Finite Automata (FA)

(i) Deterministic Finite Automata (DFA)

A DFA, \( M = \langle Q, q_0, \Sigma, F, \delta \rangle \)

Where,
- \( Q \) = set of states (finite)
- \( q_0 \in Q \) = the start/initial state
- \( \Sigma \) = input alphabet (finite)
  (use only those symbols which create a particular string)
- \( F \subseteq Q \) = the set of final state
  (Final state can be 0 (no final state) or more than 1 or it can have all states as final states.

\( \delta \) = transition function (responsible for making the transition/movement from one state to other state
If machine halt at any final state that means it is accepting that string and if it halts at any other state it simply rejects the string.

Note: Behaviour of the machine is controlled by ‘\( \delta \)’ responsible for switching of state from one to other state.
\( \delta : Q \times \Sigma \rightarrow Q \) where, \( Q \) = input state, \( \Sigma \) = input alphabet

e.g. \( \delta (q_i, a) = q_j \) where, \( q_i \) = state, \( a \) = reading and \( q_j \) = move to other state

Note: Deterministic (exactly one choice)
(If you are at a state of read the input then after that you have to do the movement)

Total: On every state we need to define the transition for every symbol of input alphabet \( \varepsilon \) i.e. we need to explain explicitly that where you want go after reading input.

Notations:
(i) State : \( q \)
(ii) Initial state : \( q_0 \)
(iii) Final state : \( q_s \)

whenever, initial state becomes final state, null string \( \lambda \) is accepted.

Note: (i) \( \Sigma \{a\}^* \rightarrow \sigma_a^a \) Therefore, total = \( a^* \{\lambda, a, a^2, a^3, \ldots \} \)

(ii) \( q \) \( a \) \( q \) Therefore, \( a^+ = \{a, a^2, a^3, \ldots \} \)

(iii) \( a, b \) \( \Sigma = \{a, b\}^* \rightarrow \{\lambda, a, b, a^2, b^2, ab, ba, \ldots \} \) (universal language)
(iv) $\Sigma = \{a, b\}^+ = \{a, b, a^2, b^2, ab, ba, \ldots\}$

(not final, so can’t accept (a))

(v) $L = \{a^n \mid n \text{ is even}\}$

(vi) All the language are same

$\Sigma = \{a, b\}^+ = \{a, b, a^2, b^2, ab, ba, \ldots\}$

$L = \{a^n \mid n \text{ is even}\}$

$L = \{a^n \mid n \text{ mod } 2 = 0\}$

(vii) $L = \{a^n \mid n \text{ is odd}\}$

Although a given FA corresponds to only one language, a given language can have many FAs that accept it. Note that you must always be careful about the empty string: should the FA accept or not. In the preceding example, the empty string is accepted because the start state is also an accept state.

Another useful technique is remembering specific symbols. In the next example you must forever remember the first symbol; so the FA splits into two pieces after the first symbol.
1. Construct the DFA for the following
   (i) \( L = \{a^mb^n | m \geq 0, n \geq 0\} = a^*b^* \)
   (ii) \( L = \{a^n | n \geq 0 \text{ and } n \neq 4\} \)
   (iii) \( L = \{a^n | n \mod 5 \geq 2\} \)
   (iv) \( L = \{w \in \{a\}^* | w \mod 7 = 1\} \)

Soln.

(i) \( q_0 \xrightarrow{a} \{\lambda, a, a^2, a^3, \ldots\} \)

Therefore, \( q_0 \xrightarrow{b} \{\lambda, b, b^2, b^3, \ldots\} \Rightarrow \{ab, a^2b, a^2b^2, ab^2, \ldots\} \)

Therefore, \( q_0 \xrightarrow{a} q_1 \xrightarrow{b} q \)

(Total concept follow, we need to define each and every state explicitly in DFA).

(ii) \( L = \{a^n | n \geq 0 \text{ and } n \neq 4\} \)

\( \{0, a, a^2, a^3, a^5, a^6, \ldots\} \)

\( \xrightarrow{a} \Sigma = \{a\} \)

\( q_0 \)

\( q_0 \xrightarrow{a} q \xrightarrow{a} q \)

\( q_1 \xrightarrow{a} q_0 \xrightarrow{a} q \)

Therefore, 5 final states, 1 non-final state

(iii) \( L = \{a^n | n \mod 5 \geq 2\} \)

\( \{0, 1, 2, 3, 4\} \)

Therefore, total 5 states.

(iv) \( L = \{w \in \{a\}^* | w \mod 7 = 1\} \)

\( w = a^m \quad \Sigma = \{a\} \quad \therefore n \mod 7 = 1 \)

\( \therefore |w| = n \quad \therefore \{0, 1, 2, 3, 4, 5, 6\} \Rightarrow \text{Total 7 states} \)
2. \( L_2 = \{ a^+ b^+ c^+ \} \)
\( \{ a^m b^n c^p \mid m \geq 1, n \geq 1, p \geq 1 \} \)

3. Design a DFA for the following language:
(Set of all binary strings which starts with 1010)
\( \{ 1010w \mid w \in \{0,1\}^* \} \)

4. Design a DFA that ends with 1010

5. Design a DFA for the following over \( \Sigma = \{a, b\} \)
   (i) The set of all strings containing exactly 3 a’s.
(ii) Atleast 3 a’s:

(iii) Atmost 3 a’s:

Transition table/tabular: It is a matrix that lists the new state given the current state and the symbol read.

Example: Transition table for the FA that accepts all binary strings that begin and end with the same symbol.

<table>
<thead>
<tr>
<th>States</th>
<th>Input 0</th>
<th>Input 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>C</td>
</tr>
</tbody>
</table>

Example: The smallest DFA accepts the language \( L = \{ x \in \{a,b\}^* | \text{length of } x \text{ is multiple of } 3 \} \). [GATE-2002]

(a) 2  (b) 3  (c) 4  (d) 5

Solution: (b)

Example: Consider the machine M [GATE-2005]

The language recognized by above machine is:

(a) \( \{ w \in \{a,b\}^* | \text{every a in } w \text{ is followed by exactly two b's} \} \)

(b) \( \{ w \in \{a,b\}^* | \text{every a in } w \text{ is followed by at least two b's} \} \)

(c) \( \{ w \in \{a,b\}^* | w \text{ contains the substring 'abb'} \} \)
(d) \( \left\{ w \in \{a, b\}^* \mid w \text{ does not contain 'aa' as a substring} \right\} \)

**Soln.** (b)

**Combining the machines:**

\[ M_1 \quad \quad \quad M_2 \]

\[ L_1 \quad \quad \quad L_2 \]

\( \overline{L}_1 \) **(complement of a FSM)**

I = the machine of language \( L \) is given

O = the machine of language \( \overline{L} \)

**Steps:**

(i) Make the final states to non-final states and

(ii) Non-final states to final states (Final \( \leftarrow \) non-final)

**Problem:** Design a DFA which accepts the set of all binary string which does not starts with 1101.

**Soln.**

**Problem:** Design a DFA for \( L = \left\{ w \in \{a, b\}^* \mid N_a(w) \text{ is even AND } N_b(w) \text{ is even} \right\} \)

**Soln.**

\( L_1 \cap L_2 \Rightarrow m \times n \)

\( A \xrightarrow{a} B \); \( C \xrightarrow{a} C \). Therefore, BC

Here, first we will check that on A state, as we read ‘a’ we move to B on C state, as we read ‘a’ we move to C.

Therefore, AC state complex move to BC state.
Finite Automata

\[ A \xrightarrow{b} A ; \ C \xrightarrow{b} D \]
\[ B \xrightarrow{b} B \quad B \xrightarrow{a} A \]
\[ D \xrightarrow{b} C \quad C \xrightarrow{a} C \]
\[ B \xrightarrow{b} B \quad B \xrightarrow{a} A \]
\[ D \xrightarrow{b} C \quad D \xrightarrow{a} D \]
\[ A \xrightarrow{b} A \quad A \xrightarrow{a} B \]
\[ D \xrightarrow{b} C \quad D \xrightarrow{a} B \]

If \( L_1 \) machine have \( m \) state and \( L_2 \) machine have \( n \) state, then the combined machine \((L_1 \cap L_2)\) have \( m \times n \) state.

- **If the OR operation is applied then**

\[
L = \left\{ w \in (a,b)^* \mid N_a(w) \text{ is even OR } N_b(w) \text{ is even} \right\}
\]

Therefore, the number of final states will change.

As here, A and C are final state, so where the A and C come, it will become final state

- \( L_1 - L_2 = \{ w \mid w \in L_1 \text{ and } w \notin L_2 \} \)

So, \( L_1 \) is final and \( L_2 \) is non final > \( L_1 \) and \( L_2 \)

If \( L_2 - L_1 \) : \( BC \) will be final.

- \( L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1) \)

\[ \therefore \quad L_1 \oplus L_2 = AD \text{ and } BC \text{ both will be final states.} \]

**Non-Deterministic Finite Automata (NFA)**

A machine, \( M = (Q, q_0, \Sigma, F, \delta) \)

Where,

- \( Q \) = the set of states (finite)
- \( q_0 \in Q \) = initial/starting state
- \( \Sigma \) = input alphabet (finite)
Finite Automata

\( F \subseteq Q \) = final state is the subset of Q
\( \delta \) = transition function \( \rightarrow \) responsible for moving the movement from one state to another state.
\( DFA = \{ \delta = Q \times \Sigma \rightarrow Q(\text{total}) \} \) in this null transition is not allowed.
\( NFA = [\delta = Q \times \Sigma] \rightarrow 2^Q \rightarrow (\text{power set of } Q) \)

- Power of set Q, \( 2^Q = P(Q) = \{\phi, \{q_0\}, \{q_1\}, \{q_0, q_1\}\} \)
- In NFA we have zero (0) choice, one choice, or more than one choice depending upon the states, so it is called non-deterministic.

**Transition in NFA:**

![Diagram of NFA transitions](image)

**Note:** If in DFA, all the states are final states, then language is considered as universal language. \( \left( \Sigma^* = \{a, b\}^* \right) \). But, in NFA, if all the states are final states then it is not necessary that it is a universal language (like \( a^* b^* \)).

- **Design a NFA for \( a^* b^* c^* \):**

  1. \( q_1 \) \( \xrightarrow{a} \) \( q_2 \) \( \lambda \) \( q_3 \) \( \lambda \) \( q_4 \) \( \lambda \) \( q_5 \) \( \lambda \) \( q_6 \) \( \lambda \) \( q_7 \) \( \lambda \) \( q_8 \) \( \lambda \) \( q_9 \)

  OR

  2. \( q_1 \) \( \xrightarrow{a} \) \( q_2 \) \( \xrightarrow{b} \) \( q_3 \) \( \xrightarrow{c} \) \( q_4 \) \( \lambda \) \( q_5 \) \( \lambda \) \( q_6 \) \( \lambda \) \( q_7 \) \( \lambda \) \( q_8 \) \( \lambda \) \( q_9 \)

In NFA, trap state is not considered because if transition is not defined for a particular I/P symbol or a state, then given string is rejected automatically through the machine i.e. check for abaab.

**e.g.**

- \( q_1 \) \( \xrightarrow{a} \) \( q_2 \) \( \xrightarrow{b} \) \( q_3 \) \( \xrightarrow{c} \) \( q_4 \) \( \lambda \) \( q_5 \) \( \lambda \) \( q_6 \) \( \lambda \) \( q_7 \) \( \lambda \) \( q_8 \) \( \lambda \) \( q_9 \)

The machine rejects abaab.

**Problem:** Design an NFA over \( \Sigma = \{0,1\} \) accepting the set of all binary strings ending with 110.

**Soln.**

![Diagram of NFA solution](image)

Here, we need not to think about other transitions.
e.g. 110110
10110

- Start with 110

- Containing 110 as substrings

- 4th symbol from the beginning is 1.

Minimum number of states by NFA is \( (n+1) \)

- 4th symbol from the ending is 1. (Minimum states)

Problem: Design a NFA over \( \Sigma = \{0,1\} \) accepting the set of all binary strings which start and end with different symbols.

Soln. 

Note: We can construct the same NFA through more than one machine, but we apply the shortest states NFA.

Problem: Define NFA that accepts all binary strings where the last symbol is 0 or which contain only 1’s?

Soln. Here is one:

Problem: Define NFA for \( a^* + (ab)^* \),
Conversion of NFA to DFA:
1. Each state is given by a set of states from the original.
2. Start state is labeled \( \{ q_0 \} \) where \( q_0 \) was original start state.
3. While (some state of DFA is missing a transition) do:
   - Compute the transition by combining the possibilities for each symbol in the set.
4. Make into accept state any set that contains at least one original accept state.

The following DFA is the result of applying the preceding algorithm to the NFA in figure below.

For example, consider the state \( \{ A, B, D \} \). On a 1, the NFA, if in state A, can go to states A or C, if in state B dies, and if in state D stays in state D. Thus, on a 1 the DFA goes from \( \{ A, B, D \} \) to \( \{ A, C, D \} \). Both of these are accept states because they contain D.

If there are \( \varepsilon \) -transitions, then you have to adjust the process slightly. Do the following:
(i) The start state becomes the old start state and every state reachable from there by \( \varepsilon \) -transitions.
(ii) When one calculates the states reachable from a state, one includes all states reachable by \( \varepsilon \) -transitions after the destination state.

This is the result of applying the preceding algorithm to the NFA given in figure below.

The start state consists of A and the two states you can reach by \( \varepsilon \) – transitions, the state is thus \( \{ A, B, C \} \)
Finite Automata with Output capability:
There are two machines that don’t have final status. These both machines have the output capacity.

Mealy Machine: A mealy machine, \( M \) is a 6 tuple system \( M = (Q, q_0, \Sigma, \Delta, \delta, \delta') \)

Where,
- \( Q \) = the set of states (finite)
- \( q_0 \in Q \) = initial/start state
- \( \Sigma \) = input alphabet \( \{a, b\} \)
- \( \Delta \) = output alphabet \( \{0, 1\} = \Delta \)
- \( \delta \) = transition function responsible for making the movement/transition from one state to another state.
- \( \delta' \) = output function responsible for producing an output corresponding to every transition.

Note: In mealy machine for each transition the output will be generated.

\( \delta = Q \times \Sigma \rightarrow Q \) (total) \( \delta' = Q \times \Sigma \rightarrow \Delta \)

Problem: Design a mealy machine which finds the complement of a binary string.

Soln. If input is 0 the output/input is 1 and vice versa.

Problem: Design a mealy machine which increments the given binary number by 1

Soln. Assume, reading of symbols should be from right to left.
**Problem:** Design a mealy machine to add two binary numbers

**Soln.**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S (sum)</th>
<th>C (carry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Moore Machine:** A moore machine,

\[ M = (Q, q_0, \Sigma, \Delta, \delta, \delta') \]

- Output capability
- Different from mealy machine in only way that they produce the output
- All components are similar like mealy machine
- Input = string (\( \Sigma \))
- Output = string (\( \Delta \))
  
  Contain those set of terminals from which the output string can be made
  
  So, it may be possible that (input string is same as output string)
  
  But in case of mealy machine, there is an output corresponding to every transition (\( \delta = Q \times \Sigma \rightarrow \Delta \))
  
  But in case of Moore machine, there is an output corresponding to every state (\( \delta': Q \rightarrow \Delta \))

**Problem:** Design a Moore machine that count the occurrence of 101 in a binary string. (Number of times 101 is available in the string).

**Soln.**

![Diagram of Moore Machine](image)

So, now we will count the number of 1’s

Therefore, 2 (2 times 101 come)

**Mealy Machine:**

- Input string = \( w = |w| = n \) then output string = \( n \)

**Moore Machine:**

- Input string = \( w = |w| = n \) then output string = \( n + 1 \)

**Problem:** Design a Moore machine which finds the remainder of decimal representation of a binary number when divided by 3.

**Soln.**

\[ \text{Remainder} \rightarrow \langle 0, 1, 2 \rangle \]

\[ \Sigma = \{ 0, 1, \ldots \} \ (\text{binary}) \]
Note: Mealy machine and Moore machines are equally powerful.

**Moore to Mealy machine (Conversion):**

\[
\begin{array}{c}
q_0 \quad 0 \\
1 \quad 2 \\
\end{array}
\Rightarrow
\begin{array}{c}
q_0 \quad 0/a \\
1/a \\
\end{array}
\]

**Note:** We need to attach the particular state output to every transition which is coming to it.

**Example:**

Mealy to Moore (Conversion):

- When there are two different output, we need to split the state into two, as there are two output.
- When all the incoming transition output are same then we need not to split the state, we just simply put that transition output into the state output.
Finite Automata

Example:

\[ q_i \xrightarrow{a/0} (Mealy) q_{i/0} \]

(need to connect both the states)

Example:

Decision Properties:
1. Finiteness / Infiniteness
   - The steps to check the condition are as follows:
     1. Select the state which can’t be reached by the initial state and delete them.
     2. Select those states from which we can’t reach to final state and delete them.
     3. If the resulting machine is free from cycles and self loops then the machine accepts finite language else the machine accepts infinite language.

Example:

contains cycle so accepts infinite language.

Example:

No cycle or self loop so accept finite language.
2. Emptyness/non-emptyness:
- The steps to check the condition is given below
  (a) Select the state which can’t be reached from the initial state and delete them.
  (b) If the resulting machine is free from final state then it will accept empty language otherwise non-empty language

Example:

The machine doesn’t contain final state so accepts empty language.

Example:

The machine contains final state so accepts non-empty language.

3. Equalness:
Two finite state machines $M_1$ and $M_2$ are said to be equal if both machines accepts same set of strings i.e.

$$M_1 = M_2 \text{ if } L(M_1) = L(M_2)$$

- Equal machines need not to contain same set of states
- Two machines $M_1$ and $M_2$ are said to be isomorphic to each other if both of them
  - accepts same language
  - contains same number of states
  - have the same properties.

Steps to check the equalness of two machine is given below:
(a) Construct the transition table that contains pair of initial states $(P, Q)$ where, $P$ is from machine $M_1$ and $Q$ is from machine $M_2$.
(b) Construct the transition table
(c) In the construction process, we get any pair of the form (final, non-final) or (non-final, final), stop constructing the table and declares that the machines are not equal else after completion of the table if no such condition generated the machines are equal.

Example:

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(A, P)$</td>
<td>$(A, P)$</td>
<td>$(C, Q)$</td>
</tr>
<tr>
<td>$(C, Q)$</td>
<td>$(D, R)$</td>
<td>$(A, P)$</td>
</tr>
<tr>
<td>$(D, R)$</td>
<td>$(B, R)$</td>
<td>$(D, R)$</td>
</tr>
<tr>
<td>$(B, R)$</td>
<td>$(D, Q)$</td>
<td>$(A, R)$</td>
</tr>
</tbody>
</table>

State $(A, R)$ is (final, non-final) so both the machines are not equal.
Minimization of Finite Automata:
The process of deletion and elimination of states whose presence or absence will not affect the language of the machine is called minimization/optimization of FA.
We start with two examples. Consider the automaton on the left:

We can see that it is not possible to ever visit state 2. States like this are called unreachable. We can simply remove them from the automaton without changing its behaviour. (This will be, indeed, the first step in our minimization algorithm.) In our case, after removing state 2, we get the automaton on the right.

As it turns out, however, removing unreachable states is not sufficient. The next example is a bit more subtle (see the automaton on the left).

We now formalize the above idea. Let $A$ be a finite automaton. We will say that $w \in \Sigma^*$ distinguishes between two states $p, q \in Q$ if either $\delta(p, w) \in F$ & $\delta(q, w) \notin F$, or $\delta(p, w) \notin F$ & $\delta(q, w) \in F$. Two states $p, q \in Q$ are called distinguishable iff there is a word that distinguishes between them. States that are indistinguishable will also be sometimes called equivalent.

Lemma-1: Suppose that $B$ is a DFA without unreachable states. Then $B$ is minimum if and only if all pairs of states are distinguishable.
Lemma-2: State indistinguishability is an equivalence relation.
Lemma-3: Let $\delta(p, a) = p'$ and $\delta(q, a) = q'$. Then, if $p', q'$ distinguishable then so are $p, q$.

To minimize $A$, after removing unreachable states, we will find the equivalence classes of the indistinguishability relation and join all states in each class into one state of the new automaton $A$. To determine transitions, if $X$ is an equivalence class, we pick any $q \in X$ and define $\delta(X, a) = Y$ where $Y$ is the equivalence class that contains $\delta(q, a)$.

To determine which states are equivalent, we will use the following method. Instead of trying to figure out which states are indistinguishable, we will try to figure out which states are distinguishable. Clearly, if $p \in F$ and $q \notin F$ then $p, q$ are distinguishable, since they can be distinguished by $\lambda$. Then we iteratively examine all pairs of states. For each pair $p, q$ we do this. If we find a symbol $a$, such that $\delta(p, q)$ and $\delta(q, a)$ are distinguishable, then we mark $p, q$ as distinguishable as well. We iterate this until no more pairs are marked. When we are done, the non-marked pairs are equivalent. The complete algorithm is given below.
Minimizing the Number of States of a DFA:

The following three statements are equivalent:

1. The set \( L \subseteq \Sigma^* \) is accepted by some finite automaton.
2. \( L \) is the union of some of the equivalence classes of a right invariant equivalence relation of finite index.
3. Let equivalence relation \( R_L \) be defined by: \( xR_L y \) if and only if for all \( z \in \Sigma^* \), \( xz \) is in \( L \) exactly when \( yz \) is in \( L \). Then \( R_L \) is of finite index.

Myhill Nerode Theorem:

Let \( A \) be any language over \( \Sigma^* \) we say that string \( x \) and \( y \) in \( \Sigma^* \) are indistinguishable by \( A \) if for every string \( z \in \Sigma^* \) either both \( xz \) and \( yz \) are in \( A \) or both \( xz \) and \( yz \) are not in \( A \).

If the number of suffix of a languages is finite (\( x \)) then

- Language \( L \) is regular
- The minimum DFA of \( L \) needs at least \( k \) states.

**Example:** Let \( L \) be the language \( 0^*10^* \). \( L \) is accepted by the DFA. Consider the relation \( R_M \) defined by \( M \).

As all states are reachable from the start state, \( R_M \) has six equivalence classes, which are

- \( C_a = (00)^* \)
- \( C_d = (00)^*01 \)
- \( C_b = (00)^*0 \)
- \( C_e = 0^*100^* \)
- \( C_c = (00)^*1 \)
- \( C_f = 0^*10^*1(0+1)^* \)

\( L \) is the union of three of these classes, \( C_b, C_e \) and \( C_f \).

The relation \( R_L \) for \( L \) has \( xR_L y \) if and only if either

(i) \( x \) and \( y \) each have no 1’s.

(ii) \( x \) and \( y \) each have one 1, or

(iii) \( x \) and \( y \) each have more than one 1.

For example, if \( x = 010 \) and \( y = 1000 \), then \( xz \) is in \( L \) if and only if \( z \) is \( n 0^* \). But \( yz \) is in \( L \) under exactly the same conditions. As another example, if \( x = 01 \) and \( y = 00 \), then we might choose \( z = 0 \) to show that \( xR_L y \) is false. That is, \( xz = 010 \) is in \( L \), but \( yz = 000 \) is not.

We may denote the three equivalence classes of \( R_L \) by \( C_1 = 0^* \), \( C_2 = 0^*10^* \), and \( C_3 = 0^*10^*1(0+1)^* \).

\( L \) is the language consisting of only one of these classes, \( C_2 \). The relationship of \( C_a \), \( C_b \), \( C_c \), \( C_d \), \( C_e \) and \( C_f \) is illustrated, in figure. For example \( C_a \cup C_b = (00)^* + (00)^*0 = 0^* = C_1 \).

From \( RL \) we may construct a DFA as follows. Pick representatives for \( C_1 \), \( C_2 \) and \( C_3 \), say \( e \), \( 1 \) and \( 11 \). Then let \( M' \) be the DFA shown in figure. For example, \( (\delta'(1), 0 = (1)) \), since if \( w \) is any string in \( (1) \) (note \( 1 \) is \( C_1 \)), say \( 0^i10^j \), then \( w0 \) is \( 0^i10^{j+1} \), which is also in \( C_1 = 0^*10^* \).
Minimization Algorithm:

**Step-I:** Remove unreachable states.

**Step-II:** Mark the distinguishable pairs of states. To achieve this task, we first mark all pairs \( p, q \) where \( p \in F \) and \( q \notin F \) as distinguishable. Then, we proceed as follows.

repeat
  for all non-marked pairs \( p, q \) do
    for each letter \( a \) do
      if the pair \( \delta(p, a), \delta(q, a) \) is marked.
        then mark \( p, q \)
    until no new pairs are marked.

**Step-III:** Construct the reduced automaton \( \hat{A} \).

We first determine the equivalence classes of the indistinguishability relation. For each state \( q \), the equivalence class of \( q \) consists of all states \( p \) for which the pair \( p, q \) is not marked in **Step-II**.

The states of \( \hat{A} \) are the equivalence classes. The initial state \( \hat{q}_0 \) is this equivalence class that contains \( q_0 \). The final states \( \hat{F} \) are these equivalence classes that consist of final states of \( A \). The transition function \( \hat{\delta} \) is defined as follows. To determine \( \hat{\delta}(X, a) \), for some equivalence class \( X \), pick any \( q \in X \) and set \( \hat{\delta}(X, a) = Y \), \( Y \) is the equivalence class that contains \( \delta(q, a) \).

**Example of minimization of DFA:**

![Diagram of DFA](image)

**Step-1:** Unreachable states in above diagram:
F and G are unreachable states, So, we will remove them.
So, now the DFA will look like

![Diagram of minimized DFA](image)

**Step-2:** Now, we will proceed to “TABLE FILLING PROCESS”

**Note:** Vertically we will read B, C, D, E (not A (first element))
Horizontally we will read A, B, C, D (not ‘E’ element)

So, in this table we will fill those states with ‘x’ (cross-marks) which are distinguishable
(Here, E and A are distinguishable, so, we have to put ‘x’ cross-mark in there common box). Similarly, in others.

Now, the transition function is considered to find out another way of distinguishable state and it helps to fill the remaining particular state in the table.

So,

\[ \delta(A, a) = B \]
\[ \delta(C, a) = D \]

Distinguishable state

\[ \Rightarrow \]

\[ \delta(A, b) = C \]
\[ \delta(C, b) = D \]

Distinguishable state

\[ \Rightarrow \]

\[ \delta(A, a) = B \]
\[ \delta(B, a) = D \]

Distinguishable state

\[ \Rightarrow \]

\[ \delta(A, a) = B \]
\[ \delta(B, a) = D \]

Distinguishable state

\[ \Rightarrow \]

\[ \delta(B, a) = D \]
\[ \delta(C, a) = D \]

Distinguishable state

\[ \Rightarrow \]

\[ \delta(D, a) = E \]
\[ \delta(E, a) = E \]

Distinguishable state

\[ \Rightarrow \]

\[ \delta(D, b) = E \]
\[ \delta(E, b) = E \]

Distinguishable state

As in the above table, we will merge ‘B’ and ‘C’ and ‘D’ and ‘E’ because they both are the non-distinguishable state.

(Minimised DFA)

Language accepting this: \( L = \{(a + b)(a + b)/(a + b)^*\} \)

\( e.g. a^2b, a^2b^2, b^2a \)

\( \epsilon - \text{NFA (NFA with } \epsilon \text{ moves)}: \)

The NFA which has a transition even for empty string \( \epsilon \) is called \( \epsilon - \text{NFA}. \)

It is a 5 tuple system \( (Q, q_0, \epsilon, \delta, F) \) everything is similar to NFA except the transition function which is given as

\[ \delta : Q \times \epsilon \cup \{\epsilon\} \rightarrow 2^Q \]
Capability of \( \epsilon \)-NFA is same as NFA and DFA. \( \epsilon \)-NFA provide programming convinience.

e.g. 

Convert NFA with \( \lambda \) transition into its equivalent DFA:

\[
\begin{array}{c}
\text{(NFA)} \\
\begin{array}{c}
\text{q}_0 \quad \text{q}_1 \quad \text{q}_2 \\
\text{a} \quad \lambda \quad \text{b} \\
\end{array}
\end{array}
\]

\( \epsilon \)-closure of \( q \) :

The set of all states reachable from 'q' by taking \( \lambda \) transitions including 'q'

\[
\begin{align*}
\epsilon \text{-closure (} q_0 \text{)} &= \{ q_0, q_1, q_2 \} \\
\epsilon \text{-closure (} q_1 \text{)} &= \{ q_1, q_2 \} \\
\epsilon \text{-closure (} q_2 \text{)} &= \{ q_2 \}
\end{align*}
\]

Firstly: We will consider 'q_0' as the initial state of an equivalent DFA i.e. \( \{ q_0, q_1, q_2 \} \rightarrow \text{initial state} \).

Now, we need to consider all the transitions a, b, c with respect to initial state of \( \epsilon \)-closure (\( q_0 \))

If we read 'a' at \( q_0 \):

\[
\begin{align*}
\delta (q_0, q_1, q_2, a) &= \epsilon \text{-closure (} q_0, a \text{)} \cup \epsilon \text{-closure (} q_1, a \text{)} \cup \epsilon \text{-closure (} q_2, a \text{)} \\
&= \epsilon \text{-closure (} q_0 \text{)} \cup \epsilon \text{-closure (} \phi \text{)} \cup \epsilon \text{-closure (} \phi \text{)} = \{ q_0, q_1, q_2 \}
\end{align*}
\]

If we read 'b' at \( q_0 \) then

\[
\begin{align*}
\delta (q_0, q_1, q_2, b) &= \epsilon \text{-closure (} q_0, b \text{)} \cup \epsilon \text{-closure (} q_1, b \text{)} \cup \epsilon \text{-closure (} q_2, b \text{)} \\
&= \epsilon \text{-closure (} q_1 \text{)} \cup \epsilon \text{-closure (} q_2 \text{)} \cup \epsilon \text{-closure (} q_2 \text{)} = \{ q_1, q_2 \}
\end{align*}
\]

If we read 'c' then at \( \epsilon \)-closure (\( q_0 \))

\[
\begin{align*}
\delta (q_0, q_1, q_2, c) &= \phi \cup \phi \cup \epsilon \text{-closure (} q_2 \text{)} = \{ q_2 \}
\end{align*}
\]

Now, we read 'a' for \( \{ q_1, q_2 \} \rightarrow \epsilon \text{-closure (} q_2, a \text{)} = \phi \cup \phi = \phi \) (trap state)

Read 'b' for \( \{ q_1, q_2 \} \)

\[
\begin{align*}
\delta (q_1, q_2, b) &= \epsilon \text{-closure (} q \text{)} \cup \phi = \{ q_1, q_2 \}
\end{align*}
\]

read 'c' 

\[
\begin{align*}
\delta (q_1, q_2, c) &= \phi \cup \epsilon \text{-closure (} q_2 \text{)} = \{ q_2 \}
\end{align*}
\]

Read 'a' at \( \epsilon \)-closure (\( q_2 \))

\[
\begin{align*}
\delta (q_2, a) &= \phi \) (trap state)
\end{align*}
\]

read 'b'

\[
\begin{align*}
\delta (q_2, b) &= \phi \) (trap state)
\end{align*}
\]

read 'c'
\( \delta(q_2, b) = \varepsilon - \text{closure}(q_2) = \{ q_2 \} \)
**SOLVED PROBLEMS**

1. State True or False with one line explanation. A FSM (Finite State Machine) can be designed to add two integers of any arbitrary length (arbitrary number of digits).

   **[GATE-1994]**

   Soln. FALSE, A FSM (Finite State Machine) can’t be designed to add two integers of any arbitrary length because FSM have finite memory and can’t store integer of any arbitrary length.

2. A finite state machine with the follows state table has a single input X and a single output Z.

<table>
<thead>
<tr>
<th>present state</th>
<th>next state, z</th>
<th>next state, z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x = 1</td>
<td>x = 0</td>
</tr>
<tr>
<td>A</td>
<td>D, 0</td>
<td>B, 0</td>
</tr>
<tr>
<td>B</td>
<td>B, 1</td>
<td>C, 1</td>
</tr>
<tr>
<td>C</td>
<td>B, 0</td>
<td>D, 1</td>
</tr>
<tr>
<td>D</td>
<td>B, 1</td>
<td>C, 0</td>
</tr>
</tbody>
</table>

   It the initial state is unknown, then the shortest input sequence to reach the final state C is

   (a) 01  (b) 10  (c) 101  (d) 110

   **[GATE-1995]**

   Ans. (b)

   Soln. The state diagram which represents the given state table is

   ![State Diagram]

   From state A to reach C, the shortest sequence is

   \[ A \rightarrow^0 B \rightarrow^0 C \]
   \[ A \rightarrow^1 D \rightarrow^0 C \]

   From B to reach C

   \[ B \rightarrow^0 C \]
   \[ B \rightarrow^1 B \rightarrow^0 C \]

   From C to reach C

   \[ C \rightarrow^0 D \rightarrow^0 C \]
   \[ C \rightarrow^1 B \rightarrow^0 C \]

   From D to reach C

   \[ D \rightarrow^0 C \]
   \[ D \rightarrow^1 B \rightarrow^0 C \]

   So 10 is the shortest input sequence to reach the final state C, whatever be the initial state.
3. Which of the following set can be recognized by a Deterministic Finite state Automaton?
(a) The numbers 1, 2, 4, 8, ..., \(2^n\), ..., written in binary
(b) The numbers 1, 2, 4, ..., \(2^n\), ..., written in unary
(c) The set of binary string in which the number of zeros is the same as the number of ones
(d) The set \{1, 101, 11011, 1110111, .......\}

**Ans. (a)**

**Soln.** The numbers 1, 2, 4, 8, ..., \(2^n\), ... are represented by 1, 10, 100, 1000, ...

The pattern in the regular expression is 1 followed by 0’s.

The regular expression for above is \(10^*\)

The DFA for above language is

![DFA Diagram](image_url)

So the numbers 1, 2, 4, 8, ..., \(2^n\), ..., written in binary can be recognized by a deterministic finite state automaton.

4. Let \(L\) be the set of all binary strings whose last two symbols are the same. The number of states in the minimum state deterministic finite state automaton accepting \(L\) is
(a) 2  
(b) 5  
(c) 8  
(d) 3

**Ans. (b)**

**Soln.** The number of states in the minimum state deterministic finite state automaton accepting all binary strings whose last two symbols are the same is 5.

![DFA Diagram](image_url)

5. Consider the regular expression \((0 + 1)(0 + 1)\ldots n\) times. The minimum state finite automaton that recognizes the language represented by this regular expression contains
(a) \(n\) states  
(b) \(n + 1\) states  
(c) \(n + 2\) states  
(d) None of these

**Ans. (b)**

**Soln.** The minimum state finite automaton that recognizes the language represent by regular expression \((0 + 1)(0 + 1)\ldots n\) times is \(n + 1\).

The language contains strings with exactly length \(n\).

\(n + 1\) states are required to count length upto \(n\). No trap state is required since we are making minimal FA, not minimal DFA. For example, for \(n = 2\) the design is shown below.

![DFA Diagram](image_url)
6. Given an arbitrary non-deterministic finite automaton (NFA) with N states, the maximum number of states in an equivalent minimized DFA is at least \[ \text{[GATE-2001]} \]
(a) \( N^2 \)  
(b) \( 2^N \)  
(c) \( 2N \)  
(d) \( N! \)

Ans. (b)

Soln. For an arbitrary NFA with N states, the maximum number of states in an equivalent minimized DFA is \( 2^N \).

7. Consider a DFA over \( \Sigma = \{a, b\} \) accepting all strings which have number of a’s divisible by 6 and number of b’s divisible by 8. What is the maximum number of states that the DFA will have? \[ \text{[GATE-2001]} \]
(a) 8  
(b) 14  
(c) 15  
(d) 48

Ans. (d)

Soln. A DFA over \( \Sigma = \{a, b\} \) accepting all strings which have number of a’s divisible by 6 and number of b’s divisible by 8 is a grid machine (product automata) having \( 6 \times 8 = 48 \) states.

8. The finite state machine described by the following state diagram with A as starting state, where an arc label is \( x/y \) and \( x \) stands for 1-bit input and \( y \) stands for 2-bit output

![State Diagram](image)

(a) Outputs the sum of the present and the previous bits of the input  
(b) Outputs 01 whenever the input sequence contains 11  
(c) Outputs 00 whenever the input sequence contains 10  
(d) None of the above \[ \text{[GATE-2002]} \]

Ans. (a)

Soln. The state diagram represents the FSM which outputs the sum of the present and previous bits of the input. State A represents last bit a 0 and B and C represents last bit is 1.

9. The smallest finite automaton which accepts the language \( L = \{x \mid \text{length of } x \text{ is divisible by 3} \} \) has
(a) 2 states  
(b) 3 states  
(c) 4 states  
(d) 5 states \[ \text{[GATE-2002]} \]

Ans. (b)

Soln. The minimal finite automaton with 3 states which accepts the language \( L = \{x \mid \text{length of } x \text{ is divisible by 3} \} \) is as follows:

![State Diagram](image)

10. Consider the following deterministic finite state automaton M.

![State Diagram](image)

Let \( S \) denote the set of seven bit binary strings in which the first, the fourth, and the last bits are 1. The number of strings in \( S \) that are accepted by M is
(a) 1  
(b) 5  
(c) 7  
(d) 8 \[ \text{[GATE-2003]} \]
Ans. (c)
Soln. The given bit pattern can be represented as:
\[
\begin{align*}
\text{1 — — 1 — — 1}
\end{align*}
\]
The four blanks can be filled in \(2^4 = 16\) ways. Therefore there are 16 such strings in this pattern. Not all of these are accepted by the machine. The strings and its acceptance is given below:

- accepted
  - 1 0 0 1 0 0 1
  - 1 0 0 1 0 1 1
  - 1 0 0 1 1 0 1
  - 1 0 0 1 1 1 1
  - 1 0 1 1 0 0 1
  - 1 1 0 1 0 0 1
  - 1 1 1 1 0 0 1

Only these seven strings given above are accepted. The other strings (9 of them) in this pattern are rejected, since they don’t reached the final state.

11. Consider the NFA M shown below:

Let the language accepted by M be L. Let \(L'\) be the language accepted by the NFA \(M'\), obtained by changing the accepting state of M to a non-accepting state and by changing the non-accepting state of M to accepting states. Which of the following statements is true?

(a) \(L = \{0, 1\}^* - L\)  (b) \(L = \{0, 1\}^*\)  (c) \(L \subseteq L\)  (d) \(L = L\)

Ans. (b)
Soln. The given machine M is

Now the complementary machine \(M\) is

\[\text{Diagram of machine M}\]
In the case of DFA, $L(M) = \overline{L(\overline{M})}$ but in the case of NFA this is not true. In fact $L(M)$ and $L(\overline{M})$ have no connection.

$\therefore$ To find $L_i = L(\overline{M})$ we have to look at $\overline{M}$ and directly find its languages.

$L_i = L(\overline{M}) = \epsilon + (0 + 1) (0 + 1)^* + \ldots = (0 + 1)^* + \ldots = (0 + 1)^*$.

12. Let $M = (K, \Sigma, \delta, s, F)$ be a finite state automaton, where $K = \{A, B\}$, $\Sigma = \{a, b\}$, $s = A$, $F = \{B\}$, 

$\delta(A, a) = A$, $\delta(A, b) = B$, $\delta(B, a) = B$ and $\delta(B, b) = A$ and $\delta(B, b) = A$

A grammar to generate the language accepted by M can be specified as $G = (V, \Sigma, R, S)$, where $V = K \cup \Sigma$, and $S = A$.

Which one of the following set of rules will make $L(G) = L(M)$?

(a) $\{A \rightarrow aB, A \rightarrow bA, B \rightarrow bA, B \rightarrow aA, B \rightarrow \epsilon\}$

(b) $\{A \rightarrow aA, A \rightarrow bB, B \rightarrow aB, B \rightarrow bA, B \rightarrow \epsilon\}$

(c) $\{A \rightarrow bB, A \rightarrow aB, B \rightarrow aA, B \rightarrow bA, B \rightarrow \epsilon\}$

(d) $\{A \rightarrow aA, A \rightarrow bA, B \rightarrow aB, B \rightarrow bA, A \rightarrow \epsilon\}$

Ans. (b)

Soln. $\delta(A, a) = A$, $A \rightarrow aA$

$\delta(A, b) = B$, $A \rightarrow bB$

$\delta(B, a) = B$, $B \rightarrow aB$

$\delta(B, b) = A$, $B \rightarrow bA$

B is final state, so $B \rightarrow \epsilon$.

13. The following finite state machine accepts all those binary strings in which the number of 1’s and 0’s are respectively.

(a) divisible by 3 and 2 (b) odd and even (c) even and odd (d) divisible by 2 and 3

Ans. (a)

Soln. The given finite state machine accepts any string $w \in (0, 1)^*$ in which the number of 1’s is multiple of 3 and the number of 0’s is multiple of 2.

14. Consider the non-deterministic finite automaton (NFA) shown in the figure.
State X is the starting state of the automaton. Let the language accepted by the NFA with Y as the only accepting state be L1. Similarly, let the language accepted by the NFA with Z as the only accepting state be L2. Which of the following statements about L1 and L2 is TRUE?

(a) \( L_1 = L_2 \)  
(b) \( L_1 \subseteq L_2 \)  
(c) \( L_2 \subseteq L_1 \)  
(d) None of these

Ans. (a)

Soln. Writing Y and Z in terms of incoming arrows (Arden’s method), we get:

\[ Y = X_0 + Y_0 + Z_1 \]
\[ Z = X_0 + Z_1 + Y_0 \]

Clearly, \( Y = Z \).

15. Consider the machine M

The language recognized by M is

(a) \( \{w \in \{a, b\}^* | \text{every a in w is followed by exactly two b's} \} \)
(b) \( \{w \in \{a, b\}^* | \text{every a in w is followed by at least two b's} \} \)
(c) \( \{w \in \{a, b\}^* \text{ w contains the substring 'abb'} \} \)
(d) \( \{w \in \{a, b\}^* \text{ w does not contain 'aa' as a substring} \} \)

Ans. (b)

Soln. (a) is false, since M is accepting “abbb”.
(b) is true
(c) is false, since “abba” contains “abb” as a substring, but is being rejected by the machine.
(d) is false, since \( \lambda \) does not contain “aa” as a substring, but \( \lambda \) is being rejected by M.

16. The following diagram represents a finite state machine which takes as input a binary number from the least significant bit

Which one of the following is TRUE?

(a) It computes 1’s complement of the input number  
(b) It computes 2’s complement of the input number  
(c) It increments the input number  
(d) It decrements the input number

Ans. (b)

Soln. The given machine, executes the algorithm for 2’s complement when input is given from LSB.
17. In the automaton below, s is the start state and t is the only final state.

Consider the strings $u = \text{abbaba}$, $v = \text{bab}$ and $w = \text{aabb}$. Which of the following statements is true?
(a) The automaton accepts $u$ and $v$ but not $w$
(b) The automaton accepts each of $u$, $v$, and $w$
(c) The automaton rejects each of $u$, $v$, and $w$
(d) The automaton accepts $u$ but rejects $v$ and $w$

[IT-2006]

Ans. (d)
Soln. $u = \text{abbaba}$: Accepted by automata.
$v = \text{bab}$: Not accepted by automata
$w = \text{aabb}$: Not accepted by automata.

18. A minimum state deterministic finite automaton accepting the language $L = \{ w \mid w \in \{0, 1\}^*, \text{number of 0's and 1's in } w \text{ are divisible by 3 and 5, respectively} \}$ has
(a) 15 states
(b) 11 states
(c) 10 states
(d) 9 states

[GATE-2007]

Ans. (a)
Soln. $L = \{ w \mid w \in \{0, 1\}^* \text{ number if 0's and 1's in } w \text{ are divisible by 3 and 5 respectively} \}$. The minimum state deterministic finite automaton accepting the language $L$ has $3 \times 5 = 15$ states.

Common Data for Q. 19 and Q. 20:
Consider the following Finite State Automaton

19. The language accepted by this automaton is given by the regular expression
(a) $b^*ab^*ab^*ab^*$
(b) $(a+b)^*$
(c) $b^*a(a+b)^*$
(d) $b^*ab^*ab^*$

[GATE-2007]
q₁ is unreachable from starting state and hence can be deleted to give the following diagram:

q₀ is the initial state. From q₀ to q₁ the regular expression is b*a. After that, every combination of a and b is accepted by q₁ or q₂. So q₁ and q₂ together can be collapsed into a permanent accept state and now the diagram becomes.

\[ r = b^*a(a + b)^* \]

20. The minimum state automaton equivalent to the above FSA has the following number of states
(a) 1  (b) 2  (c) 3  (d) 4  

Ans. (b)  
Soln. The minimum state FSA is given below for the regular expression b*a(a + b)*. So FSA contains minimum two states q₀ and q₁.

21. Consider the following DFA in which s₄ is the state and s₁, s₂, and s₃ are the final states.

What language does this DFA recognize?
(a) All strings of x and y  
(b) All strings of x and y which have either even number of x and even number of y and odd number of y  
(c) All strings of x and y which have equal number of x and y  
(d) All strings of x and y with either even number of x and odd number of y or odd number of x and even number of y
Finite Automata

Ans. (d)  
Soln. Given DFA can be redesigned as S0 as q00, S1 as q10, S2 as q11, S3 as q01. Each state is q_{ab} [a = n_x \mod 2, b = n_y \mod 2]. q00 as n_x \mod 2 = 0, n_y \mod 2 = 0 [number of x is even, number of y is even].

q01 is final state mean where number of x is even and number of y is odd. q10 is final state mean where number of x is odd and number of y is even.

22. Consider the following finite automata P and Q over the alphabet \{a, b, c\}. The start states are indicated by a double arrow and final states are indicated by a double circle. Let the languages recognized by them be denoted by L(P) and L(Q) respectively.

The automaton which recognizes the languages L(P) \cap L(Q) is:

(a)  
(b)  
(c)
23. Which of the following non-deterministic finite automata recognizes the language defined by the regular expression $R = (a + b)^* (aa + bb) (a + b)^*$?

**Common Data for Q. 23, Q.24 and Q. 25:**
Consider the regular expression: $R = (a + b)^* (aa + bb) (a + b)^*$

24. Which deterministic finite automaton accepts the language represented by the regular expression $R$?
Ans. (a)
Soln. In option (a) $S_3$ and $S_4$ together act as a permanent accept and can therefore be collapsed into a single permanent accept state as shown below:

This machine clearly accepts all strings containing the substrings ‘aa’ or ‘bb’, which is same as regular expression $R$.

25. Which one of the regular expressions given below defines the same language as defined by the regular expression $R$?

(a) $(a(ba)^* + b(ab)^*) (a + b)^*$
(b) $(a(ba)^* + b(ab)^*)^* (a + b)^*$
(c) $(a(ba)^* (a + bb) + b(ab)^* (b + aa)) (a + b)^*$
(d) $(a(ba)^* (a + bb) + b(ab)^* (b + aa)) (a + b)^*$

Ans. (c)
Soln. $R = (a + b)^*(aa + bb) (a + b)^*$ has following equivalent DFA