϕ.
- ε is a regular expression for regular language $\{\varepsilon\}$.
- If $a \in \Sigma$ (Σ represents the input alphabet), a is regular expression withlanguage $\{a\}$.
- If a and b are regular expression, a + b is also a regular expression with language $\{a, b\}$.
- If a and b are regular expression, ab (concatenation of a and b) is alsoregular.
- If a is regular expression, a* (0 or more times a) is also regular.

Closure Properties of Regular Languages

Union : If L1 and If L2 are two regular languages, their union L1 ∪ L2 willalso be regular.

For example,

$L1 = \{a^n \mid n \geq 0\}$ and $L2 = \{b^n \mid n \geq 0\}$

$L3 = L1 \cup L2 = \{a^n \cup b^n \mid n \geq 0\}$ is also regular.

Intersection : If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular. For example,

$L1= \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ and $L2= \{a^m b^n \cup b^n a^m \mid n \geq 0 \text{ and } m \geq 0\}$

$L3 = L1 \cap L2 = \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ is also regular.

Concatenation : If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular. For example,$L1 = \{a^n \mid n \geq 0\}$ and $L2 = \{b^n \mid n \geq 0\}$

$L3 = L1.L2 = \{a^m . b^n \mid m \geq 0 \text{ and } n \geq 0\}$ is also regular.

Kleene Closure : If L1 is a regular language, its Kleene closure L1* will also be regular. For example,

$L1 = (a \cup b)$

$L1^* = (a \cup b)^*$

Complement : If L(G) is regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L (G) from all possible strings.

For example, $L(G) = \{a^n \mid n > 3\}$

$L'(G) = \{a^n \mid n <= 3\}$

Note : Two regular expressions are equivalent if languages generated by them are same. For example, (a+b*)* and (a+b)* generate same language. Every string which is generated by (a+b*)* is also generated by (a+b)* andvice versa.

Applications of Regular expressions:

1. Defining Regular Languages:

Regular expressions are used to define regular languages. A regular language is a type of formal language that can be recognized by a finite automaton, and regular expressions provide a concise and expressive way to represent such languages.

2. Finite Automata:
Regular expressions and finite automata are closely related. Regular expressions can be converted into equivalent finite automata, and vice versa. This connection is formalized by the Kleene's Theorem, which states that a language is regular if and only if it can be described by a regular expression or recognized by a finite automaton.

3. Pattern Matching:
Regular expressions are used for pattern matching within strings. In the context of formal languages, this involves checking if a given string belongs to a particular regular language defined by a regular expression.

4. Lexical analysis:
Regular expressions are commonly used in the design of lexical analyzers (lexers) for compilers. Lexers are responsible for breaking down the source code into tokens, and regular expressions are used to describe the patterns of valid tokens in the programming languages.

5. Text Editors and Search algorithms:
Regular expressions are applied in text editors and search algorithms to efficiently find and manipulate patterns in textual data. This application is related to the concept of regular languages and the efficient algorithms for pattern matching.

6. String Matching algorithms:
Regular expressions are used as patterns in string matching algorithms. These algorithms, such as the Knuth-Morris-Pratt algorithm or the Boyer-Moore algorithm, are used to find occurrences of a pattern within a text.

7. Network Protocol Specification:
Regular expressions are used in the specification of network protocols and in the design of protocol analyzers. They help define the syntactic rules for valid messages or packets in a network communication protocol.

7. Database Query Languages:
Regular expressions are integrated into some database query languages to provide expressive pattern matching capabilities when searching or filtering data.

8. Automated String Processing:
Regular expressions are used in various automated string processing tasks, such as text processing, data validation, and information retrieval.

They provide a concise and powerful notation for describing patterns and sets of strings within the context of formal language theory.

Pumping Lemma:
Two Pumping Lemmas have been defined for the following:

1. Context – Free Languages
2. Regular Languages

Pumping Lemma for Regular Languages

For any regular language L, there exists an integer n, such that for all x ∈ L with |x| ≥ n, there exists u, v, w ∈ Σ∗, such that x = uvw, and

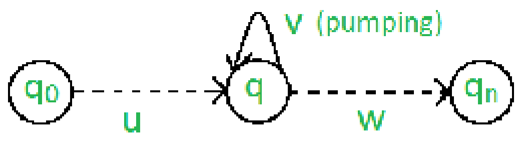
(1) |uv| ≤ n

(2) |v| ≥ 1

(3) for all i ≥ 0: uviw ∈ L

This basically indicates that even after a string v is "pumped," or added a number of times, the resulting string stays in L.
It is an evidence of language irregularity. Therefore, a language definitely fulfills pumping lemma if it is regular. L is undoubtedly not regular if it contains a minimum of one pumping string that is not in L.

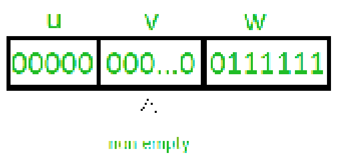
It's possible that the contrary isn't always true. That is, the language is not regular even if Pumping Lemma holds.



Ex: prove L01 = {0n1n | n ≥ 0} is irregular.
Assuming L is regular, the aforementioned rules derive from Pumping Lemma.

Now, let x ∈ L and |x| ≥ n. So, by Pumping Lemma, there exists u, v, w such that (1) – (3) hold.

We show that for all u, v, w, (1) – (3) does not hold.
If (1) and (2) hold then x = 0n1n = uvw with |uv| ≤ n and |v| ≥ 1.
So, u = 0a, v = 0b, w = 0c1n where : a + b ≤ n, b ≥ 1, c ≥ 0, a + b + c = n But, then (3) fails for i = 0
uv0w = uw = 0a0c1n = 0a + c1n ∉ L, since a + c ≠ n.



Applications of Pumping Lemma:

1. Proving Non-Regularity:

One of the primary applications of the Pumping Lemma is to prove that a given language is not regular. If a language cannot satisfy the conditions of the Pumping Lemma, then it cannot be regular

2. Identifying Non-Regular Languages

By applying the Pumping Lemma to a language, one can identify certain properties that are not satisfied if the language is not regular. This helps in understanding the limitations of regular language.

3. Compiler Design:

In the context of compiler design, the Pumping Lemma can be applied to analyze the regularity of the language defined by the lexical structure of a programming language. It helps ensure that the lexical analyzer can efficiently recognize valid tokens.

The Pumping Lemma is a powerful tool in formal language theory, and its applications extend to various areas, including language design, compiler construction, and the theoretical analysis of computational complexity.

Equivalence of Two Regular Expressions

Ex: Show the equality of the given regular expressions.

$(1 + 00^*1) + (1 + 00^*1)(1 + 10^*1)^*(1 + 10^*1) = 0^*1(0 + 10^*1)^*$

Solution:

We need to show the above LHS expression is equal to RHS expression.

We use identity rules to reduce the LHS expression to RHS.

Take $(1 + 00^*1)$ as common

$(1 + 00^*1)(\varepsilon + (1 + 10^*1)^*(1 + 10^*1))$

The above equation is of type $(\varepsilon + R^*R)$ where $R = (1 + 00^*1)$.

$(\varepsilon + R^*R)$ can be written as R^*

$(1 + 00^*1)(1 + 10^*1)^*$

Take 1 as common

$(\varepsilon + 00^*)1(1 + 10^*1)^*$

$(\varepsilon + 00^*)$ can be written 0^*

$0^*1(1 + 10^*1)^*$

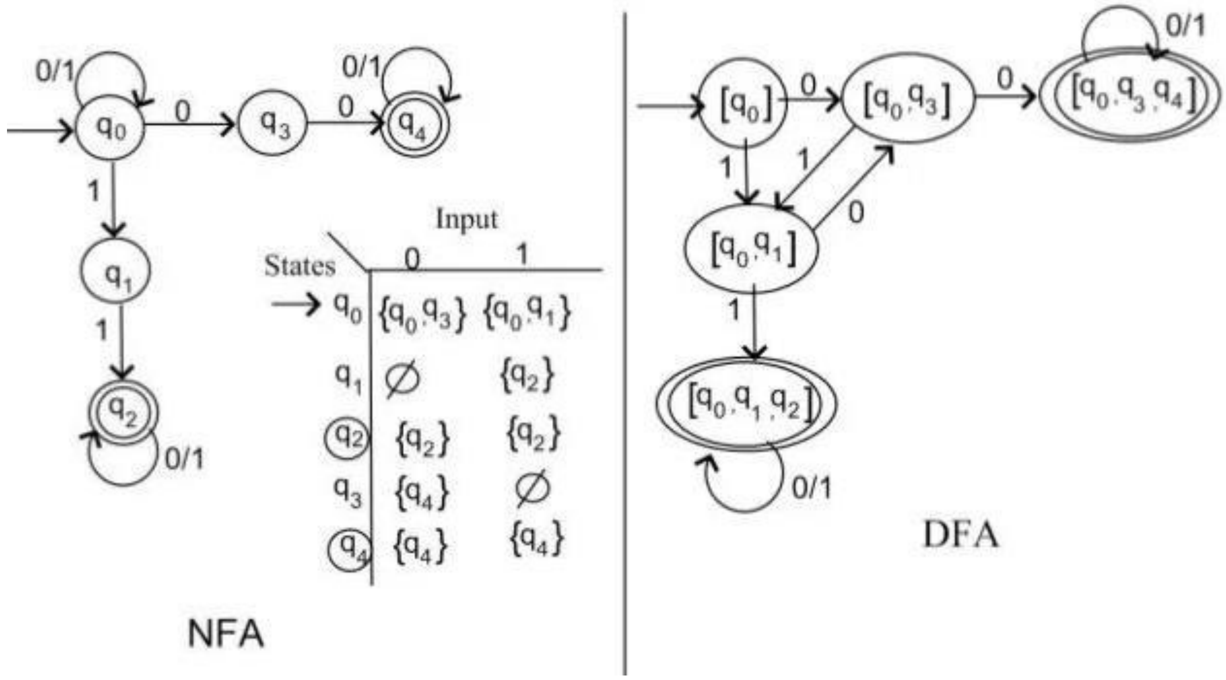
LHS = RHS

Equivalence of two finite automata

If two automata A and B accept precisely the same set of input strings, then they are considered comparable. If automata A and B are identical, then

i. If a path exists from the initial state of an A to its final state, designated a1a2..ak, then a path exists from the initial state of a B to its final state, also designated a1a2..ak.

ii. Should a path exist from the initial state of B to the ultimate state of B, designated as b1b2..bj, then a path exists from the initial state of A to the ultimate state of A, also designated as b1b2..bj.



States	Input	
	0	1
$\rightarrow q_0$	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	$\varnothing$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	$\varnothing$
q_4	$\{q_4\}$	$\{q_4\}$

Minimization of DFA

Conversion of a DFA to equivalent DFA with the fewest possible states is known as DFA minimization. Partitioning algorithms are used in DFA minimization, also known as Optimization of DFA.

Minimization of DFA

Assume a DFA D < Q, Σ, q0, δ, F > that is capable of identifying the language L. Then, for language L, the reduced DFA D < Q', Σ, q0, δ', F' > can be built as follows:

Step 1: Q (the collection of states) will be split into two sets. All final states will be included in one set, and non-final states will be included in the other. P0 is the name of this partition.

Step 2: Set up k = 1.

Step 3: Partition the various Pk-1 sets to find Pk. We will take every feasible pair of states in each set of Pk-1. We shall divide the sets into distinct sets in Pk if two states inside a set can be distinguished from one another.

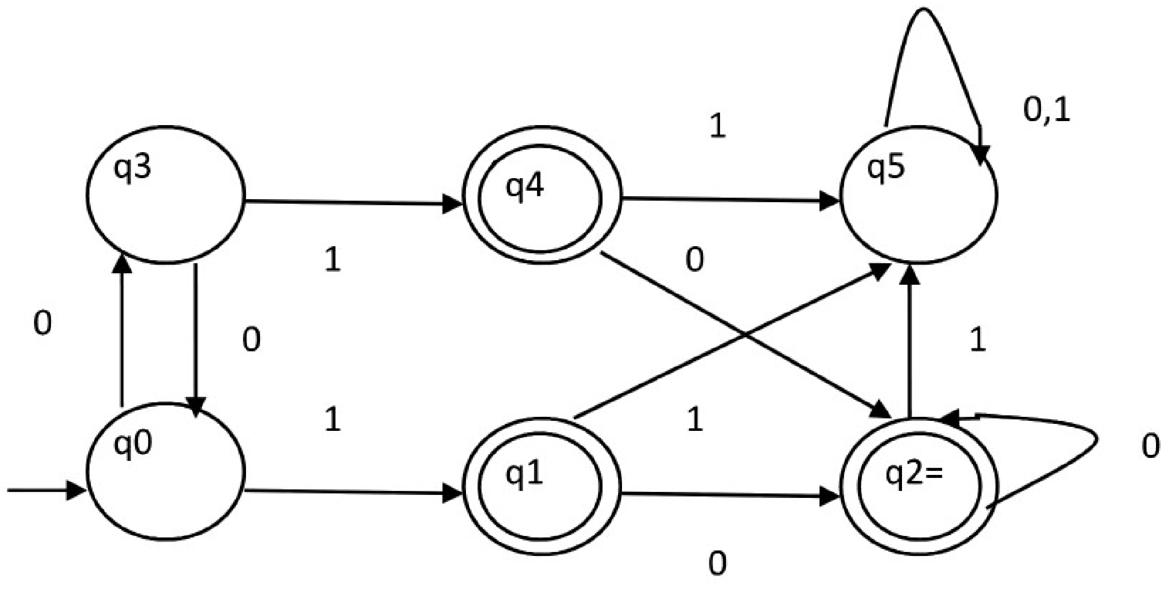
Step 4: if Pk equals Pk-1 (nopartition manipulation), stop.

Step 5: A set's states combine to form a single state. In Pk, the sets will equal the states in reduced DFA.

How may the distinguishability between two states in partition Pk be determined?
If δ (qi, a) and δ (qj, a) are in separate sets in partition Pk-1 for each given input symbol a, then two states (qi, qj) can be distinguished in partition Pk.

Ex:

. Examine the DFA that is depicted as



Step 1. P0 contains 2 states. The last DFA states, q1, q2, and q4, will be in one set, and the remaining states will be in another.

P0 = { { q1, q2, q4 }, { q0, q3, q5 } }.

Step 2. Now determine whether or not sets of partition P0 can be partitioned in order to compute P1:

i) For set { q1, q2, q4 } :
δ (q1, 0) = δ (q2, 0) = q2 and δ (q1, 1) = δ (q2, 1) = q5, So q1 and q2 are not distinguishable.
Similarly, δ (q1, 0) = δ (q4, 0) = q2 and δ (q1, 1) = δ (q4, 1) = q5, So q1 and q4 are not distinguishable.

As a result, q2 and q4 cannot be distinguished from q1 and q2, which cannot be distinguished from q1 and q4. Consequently, P1 will not partition the { q1, q2, q4 } set.

ii) For set {q0, q3, q5}:

δ (q0, 0) = q3 and δ (q3, 0) = q0
δ (q0, 1) = q1 and δ (q3, 1) = q4

Moves of q0 and q3 on input symbol 0 are q3 and q0 respectively which arein same set in partition P0. Similarly, Moves of q0 and q3 on input symbol 1 are q1 and q4 which are in same set in partition P0. So, q0 and q3 are not distinguishable.

δ(q0, 0) = q3 and δ (q5, 0) = q5 and δ (q0, 1) = q1 and δ (q5, 1) =q5
Moves of q0 and q5 on input symbol 1 are q1 and q5 respectively which are in different set in partition P0. So, q0 and q5 are distinguishable.
So, set {q0, q3, q5} will be partitioned into {q0, q3} and {q5}. So,P1 = {{q1, q2, q4}, {q0, q3}, {q5}}
To calculate P2, we will check whether sets of partition P1 can bepartitioned or not:

iii)For set {q1, q2, q4} :

δ (q1, 0) = δ (q2, 0) = q2 and δ (q1, 1) = δ (q2, 1) = q5, So q1 and q2 are not distinguishable.
Similarly, δ (q1, 0) = δ (q4, 0) = q2 and δ (q1, 1) = δ (q4, 1) = q5, So q1 and q4 are not distinguishable.
Since, q1 and q2 are not distinguishable and q1 and q4 are also not distinguishable, so q2 and q4 are not distinguishable. So, {q1, q2, q4} setwill not be partitioned in P2.

iv)For set {q0, q3}:

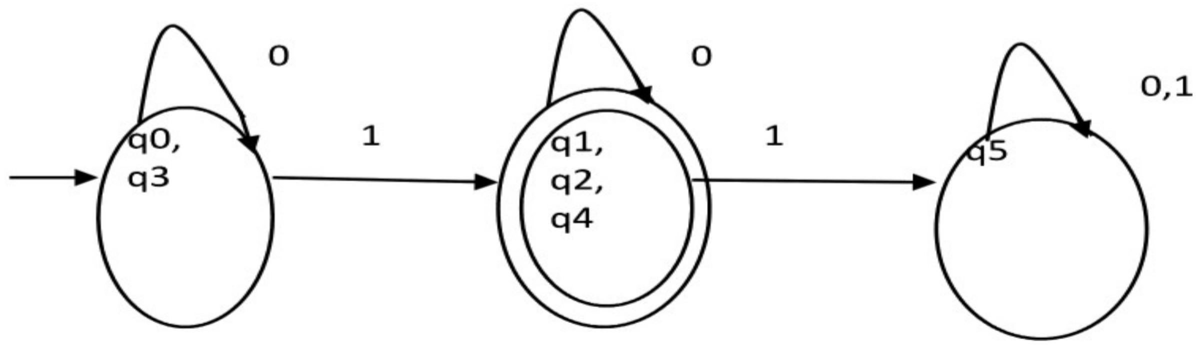
δ (q0, 0) = q3 and δ (q3, 0) = q0
δ (q0, 1) = q1 and δ (q3, 1) = q4

Moves of q0 and q3 on input symbol 0 are q3 and q0 respectively which arein same set in partition P1. Similarly, Moves of q0 and q3 on input symbol 1 are q1 and q4 which are in same set in partition P1. So, q0 and q3 are not distinguishable.

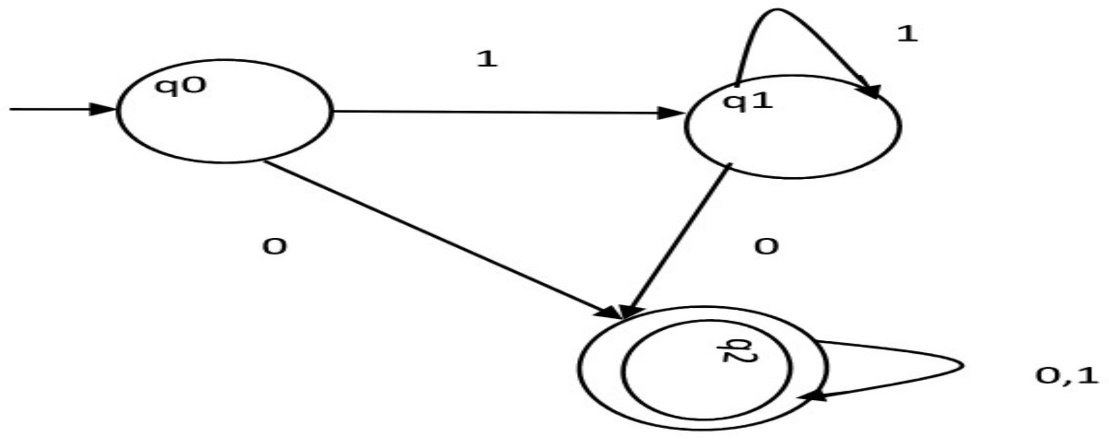
v) For set {q5}:

Since we have only one state in this set, it can't be further partitioned.
So,P2 = {{q1, q2, q4}, {q0, q3}, {q5}}
Since, P1=P2. So, this is the final partition. Partition P2 means that q1, q2and q4 states are merged into one.

In the same way, q0 and q3 combine to form q3. Figure 2 displays the minimized DFA that corresponds to the DFA of Figure 1 as:



Ex:
Examine the provided DFA. Which statement below is not true?
1. L(A)'s complement is independent of context.
2. L (A) = L ((11 * 0 + 0) (0 + 1)* 0* 1*)
3. A is the minimal DFA for the language that A has approved.
4. A takes any string longer than { 0, 1 } by at least two lengths.



A. 1 and 3 only
B. 2 and 4 only
C. 2 and 3 only
D. 3 and 4 only

Solution:

It will take all strings with a minimum length of two, according to statement 4. However, it takes 0 (which has length 1). Thus, 4 is untrue.

According to Statement 3, the DFA is negligible. We will verify with the previously mentioned algorithm.

P0 = {{q2}, {q0, q1}}
P1 = {q2}, {q0, q1}}.

P0 equals P1, hence P1 is ultimate DFA. Q0 , Q1 are combinable. Then there will be 2 states for minimal DFA. As a result, statement 3 is likewise not true.
Thus, (D) is the correct choice.