Module 1

...ormal Language Theory and Regular Languages

FORMAL LANGUAGE THEORY

...es 1) Define the basic terminologies of automata ...ory.

Or

...ite short notes on the following:
- Alphabets
- Strings
- Languages

...s: Basic Terminologies of Automata Theory

...e basic terminologies of automata theory are:

Alphabets: An alphabet is any finite, nonempty set of symbols. For alphabet the symbol Greek letter sigma Σ is used. **For examples,**

i) $\Sigma = \{0, 1\}$, set of binary symbols.

ii) $\Sigma = \{a, b, c, d, \ldots x, y, z\}$, set of all lowercase English letters.

iii) $\Sigma = \{A, B, C, D, E\}$, let of first five letters of the uppercase English letters.

iv) $\Sigma = \{0, 1, 2, \ldots, 9, a, b, c, \ldots, z\}$ set of alphanumeric symbols.

Strings: A string is the finite ordering of symbols chosen from some alphabets Σ. A string over an alphabet is a finite sequence of symbols from that alphabet, which is usually written next to one another and not separated by commas. **For examples,**

i) 01101 is a string from the binary alphabet $\Sigma = \{0, 1\}$. The string 1111 is another string chosen from these alphabets.

ii) abab, aabb, ab, ba, a, are all strings over an alphabet $\Sigma = \{a, b\}$.

iii) abcd, bcad, cbad, abcda are some of the strings from the alphabet $\Sigma = \{a, b, c, d\}$.

Languages: A set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet, is called a language. If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a language over Σ. **For examples,**

i) $L_1 = \{w \in (0, 1)^*$: w has an equal number of 0's and 1's$\}$

ii) $L_2 = \{w \in \Sigma^*: w = w^R\}$ where w^R is the reverse string of w.

iii) $L_3 = \{a, ab, ab^2, \ldots\}$.

Ques 2) Explain the concept of power of alphabet.

Ans: Power of Alphabet

If Σ is an alphabet, one can express the set of all strings of a certain length from that alphabet by using an exponential notation. Let Σ be the alphabet. Then Σ^m is the set of all strings of length m with symbols from Σ. **For example,** let $\Sigma = \{a, b\}$. Then,

i) $\Sigma^0 = \{\varepsilon\}$, empty string is the only string of length 0.

ii) $\Sigma^1 = \{a, b\}$, set of all strings over $\Sigma = \{a, b\}$ of length 1.

iii) $\Sigma^2 = \{ab, ba, aa, bb\}$, set of all strings of length 2.

iv) $\Sigma^3 = \{aaa, aab, aba, baa, abb, bab, bba, bbb\}$, set of all strings of length 3.

Ques 3) Differentiate between the L^* and L^+.

Ans: The set of all **possible** strings over Σ is denoted by Σ^* (L^*), where operator * is called as **Kleeny-closure** operator and the meaning of Σ^* is given by:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots\ldots$$

For example, if $\Sigma = (0, 1)$, then using the previous definitions of $\Sigma^0, \Sigma^1, \Sigma^2, \ldots$ one has,

$\Sigma^* = \{\varepsilon\} \cup \{0, 1\} \cup \{00, 01, 10, 11\} \cup \{000, 001, 010, 011, 011, \ldots\} \cup \ldots$

$= \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 011, \ldots\ldots\ldots\ldots\}$

(This set contains infinite many strings)

From the set Σ^*, if one exclude the empty string (ε) then he/she has a set of non-empty strings over alphabet Σ. That one denoted by Σ^+ (Where + is called Positive-closure operator and also denoted by L^+). Therefore,

$$\Sigma^+ = \Sigma^* - \{\varepsilon\} \quad [\therefore \Sigma^1 \cup \Sigma^2 \cup \ldots\ldots]$$

Conversely, one can say $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

Ques 4) What is length of a string? Give an example.

Ans: Length of String

Let u be a string. Then the length of the string u is denoted by $|u|$ and defined as "the number of symbols in the string". **For examples,**

i) If u = 000111, then $|u| = 6$

ii) If u = abab, then $|u| = 4$

Ques 5) Define empty string.

Or

What is null string?

Ans: Empty or Null String
The string having no symbol is called the empty (or null) string. In other words, "if the length of a string is zero", then it is called **empty string** or **null string**. It is denoted by ε or λ. Hence $|\varepsilon| = |\lambda| = 0$.

Ques 6) What is substring? Give an example.

Ans: Substring
A string u if it appears within another string w then u is called a substring of w. **For example,** let w = abbaba be a string over an alphabet set $\Sigma = \{a, b\}$, then a, ab, abb, ... are all substrings of w.

Ques 7) Discuss the concatenation operation on strings.

Ans: Concatenation of Strings
Let u and v be any two strings of alphabet Σ. The concatenation of u and v denoted by uv is a new string obtained by u followed by v.

Let $u = a_1a_2 \ldots a_i$ and $v = b_1b_2 \ldots b_j$, then $uv = a_1a_2 \ldots a_i b_1b_2 \ldots b_j$. Since the length of u is i and the length of v is j, then the length of uv is i + j. Concatenation with empty string does not change the string. That is, ε is called the **identity for concatenation** as $u\varepsilon = u = \varepsilon u$.

If w = uv then u is called **prefix** of w and v is called **suffix** of w. Generally, if w = uv, then u and v are substrings of w. Also $u\varepsilon = u = \varepsilon u$, therefore every string is a substring of itself, is a prefix of itself, and is a suffix of itself. **For example,** if w = 0111, then:

i) Set of prefixes = {ε, 0, 01, 011, 0111}, and
ii) Set of suffixes = {ε, 1, 11, 111, 0111}.

Ques 8) Given Σ = {a, b} obtain Σ*.
1) **Give an example of a finite language in Σ.**
2) **Given $L = \{a^nb^n : n \geq 0\}$, check if the strings aabb, aaaabbbb, abb are in the language L.**

Ans: Σ = {a,b}. Therefore, we have Σ* = {λ, a, b, aa, ab, ba, bb, aaa,}

1) {a, aa, aab} is an example of a finite language in Σ.
2) aa bb → a string in L. (n = 2)
   aaaa bbbb → a string in L. (n = 4)
   abb → not a string in L (since there is no n satisfying this).

Ques 9) Given $\{a^nb^n : n \geq 0\}$. Obtain:
1) L^2
2) L^R

Ans: Given $L = \{a^nb^n : n \geq 0\}$:
1) $L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}$
   Where, n and m are unrelated.
   For example, the string aabbaaabbb is in L^2.
2) Reverse of L is given by:
   $L^R = \{b^n a^n : n \geq 0\}$

REGULAR LANGUAGE

Ques 10) Formally define the regular language and regular set.

Ans: Regular Languages
The regular language are those languages that can be constructed from the 'big three' set operations, *viz.*,
1) Union, 2) Concentration, 3) Kleene star.

Let Σ be an alphabet, the class of 'regular languages' over Σ is defined as:
1) ϕ is a regular language.
2) For each σ ∈ Σ, (σ) is a regular language.
3) For any natural number n ≥ 2 if $L_1, L_2, L_3 \ldots L_n$ are regular languages, then $L_1 \cup L_2 \cup L_3 \ldots \cup L_n$ also a regular language.
4) For any natural number n ≥ 2 if $L_1, L_2, L_3 \ldots L_n$ are regular languages then $L_1 \circ L_2 \circ L_3 \ldots \circ L_n$ also regular language.
5) If L is a regular language then L* is also a regular language.

Regular Set
Any set represented by a regular expression is known as regular set. Let us consider the **table 1.1** (below):

Table 1.1: Some Regular Sets Corresponding to Given Regular Expressions

Regular Expression	Regular Set
a	{a}
abb	{abb}
a*	{∧, a, aa, aaa, aaaa, ...}
(a + b)* or (a|b)*	{∧, a, b, aa, ab, ba, bb, aaa, ...}
(a + b)* bb	{bb, abb, bbb, aabb, abbb, babb, bbbb, ...}

Ques 11) Let Σ = {0, 1}, then check the regularity of language $L = \{0^k1^k \mid k \geq 0\}$.

Ans: The language L consists of following set of string {ε, 01, 0011, 000111, ...}. We shall prove the regularity of L by method of contradiction. Thus, assume L is regular and n is any constant then string Z can be written as:

$$Z = 0^n . 1^n \text{ i.e., } |Z| = 2n \text{ so } |Z| > n$$

Now break the string Z into substrings u, v, and w (**figure 1.1**), i.e.,

$$Z = 0^n . 1^n = u.v.w$$

Where, substrings fulfill the condition such that

$$\underbrace{(000\ldots\ldots.0}_{u}\ \underbrace{0\ldots\ldots000)}_{v}\ .\ \underbrace{(111\ldots\ldots\ldots.1111)}_{w}$$

Figure 1.1

We observe that string u.v consist only of 0's so $|u.v| \leq n$ and $|v| \geq 1$. For the verification of the base case of pumping lemma, i.e., string $Z = u.v^i.w \Rightarrow u.w$ (for i = 0). Lemma says if L is regular then u.w ∈ L. Since substring u contains 0's that are fewer than n (because of few 0's are part of substring v) and substring w contains exactly n, 1's. So, string u.w does not have equal numbers of 0's followed by equal number of 1's. Hence, we obtain a contradiction, therefore L is not regular.

Ques 12) Is the following language regular? Justify your answer $L = \{0^{2n} \mid n \geq 1\}$.

Ans: This is a language length of string is always even, i.e.,

$n = 1$; $L = 00$

$n = 2$ $L = 00\ 00$ and so on.

Let $L = uvw$

$L = 0^{2n}$

$|z| = 2^n = uv^i w$

If we add 2n to this string length:

$|z| = 4n = uv.vw$

= even length of string.

Thus even after pumping 2n to the string we get the even length. So the language L is regular language.

Ques 13) Prove $L = \{a^P \mid p \text{ is a prime}\}$ **is not regular.**

Or

Show that L= $\{0^P/P$ is a prime number$\}$ is not regular. **(2018 [4.5])**

Ans: Let us assume L is a regular and P is a prime number.

$L = a^P$

$|z| = uvw$ $i = 1$

Now consider $L = uv^i w$ where $i = 2$

$= uv.\ vw$

Adding 1 to P we get, $P < |uvvw|$

$P < P + 1$

But P + 1 is not a prime number. Hence what we have assumed becomes contradictory. Thus L behaves as it is not a regular language.

Ques 14) Show that $L = \{0^n 1^{n+1} \mid n > 0\}$ **is not regular.**

Ans: Let us assume that L is a regular language.

$|z| = |uvw| = 0^n 1^{n+1}$

Length of string $|z| = n + n + 1 = 2n + 1$. That means length is always odd.

By pumping lemma

$= |uv.vw|$

That is if we add 2n + 1

$2n + 1 < (2n + 1) + 2n + 1$

$2n + 1 < 4n + 2$

But if n = 1 then we obtain 4n + 2 = 6 which is no way odd. Hence the language becomes irregular.

Even if we add 1 to length of $|z|$, then

$|z| = 2n + 1 + 1 = 2n + 2$

= even length of the string

So this is not a regular language.

FINITE STATE AUTOMATA

Ques 15) Describe finite state automata. List the elements of finite automata.

Or

What do you mean by finite automata? Also write its mathematical definition.

Ans: Finite State Automata / Finite Automata

A Finite Automata or finite-State Automaton (FSA) is an abstract machine having:

1) A **finite set** of states. These carry no further structure and provide a simple form of memory.
2) A **start state** and a set of **final states**.
3) A finite set of **input symbols** (alphabet)
4) A finite set of **transition rules** which specify how the machine, when in a particular state, responds to a particular input symbol. The response may be to change state and/or produce an output (action).

A FSA processes an input string over its alphabet. Each symbol is processed in turn. The FSA starts in its initial state and uses the transition rules to determine the next state and output (if any) from its current state and the symbol just read. If there is no rule defined for the current state and input symbol, the machine halts. If the entire string has been processed and the machine is in a final state, the FSA is said to accept the string.

Mathematical Definition

Mathematically, a finite automaton can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

1) Q is a finite non-empty set of states.
2) Σ is a finite non-empty set of inputs called input alphabet.
3) δ is a function which maps $Q \times \Sigma$ into Q and is usually called direct transition function. This is the function which describes the change of states during the transition. This mapping is usually represented by a transition table or a transition diagram.
4) $q_0 \in Q$ is the initial state.
5) $F \subseteq Q$ is the set of final states. It is assumed here that there may be more than one final state.

The transition function which maps $Q \times \Sigma^*$ into Q (i.e., maps a state and a string of input symbols including the empty string into a state) is called **indirect transition function**. We shall use the same symbol δ to represent both types of transition functions and the difference can be easily identified by nature of mapping (symbol or a string), i.e., by the argument. δ is also called **next state function**.

Elements of Finite Automata

The various components of finite automata are shown in **figure 1.2**:

1) **Input Tape:** Input tape is divided into cells (squares), which can hold one symbol from input alphabet. Input tape is a linear tape having some cells which can hold an input symbol from Σ.

2) **Finite Control:** It indicates the current state and decides the next state on receiving a particular input from the input tape. The tape reader reads the cells one by one from left to right and at any instance only one input symbol is read.

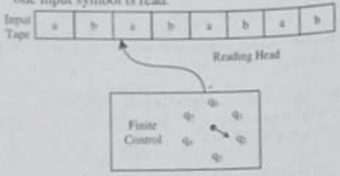


Figure 1.2: Block Diagram of Finite Automata

3) **Reading Head:** The reading head examines read symbol and the head moves to the right side with or without changing the state. When the entire string is read and if finite control is in final state then the string is accepted otherwise rejected.

Ques 16) Explain the state diagram of finite state automata with example.

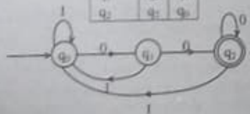
Ans: State Diagram of a FSA

Finite state automata can be defined by means of its state diagram. The state diagram is a labeled directed graph, where:

1) The vertices of the graph are the states in Q.
2) The arrow from the q_i to q_j with label a represents a transition or next-state function, which defines $\delta(q_i, a) = q_j$.
3) The initial state q_0 is represented by an arrow.
4) The final state or accepting state is represented by a double circle.

For example, the finite state automata and state diagram that accepts the language of all strings of 0's and 1's that ends with 00.

State	Input	
	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_0



The automata M is defined by representing its five parts, which is as follows:

1) $Q = \{q_0, q_1, q_2\}$.
2) $\Sigma = \{0, 1\}$, input symbols.
3) q_0 is the initial state.
4) $F = \{q_2\}$, is the final state.
5) δ, the transition function is represented by the table as shown in the **figure**.

This defines the automaton M, which accepts all the strings of 0's and 1's that ends with double 0. In this the final state q_2, is represented by a double circle, and the initial state q_0 is represented by special arrow.

Ques 17) Let Σ = {a, b}. Draw the state diagram of a finite state automaton M that accepts the given set of strings having odd number of a's.

Ans: Let M be a required DFA which contains two distinct states q_0, q_1 as shown in **figure 1.3**.

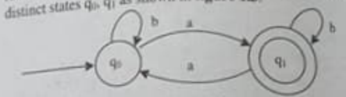


Figure 1.3

Ques 18) How design the finite automata?

Ans: Designing Finite Automata

The most important step in the designing of a finite automaton is to understand the language that users are working with. This requires that one can first understand what words are in and not in the language and that you next look for patterns among the words. This is necessary because two languages that look the same may be quite different.

Carefully listing the words in the language is the first step to understanding the language. It is especially important to be careful when the language is based on some formula because it is easy to get things mixed-up and have the right proportions but of the wrong symbols. When user produce lists of words, he should start with the smallest words and work-up. He should first check if the empty word (λ) is in the language.

Then users check to see if words of just one symbol are in the language. Then he should check the words of length 2, and so on. It may take as many as 10 words (or maybe even more for a complex description) to gives a clear picture of the language.

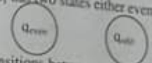
Table 1.2: Languages and Sample Words

$L_1 = \{a^n b^n : n \geq 0\}$	λ, ab, aabb, aaabbb, aaaabbbb, ...
$L_2 = \{a^n b^m : n \geq 0$ and $m \geq 0\}$	λ, a, b, aa, ab, bb, aaa, aab, abb, bbb, ...
$L_3 = \{(ab)^n : n \geq 0\}$	λ, ab, abab, ababab, abababab, ...
$L_4 = \{w : \#_a(w) = \#_b(w)\}$	λ, ab, ba, aabb, abab, abba, baab, baba, bbaa, ...
$L_5 = \{a^n b^m : n = m + 3\}$	aaa, aaaab, aaaaabb, aaaaaabbb, aaaaaaabbbb, ...
$L_6 = \{a^n b^m : n = m \bmod 2\}$	λ, ab, bb, abbb, bbbb, abbbbb, bbbbbb, ...
$L_7 = \{a^n b^m : n < m + 1\}$	λ, b, ab, bb, abb, aabb, bbb, abbb, aabbb, aaabbb, ...
$L_8 = \{a^n : n \bmod 5 = 2\}$	aa, aaaaaaa, aaaaaaaaaaaa, aaaaaaaaaaaaaaaaa, ...

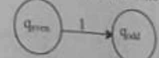
Table 1.2 has two columns. The first gives the language descriptions, and the second gives some of the smaller words that are in the language. Try to determine these lists before you look at the second column. In designing a finite automaton, we begin by considering the words in the language, and creating the states and transitions we need to accept these words. When designing a finite automaton, you should consider states as having a meaning.

For example, suppose that the alphabet set is Σ={0,1}. The language consists of all strings with an odd number of 1's. Construct a finite automaton E1 to recognize this language. You have to remember the number of 1's you have read is even or odd. If you read a 1 flip the answer, but if you read zero leave the answer as it is.

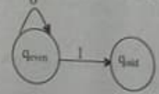
The machine E_1 has two states either even or odd state.



Define the transitions between states by seeing how to go from one state to another upon reading a symbol. If you are in even state and you read one then move to odd state.

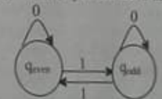


If you are in even state and you read Zero then stay in even state.

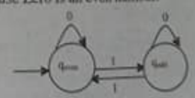


If you are in odd state and you read one then move to even state.

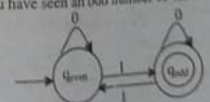
If you are in odd state and you read Zero then stay into the odd state.



Next, set the start state to be the state corresponding to the possibility to read the empty string ϵ which have zero symbols. In this case the start state will be the even state q_{even} because Zero is an even number.



Next, set the accept states to be corresponding to the possibilities you want to accept the input string. In our machine E_1, set the accept state to be the odd state q_{odd} when you have seen an odd number of 1s.



Ques 19) Discuss various types of finite state automata.

Or

Define Non-Deterministic Finite Automata?

(2017 [1.5], 2019[01])

Or

Write the notations for the language accepted by DFA and NFA.

(2017 [02])

Or

Write the notation for the language defined by a DFA.

(2019[01])

Ans: Types of Finite State Automata

There are two types of finite automata as shown in **figure** below:

1) **Deterministic Finite State Automata (DFA):** The finite automaton is called **deterministic finite automata** if there is only one path for a specific input from current state to next state. A deterministic finite automaton is a collection of following things and notation denoted as $A = (Q, \Sigma, \delta, q_0, F)$:
   i) The finite set of states which can be denoted by Q.
   ii) The finite set of input symbols Σ.
   iii) The start state q_0 such that $q_0 \in Q$.
   iv) A set of final states F such that $F \in Q$, and
   v) The mapping function or transition function denoted by δ. Two parameters are passed to this transition function – one is current state and other is input symbol. The transition function returns a state which can be called as next state.

   For example, $q_1 = \delta(q_0, a)$ means from current state q_0, with input a the next state transition is q_1.

2) **Non-Deterministic Finite Automaton (NFA):** Non determinism is the ability to change states in a way that is only partially determined by current state and input symbol. That is, several possible "next states" are possible for a given combination of current symbol and input symbol.

The automaton, as it reads input string may choose at each step to go into any one of legal next states; the choice is not determined by anything and is therefore called **nondeterministic**.

Hence, if some moves of the machine cannot be determined uniquely by the input symbol and the present state. Such machines are called non-deterministic finite automata.

A Non-deterministic Finite Automaton (NFA) is very similar to Deterministic Finite Automaton. Just like a DFA, a NFA has states, transitions, an initial state and a set of final states. The differences is that in a DFA, each state has at most one transition for a given symbol, while a NFA can have any number of transitions for a given symbol.

A Non-Deterministic Finite Automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where,
i) Q is a finite non-empty set of states.
ii) Σ is a finite non-empty set of inputs.
iii) δ is the transition function mapping from $Q \times \Sigma$ into 2^Q which is the power set of Q, the set of all subsets of Q.
iv) $q_0 \in Q$ is the initial state, and
v) $F \subseteq Q$ is the set of final states.

Ques 20) Write a string belong to the language L^3 if L={0, 1}. **(2019[02])**

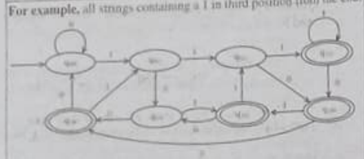
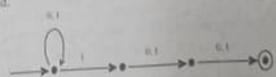
Ans: String belong to the Language L^3 if L = {0, 1}
For calculating L^2, we pick two strings from L, with repetitions allowed, so there are four choices. These four selections give us:
$L^2 = \{00, 01, 10, 11\}\{0,1\}$
Similarly, L^3 is the set of strings that may be formed by making three choices of the two strings in L and gives us:
$L^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

Ques 21) Differentiate DFA and NFA. **(2017 [1.5])**

Or

Compare its ability with Deterministic Finite Automata in accepting languages.

Ans: Difference between Deterministic and Non – Deterministic Finite Automata
The main difference between the deterministic and non-deterministic automata is in δ. The other differences are given in the following table:

Deterministic Finite Automata	Non – Deterministic Finite Automata
For Every symbol of the alphabet, there is only one state transition in DFA.	There may be more than one state transition in NFA.
DFA cannot use empty string transition.	NFA can use empty string transition.
DFA can be understood as one machine.	NFA can be understood as multiple little machines computing at the same time.
DFA will reject the string if it ends at other than accepting state.	NFA is easier to construct.
If all of the branches of NFA dies or rejects the string, we can say that NFA reject the string.	It is more difficult to construct DFA.
DFA requires more space.	NFA requires less space.
For example, all strings containing a 1 in third position from the end.  Figure 1.4: DFA	**For example,** all strings containing a 1 in third position from the end.  Figure 1.5: NFA

Ques 22) How DFA can be represented using state diagram?

Or

How transition table is used to represent DFA?

Ans: Representation DFA Using State Diagrams and Transition Table
The strings and languages can be accepted by a finite automaton, when it reaches to a final state. There are two preferred notations for describing automata, which are as follows:
1) **State Diagram:** Finite state automata can be defined by means of its state diagram. The state diagram is a labeled directed graph, where:
   i) The **vertices** of the graph are the states in set of states (Q).
   ii) The arrow from the q_i to q_j with label a represents a transition or next-state function, which defines $\delta(q_i, a) = q_j$.
   iii) The **initial state** q_0 is represented by an arrow.
   iv) The **final state** or accepting state is represented by a double circle.
   v) A transition function $Q \times A \rightarrow Q$ with Q as state and A as input from Σ^*.

The notations used in transition diagram are as shown:

Symbol	Description
q_i	Represents the state
$\longrightarrow$	Represents transition from one state to another
Start $\rightarrow q_i$	Start state
q_n	Final state

For example, FA can be represented using transition diagram as **shown figure 1.6.**

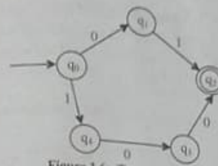


Figure 1.6: Transition Diagram

The machine initially is in start state q_0 then on receiving input 0 it changes to state q_1. From q_0 on receiving input 1 the machine changes its state to q_4. The state q_2 is a final state or accept state. When the input for transition diagram reach to a final state at end of input string then it is said that the given input is accepted by transition diagram. **For example,** a finite automaton M is shown in the state diagram below:

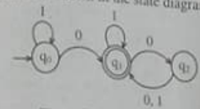


Figure 1.7: State Diagram

In this case, $Q = \{q_0, q_1, q_2\}$, Σ(alphabet) = {0, 1}, F (set of accept states) = $\{q_1\}$ and q_0 is the initial state.

2) **Transition Table:** This is a tabular representation of finite automata. State transition table is a table showing what state a finite state machine will move to, based on the current state and other inputs. A state table is essentially a truth table in which some of the inputs are the current state, and the outputs include the next state, along with other outputs.

For transition table, the transition function is used. **For example,** for **figure 1.7**, the transition function δ can be described by a transition function table, as follows:

State	Input Symbol	
	0	1
q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_1	q_1

The transition function can be represented as δ(current state, current input symbol) → next state. If q_0 is the current state and 0 is the current input symbol, then the transition function is $\delta(q_0, 0) \rightarrow q_1$. One can check this by comparing the transition table with the state diagram as shown in **figure 1.7**.

Ques 23) How strings are processed in DFA? Explain with an example.

Ans: Processing of Strings in DFA
A string x is accepted by a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ if $\delta(q_0, x) = q$ for some $q \in F$. This is basically the acceptability of a string by the final state. A final state is also called an **accepting state**.

For example, consider the finite state machine whose transition function δ is given in **table 1.3** in the form of a transition table. Here, $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $F = \{q_0\}$. The entire sequence of states for the input string 110001 is as follows.

Table 1.3: Transition Function Table

States	Inputs	
	0	1
$\rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

The sequence of states for the input string 110001 is shown below:

$$\begin{aligned}\delta(q_0, \underset{\downarrow}{1}10101) &= \delta(q_1, \underset{\downarrow}{1}0101)\\ &= \delta(q_0, \underset{\downarrow}{0}101)\\ &= \delta(q_2, \underset{\downarrow}{1}01)\\ &= \delta(q_3, \underset{\downarrow}{0}1)\\ &= \delta(q_1, \underset{\downarrow}{1})\\ &= \delta(q_0, \Lambda) = q_0\end{aligned}$$

The symbol ↓ indicates the current input symbol being processed by the machine.

Hence,
$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_0$$

Ques 24) Construct a finite state automata that accept the set of natural numbers x which are divisible by 3.

Ans: Let $M = (Q, \Sigma, q_0, \delta, F)$ be a machine with $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $F = \{q_0\}$ and δ is defined as:
$\delta(q_0, a) = q_0$; $\delta(q_1, a) = q_1$; $\delta(q_2, a) = q_2$ for $a \in \{0, 3, 6, 9\}$
$\delta(q_0, b) = q_1$; $\delta(q_1, b) = q_2$; $\delta(q_2, b) = q_0$ for $b \in \{1, 4, 7\}$
$\delta(q_0, c) = q_2$; $\delta(q_1, c) = q_0$; $\delta(q_2, c) = q_1$ for $c \in \{2, 5, 8\}$

The transition diagram for this machine is given in **figure 1.8**.

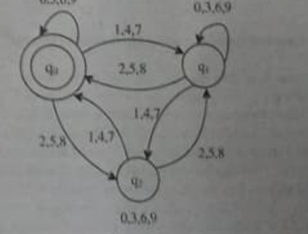


Figure 1.8

The computation of the string w = 150 is:
$\delta(q_0, 150) = \delta(\delta(\delta(q_0, 1), 5, 0)) = \delta(\delta(q_1, 5), 0) = \delta(q_0, 0)$
$= q_0$ (accepting state) and

For w = 116,
$\delta(q_0, 116) = \delta(\delta(\delta(q_0, 1), 1), 6) = \delta(\delta(q_1, 1), 6)$
$= \delta(q_2, 6) = q_2$ (non-accepting state).

Ques 25) Design DFA which accepts odd number of 1's and any number of 0's.

Ans: In the problem statement, it is indicated that there will be a state which is meant for odd number of 1's and that will be the final state. There is no condition on number of 0's.

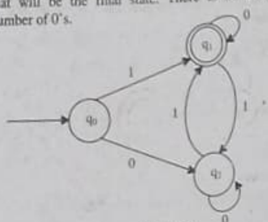


Figure 1.9

At the start, if we read input 1 then we will go to state q_1 which is a final state as we have read odd number of 1's. There can be any number of zeros at any state and therefore the self-loop is applied to state q_2 as well as to state q_1.

Ques 26) Let M be the deterministic finite automaton (Q, Σ, δ, q, F), where
$Q = \{q_0, q_1\}$,
$\Sigma = \{a, b\}$,
$q = q_0$,
$F = \{q_0\}$,

And δ is the function is given in table below.

q	σ	δ(q, σ)
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0

Generate finite automata for the above system.

Ans: It is then easy to see that L (M) is the set of all strings in {a, b}* that have an even number of b's. For M passes from state q_0 to q_1 or from q_1 back to q_0 when a 'b' is read, but M essentially ignores a's, always remaining in its current state when a is read. Thus M counts b's modulo 2, and since q_0 (the initial state) is also the sole final state; M accepts a string if and only if the number of b's is even.

If M is given the input aabba, its initial configuration is (q_0, aabba). Then

$\delta(q_0, aabba) \to \delta(q_0, abba)$
$\to \delta(q_0, bba)$
$\to \delta(q_1, ba)$
$\to \delta(q_0, a)$
$\to \delta(q_0, \varepsilon)$

Therefore $(q_0, aabba) \to \delta^*(q_0, \varepsilon)$, and so aabba is accepted by M.

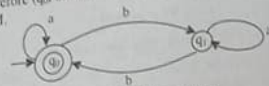


Figure 1.10: State Diagram

The tabular representation of the transition function used in this example is not the clearest description of a machine. We generally use a more convenient graphical representation called the state diagram (**figure 1.10**).

The state diagram is a directed graph, with certain additional information incorporated into the picture. States are represented by nodes, and there is an arrow labeled with a from node q to q' whenever δ(q, a) = q'. Final states are indicated by double circles, and the initial state is shown by →.

Ques 27) Can we use finite state automata to evaluate 1's complement of a binary number? Design a machine to perform the same. (2017 [03])

Ans: Yes, we can use the finite state automata to evaluate 1's complement of a binary number.

In 1's complement if the input is 0, the output must be 1 and vice-versa. Let us construct a state table (**Table 1.4**) which could solve our purpose.

Table 1.4 Mealy machine of Example

PRESENT STATE	NEXT STATE			
	x = 0	OUTPUT	x = 1	OUTPUT
$\to q_0$	q_0	1	q_0	0

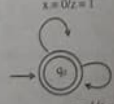


Figure: 1.11 Transition diagram of Table 1.4

Now we can see for input string '0100' we have '1011' which is 1's complement of '0100'.

Ques 28) Design a Finite state automata which accepts all strings over {0, 1} with odd number of 1's and even number of 0's. (2017 [05])

Ans: This FA will consider four different stages for input 0 and 1. The stages could be
even number of 0 and even number of 1,

even number of 0 and odd number of 1,
odd number of 0 and even number of 1,
odd number of 0 and odd number of 1.
Let us try to design the machine

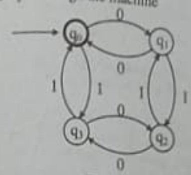


Figure: 1.12

Here q_0 is a start state as well as final state. Note carefully that a symmetry of 0's and 1's is maintained. We can associate meanings to each state as.
q_0 : state of even number of 0's and even number of 1's.
q_1 : state of odd number of 0's and even number of 1's.
q_2 : state of odd number of 0's and odd number of 1's.
q_3 : state of even number of 0's and odd number of 1's.
The transition **table** can be as follows-

	0	1
$\to q_0$	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_2	q_1

Ques 29) Show the changes needed to convert the above designed automata to accept even number of 1's and odd number of 0's. (2017 [04])

Ans: The states of DFA have to count odd number of 0's and even number of 1's. The states is used to remember the number of 0's seen so far is even or odd and also to remember the number of 1's seen so far is even or odd. Hence four states are needed. Given any string of 0's and 1's, after reading a character, we can either have seen:
1) Both the number of 0's and 1's are even (state q_0)
2) Both the number of 0's and 1's seen are odd (state q_1)
3) The number of 0's seen is odd and number of 1's seen is even (state q_2)
4) The number of 0's seen is even and number of 1's seen is odd (stateq_3)

q_0 is the initial state. We want to accept a given string if it has an odd number of 0's and an even number of 1's so q_2 should be an accepting state. Let us check for an input 01001.
$\delta(q_0, 01001) - \delta(q_2, 1001) - \delta(q_1, 001) = \delta(q_3, 01) - \delta(q_1, 1) - q_2$

Hence 01001∈L (M)

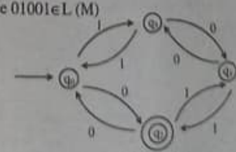


Figure 1.13

Ques 30) Design a DFA that accept the language $L = \{a^n b : n \geq 0\}$

Ans: The state in DFA is used to remember many number of a's followed by a single b. The minimum requirement of states is 2, since if we start with q_0 if the machine see b, it changes its state to q_1 (an accepting state), and if sees 'a' it should stay back in same state q_0. That is $\delta(q_0, a) = q_0$. Hence we have

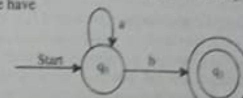


Figure 1.14

If the next input is a, b we define another state q_2, non-accepting state, such that

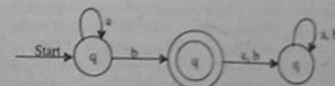


Figure 1.15

If abab is an input, then
$\delta(q_0, abab) = \delta(q_0, bab) = \delta(q_1, ab) = \delta(q_2, b) = q_2$

A non-accepting state. Hence abab ∉ L(M). If the input is a^3b, then
$\delta(q_0, a^3b) = \delta(q_0, a^2b) = \delta(q_0, ab) = \delta(q_0, b) = q_1$

Hence $a^3b \in L(M)$.

Ques 31) Given Σ = {a, b}. Construct a DFA that recognize the language $L = \{b^m ab^n : m, n > 0\}$.

Ans: Given language has a string with exactly one 'a' in between b's. So minimum requirement of states required to accept the string in L is 4. DFA with dead state q_4 (non-accepting state) is given by **figure 1.16**.

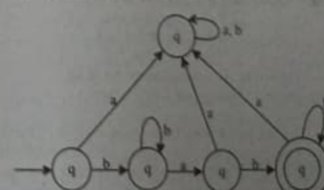


Figure 1.16

For an input b^2ab^3, we have
$\delta(q_0, b^2ab^3) = \delta(q_1, bab^3) = \delta(q_2, ab^3) = \delta(q_3, b^3) = \delta(q_3, b^2)$
$= \delta(q_3, b) = q_3$, an accepting state.
Hence $b^2ab^3 \in L(M)$.

Ques 32) Explain the processing of string in NFA with example.

Or

Compare the transition functions of NFA and DFA. **(2018 [03])**

Ans: Processing of Strings in NFA
Like the DFA, an NFA has a finite set of states, a finite set of input symbols, one start state and a set of accepting states. It also has a transition function, which one shall commonly call δ.

The **difference between the DFA and the NFA** is in the type of δ. For the NFA, δ is a function that takes a state and input symbol as arguments (like the DFA's transition function), but returns a set of zero, one, or more states (rather than returning exactly one state, as the DFA must).

For example, consider the non-deterministic automaton (NFA) whose transition diagram is given in **figure 1.17.** The sequence of states for the input string 0100 is given in **figure 1.17**. Hence $\delta(q_0, 0100) = \{q_0, q_3, q_4\}$. Since q_4 is an accepting state, the input string 0100 will be accepted by the non-deterministic automaton.

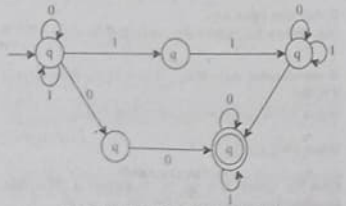


Figure 1.17: Transition System for NFA

For example, consider the **figure 1.17** shows a finite automaton which accepts exactly those strings that have the symbol 1 in the second last **position**.

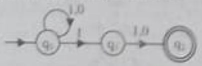


Figure 1.18: NFA Accepting the Set of All Strings whose Second Last Symbol is 1

State q_0 is the initial state from where it moves when it sees a 1 and guesses that there is only one more symbol to follow. Since it is possible that there is more than one symbol to be examined, there are transitions from q_0 to itself on reading either a 0 or a 1.

Now there are two possible transitions labelled 1 out of q_0 and hence NFA has two options – it can move to **either** q_0 or q_1. In fact at each such point, it starts a new thread corresponding to each of the new transitions.

Also note that there are no transitions out of q_2. Whenever this happens, the corresponding thread just ceases to exist. This is illustrated in **figure 1.18**.

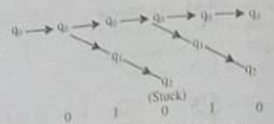


Figure 1.19: States of the NFA while Processing the String 01010

Consider the input string 01010. The NFA starts from initial state q_0 and remains there on reading the 0. On reading a 1 next it may go either to q_0 or to q_1 as seen in the third column of **figure 1.19**. Now the next symbol 0 is read. One needs to consider transitions out of both q_0 and q_1. q_0 goes to itself while q_1 moves to q_2. When the next symbol 1 is read one needs to consider two transitions once again – one out of q_0 and the other out of q_2. q_0 can go either to q_0 or to q_1 whereas q_2 has no transition on 0 or 1 and hence it "dies". With the last input 0 q_0 goes to q_0 while q_1 goes to q_2. Since q_2 is an accepting state the NFA accepts 0101.

Ques 33) Draw the NFA for the language $L = \{a^n b^m \mid n, m >= 0\}$. **(2019[02])**

Ans: NFA for $L = \{a^n b^m \mid n, m>=1\}$
$L = \{a^n b^m / n, m \geq 1\}$
$= \{ab, aab, abb, aabb, aaabb.......\}$

NFA is shown in **figure 1.20**.

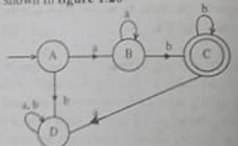


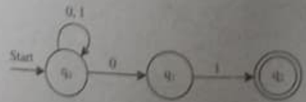
Figure 1.20: NFA

Ques 34) Design the NFA accepting the string that end in 01.
Ans: An NFA accepting strings that end in 01 is given by:
$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$

Where the transition function δ is given by the **table below**:

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	φ	$\{q_2\}$
$*q_2$	φ	φ

The transition diagram is **shown below**:



Ques 35) Construct NFA without ε-transitions from the following NFA:
$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \delta, q_0, \{q_2\})$ and **(2019[06])**
$\delta(q_0, a) = \{q_0\}, \delta(q_0, b) = \{q_1\}, \delta(q_0, c) = \{q_2\},$
$\delta(q_1, \varepsilon) = \{q_0\}, \delta(q_1, a) = \{q_1\}, \delta(q_1, b) = \{q_2\},$
$\delta(q_2, \varepsilon) = \{q_1\}, \delta(q_2, a) = \{q_2\}, \delta(q_2, c) = \{q_0\}.$

Ans: The NFA with ε-transitions is shown in **figure 1.21**:

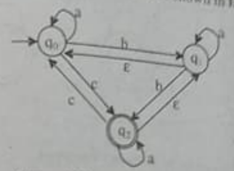


Figure 1.21: NFA with ε-Transitions

Transition state table for the above NFA is shown below:

States	ε	a	b	c
q_0	–	q_0	q_1	q_2
q_1	q_0	q_1	q_2	–
q_2	q_1	q_2	–	q_0

Ques 36) Determine an NFA with three states that accepts the language $\{ab, abc\}^*$.

Ans: NFA for the language
$L = \{ab, abc\}^*$

Should be such that it accepts "ab" or "abc" in the first step and then this is looped with initial state so that any combination of "ab" and "abc" can be accepted.

Hence we have the NFA as:
$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \delta, q_0, \{q_1\})$

This is shown below:

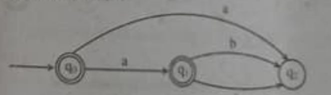


Figure 1.22

Ques 37) Determine an NFA that accepts the language:
$L(aa^*(a + b))$

Ans: NFA is given by:
$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$

Hence the transition diagram is shown below:

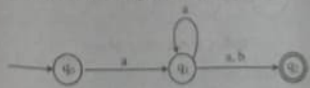


Figure 1.23

Ques 38) Explain the NFA with ε moves.
Or
Define the NFA with epsilon transition.
Or
Write the notations for the language accepted by ε-NFA. **(2017 [01])**

Ans: NFA with ε Moves/ NFA with Epsilon Transition
One shall now introduce another extension of the finite automaton. The new "feature" is that one allows a transition on ε, the empty string. In effect, a NFA is allowed to make a transition spontaneously, without receiving an input symbol. Like the non-determinism added, this new capability does not expand the class of languages that can be accepted by finite automata, but it does give us some added "programming convenience".

One will see that when one take up regular expressions, how NFA's with ε-transitions, which one calls ε-NFA's, are closely related to regular expressions and useful in proving the equivalence between the classes of languages accepted by finite automata and by regular expressions.

The symbol ε is a character used to indicate null string, i.e., the string which is used simply for transition from one state to other without any input.

A ε-NFA is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$

Where,
1) Q is a set of states
2) Σ is the alphabet of input symbols
3) $q_0 \in Q$ is the initial state
4) $F \subseteq Q$ is the set of final states
5) $\delta: Q \times \Sigma_\varepsilon \rightarrow P(Q)$ is the transition function

Note: ε is never a member of Σ. Σ_ε is defined to be $(\Sigma \cup \varepsilon)$

It is already known that how to obtain a regular expression (RE) from the given finite automata. Let now discuss how to obtain a finite automata from a given regular expression (RE). This algorithm needs to construct a NFA with ε-moves from the given regular expression (RE) ε as the first step. In the next step, the algorithm converts the NFA into an equivalent DFA, by first eliminating ε-transitions from the constructed NFA.

For example, consider the following automata.

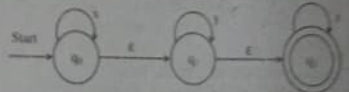


Figure 1.24: Finite Automata with ε Moves

The important aspect for building the transition table is to compute the δ function. The $\hat{\delta}$ the transition function, maps $Q \times (\Sigma \cup \{\varepsilon\})$ to 2^Q.

The intention is that $\delta(q, a)$ will consist of all states p such that there is a transition labeled a from q to p where a is either ε or the symbol in ε. Let us design the transition table with δ function for **figure 1.24**.

	x	y	z	ε
q_0	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$	ϕ

Algorithms to Remove ε-Transition from NFA

Step 1) Find all edges starting from b.

Step 2) Duplicate all these edges starting from a without changing edge labels.

Step 3) If a is the initial stage make b as the initial state.

Step 4) If b is the final state make a as the final state.

Ques 39) Change the following NFA with epsilon into NFA without epsilon.

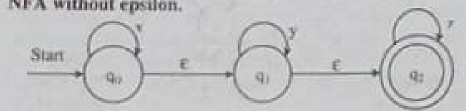


Figure 1.25: Finite Automata with ε Moves

Ans: The important aspect for building the transition table is to compute the δ function. The δ the transition function, maps $Q \times (\Sigma \cup \{\varepsilon\})$ to 2^Q. The intention is that $\delta(q, a)$ will consist of all states p such that there is a transition labeled a from q to p where a is either ε or the symbol in ε. Let us design the transition table with δ function for **figure 1.26**.

	x	y	z	ε
q_0	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_1\}$
q_2	ϕ	ϕ	$\{q_2\}$	ϕ

Let $\delta''(q, w)$ will be all states p such that one can go from q to p along a path labeled w, sometime including edges labeled ε. While constructing δ'' we have to compute the set of states reachable from a given state q using ε transitions only. Let us use ε CLOSURE (q) to denote the set of all vertices p such that there is a path from q to p labeled ε.

The δ'' can be interpreted as follows:

$\delta''(q, \varepsilon) = \varepsilon\text{-CLOSURE}(q)$

For w in Σ^* and a in Σ, $\delta''(q, wa) = \varepsilon$ CLOSURE (p)

Where $p = \{p \mid$ for some r in $\delta''(q, w)p$ is in $\delta(r, a)\}$

For the transition table

$$\delta''(q_0, \varepsilon) = \varepsilon\text{-CLOSURE}\{q_0\} = \{q_0, q_1, q_2\}$$

Thus,

$$\begin{aligned}\delta''(q_0, x) &= \varepsilon\text{-CLOSURE}(\delta(\delta''(q_0, \varepsilon), x))\\ &= \varepsilon\text{-CLOSURE}(\delta(\{q_0, q_1, q_2\}, x))\\ &= \varepsilon\text{-CLOSURE}(\delta(q_0, x) \cup \delta(q_1, x) \cup \delta(q_2, x))\\ &= \varepsilon\text{-CLOSURE}(\{q_0\} \cup \phi \cup \phi)\\ &= \varepsilon\text{-CLOSURE}[\{q_0\}]\end{aligned}$$

$\delta(((q0, x) = \{q_0, q_1, q_2\}$

$$\begin{aligned}\delta(((q0, xy) &= \varepsilon\text{-CLOSURE}(\delta(\delta(((q0, x), y))\\ &= \varepsilon\text{-CLOSURE}(\delta(\{q0, q1, q2\}, y))\\ &= \varepsilon\text{-CLOSURE}(\{q1, q2\}) = \{q_1, q_2\}\end{aligned}$$

Thus, the transition table can be drawn as:

	x	y	z
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

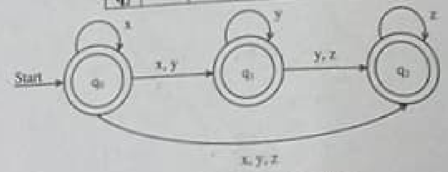


Figure 1.26: NFA without ε Transitions

Ques 40) Obtain an equivalent NFA without ε-transitions from the NFA shown in figure 1.27:

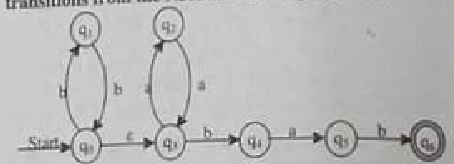


Figure 1.27

Ans: One just needs to eliminate the ε-transition between q_0 and q_3. This will add two new edges from q_0 to q_2 on a and q_0 to q_4 on b. Moreover q_3 will become initial state as q_0 is an initial state. The NFA without ε-transition is as shown in **figure 1.28**:

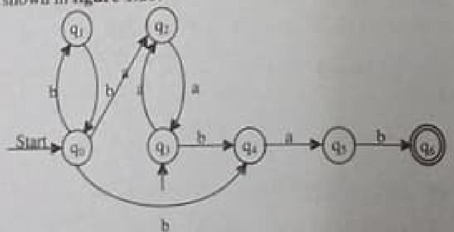


Figure 1.28: NFA without ε-Transitions for NFA given in figure 1.27

Ques 41) Construct an NFA without ε -moves corresponding to the following NFA:

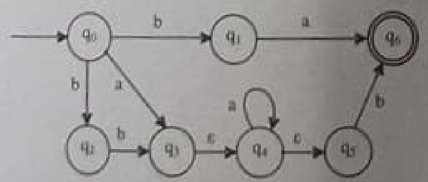


Figure 1.29: FA with ε -Moves

Ans: Constructed NDFA with ε -moves is:

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{a, b\}, \delta, q_0, \{q_6\})$

Let us determine ε-closure of all the states of M, i.e., $q_0, q_1, q_2, q_3, q_4, q_5$, and q_6 as below:

ε-closure $(q_0) = \{q_0\}$

ε-closure $(q_1) = \{q_1\}$

ε-closure $(q_2) = \{q_2\}$

ε-closure $(q_3) = \{q_3, q_4, q_5\}$

ε-closure $(q_4) = \{q_4, q_5\}$

ε-closure $(q_5) = \{q_5\}$

ε-closure $(q_6) = \{q_6\}$

Let the NDFA without ε-moves equivalent to NDFA having ε -moves be defined as:

$M' = (\{\{q_0\}, \{q_1\}, \{q_2\}, \{q_3, q_4, q_5\}, \{q_4, q_5\}, \{q_5\}, \{q_6\}\}, \{a, b\}, \delta', q_0', F')$,

Where, δ' is defined as:

$\delta'(\{q_0\}, a) = \{q_3, q_4, q_5\}$,	$\delta'(\{q_0\}, b) = \{q_1\} \cup \{q_2\} = \{q_1\}, \{q_2\}$
$\delta'(\{q_1\}, a) = \{q_6\}$,	$\delta'(\{q_1\}, b) = \{\phi\}$
$\delta'(\{q_2\}, a) = \{\phi\}$,	$\delta'(\{q_2\}, b) = \{q_3, q_4, q_5\}$
$\delta'(\{q_3, q_4, q_5\}, a) = \{q_4, q_5\}$,	$\delta'(\{q_3, q_4, q_5\}, b) = \{q_6\}$
$\delta'(\{q_4, q_5\}, a) = \{q_4, q_5\}$,	$\delta'(\{q_4, q_5\}, b) = \{q_6\}$
$\delta'(\{q_5\}, a) = \{\phi\}$,	$\delta'(\{q_5\}, b) = \{q_6\}$
$\delta'(\{q_6\}, a) = \{\phi\}$,	$\delta'(\{q_6\}, b) = \{\phi\}$

Let us assume, $\{q_0\} = q_A$, $\{q_1\} = q_B$, $\{q_2\} = q_C$, $\{q_3, q_4, q_5\} = q_D$, $\{q_4, q_5\} = q_E$, $\{q_5\} = q_F$, $\{q_6\} = q_G$.

By using above transitions δ', following finite automaton (NDFA without ε-moves) can be drawn:

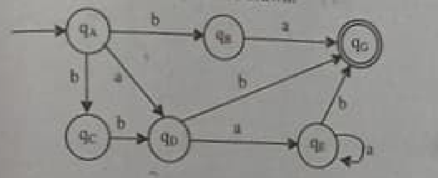


Figure 1.30: Finite Automaton without ε -Moves

The transition table is shown below:

Table 1.5: Transition Table of NDFA without ε- Moves

Present State	Inputs	
	a	b
$\to q_A$	q_D	q_B, q_C
q_B	q_F	–
q_C	–	q_D
q_D	q_E	q_G
q_E	q_E	q_G
(q_G)	–	–

Ques 42) Prove the equivalence of DFA and NFA.

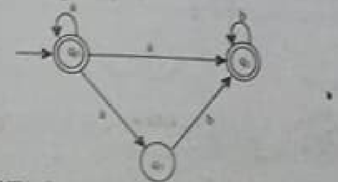
Ans: Equivalence of DFA and NFA

A DFA can simulate the behavior of NFA by increasing the number of states. In other words, a DFA $(Q, \Sigma, \delta, q_0, F)$ can be viewed as an NFA $(Q, \Sigma, \delta', q_0, F)$ by defining $\delta'(q, a) = \{\delta(q, a)\}$.

Any NFA is a more general machine without being more powerful.

Let, $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA which accepts the language L(M). There should be equivalent DFA denoted by $M' = (Q', \Sigma, \delta', q_0', F)$ such that $L(M) = L(M')$.

Consider the following NFA.



In the above NFA, from state q_0, there are 3 possible moves for input symbol, a i.e., to q_0, q_1, q_2. When we simulate this NFA using a program, the move from q_0 for symbol a is non-deterministic. That means, which transition is to be made next cannot be determined in one move.

But in a DFA, from a state, for an input symbol, only one transition exists. So it is very easy to simulate a DFA using a program. So we need to convert an NFA to an equivalent DFA.

For every NFA, there exists an equivalent DFA.

Ques 43) Write the steps of converting NFA into DFA.

Ans: Conversion of NFA into DFA

The steps involved in the conversion of NFA to DFA are,

1) Transform the NFA with Epsilon transitions to NFA without epsilon transitions.
2) Convert the resulting NFA to DFA.

The conversion method of resulting NFA into DFA is as follows:

Step1) The start state of NFA M will be the start of DFA M'. Hence add q_0 of NFA (start state) to Q'. Then find the transitions from this start state.

Step2) For each state $[q_1, q_2, q_3, \ldots\ldots\ldots q_i]$ in Q' the transitions for each input symbol Σ can be obtained as,

i) $\delta'([q_1, q_2, \ldots\ldots\ldots q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \ldots\ldots\ldots \delta(q_i, a) = [q_1, q_2, \ldots\ldots\ldots q_k]$ may be some state.

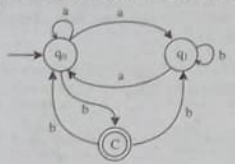
ii) Add the state $[q_1, q_2, \ldots\ldots\ldots q_k]$ to DFA if it is not already added in Q'.

iii) Then find the transitions for every input symbol from Σ for state $[q_1, q_2, \ldots\ldots\ldots q_k]$. If we get some state $[q_1, q_2, \ldots\ldots\ldots q_n]$ which is not in Q' of DFA then add this state to Q'.

iv) If there is no new state generating then stop the process after finding all the transitions.

Step3) For the state $[q_1, q_2, \ldots\ldots q_n] \in Q'$ of DFA if any one state q_i is a final state of NFA then $[q_1, q_2, \ldots\ldots q_n]$ becomes a final state. Thus the set of all the final states $\in F'$ of DFA.

Ques 44) Convert the following NFA into DFA.



Ans:

Step 1: Transform the NFA with Epsilon transitions to NFA without epsilon transitions.

Note that above NFA does not contain any epsilon transitions.

Step 2: Convert the resulting NFA to DFA.

Consider the start state q_0.

Seek all the transitions from state q_0 for all symbols a and b.

Step i)
We get,

$\delta(q_0, a) = \{q_0, q_1\}$
$\delta(q_0, b) = \{q_2\}$

Now we got,

Current State	Input	Symbol
	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_2

Step ii) a) $\delta(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, a)$
$= \delta(q_0, q_1) \cup q_0$
$= \{q_0, q_1\}$

$\delta(\{q_0, q_1\}, b) = \delta(q_0, b) \cup \delta(q_1, b)$
$= q_2 \cup q_1$
$= \{q_0, q_1\}$

Now we got,

Current State	Input	Symbol
	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_2
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Step ii) b): $\delta(\{q_1, q_2\}, a) = \delta(q_1, a) \cup \delta(q_2, a)$
$= q_0 \cup \phi$
$= q_0$

$\delta(\{q_1, q_2\}, b) = \delta(q_1, b) \cup \delta(q_2, b)$
$= q_1 \cup \{q_0, q_1\}$
$= \{q_0, q_1\}$

Now we got,

Current State	Input	Symbol
	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_1\}$	q_0	$\{q_0, q_1\}$

Step iii) c) $\delta(q_2, a) = \phi$
$\delta(q_2, b) = \{q_0 \cup q_1\}$.

Now we got,

Current State	Input	Symbol
	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_2
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_1, q_2\}$	q_0	$\{q_0, q_1\}$
$*q_2$	ϕ	$\{q_0, q_1\}$

Thus there are no more new states. The above is the transition table for the new NFA. The start state is q_0. The final states are q_2 and $\{q_1, q_2\}$, since they contain the final state, q_2 of the NFA.

Let we say,
q_0 as A
$\{q_0, q_1\}$ as B
$\{q_1, q_2\}$ as C
q_2 as D.

The new transition table is,

Current State	Input	Symbol
	a	b
$\rightarrow$ A	B	D
B	B	C
*C	A	B
*D	ϕ	B

The transition diagram for the DFA is

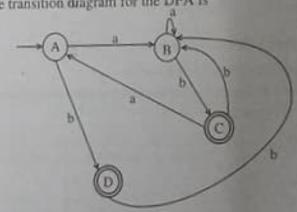


Figure 1.31

Ques 45) Prove that every NFA has an equivalent DFA.

Ans: Suppose that $N = (Q, \Sigma, \delta, q_0, F)$ is an NFA. We construct a DFA M that simulates N pretty much as in the proof of the previous theorem. In particular, if R is the set of states that N could be in after reading a string w and using any number of ε transitions, then M will be in state R.

One difference from the previous theorem is that M will not start in state $\{q_0\}$ but in state $E(\{q_0\})$. This will account for the fact that N can use any number of ε transitions even before reading the first input symbol.

The other difference is that from state R, M will have a transition labelled a going to state
$\{q \in Q \mid q \in E(\delta(r, a)), \text{ for some } r \in R\}$.

This will account for the fact that N can use any number of ε transitions after reading each input symbol (including the last one).

This is what we get: $M = (Q', \Sigma, \delta', q_0', F')$ where
$Q' = \mathcal{P}(Q)$

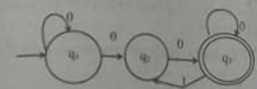
$q_0' = E(\{q_0\})$.
$F' = \{R \subseteq Q \mid R \cap F \neq \phi\}$

and δ' is defined as follows:

$$\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a)), \text{ for } R \subseteq Q \text{ and } a \in \Sigma.$$

It should be clear that $L(M) = L(N)$.

Ques 46) Convert the following NFA to DFA. **(2019[03])**

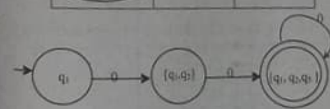


Ans: NFA to DFA Transition

	0	1
$\rightarrow q_1$	q_1, q_2	–
q_2	q_3	–
q_3	q_3	q_2

New Transition table is shown below:

δ	0	1
$\rightarrow q_1$	$\{q_1, q_2\}$	–
$\{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	–
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$



REGULAR GRAMMAR

Ques 47) Define the regular grammar.

Ans: Regular Grammar
Formally a grammar consists of a set of non-terminals (or variables) V, a set of terminals Σ (the alphabet of the language), a start symbol S, which is a nonterminal, and a set of rewrite rules (productions) P. A production has in general the form $\gamma \rightarrow \alpha$, where γ is a string of terminals and non-terminals with at least one nonterminal in it and α is a string of terminals and non-terminals. A grammar is regular if and only if γ is a single nonterminal and α is a single terminal or a single terminal followed by a single nonterminal, that is a production is of the form X -> a or X -> aY, where X and Y are non-terminals and a is terminal.

For example, $\Sigma = \{a, b\}$, $V = \{S\}$ and $P = \{S \rightarrow aS, S \rightarrow bS, S \rightarrow \Lambda\}$ is a regular grammar and it generates all the strings consisting of a's and b's including the empty string.

A regular grammar is defined as:
$G = (V, T, P, S)$

where, V = set of symbols called non-terminals which are used to define the rules.
T = a set of symbols called terminals.
P = a set of production rules.
S = a start symbol which ∈ v.

Ques 48) Prove that for any language $L \subseteq \Sigma^*$, L is regular if and only if there is a regular grammar G so that $L(G) = L - \{\Lambda\}$

Ans: First, suppose that L is regular, and let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA accepting L. Define the grammar $G = (V, \Sigma, S, P)$ by letting $V = Q$, $S = q_0$, and

$P = \{B \rightarrow aC \mid \delta(B, a) = C\} \cup \{B \rightarrow a \mid \delta(B, a) = F \text{ from some } F \in A\}$

Suppose that $x = a_1a_2 \ldots a_n$ is accepted by M, and $n \geq 1$. Then there is a sequence of transitions

$$\rightarrow q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \ldots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n$$

Where $q_n \in A$. By definition of G, we have the corresponding derivation
$S = q_0 \Rightarrow a_1q_1 \Rightarrow a_1a_2q_2 \Rightarrow \ldots \Rightarrow a_1a_2 \ldots a_{n-1}q_{n-1} \Rightarrow a_1a_2 \ldots a_{n-1}a_n$

Similarly, if x is generated by G, it is clear that x is accepted by M.

Conversely, suppose $G = (V, \Sigma, S, P)$ is a regular grammar generating L. To some extent we can reverse the construction above: We can define states corresponding to all the variables and create a transition

$$B \xrightarrow{a} C$$

for every production of the form $B \rightarrow aC$. Note that this is likely to introduce non-determinism. We can handle the other type of production by adding one extra state f, which will be the only accepting state, and letting every production $B \rightarrow a$ correspond to a transition $B \xrightarrow{a} f$

Our machine $M = (Q, \Sigma, q_0, A, \delta)$ is therefore an NFA. Q is the set $V \cup \{f\}$, q_0 the start symbol S, and A the set $\{f\}$.
For any $q \in V$ and $a \in \Sigma$,

$$\delta(q,a) = \begin{cases} \{p \mid \text{the production } q \to ap \text{ is in } P\} & \text{if } q \to a \text{ is not in } P \\ \{p \mid q \to ap \text{ is in } P\} \cup \{f\} & \text{if } q \to a \text{ is in } P \end{cases}$$

There are no transitions out of f.
If $x = a_1a_2 \ldots a_n \in L = L(G)$, then there is a derivation of the form
$S \Rightarrow a_1q_1 \Rightarrow a_1a_2q_2 \Rightarrow \ldots \Rightarrow a_1a_2 \ldots a_{n-1}q_{n-1} \Rightarrow a_1a_2 \ldots a_{n-1}a_n$
and according to our definition of M, there is a corresponding sequence of transitions

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} f$$

which implies that x is accepted by M.

On the other hand, if $x = a_1 \ldots a_l$ is accepted by M, then $l \geq 1$ because f is the only accepting state of M. The transitions causing x to be accepted look like.

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} f$$

These transitions correspond to a derivation of x in the grammar, and it follows that $x \in L(G)$.

Ques 49) Discuss the equivalence of RGs and DFA.

Ans: Equivalence of RGs and DFA
A regular grammar or type-3 defines the language called regular language that is accepted by finite Automata. A Regular Grammar G consists of 4 tuples (V, T, P, S).

Linear Grammar: A grammar is called linear in which at most one non-terminal can occur on the right side of any production rule. Following are the types of Linear Grammar:

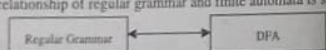
1) **Right Linear Grammar:** A right linear grammar is a grammar G = (V, T, P, S) such that all the production rules P are one of the following forms:
   A -> a
   A -> aB

   In this, A and B are variables in V i.e. A and B belongs to variable V and a is a terminal. The left-hand side of production rule in right linear grammar consists of only one symbol from set of variables, and right hand side contains either strings of terminals or only one variable present at rightmost position.

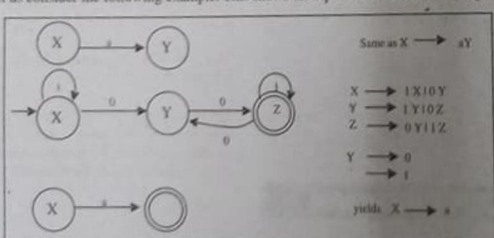
2) **Left Linear Grammar:** A left linear grammar is a grammar G = (V, T, P, S) such that all the production rules P are one of the following forms:
   A -> a
   A -> Ba

   In this A and B are variables in V i.e. A and B belongs to variables and a is a terminal.

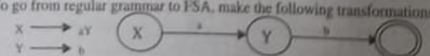
Finite Automata are the simplest model accepted the language called regular language. The term finite in finite automata is that it has a limited number of states and the limited number of alphabets in the strings. Finite Automata consists of 5 tuples. The relationship of regular grammar and finite automata is shown below:



For example, let us consider the following example. This shows the equivalence of FSA and regular grammars.

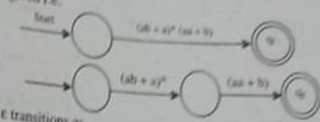


To go from regular grammar to FSA, make the following transformations:

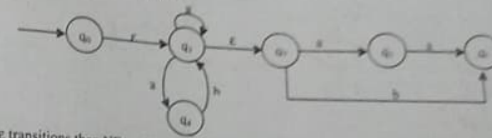


Ques 50) Construct a regular grammar for $(ab + a)^* (aa + b)$.

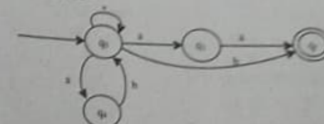
Ans: We will first construct FA for given r.e.



This FA can be constructed using ε transitions as
i.e.,



If we eliminate ε transitions then NFA will be –



The transition table can be

State \ Input	a	b
q_0	$[q_0, q_3, q_4]$	q_f
q_3	q_f	ϕ
q_4	ϕ	q_f
q_f	ϕ	ϕ

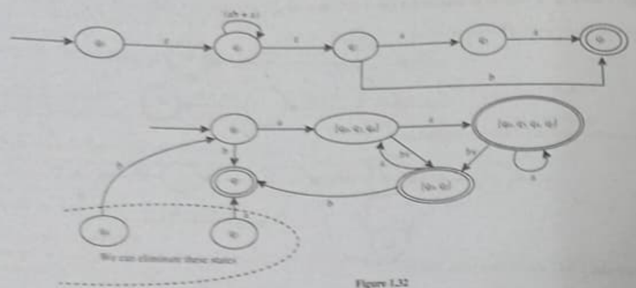
We can convert this NFA to equivalent DFA as:

$\delta'(q_0, a) = [q_0, q_3, q_4] = [q_0, q_3, q_4] \to$ new state generated
$\delta'(q_0, b) = \delta(q_0, b) = [q_f]$

We will find input transitions on $[q_0, q_3, q_4]$
$\delta'([q_0, q_3, q_4], a) = \delta(q_0, a) \cup \delta(q_3, a) \cup \delta(q_4, a) = \{q_0, q_3, q_4\} \cup \{q_f\} \cup \phi = \{q_0, q_3, q_4, q_f\} = [q_0, q_3, q_4, q_f] \to$ new state
$\delta'([q_0, q_3, q_4], b) = \delta(q_0, b) \cup \delta(q_3, b) \cup \delta(q_4, b) = \{q_f\} \cup \phi \cup \{q_0\} = \{q_0, q_f\} = [q_0, q_f] \to$ new state
$\therefore \delta'([q_0, q_3, q_4, q_f], a) = \delta(q_0, a) \cup \delta(q_3, a) \cup \delta(q_4, a) \cup \delta(q_f, a) = \{q_0, q_3, q_4\} \cup \{q_f\} \cup \phi \cup \phi = \{q_0, q_3, q_4, q_f\} = [q_0, q_3, q_4, q_f]$
$\delta'([q_0, q_3, q_4, q_f], b) = \delta(q_0, b) \cup \delta(q_3, b) \cup \delta(q_4, b) \cup \delta(q_f, b) = \{q_f\} \cup \phi \cup \{q_0\} \cup \phi = \{q_0, q_f\} = [q_0, q_f]$
$\therefore \delta'([q_0, q_f], a) = \delta(q_0, a) \cup \delta(q_f, a) = \{q_0, q_3, q_4\} \cup \phi = [q_0, q_3, q_4]$
$\delta'([q_0, q_f], b) = \delta(q_0, b) \cup \delta(q_f, b) = \{q_f\} \cup \phi = [q_f]$

The transition table for this DFA will be:

State \ Input	a	b
$\to [q_0]$	$[q_0, q_3, q_4]$	$[q_f]$
$[q_3]$	$[q_f]$	ϕ
$[q_4]$	ϕ	$[q_f]$
$[q_f]$	ϕ	ϕ
$[q_0, q_3, q_4]$	$[q_0, q_3, q_4, q_f]$	$[q_0, q_f]$
$[q_0, q_3, q_4, q_f]$	$[q_0, q_3, q_4, q_f]$	$[q_0, q_f]$
$[q_0, q_f]$	$[q_0, q_3, q_4]$	$[q_f]$



We can eliminate these states

Figure 1.32

Now we can rename the states as:

$[q_0]$	$= S$
$[q_0, q_1, q_2]$	$= A$
$[q_0, q_1, q_2, q_3]$	$= B$
$[q_0, q_3]$	$= C$
$[q_3]$	$= D$

The DFA with renamed states will be:

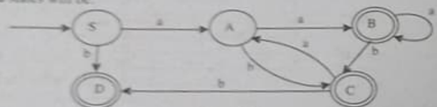


Figure 1.33

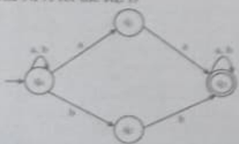
The regular grammar can be:

$S \to aA$	$S \to bD$	$S \to b$
$A \to aB$	$A \to a$	$A \to bC$
$A \to b$	$B \to aB$	$B \to a$
$B \to bC$	$B \to b$	$C \to aA$
$C \to a$	$C \to bD$	$C \to b$
$D \to \epsilon$		

Ques 51) Construct Regular grammar for the regular expression: (2017 [05])

$$L = (a + b)^*(aa + bb)(a + b)^*$$

Ans: The NFA for the RE is



There are four states in the FA. So in the regular grammar, there are four non-terminals

Let us take them as A (for q_0), B (for q_1), C (for q_2), and D (for q_3).

Now, we have to construct the production rules of the grammar. For the state q_0, the production rules are

$A \to aA$, $A \to bA$, $A \to aB$, $A \to bC$

For the state q_1, the production rules are

$B \to aD$, $B \to a$ (as D is the final states).

For the state q3, the production rules are

$C \to bD$, $C \to b$ (as D is the final state).

For the state q_3, the production rules are

$D \to aD$, $D \to bD$, $D \to a/b$.

The grammar

$= \{V_N, \Sigma, P, S\}$

$V_N = \{A, B, C, D\}$ $\Sigma = \{a, b\}$

$P: A \to aA/bA/bB/bC$

$B \to aD/a$

$C \to bD/b$

$D \to aD/bD/a/b.$

Ques 52) Construct a regular grammar for

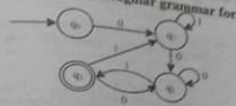


Figure 1.34

Ans: The equivalent regular grammar can be denoted by
$G = \{V, T, P, S\}$ where
$V = \{A_0, A_1, A_2, A_3\}$

The production rules can be:

$A_0 \to 0A_1$
$A_1 \to 1A_1$
$A_1 \to 0A_2$
$A_2 \to 0A_2$
$A_2 \to 1A_3$
$A_2 \to 1$
$A_3 \to 1A_1$
$A_3 \to 0A_2$

Ques 53) Construct a finite automata recognizing L(G) where G is the grammar.

$S \to aS \mid bA \mid b$

$A \to aA \mid bS \mid a$

Ans: We can have the FA M as

$M = (\{q_0, q_1, q_f\}, \{a, b\}, \delta, q_0, \{q_f\})$

Where q_0 and q_1 correspond to S and A and q_f is a final state.

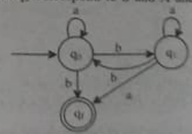


Figure 1.35

This is basically NFA because from q_0 with b there are two next states q_1 and q_f.

Ques 54) Construct a deterministic finite automaton equivalent to the grammar

$S \to aS \mid bS \mid aA$

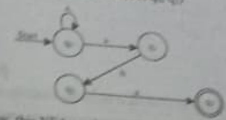
$A \to bB$

$B \to aC$

$C \to A$

Ans: The DFA is denoted by,

$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, q_3)$



Now, this NFA can be converted to equivalent DFA

$\delta'([q_0], a) = \{q_0, q_1\} = [q_0, q_1]$
$\delta'([q_0], b) = \{q_0\} = [q_0]$
$\delta'([q_1], a) = \phi$
$\delta'([q_1], b) = \{q_2\} = [q_2]$
$\delta'([q_2], a) = \{q_1\} = [q_1]$
$\delta'([q_2], b) = \phi$

$\delta'([q_0, q_1], a) = \delta([q_0], a) \cup \delta([q_1], a) = \{q_0, q_1\} = [q_0, q_1]$
$\delta'([q_0, q_1], b) = \delta([q_0], b) \cup \delta([q_1], b) = \{q_0, q_2\} = [q_0, q_2]$
$\delta'([q_0, q_2], b) = \delta([q_0], a) \cup \delta([q_2], a)$
$= [q_0, q_1] \cup [q_1] = [q_0, q_1, q_2]$

$\delta'([q_0, q_2], b) = \delta(q_0, b) \cup \delta(q_2, b) = [q_0] \cup \phi = [q_0]$
$\delta'([q_0, q_1, q_2], a) = \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)$
$= [q_0, q_1] \cup \phi \cup \phi = \{q_0, q_1\} = [q_0, q_1]$

$\delta'([q_0, q_1, q_2], b) = \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)$
$= \{q_0, q_1\} \cup \{q_2\} \cup \phi = \{q_0, q_1, q_2\}$
$= [q_0, q_1, q_2]$

$\delta'([q_0, q_1, q_2], a) = \delta([q_0], a) \cup \delta([q_1], a) \cup \delta([q_2], a)$
$= [q_0, q_1] \cup \phi \cup \{q_1\} = \{q_0, q_1, q_2\}$
$= [q_0, q_1, q_2]$

$\delta'([q_0, q_1, q_2], b) = \delta([q_0], b) \cup \delta([q_1], b) \cup \delta([q_2], b)$
$= [q_0] \cup \{q_2\} \cup \phi = \{q_0, q_2\} = [q_0, q_2]$

The transition table will be:

	Input / State	a	b
→	$[q_0]$	$[q_0, q_1]$	$[q_0, q_2]$
	$[q_1]$	ϕ	$[q_2]$
	$[q_2]$	$[q_1]$	ϕ
*	$[q_3]$	ϕ	ϕ
	$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_2]$
	$[q_0, q_2]$	$[q_0, q_1, q_2]$	$[q_0]$
*	$[q_0, q_1, q_3]$	$[q_0, q_1]$	$[q_0, q_1, q_2]$
	$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_2]$

The final states are marked by *.

Module 2

More on Regular Languages

REGULAR EXPRESSION

Ques 1) Define the regular expression with example.

Ans: Regular Expressions

Regular expressions are useful for representing certain sets of stings in an algebraic fashion. Actually these describe the language accepted by finite state automata.

A regular expression is said to denote a formal language L over the alphabet and is defined by the following rules:

1) ∈ is a regular expression denoting the language which consists of null string.
2) If 'a' is a symbol in the alphabet then 'a' is also a regular expression consists of {a}, i.e., any terminal symbol is also a regular expression.
3) If R and S are regular expressions over the alphabet then (R)/(S) is also a regular expression, which is the Union of the languages represented by R and S, i.e., $L(R) \cup L(S)$.
4) If R and S are regular expressions over the alphabet then (R)(S) is also a regular expression, denoting L(R)L(S).
5) R* is a regular expression representing the language L(R) with zero or more occurrences of the string.
6) If R is regular expression then (R) is also a regular expression.

The language denoted by a regular expression is said to be **regular set**.

Regular expressions can be viewed as language generators in such a way that the strings in the language can be produced from left to right – i.e., we can imagine each symbol of the generated string to be output as it is determined. Context free grammars also language generators with some set of rules.

For example,

1) The regular expression a* denotes ε, a, aa, aaa, aaaa, ……
   (Remember the operation of * to be zero or more occurrences, hence, the string 'a' can occur in zero or more cases, so ε is also included)
2) a* b denotes b, ab, aab, aaab, ……………
   (a can occur in zero or more cases along with b)
3) a* b* denotes ε, ab, aab, abb, aabb
   (a and b can occur in zero or more cases)
4) a* ab denotes {aab, aaab,……}

There is a close relationship between a finite automata and the regular expression we can show this relation in **figure 2.1:**

Figure 2.1: Relationship between FA and regular expression

The **figure 2.1** shows that it is convenient to convert the regular expression to NFA with ε moves. Let us see the theorem based on this conversion.

Ques 2) Write the identities rules for regular expression.

Ans: Identities Rules for Regular Expression

Identities rules are useful for simplifying regular expressions:

I_1 $\phi + R = R$
I_2 $\phi R = R\phi = \phi$
I_3 $\Lambda R = R\Lambda = R$
I_4 $\Lambda^* = \Lambda$ and $\phi^* = \Lambda$
I_5 $R + R = R$
I_6 $R^*R^* = R^*$
I_7 $RR^* = R^*R$
I_8 $(R^*)^* = R^*$
I_9 $\Lambda + RR^* = R^* = \Lambda + R^*R$
I_{10} $(PQ)^*P = P(QP)^*$
I_{11} $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$
I_{12} $(P + Q)R = PR + QR$ and $R(P + Q) = RP + RQ$

We have found a method of describing the possible tokens of programming languages that can appear in the input stream of lexical analysis, i.e., Regular Expression.

Ques 3) Find a regular expression corresponding to the language of all strings over the alphabet {a, b} that contain exactly two a's.

Ans: A string in this language must have only two a's. Since any string of b's can be placed in front of the first a, behind the second a and between the two a's, and since an arbitrary string of b's can be represented by the regular expression b*, b*a b*a b* is a regular expression for this language.

Ques 4) Find a regular expression corresponding to the language of strings of even lengths over the alphabet {a, b}.

Ans: Since any string of even length can be expressed as the concatenation of strings of length 2 and the strings of length 2 available to us are aa, ab, ba, bb, a regular expression corresponding to the language is (aa + ab + ba + bb)*. Note that 0 is an even number. Hence the string ε is in this language.

Ques 5) Describe as simply as possible in English the language corresponding to the regular expression $((a + b)^3)^* (\varepsilon + a + b)$.

Ans: $((a + b)^3)$ represents the strings of length 3. Hence $((a + b)^3)^*$ represents the strings whose length is multiple of 3. Since $((a + b)^3)^* (a + b)$ represents the strings of length 3n + 1, where n is a natural number, the given regular expression represents the strings of length 3n and 3n + 1, where n is a natural number.

Ques 6) Describe as simply as possible in English the language corresponding to the regular expression (b + ab)*(a + ab)*.

Ans: (b + ab)* represents strings which do not contain any substring aa and which end in b, and (a + ab)* represents strings which do not contain any substring bb. Hence altogether it represents any string consisting of a substring with no aa followed by one b followed by a substring with no bb.

Ques 7) Describe the following sets by regular expressions:

1) {101},
2) {abba},
3) {01, 10},
4) {Λ, ab},
5) {abb, a, b, bba},
6) {Λ, 0, 00, 000,...}, and
7) {1,11, 111,...}.

Ans:

1) Now, {1}, {0} are represented by 1 and 0, respectively. 101 is obtained by concatenating 1, 0 and 1. So, {101} is represented by 101.
2) abba represents {abba}.
3) As {01, 10} is the union of {01} and {10}, we have {01,10} represented by 01 + 10.
4) The set {Λ, ab} is represented by Λ + ab.
5) The set {abb, a, b, bba} is represented by abb + a + b + bba.
6) As {Λ, 0, 00, 000, ...} is simply {0}*, it is represented by 0*.
7) Any element in {1,11,111,...} can be obtained by concatenating 1 and any element of {1}*. Hence 1(1)* represents {1, 11, 111,...}.

Ques 8) Describe the following sets by regular expressions:

1) L_1 = the set of all strings of 0's and 1's ending in 00.
2) L_2 = the set of all strings of 0's and 1's beginning with 0 and ending with 1.
3) L_3 = {Λ, 11, 1111, 111111,...}.

Ans:

1) Any string in L_1 is obtained by concatenating any string over {0, 1} and the string 00. {0, 1} is represented by 0 + 1. Hence L_1 is represented by (0 + 1)* 00.
2) As any element of L_2 is obtained by concatenating 0, any string over {0, 1} and 1, L_2 can be represented by 0(0 + 1)*1.
3) Any element of L_3 is either Λ or a string of even number of 1's, i.e., a string of the form $(11)^n$, n ≥ 0. So L_3 can be represented by (11)*.

Ques 9) Write the process of constructing regular grammar from regular expression. Also give an example.

Ans: Construction of Regular Grammar from Regular Expression

We can convert the regular expression into its equivalent regular grammar by using following method:

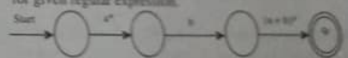
Step 1) Construct a NFA with ε from given regular expression.

Step 2) Eliminate ε transitions and convert it to equivalent DFA.

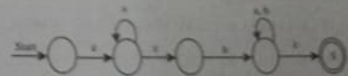
Step 3) From constructed DFA, the corresponding states become non-terminal symbols and transitions made are equivalent to production rules.

For example, construct a regular grammar for the regular expression a*b (a + b)*.

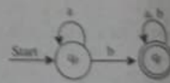
We will first construct a DFA in a straightforward manner for given regular expression.



Now we will convert it to NFA with ε transitions.



Now eliminate ε moves.



We can write the regular grammar as

$G = (V, T, P, A_0)$ where

$V = \{A_0, A_1\}$, $T = \{a, b\}$

$P = \{A_0 \rightarrow aA_0$

$A_0 \to bA_1$

$A_0 \to b$

$A_1 \to aA_1$

$A_1 \to bA_1$

$A_1 \to a$

$A_1 \to b$

and A_0 is a start symbol.

Thus G is required regular grammar.

Ques 10) Prove that the regular expression can also be represented by its equivalent deterministic finite automata.

Ans: Equivalance of DFA and Regular Expressions(REs)

Let, L be the set of the language accepted by the DFA.

The DFA can be denoted by the
$M = (\{q_1, q_2, \dots q_n\}, \Sigma, \delta, q_1, F)$

Let,

r_{ij}^k denotes the set of the strings x such that $\delta(q_i, x) = q_j$.

The q_i and q_j indicates source state to target state respectively. This inputs are going through the states of finite automata means that with some input entering into the states and coming out of it. The value of k is always less than i or j.

The r_{ij}^k is denoted by,

$$r_{ij}^k = r_{ik}^{k-1}(r_{kk}^{k-1})^* r_{kj}^{k-1} \cup r_{ij}^{k-1}$$

$$r_{ij}^0 = \begin{cases} \{\alpha \mid \delta(q_i, \alpha) = q_j\} & \text{if } i \neq j \\ \{\alpha \mid \delta(q_i, \alpha) = q_j\} \cup \{\varepsilon\} & i = j \end{cases}$$

We have to show that for each i, j, k there exists a regular expression r_{ij}^k denoting the language r_{ij}^k. We will put the induction on k basis (k = 0). The r_{ij}^0 is a set of strings each of which is either ε or a single symbol. The r_{ij}^0 is based on $\delta(q_i, a) = q_j$. The r_{ij}^0 denotes the set of such a's, if there is no such a then it will be taken a's ϕ. If i = j the all the a's + ε will be the set.

Induction: The formula for getting the language r_{ij}^k is given by regular expression.

$$r_{ij}^k = (r_{ik}^{k-1})(r_{kk}^{k-1})^*(r_{kj}^{k-1}) + r_{ij}^{k-1}$$

which completes the induction

To get the final regular expression, we have to simply get the language r_{ij}^k, where i indicates start state, and j indicates final state and n will be number of items. If there are p number of paths, leading to final state, the

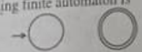
$$r_{ij}^k = r_{ij1}^n + r_{ij2}^n + \dots + r_{ijp}^n$$

where F is a set of final states $F = \{q_{j1}, q_{j2}, q_{jp}\}$

Ques 11) Prove that for every regular expression there is an equivalent finite automaton.

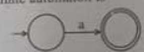
Ans: The proof hinges on the fact that regular expressions are defined recursively, so that, once the basic steps are shown for constructing finite automata for the primitive elements of regular expressions, finite automata for regular expressions of arbitrary complexity can be constructed by showing how to combine component finite automata to simulate the basic operations. For convenience, we shall construct finite automata with a unique accepting state. (Any non-deterministic finite automaton with ε moves can easily be transformed into one with a unique accepting state by adding such a state, setting up an ε transition to this new state from every original accepting state, and then turning all original accepting states into rejecting ones).

For the regular expression ϕ denoting the empty set, the corresponding finite automaton is

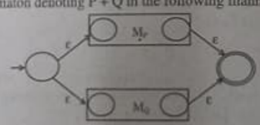


For the regular expression ε denoting the set {ε}, the corresponding finite automaton is

For the regular expression a denoting the set {a}, the corresponding finite automaton is

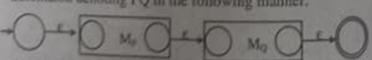


If P and Q are regular expressions with corresponding finite automata M_P and M_Q, then we can construct a finite automaton denoting P + Q in the following manner:



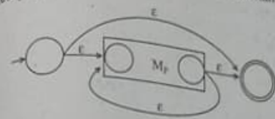
The ε transitions at the end are needed to maintain a unique accepting state.

If P and Q are regular expressions with corresponding finite automata M_P and M_Q, then we can construct a finite automaton denoting PQ in the following manner:



Finally, if P is a regular expression with corresponding finite automaton M_P, then we can construct a finite automaton denoting P^* in the following manner:

Again, the extra ε transitions are here to maintain a unique accepting state.



It is clear that each finite automaton described above accepts exactly the set of strings described by the corresponding regular expression (assuming inductively that the sub-machines used in the construction accept exactly the set of strings described by their corresponding regular expressions). Since, for each constructor of regular expressions, we have a corresponding constructor of finite automata, the induction step is proved and our proof is complete.

Ques 12) Construct an FSA equivalent to the regular expression

$$(0 + 1)^* (00 + 11) (0 + 1)^*$$

Ans: $(0 + 1)^* (00 + 11)(0 + 1)^*$ is of the form r s r, where $r = (0 + 1)^*$ and $s = 00 + 11$. We express $r = (0 + 1)^* = (r_1 + r_2)^*$ where $r_1 = 0$, $r_2 = 1$ and $s = s_1 + s_2$ where $s_1 = 00$ and $s_2 = 11$.

The automaton for $r_1 = 0$ is:

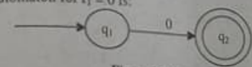


Figure 2.2 (a)

The automaton for $r_2 = 1$ is:

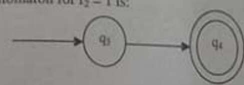


Figure 2.2 (b)

The construction for $r = (r_1 + r_2)^*$ is:

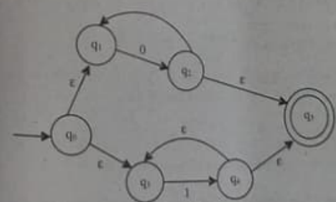


Figure 2.2 (c)

The automaton for $s_1 = 00$ is:

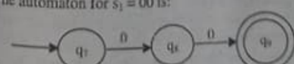


Figure 2.2 (d)

The automaton for $s_2 = 11$ is:

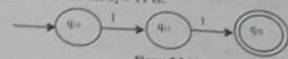


Figure 2.2 (e)

The automaton for $s_1 + s_2$ is:

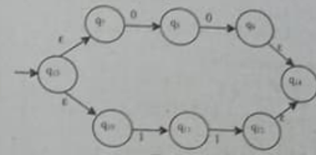


Figure 2.2 (f)

Hence, the automaton for $(r_1 + r_2)^* (s_1 + s_2) (r_1 + r_2)^*$ is:

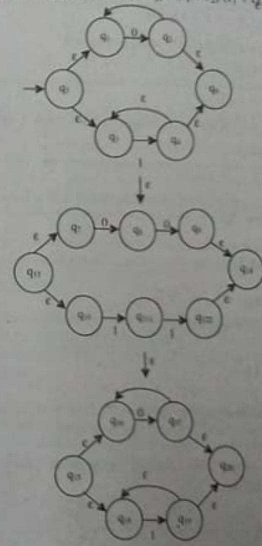


Figure 2.3

Ques 13) Construct regular expression from given finite automata.

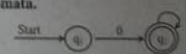


Figure 2.4

Ans: As we have seen in previous example, we are following the important formula as

$$r_{ij}^k = r_{ik}^{k-1}(r_{kk}^{k-1})^*(r_{kj}^{k-1}) + r_{ij}^{k-1}$$

Let us compute when k = 0

Table 2.4

	k = 0
r_{11}	ε
r_{12}	0
r_{21}	ϕ
r_{22}	$1+\varepsilon$

Now we will calculate for k = 1

	k = 1
	Computation
r_{11}	$i=1, j=1, k=1$ $r_{11}^1 = r_{11}^0 (r_{11}^0)^* r_{11}^0 + r_{11}^0 = \varepsilon(\varepsilon)^*(\varepsilon)+\varepsilon$ $r_{11}^1 = \varepsilon$ $k=1$
r_{12}	$i=1, j=2, k=1$ $r_{12}^1 = r_{11}^0 (r_{11}^0)^* r_{12}^0 + r_{12}^0 = \varepsilon 0 + 0 = 0+0$ $r_{12}^1 = 0$
r_{21}	$i=2, j=1, k=1$ $r_{21}^1 = r_{21}^0 (r_{11}^0)^* r_{11}^0 + r_{21}^0 = \phi.\varepsilon.\varepsilon+\phi = \phi+\phi \therefore \phi\varepsilon=\phi$ $r_{21}^1 = 0$
r_{22}	$i=2, j=2, k=1$ $r_{22}^1 = r_{21}(r_{11})^* r_{12}^0 + r_{22}^0 = \phi(\varepsilon)^*0+(1+\varepsilon) = 1+\varepsilon$

Now for calculating regular expression we should compute for the path from start state to final. That is from q_1 to q_2.

Considering $i=1, j=2, k=2$

$$\therefore \quad r_{12}^2 = r_{12}^1 (r_{22}^1)^* r_{22}^1 + r_{12}^1$$
$$= (0.1^*(1+\varepsilon)) + 0$$

$r_{12}^2 = 01^* + 0$

his is a final regular expression. This is a language ginning wih 0 and followed by any number of 1's.

es 14) Find the regular expression for the lowing DFA.

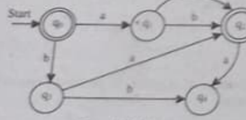


Figure 2.5: DFA

ince, q_0 is the initial and final state, we will need to the regular expression ε to see that it also accepts trings. Look at sate q_4. Since it is a dead end we consider transitions from q_2 and q_3 to form the xpression.

q_2 is also a final state and there are three different paths from q_0 to q_2. These are,

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2$$
$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2$$

and

$$q_0 \xrightarrow{b} q_3 \xrightarrow{a} q_2$$

So the complete regular expression will be

ε | aa | ab | ba

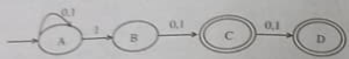
Ques 15) Write regular expression for the language L = {1ⁿ0ᵐ |n>=1, m>=0}. (2019[03])

Ans: A regular expression for $L_1 = \{1^n : n>=1\}$ is 1 +11 + 111 +....

A regular expression for $L_2 = \{0^m : m>=0\}$ is λ + 0 + 00 + 000 +....

Thus, a regular expression for $L = L_1 L_2$ is (1 +11 + 111 +....)(λ + 0 + 00 + 000 +....)

Ques 16) Construct regular expression corresponding to the following state diagram: (2019[4.5])



Ans: We have replaced A, B, C and D with q1, q2, q3 and q4 respectively.

Let us write down the equations for each state:

$q1 = q_1 0 + q_1 1 + \varepsilon \quad = q1\,(0+1)+\varepsilon$
(Initial State)

$q2 = q_1 1$

$q3 = q_2 0 + q_2 1 = q_2\,(0+1)$

$q4 = q_3 0 + q_3 1 = q_3\,(0+1)$

Since final states are q3 and q4, we are interested in solving q3 and q4 only.

Let us see q1 first.

$q1 = q1\,(0+1)+\varepsilon$

We will compare R = Q + RP with above equation, so R= q1, Q = ε, R =(0 + 1). So we can write:

$q1 = \varepsilon\,(0+1)^*$

Or, $q1 = (0+1)^*$

Thus, the regular expression is $q1 = (0+1)^*$.

Ques 17) Formally define the regular language and regular set.

Ans: Regular Languages

A regular language (also known as a **regular set** or a **regular event** or a **rational language**) is a language that can be expressed with a regular expression or a deterministic or non-deterministic finite automata or state machine.

As in Chomsky Hierarchy, Regular Languages are the most restricted types of languages and are accepted by finite automata. Regular languages are a subset of the set of all strings. Regular languages are used in parsing and designing programming languages and are one of the first concepts taught in computability courses. These are useful for helping computer scientists to recognize patterns in data and group certain computational problems together – once they do that, they can take similar approaches to solve the problems grouped together.

The regular language is defined as follows:

Definition: Let Σ be an alphabet the class of 'regular languages' over Σ is defined as:

1) ϕ is a regular language.
2) For each $\sigma \in \Sigma$, (σ) is a regular language.
3) For any natural number $n \geq 2$ if $L_1, L_2, L_3 ... L_n$ are regular languages, then $L_1 \cup L_2 \cup L_3 ... \cup L_n$ also a regular language.
4) For any natural number $n \geq 2$ if $L_1, L_2, L_3 ... L_n$ are regular languages then $L_1 \circ L_2 \circ L_3 \_\_\_ \circ L_n$ also regular language.
5) If L is a regular language then L^* is also a regular language.

Ques 18) Describe the properties of regular languages.

Or

List the closure properties of Regular sets. (2017 [04])

Or

Define the decision properties of regular languages.

Or

Which of the following operations are closed under regular sets Justify your answer: (2019[4.5])
i) Complementation
ii) Set difference
iii) String reversal
iv) Intersection

Ans: Properties of Regular Languages

The most important of properties of regular languages include,

1) **Closure Properties of Regular Languages:** The closure properties of regular sets are as follows:
   i) The regular sets are closed under union, i.e., if L_1, L_2 are regular sets, then $L_1 \cup L_2$ is also regular.
   ii) The regular sets are closed under concatenation, i.e., if L_1, L_2 are regular sets, then L_1L_2 is also regular.
   iii) The regular sets are closed under keen closure, i.e., if L is a regular set, then L^* is also regular.
   iv) The class of regular sets is closed under complementation, i.e., if L is a regular set, and L $\subseteq \Sigma^*$, then $\Sigma^* - L$ is a regular set.
   v) The regular sets are closed under intersection, i.e., if L_1, L_2 are regular sets, then $L_1 \cap L_2$ is also regular.
   vi) The reversion of a regular set is regular, i.e., if L is a regular set, then L^T is also regular.

2) **Decision Properties of Regular Languages:** Some other important facts about regular languages are called "**decision properties**". These properties give us algorithms for answering important questions about automata. **For example,** a central example is an algorithm for deciding whether two automata define the same languages.

The decision algorithm for regular sets requires the following points to be remembered:

i) An algorithm must always terminate to be called an algorithm. Basically, an algorithm needs to have the following four characteristics:
   a) An algorithm must be written using a finite number of unambiguous (one and only one meaning) steps.
   b) For every possible input, only a finite number of steps are to be performed, and the algorithm is supposed to produce a result.
   c) Every time, the same and correct result is to be produced for the same input.
   d) Each step of the algorithm must have the properties as explained in (i), (ii) and (iii).

ii) A regular language is just a set of strings over a finite alphabet. Every regular set can be represented by a regular expression, and accepted by a minimum state DFA.

   We choose DFAs represented by usual notations so that we can analyze every DFA, and even simulate them.

Ques 19) State the pumping lemma for regular language.

Ans: Pumping Lemma for Regular Languages

Properties of regular languages are that whether certain language is regular or not, this is shown by the pumping lemma.

Theorem: Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, one can break w into three strings, w = xyz, such that:

1) $y \neq \varepsilon$.
2) $|xy| \leq n$.
3) For all $k \geq 0$ the string xy^kz is also in L.

That is, one can always find a non-empty string y not too far from the beginning of w that can be "pumped", i.e., repeating y any number of times, or deleting it (the case k = 0), keeps the resulting string in the language L.

Ques 20) Discuss the closure under simple set operations.

Ans: Closure under Simple Set Operations

1) **Union:** If L1 and If L2 are two regular languages, their union L1 $\cup$ L2 will also be regular. **For example,** L1 = $\{a^n \mid n \geq 0\}$ and L2 = $\{b^n \mid n \geq 0\}$, then L3 = L1 $\cup$ L2 = {an $\cup$ bn | n $\geq$ 0} is also regular.

2) **Intersection:** If L1 and If L2 are two regular languages, their intersection $L1 \cap L2$ will also be regular. **For example,** $L1 = \{a^n b^m \mid n \geq 0 \text{ and } m \geq 0\}$ and $L2 = \{a^m b^n \cup b^n a^m \mid n \geq 0 \text{ and } m \geq 0\}$, then $L3 = L1 \cap L2 = \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ is also regular.

3) **Concatenation:** If L1 and If L2 are two regular languages, their concatenation L1L2 will also be regular. **For example,** $L1 = \{a^n \mid n \geq 0\}$ and $L2 = \{b^n \mid n \geq 0\}$, then $L3 = L1L2 = \{a^m . b^n \mid m \geq 0 \text{ and } n \geq 0\}$ is also regular.

4) **Kleene Closure:** If L1 is a regular language, its Kleene closure L1* will also be regular. **For example,** $L1 = (a \cup b)$, then $L1^* = (a \cup b)^*$

5) **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,**
$L(G) = \{a^n \mid n > 3\}$
$L'(G) = \{a^n \mid n <= 3\}$

Ques 21) Discuss about the Closure under Other Operations.

Or

Explain the Homomorphism and Inverse Homomorphism with example.

Ans: Closure under Other Operations

In addition to the standard operations on languages, one can define other operations and investigate closure properties for them. There are many such results; several typical ones are as:

1) **Homomorphism:** A string homomorphism is a function on strings that works by substituting a particular string for each symbol.

Suppose Σ and Γ are alphabets. Then a function:
$h : \Sigma \rightarrow \Gamma^*$

is called a homomorphism. In other words, a homomorphism is a substitution in which a single letter is replaced with a string. The domain of the function h is extended to strings in an obvious fashion; if

$w = a_1a_2 \ldots a_n$

Then,
$h(w) = h(a_1) h(a_2) \ldots h(a_n)$

If L is language on Σ, then its homomorphic image is defined as:
$h(L) = \{h(w) : w \in L\}$.

Example: Let $\Sigma = \{a, b\}$ and $\Gamma = \{a, b, c\}$, and define h by
$h(a) = ab$,
$h(b) = bbc$.

Then $h(aba) = abbbcab$. The homomorphic image of $L = \{aa, aba\}$ is the language $h(L) = \{abab, abbbcab\}$.

If we have a regular expression r for a language L, then a regular expression for h(L) can be obtained by simply applying the homomorphism to each Σ symbol of r.

2) **Inverse Homomorphism:** Homomorphism may also be applied "backwards" and in this mode they also preserve regular languages. That is, suppose h is a homomorphism from some alphabet Σ to strings in another (possibly the same) alphabet T. Let L be a language over alphabet T. Then $h^{-1}(L)$, read "h inverse of L", is the set of strings w in Σ^* such that h(w) is in L.

3) **Figure 2.6** suggests the effect of a homomorphism on a language L in part (a), and the effect of an inverse homomorphism in part (b).

Figure 2.6: Homomorphism Applied in the Forward and Inverse Direction

Ques 22) How regular expression can be converted into finite automata?

Ans: Conversion of Regular Expression (RE) to Finite Automata (FA)

To the regular expressions over the alphabet Σ belong the following:

1) Every element of the alphabet Σ and symbol ε.
2) Expressions $(a \cup b)$, (ab), and a^*, where a and b, in turn, are regular expressions over Σ.
3) Empty set ϕ is often considered a regular expression.

By using these "atoms" we can construct any, even a very complex, regular expression. Thus, for converting a regular expression into a finite automaton we must learn how to represent these elementary, atomic constructions as automata. Let's consider each of them.

1) **Single Character a of the Used Alphabet:** The machine for the recognition of a character is constructed in an obvious way (**figure 2.7**).

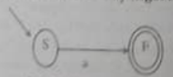


Figure 2.7: Machine that Recognizes a Single Character of the Alphabet

By changing the transition labeled a to the ε-transition we will get the machine that recognizes the empty string ε.

2) **Union of Two Regular Expressions a and b [i.e., $(a \cup b)$]:** Let us suppose that for expressions a and b we have already constructed the appropriate A and B machines.

The task is to unite them correctly. The solution is shown in **figure 2.8.** Here, it is possible (and actually necessary) to use non-deterministic behavior.

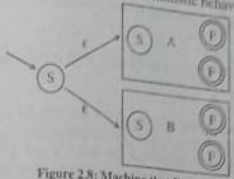


Figure 2.8: Machine that Recognizes the Union of Two Regular Expressions

Let us suppose that on the machine's input there is a string that corresponds to the regular expression b. The first step splits the world into two new ones. In the first world the machine transfers to the initial state of block A, and in the second, to the initial state of block B. Block A does not recognize the input expression and the machine in the first world ends its work in an ordinary state. Block B in the second world recognizes the input expression, which means the acceptance of the expression by the non-deterministic machine.

3) **Concatenation of Expressions (ab):** Here, we must place blocks A and B sequentially, as in this case the machine has to recognize two expressions individually (**figure 2.9**). Favorable states of machine A are connected to the initial state of machine B:

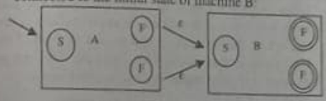


Figure 2.9: Machine that Recognizes the Concatenation of Two Regular Expressions

4) **Machine for the Kleene Closure ("Star"):** This machine looks a little more extravagant than the previous ones. Since syntax a^* means "zero or more occurrences", the initial state of the machine must be also favorable. If there is a string in the input, then it is necessary to process it entirely, and then return to the initial state of the machine (**figure 2.10**).

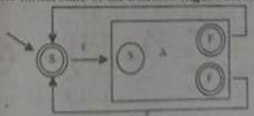


Figure 2.10: Machine Corresponding to the Kleene Closure

From the theory of regular expressions we know that any concatenation or union should be put into parentheses. That is why it is always possible to find the most interior atomic sub-expression and convert it to the machine. Then we continue the converting process for the exterior sub-expression, and so on. It is like the process of parsing and evaluating arithmetical expressions with parentheses, but instead of executing the arithmetical operations, we translate textual constructions into states and transition rules. In practice, regular expressions usually are not overloaded with parentheses, but there exists the order of operations precedence, so it is possible to place missing brackets automatically if desired.

To explain the process of converting a regular expression into a finite-state machine, we consider a simple **example**.

Let us construct an automaton corresponding to the expression $(a \cup b)^*c$ (**figure 2.11**).

The process is as follows:

1) The constructing process begins from two atomic automata that recognize the single characters a and b.
2) In the second step they are joined in accordance with **figure 2.11** to get the mechanism that recognizes the union $a \cup b$.
3) The next step is the implementation of the Kleene closure (**figure 2.11**).
4) In the last step the sequential connection of the machines is used to get the concatenation of expressions $(a \cup b)^*$ and c. During this operation we join the favorable states of the first machine with the initial state of the second one. The fact that, in this case, the favorable state of the first machine is simultaneously its initial state does not play any role.

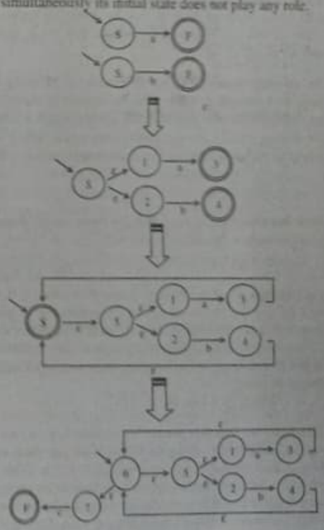


Figure 2.11: Constructing a Machine for the Expression $(a \cup b)^*c$.

Ques 23) Discuss the necessary conditions for regular languages.

Or

Prove that a language is regular language if and only if some finite state machine recognizes it.

Or

Prove that the language accepted by any finite automaton is regular.

Ans: Let $L \subseteq \Sigma^*$ be accepted by the FA $M = (Q, \Sigma, q_0, A, \delta)$. What this means is that $L = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in A\}$. By considering the individual elements of A, we can express L as the union of a finite number of sets of the form $\{x \in \Sigma^* \mid \delta^*(q_0, x) = q\}$; because finite unions of regular sets are regular, it will be sufficient to show that for any two states p and q, the set

$$L(p, q) = \{x \in \Sigma^* \mid \delta^*(p, x) = q\} \text{ is regular.}$$

In looking for ways to use mathematical induction, we have often formulated statements involving the length of a string. At first this approach does not seem promising, because we are trying to prove a property of a language, not a property of individual strings in the language. However, rather than looking at the number of transitions in a particular path from p to q (i.e., the length of a string in L(p, q)), we might look at the number of distinct states through which M passes in moving from p to q. It turns out to be more convenient to consider, for each k, a specific set of k states, and to consider the set of strings that cause M to go from p to q by going through only states in that set. If k is large enough, this set of strings will be all of L(p, q).

To simplify notation, let us the states of M using the integers 1 through n, where n is the number of states. Let us also formalize the idea of a path going through a state s: for a string x in Σ^*, we say x represents a path from p to q going through s if there are non-null strings y and z so that

$$x = yz \qquad \delta^*(p, y) = s$$
$$\delta^*(s, z) = q$$

Note that a path can go to a state, or from a state, without going through it. In particular, the path

$$p \xrightarrow{a} q \xrightarrow{b} r$$

goes through q, but not through p or r. Now, for $j \geq 0$, we let L(p, q, j) be the set of strings corresponding to paths from p to q that go through no state numbered higher than j. No string in L(p, q) can go through a state numbered higher than n, because there are no states numbered higher than n. In other words,

$$L(p, q, n) = L(p, q)$$

The problem, therefore, is to show that L(p, q, n) is regular, and this will obviously follow if we can show that L(p, q, j) is regular for every j with $0 \leq j \leq n$. (This is where the induction comes in.) In fact, there is no harm in asserting that L(p, q, j) is regular for every $j \geq 0$; this is not really a stronger statement, since for any $j \geq n$, L(p, q, j) = L(p, q, n), but this way it will look more like an ordinary induction proof.

For the basis step we need to show that L(p, q, 0) is regular. Going through no state numbered higher than 0 means going through no state at all, which means that the string can contain no more than one symbol.

Therefore,

$L(p, q, 0) \subseteq \Sigma \cup \{\varepsilon\}$ and L(p, q, 0) is regular because it is finite.

The induction hypothesis is that $0 \leq k$ and that for every p and q satisfying $0 \leq p, q \leq n$, the language L(p, q, k) is regular. We wish to show that for every p and q in the same range, L(p, q, k + 1) is regular. As we have already observed, L(p, q, k + 1) = L(p, q, k) if $k \geq n$, and we assume for the remainder of the proof that $k < n$.

A string x is in L(p, q, k + 1) if it represents a path from p to q that goes through no state numbered higher than k + 1. There are two ways this can happen:

1) The path can bypass the state k + 1 altogether, in which case it goes through no state higher than k, and $x \in L(p, q, k)$.

2) The path can go through k + 1 and nothing higher. In this case, it goes from p to the first occurrence of k + 1, then loops from k + 1 back to itself zero or more times, then goes from the last occurrence of k + 1 to q. (**figure 2.12**). This means that we can write x as yzw, where y corresponds to the path from p to the first occurrence of k + 1, z to all the loops from k +1 back to itself, and w to the path from k + 1 to q. The crucial observation here is that in each of the two parts y and w, and in each of the individual loops making up z, the path does not go through any state higher than k; in other words,

$$y \in L(p, k+1, k) \qquad w \in L(k+1, q, k) \qquad z \in L(k+1, k+1, k)^*.$$

It follows that in either of the two cases,

$$x \in L(p, q, k) \cup L(p, k+1, k)L(k+1, k+1, k)^* L(k+1, q, k)$$

On the other hand, it is also clear that any string in this right-hand set is an element of L (p, q, k + 1), since the corresponding path goes from p to q without going through any state higher than k + 1.

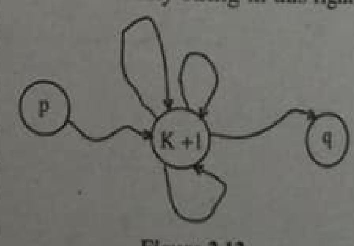


Figure 2.12

Therefore,

$$L(p, q, k+1) = L(p, q, k) \cup L(p, k+1, k)L(k+1, k+1, k)^* L(k+1, q, k)$$

Each of the languages appearing in the right side of the formula is regular because of the induction hypothesis, and L(p, q, k + 1) is obtained from them by using the operations of union, concatenation, and Kleene*. Therefore, L(p, q, k + 1) is regular.

Ques 24) Design a FA from given regular expression 10 + (0 + 11)0*1.

Ans: First we will construct the transition diagram for given regular expression.

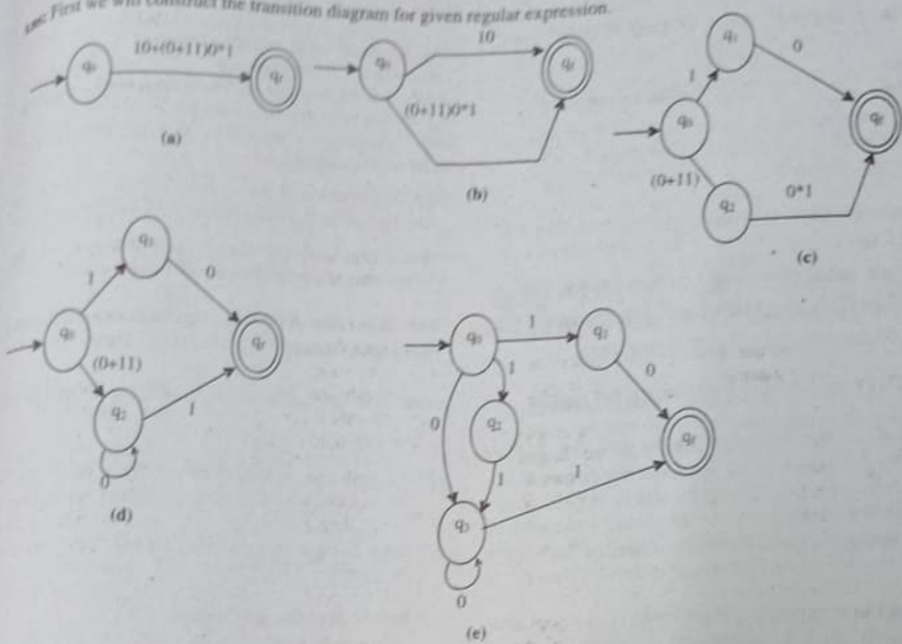


Figure 2.13

Now we have got NFA without ε. Now we will convert it to required DFA for that, we will first write a transition table for this NFA.

State \ Input	0	1
$\{q_0\}$	$\{q_3\}$	$\{q_1, q_2\}$
$\{q_1\}$	$\{q_f\}$	ϕ
$\{q_2\}$	ϕ	$\{q_3\}$
$\{q_3\}$	$\{q_3\}$	$\{q_f\}$
$\{q_1, q_2\}$	$\{q_f\}$	$\{q_3\}$
$\{q_f\}$	ϕ	ϕ

The equivalent DFA will be

State \ Input	0	1
q_0	q_3	$\{q_1, q_2\}$
q_1	q_f	ϕ
q_2	ϕ	q_3
q_3	q_3	q_f
q_f	ϕ	ϕ

Ques 25) Let $\Sigma = \{0, 1\}$, then check the regularity of language $L = \{0^k1^k \mid k \geq 0\}$.

Ans: The language L consists of following set of string {ε, 01, 0011, 000111, ...}. We shall prove the regularity of L by method of contradiction. Thus, assume L is regular and n is any constant then string Z can be written as:

$$Z = 0^n . 1^n \text{ i.e., } |Z| = 2n \text{ so } |Z| > n$$

Now break the string Z into substrings u, v, and w (**figure 2.14**), i.e.,

$$Z = 0^n . 1^n = u.v.w$$

Where, substrings fulfill the condition such that

$$(\underbrace{000.........0}_{u}\ \underbrace{0......000}_{v}) . (\underbrace{111..........1111}_{w})$$

Figure 2.14

We observe that string u.v consist only of 0's so $|u.v| \leq n$ and $|v| \geq 1$. For the verification of the base case of pumping lemma, i.e., string $Z = u.v^i.w \Rightarrow u.w$ (for $i = 0$). Lemma says if L is regular then $u.w \in L$. Since substring u contains 0's that are fewer than n (because of few 0's are part of substring v) and substring w contains exactly n, 1's. So, string u.w does not have equal numbers of 0's followed by equal number of 1's. Hence, we obtain a contradiction, therefore L is not regular.

Ques 26) Is the following language regular? Justify your answer $L = \{0^{2n} \mid n \geq 1\}$.

Ans: This is a language length of string is always even, i.e.,

$n = 1$; $L = 00$
$n = 2$ $L = 00\ 00$ and so on.

Let $L = uvw$
$L = 0^{2n}$
$|z| = 2^n = uv^iw$

If we add 2n to this string length:
$|z| = 4n = uv.vw$
= even length of string

Thus even after pumping 2n to the string we get the even length. So the language L is regular language.

Ques 27) Prove $L = \{a^p \mid p \text{ is a prime}\}$ is not regular.

Ans: Let us assume L is a regular and P is a prime number.

$L = a^P$
$|z| = uvw$ $i = 1$

Now consider $L = uv^iw$ where $i = 2$
$= uv.\ vw$

Adding 1 to P we get, $P < |uvvw|$
$P < P + 1$

But P + 1 is not a prime number. Hence what we have assumed becomes contradictory. Thus L behaves as it is not a regular language.

Ques 28) Show that $L = \{0^n 1^{n+1} \mid n > 0\}$ is not regular.

Ans: Let us assume that L is a regular language.
$|z| = |uvw| = 0^n 1^{n+1}$

Length of string $|z| = n + n + 1 = 2n + 1$. That means length is always odd.

By pumping lemma
$= |uv.vw|$

That is if we add 2n + 1
$2n + 1 < (2n + 1) + 2n + 1$
$2n + 1 < 4n + 2$

But if n = 1 then we obtain 4n + 2 = 6 which is no way odd. Hence the language becomes irregular.

Even if we add 1 to length of $|z|$, then
$|z| = 2n + 1 + 1 = 2n + 2$
= even length of the string

So this is not a regular language.

Ques 29) Write the applications of pumping lemma.
Or
Write the use of pumping lemma.

Ans: Applications of Pumping Lemma
The pumping lemma statement can be used to prove that certain sets are not regular. The steps needed for proving that a given set is not regular are given as follows:
Step 1: Assume L as regular. Let n be the number of states in the corresponding FA.

Step 2: Choose a string w such that $|w| \geq n$. Use the pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Step 3: Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

Ques 30) Is the following language regular? Justify your answer. $L = \{0^{2n} \mid n \geq 1\}$.

Ans: This is a language length of string is always even, i.e.,
$n = 1$; $L = 00$
$n = 2$ $L = 00\ 00$ and so on.

Let $L = uvw$
$L = 0^{2n}$
$|z| = 2^n = uv^iw$

If we add 2n to this string length:
$|z| = 4n = uv.vw$
= even length of string.

Thus even after pumping 2n to the string we get the even length. So the language L is regular language.

Ques 31) Prove $L = \{a^P \mid p \text{ is a prime}\}$ is not regular.

Ans: Let us assume L is a regular and P is a prime number.
$L = a^P$
$|z| = uvw$ $i = 1$

Now consider $L = uv^iw$ where $i = 2$
$= uv.\ vw$

Adding 1 to P we get,
$P < |uvvw|$
$P < P + 1$

But P + 1 is not a prime number. Hence what we have assumed becomes contradictory. Thus L behaves as it is not a regular language.

Ques 32) Show that $L = \{0^n 1^{n+1} \mid n > 0\}$ is not regular.

Ans: Let us assume that L is a regular language.
$|z| = |uvw|$
$= 0^n 1^{n+1}$

...gth of string $|z| = n + n + 1 = 2n + 1$. That means length ... always odd.

...y pumping lemma

$= |uv.vw|$

...hat is if we add $2n + 1$

$...n + 1 < (2n + 1) + 2n + 1$

$...n + 1 < 4n + 2$

...e if $n = 1$ then we obtain $4n + 2 = 6$ which is no way ...dd. Hence the language becomes irregular.

...en if we add 1 to length of $|z|$, then

$|z| = 2n + 1 + 1$

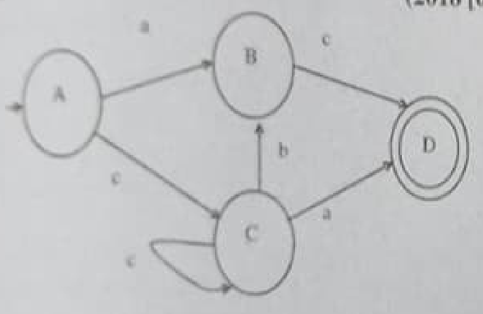
$= 2n + 2$

= even length of the string

So this is not a regular language.

The simple way to know whether given language is regular or not is that try to draw finite automata for it, if you can easily draw the FA for the given L then that language is surely the regular otherwise not.

Ques 33) What is the regular expression for the DFA. **(2018 [03])**



Figure

Ans: Regular Expression = ac+cc*(a+bc). So the complete regular expression will be $\varepsilon | ac | cc^* a | cc^* bc$

Ques 34) Verify that the following languages is not regular: **(2019[4.5])**

$\{a^n b^{2n} \mid n > 0\}$

Ans: If L were regular then there would exist some k such that any string w where $|w| \geq k$ must satisfy the condition of the theorem.

Let $w = a^{\lceil k/2 \rceil} b^{\lceil k/2 \rceil}$

Since $|w| \geq k$, w must satisfy the condition of the pumping theorem. So, for some x, y, and z, w = xyz, $|xy| \leq k$, $y \neq \varepsilon$, and $\forall q \geq 0$, $xy^q z$ is in L. We show that no such x, y, and z exist. There are 3 cases for where y could occur; we divide w into two regions.

aaaa....aaaaaa|bbbbb.....bbbbbb

1 | 2

So y can fall in:

1) (1): $y = a^p$ for some p. Since $y \neq \varepsilon$, p must be greater than 0. Let q = 2. The resulting string is $a^{k+p}b^k$. But this string is not in L, since it has more a's than b's.

2) (2): $y = b^p$ for some p. Since $y \neq \varepsilon$, p must be greater than 0. Let q = 2. The resulting string is $a^k b^{k+p}$. But this string is not in L, since it has more b's than a's.

3) (1, 2): $y = a^p b^r$ for some non-zero p and r. Let q = 2. The resulting string will have interleaved a's and b's and so is not in L.

So there exists one string in L for which no x, y, z exist. So L is not regular.

MINIMIZATION OF DFA

Ques 35) What is Minimization of DFA?

Ans: Minimization of DFA

A finite automaton M is determined by giving the following five items:

1) A finite set Q of states,
2) A finite set Σ of input symbols,
3) An initial state ($\in$ Q),
4) A set F ($\subseteq$ Q) of accepting states, and
5) A state transition function δ.

If δ is a mapping from $Q \times \Sigma$ into Q then M is said to be deterministic. If is a mapping from $Q \times \Sigma$ into 2^Q then M is said to be non-deterministic. The domain of $\hat{\delta}$ can be naturally extended from $Q \times \Sigma$ to $Q \times \Sigma^*$.

The definition of the language accepted by M is as usual and we omit it. Two finite automata are said to be equivalent if they accept the same language. A DFA (NFA) M is said to be minimal if there is no DFA (NFA) M′ that is equivalent to M and has fewer states than M.

It is well known that a DFA M is minimal if:

1) All its states are reachable from the initial state, and
2) There are no two equivalent states (two states q_1 and q_2 are said to be equivalent if for all $x \in \Sigma^*$, $\delta(q_1, x) \in F$ iff $\delta(q_2, x) \in F$).

For an NFA M of n states $\Delta(M, n)$ denotes the number of states of the minimal DFA that is equivalent to M. NFA's should also be minimal.

Definition: Two states q_1 and q_2 are equivalent (denoted by $q_1 \equiv q_2$) if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states, or both of them are non-final states for all $x \in \Sigma^*$.

Definition: Two states q_1 and q_2 are k-equivalent ($k \geq 0$) if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both non-final states for all strings x of length k or less. In particular, any two final states are 0-equivalent and any two non-final states are also 0-equivalent.

Ques 36) Write the process of minimal state finite automata?

Ans: Minimal State Finite Automata

Minimization of automata refers to detect those states of automata whose presence or absence in a automata does not affect the language accepted by automata. These states are like **Unreachable states, Dead states, Non-distinguishable states** etc. The minimization problem suggest the way that how we will find a minimum state DFA equivalent to given DFA. Since there is essentially an unique minimum state DFA for every regular expression Myhill Nerode theorem. Two finite automata are said to be equivalent if they accept the same language. A DFA (NFA) M is said to be minimal if there is no DFA (NFA) M' that is equivalent to M and has fewer states than M. It is well known that a DFA M is minimal if:

3) All its states are reachable from the initial state, and
4) There are no two equivalent states (two states q_1 and q_2 are said to be equivalent if for all $x \in \Sigma^*$, $\delta(q_1, x) \in F$ iff $\delta(q_2, x) \in F$).

For an NFA M of n states $\Delta(M, n)$ denotes the number of states of the minimal DFA that is equivalent to M. NFA's should also be minimal.

Ques 37) Suppose $p, q \in Q$, and x and y are strings with $x \in L_p$ and $y \in L_q$ (in other words, $\delta^*(q_0, x) = p$ and $\delta^*(q_0, y) = q$). Then prove that the following three statements are all equivalent:

1) $p = q$.
2) $L/x = L/y$ (i.e., x I_L y, or x and y are indistinguishable with respect to L).
3) For any $z \in \Sigma^*$, $\delta^*(p, z) \in A \Leftrightarrow \delta^*(q, z) \in A$ (i.e., $\delta^*(p, z)$ and $\delta^*(q, z)$ are either both in A or both not in A).

Ans: Proof

To see that statements 2 and 3 are equivalent, we begin with the formulas:

$\delta^*(p, z) = \delta^*(\delta^*(q_0, x), z) = \delta^*(q_0, xz)$
$\delta^*(q, z) = \delta^*(\delta^*(q_0, y), z) = \delta^*(q_0, yz)$

Saying that $L/x = L/y$ means that a string z is in one set if and only if it is in the other, or that $xz \in L$ if and only if $yz \in L$; since M accepts L, this is exactly the same as statement 3.

Now if statement 1 is true, then L_p and L_q are both subsets of the same equivalence class. This means that x and y are equivalent, which is statement 2. The converse is also true, because we know that if L_p and L_q are not both subsets of the same equivalence class, then they are subsets of the different equivalence classes, so that statement 2 does not hold.

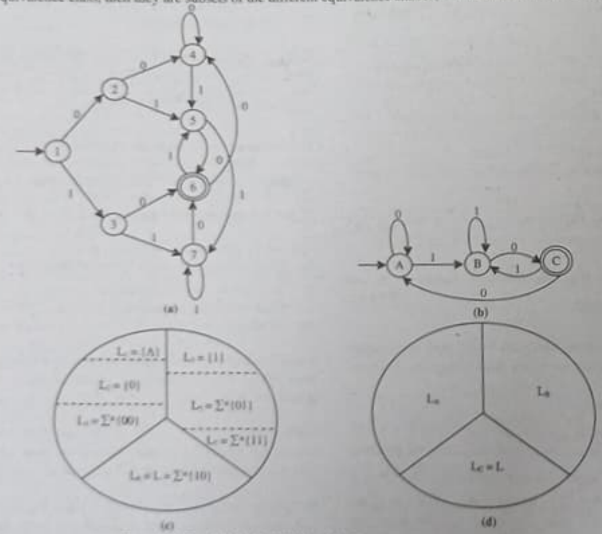


Figure 2.15: Two FAs for $\{0, 1\}^*\{10\}$ and the Corresponding Partitions of $\{0, 1\}^*$

Let us now consider how it can happen that $p \not\equiv q$. According to the lemma, this means that for some z, exactly one of the two states $\delta^*(p, z)$ and $\delta^*(q, z)$ is in A.

The simplest way this can happen is with $z = \Lambda$, so that only one of the states p and q is in A. Once we have one pair (p, q) with $p \not\equiv q$, we consider the situation where $r, s \in Q$ and for some $a \in \Sigma$, $\delta(r, a) = p$ and $\delta(s, a) = q$. We may write:

$$\delta^*(r, az) = \delta^*(\delta^*(r, a), z) = \delta^*(\delta(r, a), z) = \delta^*(p, z)$$

and similarly, $\delta^*(s, az) = \delta^*(q, z)$. Since $p \not\equiv q$, then for some z, exactly one of the states $\delta^*(p, z)$ and $\delta^*(q, z)$ is in A; therefore, exactly one of $\delta^*(r, az)$ and $\delta^*(s, az)$ is in A, and $r \not\equiv s$.

These observations suggest the following recursive definition of a set S, which will turn-out to be the set of all pairs (p, q) with $p \not\equiv q$.

1) For any p and q for which exactly one of p and q is in A, (p, q) is in S.
2) For any pair $(p,q) \in S$, if (r, s) is a pair for which $\delta(r, a) = p$ and $\delta(s, a) = q$ for some $a \in \Sigma$, then (r, s) is in S.
3) No other pairs are in S.

It is not difficult to see from the comments preceding the recursive definition that for any pair $(p, q) \in S$, $p \not\equiv q$. δ contains all such pairs by establishing the following statement: For any string $z \in \Sigma^*$, every pair of states (p, q) for which only one of the states $\delta^*(p, z)$ and $\delta^*(q, z)$ is in A is an element of S.

We do this by using structural induction on z. For the basis step, if only one of $\delta^*(p, \Lambda)$ and $\delta^*(q, \Lambda)$ is in A, then only one of the two states p and q is in A and $(p, q) \in S$ because of statement 1 of the definition.

Now suppose that for some z, all pairs (p, q) for which only one of $\delta^*(p, z)$ and $\delta^*(q, z)$ is in A are in S. Consider the string az, where $a \in \Sigma$, and suppose that (r, s) is a pair for which only one of $\delta^*(r, az)$ and $\delta^*(s, az)$ is in A. If we let $p = \delta(r, a)$ and $q = \delta(s, a)$, then we have:

$$\delta^*(r, az) = \delta^*(\delta(r, a), z) = \delta^*(p, z)$$
$$\delta^*(s, az) = \delta^*(\delta(s, a), z) = \delta^*(q, z)$$

The assumption on r and s is that only one of the states $\delta^*(r, az)$ and $\delta^*(s, az)$ and therefore only one of the states $\delta^*(p, z)$ and $\delta^*(q, z)$, is in A. Induction hypothesis therefore implies that $(p, q) \in S$, and it then follows from statement 2 in the recursive definition that $(r, s) \in S$.

Ques 38) Discuss the different properties of minimization of finite automata.

Ans: Properties of Minimization of Finite Automata

Property 1: The relations we have defined, i.e., equivalence and k-equivalence, are equivalence relations, i.e., they are reflexive, symmetric and transitive.

Property 2: Induce partitions of Q. These partitions can be denoted by π and π_k, respectively. Elements of π_k are k-equivalence classes.

Property 3: If q_1 and q_2 are k-equivalent for all $k \geq 0$, then they are equivalent.

Property 4: If q_1 and q_2 are (k + 1)-equivalent, then they are k-equivalent.

Property 5: $\pi_n = \pi_{n+1}$ for some n. (π_n denotes the set of equivalence classes under n-equivalence)

Ques 39) Write the steps of construction of minimum automaton.

Ans: Steps of Construction of Minimum Automaton

Step 1: (Construction of π_0). By definition of 0-equivalence, $\pi_0 = \{Q_1^0, Q_2^0\}$, where Q_1^0 is the set of all final states and $Q_2^0 = Q - Q_1^0$.

Step 2: (Construction of π_{k+1} from π_k). Let Q_i^k be any subset in π_k. If q_1 and q_2 are in Q_i^k, they are (k + 1)-equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are k-equivalent. Find out whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are in the same equivalence class in π_k for every $a \in \Sigma$. If so, q_1 and q_2 are (k + 1)-equivalent. In this way, Q_i^k is further divided into (k + 1)-equivalence classes. Repeat this for every Q_i^k in π_k to get all the elements of π_{k+1}.

Step 3: Construct π_n for n = 1, 2, ... until $\pi_n = \pi_{n+1}$.

Step 4: (Construction of minimum automaton). For the required minimum state automaton, the states are the equivalence classes obtained in step 3, i.e., the elements of π_n. The state table is obtained by replacing a state q by the corresponding equivalence class [q].

The number of equivalence classes is less than or equal to |Q|. Consider an equivalence class $[q_1] = \{q_1, q_2, ..., q_k\}$. If q_1 is reached while processing $w_1w_2 \in T(M)$ with $\delta(q_0, w_1) = q_1$, then $\delta(q_1, w_2) \in F$. So $\delta(q_i, w_2) \in F$ for i = 2, ..., k. Thus we see that q_i, i = 2, ..., k is reached on processing some $w \in T(M)$ iff q_1 is reached on processing w, i.e., q_1 of $[q_1]$ can play the role of q_2, ..., q_k. The above argument explains why we replace a state by the corresponding equivalence class.

Ques 40) Let the language $L = \{x \in \{0, 1\}^*$, the second symbol from the right is a 1, then the corresponding regular expression is $(0 + 1)^*1.(0 + 1)$ and the possible DFA that accepts given regular expression is shown in figure 2.16:

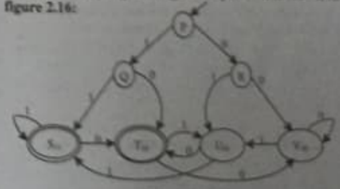


Figure 2.16: (M)

Now, is this is a minimum states DFA M? If not, then construct the minimum states DFA.

Where, M = ({P, Q, R, S, T, U, V}, {0, 1}, δ, {P}, {S, T})

Ans: Find Equivalence Classes

1) **0-Equivalence Classes:** Test the transition of states over the symbol ∈ and make groups of the states which are equivalent and which are distinguishable.

Since, $PR_0Q \Rightarrow \delta(P, \in) = P(\notin F)$ and $\delta(Q, \in) = Q(\notin F)$ so they are distinguishable.

And, $PR_0R \Rightarrow \delta(P, \in) = P(\notin F)$ and $\delta(R, \in) = R(\notin F)$ so they are distinguishable.

And, $PR_0S \Rightarrow \delta(P, \in) = P(\notin F)$ and $\delta(S, \in) = S(\notin F)$ so they are distinguishable.

Similarly test all other states with each other we find states P, Q, R, U, and V are distinguishable so, they form a group

{P, Q, R, U, V}

Only states S and T are equivalence, because

$SR_0T \Rightarrow \delta(S, \in) = S(\in F)$ and $\delta(T, \in) = T(\in F)$ so they are equivalence and form another group {S, T}

Hence, 0-equivalence classes are,

{P, Q, R, U, V} and {S, T}

2) **1-Equivalence Classes:** Test the equivalence relation between states of groups for the strings of length less then or equal to one (i.e., for the symbols ∈, 0, and 1).

First take-up the group of states (P, Q, R, U, V) and test for equivalence,

Symbol ∈ is not able to distinguish between states of above group so, test equivalence with respect to another symbols 0 and 1.

$PR_1Q \Rightarrow \delta(P, 0/1) = R/Q(\notin F)$ and $\delta(Q, 0/1) = T/S(\in F)$ so they are distinguishable.

And.$PR_1R \Rightarrow \delta(P,0/1) \notin F$ and $\delta(R,0/1) \in F$;

so they are distinguishable.

And.$PR_1S \Rightarrow \delta(P,0/1) \notin F$ and $\delta(S,0/1) \in F$;

so they are distinguishable.

And.$PR_1U \Rightarrow \delta(P,0/1) \notin F$ and $\delta(U,0/1) \in F$;

so they are distinguishable.

And.$PR_1V \Rightarrow \delta(P,0/1) \notin F$ and $\delta(V,0/1) \notin F$;

so they are distinguishable.

Similarly test for other pairs of states in this group, we find states Q and U are equivalent because,

$QR_1U \Rightarrow \delta(Q, 0/1) \in F$ and $\delta(U, 0/1) \in F$; so they places in other group.

Test equivalence relation for other group of states {S, T}, we see that

$SR_1T \Rightarrow \delta(S, 0/1) \in F$ and $\delta(T, 0/1) \notin F$; since they are distinguishable so they will not be in the same group. Hence 1-equivalence classes are,

{P, R, V}, {Q, U}, {S} and {T}

(this is a refinement of 0-equivalence class)

3) **2-Equivalence Classes:** Now test the equivalence relation for the strings ε, 0, 1, 00, 01, 10, 11 (all strings of length ≤ 2). Since we form the groups of states w.r.t strings ε, 0 and 1 so further it cannot be distinguish. Now test the equivalence relation between states of the groups only the over remaining strings. We say it xy, where x and y are either 0 or 1.

Since, $PR_2R \Rightarrow \delta(P, 00/01) \notin F$ and $\delta(R, 00/01) \notin F$; so they are distinguishable.

and, $PR_2V \Rightarrow \delta(P, 00/01) \notin F$ and $\delta(V, 00/01) \notin F$; so they are distinguishable.

and, $RR_2V \Rightarrow \delta(R, 00/01) \notin F$ and $\delta(V, 00/01) \notin F$; so they are distinguishable. Since, we could not find any equivalent of states in this group so there is no split in the group.

Next, test for equivalence in other group {Q, U}.

$QR_2U \Rightarrow \delta(Q, 00/01) \notin F$ and $\delta(U, 00/01) \in F$; so they are distinguishable.

So there is no split in this group also.

Since, group {S} and {T} contains single state so further there is no split in these groups.

Hence, 2-equivalence classes are,

{P, R, V}, {Q, U}, {S} and {T}

That is similar to 1-equivalence classes and since there is no further refinement. So, process to find equivalence classes terminate. Hence, equivalent states are ⇒ {P, R, V}, {Q, U}, {S} and {T}

Therefore, minimum state DFA has just above 4-states. Transition nature of groups is given by the transitions of all the states in the group that return on the same group or some other group.

Thus, we obtain the transition table shown in **table 2.1**:

Table 2.1

State	Input Symbol	
	0	1
→{P, R, V}	Return on same group {P, R, V}	Return on group {Q, U}
• {Q, U}	Return on group {T}	Return on group {S}
{T}	Return on group {V}	Return on group {U}
{S}	Return on group {T}	Return on group {S}

And the minimum states DFA is shown in **figure 2.17**. [whose language is also the language expressed by the regular expression (0 + 1)*1(0 + 1)]

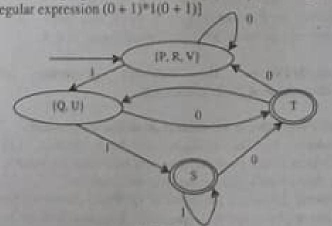


Figure 2.17

Ques 41) Minimise the following DFA. (2019[06])

δ	a	b
→ P0	P0	P1
P1	P2	P1
P2	P3	P1
*P3	P3	P4
*P3	P5	P4
*P5	P3	P4

Ans: The DFA is shown in **figure 2.18**

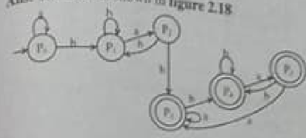


Figure 2.18: DFA

The transition is shown in below:

δ	a	b
→ P_0	P_0	P_1
P_1	P_2	P_1
P_2	P_3	P_1
*P_3	P_3	P_4
*P_4	P_5	P_4
*P_5	P_3	P_4

Now,

$\pi_0 = \{\{P_3\}, \{P_4\}, \{P_5\}, \{P_0, P_1, P_2\}\}$

$\delta(P_0, a) = P_0 \in \{P_0, P_1, P_2\}$

$\delta(P_1, a) = P_2 \in \{P_0, P_1, P_2\}$

Similarly,

$\delta(P_0, b) = P_1 \in \{P_0, P_1, P_2\}$

$\delta(P_1, b) = P_1 \in \{P_0, P_1, P_2\}$

Thus, $\{P_0, P_1\}$ are equivalent.

Similarly checking equivalent of $\{P_0, P_2\}$ which are not equivalent. Thus $\pi_1 = \{\{P_3\}, \{P_4\}, \{P_5\}, \{P_0, P_1\}, \{P_2\}\}$

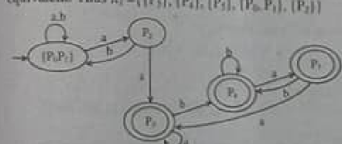


Figure 2.19: Minimization of DFA

Ques 42) Construct a minimum state automata equivalent to the finite automaton given in figure 2.20.

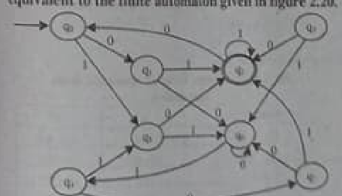


Figure 2.20: FA of Example

Ans: It will be easier if we construct the transition table given in **table below**.

Table: Transition Table		
State/Σ	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

By applying step 1, we get

$$Q_1^0 = F = \{q_2\},\ Q_2^0 = Q - Q_1^0$$

So,

$$\pi_0 = \{\{q_2\}, \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\}$$

$\{q_2\}$ in π_0 cannot be further partitioned. So, $Q'_1 = \{q_2\}$. Consider q_0 and $q_1 \in Q_2^0$. The entries under 0-column corresponding to q_0 and q_1 are q_1 and q_6; they lie in Q_2^0. The entries under 1-column are q_5 and q_2. $q_2 \in Q_1^0$ and $q_5 \in Q_2^0$. Therefore, q_0 and q_1 are not 1-equivalent. Similarly, q_0 is not 1-equivalent to q_3, q_5 and q_7.

Now, consider q_0 and q_4. The entries under 0-column are q_1 and q_7. Both are in Q_2^0. The entries under 1-column are q_5, q_5. So q_4 and q_0 1-equivalent. Similarly, q_6 is 1-equivalent to q_0. $\{q_0, q_4, q_6\}$ is a subset in π_1. So, $Q'_2 = \{q_0, q_4, q_6\}$.

Repeat the construction by considering q_1 and any one of the states q_3, q_5, q_7. q_1 is not 1-equivalent to q_3 or q_5 but 1-equivalent to q_7. Hence, $Q'_3 = \{q_1, q_7\}$. The elements left over in Q_2^0 are q_3 and q_5. By considering the entries under 0-column and 1-column, we see that q_3 and q_5 are 1-equivalent. So $Q'_4 = \{q_3, q_5\}$.

Therefore,

$$\pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

$\{q_2\}$ is also in π_2 as it cannot be partitioned further. Now the entries under 0-column corresponding to q_0 and q_4 are q_1 and q_7, and these lie in the same equivalence class in π_1. The entries under 1-column are q_5, q_5. So q_0 and q_4 are 2-equivalent. But q_0 and q_6 are not 2-equivalent. Hence, $\{q_0, q_4, q_6\}$ is partitioned into $\{q_0, q_4\}$ and $\{q_6\}$. q_1 and q_7 are 2-equivalent. q_3 and q_5 are also 2-equivalent. Thus, $\pi_2 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$. q_0 and q_4 are 3-equivalent. q_1 and q_7 are 3-equivalent. Also, q_3 and q_5 are 3-equivalent. Therefore,

$$\pi_3 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

As $\pi_2 = \pi_3$, π_2 gives the equivalence classes, the minimum state automaton is:

$$M' = (Q', \{0, 1\}, \delta', q'_0, F')$$

where

$$Q' = \{[q_2], [q_0, q_4], [q_6], [q_1, q_7], [q_3, q_5]\}$$
$$q'_0 = [q_0, q_4], \qquad F' = [q_2]$$

and δ' is given by **table below**.

Table: Transition Table of Minimum State Automaton

State/Σ	0	1
$[q_0, q_4]$	$[q_1, q_7]$	$[q_5, q_3]$
$[q_1, q_7]$	$[q_6]$	$[q_2]$
$[q_2]$	$[q_0, q_4]$	$[q_2]$
$[q_3, q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0, q_4]$

Note: The transition diagram for the minimum state automaton is given in **figure 2.20**. The states q_0 and q_4 are identified and treated as one state. (So also are q_1, q_7, and q_3, q_5.) But the transitions in both the diagrams (i.e., **figure 2.20**) are the same. If there is an arrow from q_i to q_j with label a, then there is an arrow from $[q_i]$ to $[q_j]$ with the same label in the diagram for minimum state automaton. Symbolically, if $\delta(q_i, a) = q_j$, then $\delta'([q_i], a) = [q_j]$.

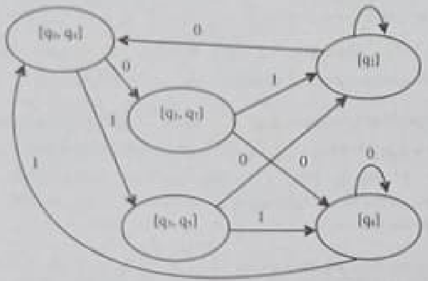


Figure 2.21: Minimum State Automaton

Ques 43) Construct the minimum state automaton equivalent to the transition diagram given by figure 2.21.

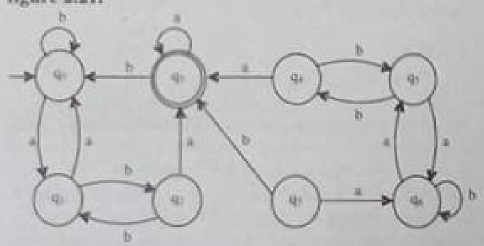


Figure 2.22

Ans: We construct the transition table given in **table 2.2**. Since there is only one final state q_3, $Q_1^0 = \{q_3\}$, $Q_2^0 = Q - Q_1^0$. Hence, $\pi_0 = \{\{q_3\}, \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}\}$. As $\{q_3\}$ cannot be partitioned further, $Q'_1 = \{q_3\}$.

Now q_0 is 1-equivalent to q_1, q_5, q_6, but not to q_2, q_4, q_7 and so $Q'_2 = \{q_0, q_1, q_5, q_6\}$. q_2 is 1-equivalent to q_4.

Hence $Q'_3 = \{q_2, q_4\}$. The only element left over in Q_2^0 is q_7. Therefore, $Q'_4 = \{q_7\}$. Thus,

Table 2.2: Transition Table for Example 6

State/Σ	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

$$\pi_1 = \{\{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4\}, \{q_7\}\}$$
$$Q_1^2 = \{q_3\}$$

q_0 is 2-equivalent to q_6 but not to q_1 or q_5. So,

$$Q_2^2 = \{q_0, q_6\}$$

As q_1 is 2-equivalent to q_5,

$$Q_3^2 = \{q_1, q_5\}$$

As q_2 is 2-equivalent to q_4,

$$Q_4^2 = \{q_2, q_4\}, \; Q_5^2 = \{q_7\}$$

Thus,

$$\pi_2 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$$
$$Q_1^3 = \{q_3\}$$

As q_0 is 3-equivalent to q_6,

$$Q_2^3 = \{q_0, q_6\}$$

As q_1 is 3-equivalent to q_5,

$$Q_3^3 = \{q_1, q_5\}$$

As q_2 is 3-equivalent to q_4,

$$Q_4^3 = \{q_2, q_4\}, \; Q_5^3 = \{q_7\}$$

Therefore,

$$\pi_3 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$$

As $\pi_3 = \pi_2$, π_2 gives us the equivalence classes, the minimum state automaton is $M' = (Q', \{a, b\}, \delta', q'_0, F')$, where

$$Q' = \{[q_3], [q_0, q_6], [q_1, q_5], [q_2, q_4], [q_7]\}$$
$$q'_0 = [q_0, q_6], \qquad F' = [q_3]$$

δ' is given by **table 2.3**.

Table 2.3: Transition Table of Minimum State Automaton

State/Σ	a	b
$[q_0, q_6]$	$[q_1, q_5]$	$[q_0, q_6]$
$[q_1, q_5]$	$[q_0, q_6]$	$[q_2, q_4]$
$[q_2, q_4]$	$[q_3]$	$[q_1, q_5]$
$[q_3]$	$[q_3]$	$[q_0, q_6]$
$[q_7]$	$[q_0, q_6]$	$[q_3]$

Ques 44) Find minimum finite-state automata for the DFA shown in figure 2.22.

	0	1
→a	b	f
b	g	c
*c	a	c
e	h	f
f	c	g
g	g	e
h	g	c

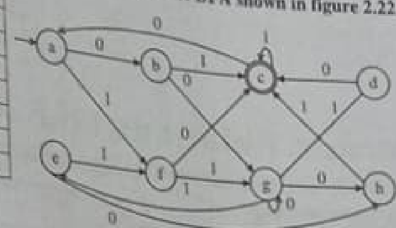


Figure 2.23: DFA

Ans: Any two final states are 0-equivalent, and any two non-final states are also 0-equivalent.

$\Pi_0(1, 2) = \{\{c\}, \{a, b, c, d, e, f, g, h\}\}$

	a	b	d	e	f	g	h
0	2	2	1	2	1	2	2
1	2	1	2	2	2	2	1

From the above table, we find a, e, g are 1-equivalent, b, h are 1 equivalent and d, f are 1-equivalent. Hence,
$\Pi_1(1, 3, 4, 5) = \{\{c\}, \{a, e, g\}, \{b, h\}, \{d, f\}\}$

Using the new classes, we find whether they are 2-equivalent.

	a	b	d	e	f	g	h
0	4	3	1	4	1	3	3
1	5	1	3	5	3	3	1

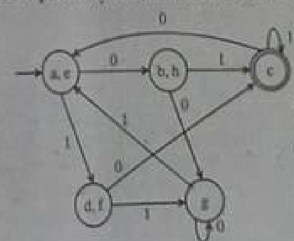
$\Pi_2(1, 6, 7, 8, 9) = \{\{c\}, \{a, e\}, \{b, h\}, \{d, f\}, \{g\}\}$

	a	b	d	e	f	g	h
0	7	9	1	7	1	9	9
1	8	1	9	8	9	6	1

$\Pi_3(1, 6, 7, 8, 9) = \{\{c\}, \{a, e\}, \{b, h\}, \{d, f\}, \{g\}\}$

	a	b	d	e	f	g	h
0	7	9	1	7	1	9	9
1	8	1	9	8	9	6	1

From the above two relations, Π_2 and Π_3 are same. Hence, the final set of states are the sets 1, 6, 7, 8, 9, where {a, e}, {b, h}, {d, f}, {g}, {c} are all 3-equivalent. The minimised DFA is depicted in **figure 2.24**.



	0	1
→{a, e}	{b, h}	{d, f}
{b, h}	{g}	{c}
*{c}	{a, e}	{c}
{d, f}	{c}	{g}
{g}	{g}	{a, e}

Figure 2.24: Minimum State of DFA