VIGNAN'S UNIVERSITY :: VADLAMUDI

Faculty	**:**	**Mr. S Deva Kumar**
Subject	**:**	**Formal Languages and Automata Theory**
Code	**:**	**CS224**
Year & Branch	**:**	**II CSE**

UNIT – I

PRILIMINARIES

Symbol:

A symbol is an abstract entity. There is no formal definition for a symbol.

Examples: letters, digits and special characters etc.

Alphabet:

A finite non-empty set of symbols is called 'alphabet'. It is denoted by $\sum$.

Example**:** $\sum = \{a,b,\ldots z\}$, $\sum = \{a,b\}$, $\sum = \{0,1\}$

String:

A sentence over an alphabet is a finite sequence of the symbols from an alphabet.

Example:

1.. In $\sum = \{a,b\}$ is the alphabet

ab,abb are valid strings, where as abc is not a string because 'c' doesn't belongs to $\sum$

2. In $\sum = \{0, 1\}$ is the alphabet

00,110,1011 are valid strings.

Note: The length of any string over the given alphabet is at least 1.

Assumption:

In theory of computation, there exists a string of length zero called as 'epsilon' i.e. '$\in$' .

$S.\in = S$

$\in .S = S$

$\in .S.\in = S$

Prefix: Any sequence of leading symbols over the given string is called 'prefix'.

Example: For the string 'cse', the possible prefixes are $\in$,c, cs, cse.

Suffix: Any sequence of trailing symbols over the given string is called 'suffix'.

Example: For the string 'cse', the possible suffixes are $\in$, e, se, cse

Substring: Any sequence of symbols over the given string is called 'substring'.

Example: For the string 'computer' the possible substrings are $\in$, com, put, te, mp,….

Power of an Alphabet:

Assume input alphabet $\sum = \{a,b\}$

$\sum 0 = \{\in\}$ i.e. the set of all strings possible over the alphabet of length Zero

Similarly,

$\sum 1 = \{a,b\}$

$\sum 2 = \{aa,ab,bb,ba\}$

$\sum 3 = \{$set of all strings possible over the alphabet of lenth 'n'$\}$

$\sum^* = \sum 0 \cup \sum 1 \cup \sum 2 \cup \sum 3 \ldots . \sum n$

$\sum^*$= set of all strings over $\sum$ including '$\in$'

Here '*' is the kleene closure operator

- $\sum^+ = \sum^* - \in$

  $= \sum^1 \cup \sum^2 \cup \ldots\ldots \cup \sum^n$

Here '+' is the positive closure operator

$\therefore \sum^* = \sum^+ \cup \in$

Language:

Language is a set of strings formed from some specific alphabet set. Language over $\sum$, L is any subset of $\sum^*$.

Example: $\sum = \{a, b\}$

$\sum^* = \{\in, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$

$L = \{a, aa, aab\}$ is a finite language on $\sum$

Star-closure of language $L \subseteq \Sigma^*$

$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$ Remember that $L^0 = \{\in\}$

set of all strings by concatenating strings from L ... strings are "building blocks" rather than elements from Σ

Positive Closure

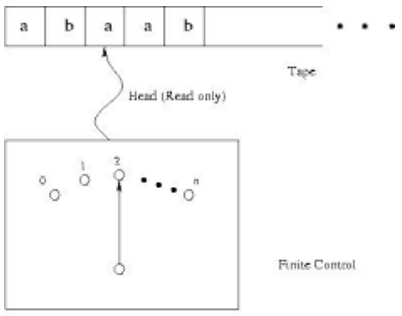
$L+ = L1 \cup L2 \cup \ldots = L^* - L0$

FINITE AUTOMATA

- A Finite Automata is the mathematical model of a digital computer. Finite Automata are used as string or language acceptors. They are mainly used in pattern matching tools like LEX and Text editors.
- The Finite State System represents a mathematical model of a system with certain input. The model finally gives a certain output. The output given to the machine is processed by various states. These states are called intermediate states.
- A good example of finite state systems is the control mechanism of an elevator. This mechanism only remembers the current floor number pressed, it does not remember all the previously pressed numbers.
- The finite state systems are useful in design of text editors, lexical analyzers and natural language processing. The word "automaton" is singular and "automata" is plural.
- An automaton in which the output depends only on the input is called an automaton without memory.
- An automaton in which the output depends on the input and state is called as automation with memory.

1. FINITE AUTOMATON MODEL

The Finite Automata can be represented as,



Finite Automaton

i) **Input Tape:** Input tape is a linear tape having some cells which can hold an input symbol from $\sum$.

ii) **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. 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. The finite automaton can be represented by a transition diagram in which the vertices represent the states and the edges represent transitions.

- A Finite Automaton (FA) consists of a finite set of states and set of transitions among states in response to inputs.
  - Always associated with a FA is a transition diagram, which is nothing but a 'directed graph'.
  - The vertices of the graph correspond to the states of the FA.
  - The FA accepts a string of symbols from $\sum$, x if the sequence of transitions corresponding to symbols in x leads from the state to an accepting state.

Finite Automata (FA) → FA without output or Language Recognizers (e.g. DFA and NFA)

Finite Automata (FA) → FA with output or Transducers (e.g. Moore and Mealy machines)

1.1 Deterministic Finite Automata (DFA):

i. In Deterministic Finite Automata no input symbol causes to move more than one state or a state does not contain more than one transition from same input symbol.

ii. Each and every state has to consume all the input symbols present in $\sum$.

iii. A Deterministic Finite Automaton is represented by a 5-Tuple machine:

i.e $M = (Q, \Sigma, \delta, q_0, F)$

where

Q is finite set of states

Σ is finite input alphabet

δ is transition mapping function i.e $Q \times \Sigma \rightarrow Q$

$q_0 \in Q$ Initial state

$F \subseteq Q$ Set of final states

Acceptance by an Automaton:

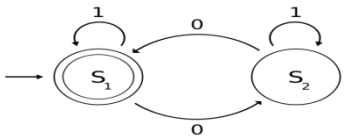
- A string "w" is said to be accepted by a finite automation M=(Q, Σ, δ, q_0, F)

  If δ(q_0, w)=p for some p in F. The language accepted by M, designated L(M), is the set {w|δ(q_0, w) is in F}.
- A language is a regular set, if it is the set accepted by some automaton.
- There are two preferred notations for describing automata
  1. Transition Diagram
  2. Transition Table

Examples:

1. Construct DFA for accepting the set of all strings containing even number of 0s over an alphabet {0,1}.

   Transition Diagram:



   DFA tuples are M = (*Q*, Σ, δ, q_0, *F*)

   where

   Q = {S_1, S_2}, Σ = {0, 1}, q_0 = S_1, *F* = {S_1}, and

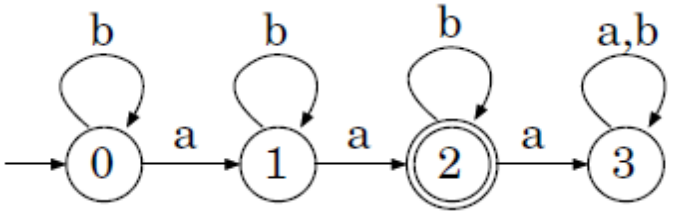
   δ is defined by the following state transition table

   Transition Table: δ:Q x Σ → Q

	0	**1**
***S*1**	S_2	S_1
***S*2**	S_1	S_2

2. Construct DFA for all accepting strings over {a,b} that contains exactly 2 a's.

   Transition Diagram:



DFA tuples *are* $M = (Q, \Sigma, \delta, q_0, F)$

where

$Q = \{0,1,2,3\}, \Sigma = \{a,b\}, q_0 = 0, F = \{2\}$, and

δ is defined by the following state transition table

Transition Table : $\delta: Q \times \Sigma \rightarrow Q$

	a	**b**
0	1	0
1	2	1
2	*3*	*2*
3	*3*	*3*

1.2 Non-Deterministic Finite Automata (NFA):

i. In NFA, one input symbol causes to move more than one state or a state may contain more the one transition from same input symbol.
ii. It is not compulsory that all the states have to consume all input symbols in $\sum$.
iii. An Non-Deterministic Finite Automata is represented by a 5 – tuple.
$M = (Q, \Sigma, \delta, q_0, F)$

where

Q is finite set of states

Σ is finite input alphabet

δ is transition mapping function i.e $Q \times \Sigma \rightarrow 2^Q$

$q_0 \in Q$ Initial state

$F \subseteq Q$ Set of final states

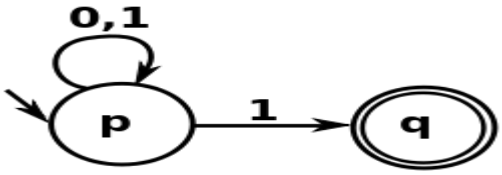
1.2.1 Acceptance by NFA:

An NFA accepts a string "w" if it is possible to make any sequence of choices of next state, while reading the characters of w, and go from start state to any accepting state.

Example:

1. Construct an NFA for the set of all strings over the alphabet {0,1}containing the string ends with a 1.

Transition Diagram:



NFA tuples are M = (Q, Σ, δ, q_0, F)

where

Q = {p, q}, Σ = {0, 1}, q_0 = p, F = {q}, and

δ is defined by the following state transition table

Transition Table: $\delta: Q \times \Sigma \rightarrow 2^Q$

	0	**1**
p	{p}	{p,q}
q	∅	∅

1.3 Differences between DFA and NFA

- In the case of DFA, the transition function gives exactly one state, when applied an input symbol.
- In the case of NFA, there can be several possible next states, and the automation 'guesses' (always correctly) which next state (of the set of possible next states) will lead to acceptance of the input string.
- DFA is a particular case of NFA so, transition function in NFA is $\delta: Q \times \Sigma \rightarrow 2^Q$

Note:

1) All DFAs are NFAs.
2) All NFAs are not be DFAs.

2. Equivalence between NFAs and DFAs

Conversion of NFA to DFA

Let N = (Q_N, ∑, δ_N, q0, FN) be a NFA.
We have to construct the DFA, D= (Q_D, ∑, δ_D, {q_0}, FD) Such that L (D) =L (N).

∴ Language accepted by DFA should be same as language accepted by NFA.

Step 1: Convert the given transition system into state transition table where each state corresponds to a row and each input symbol corresponds to a column.

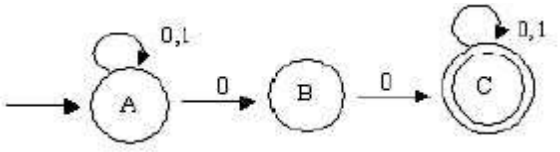
Step 2: Construct the succession table which lists subsets of states reachable from the set of initial states.

Step 3: the transition graph given by the successor table is the required deterministic system.

⇨ The final states contain some final state of NFA. If possible we can reduce the number of states.

Example:

1. Construct DFA equivalent to the following NFA.



Sol:

Equivalent DFA construction: $D=(Q_D, \sum, \delta_D, \{q_0\}, F_D)$

$Q_D=\{\{A\},\{A,B\},\{A,C\},\{A,B,C\}\}$

$\sum=\{0,1\}$

$F_D=\{\{A,C\},\{A,B,C\}\}$

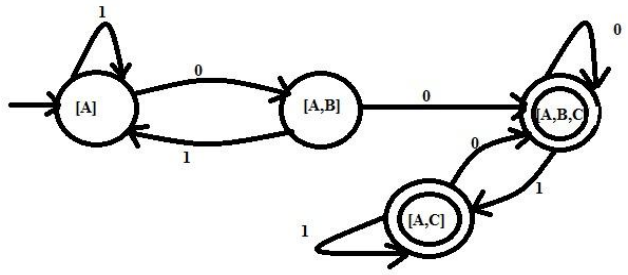
(These are final states as C is the final state of NFA and all these states contain C)

The transition function δ_D is

Transition Table:

State/$\sum$	0	1
→[A]	[A,B]	[A]
[A,B]	[A,B,C]	[A]
*[A,B,C]	[A,B,C]	[A,C]
*[A,C]	[A,B,C]	[A,C]

Transition Diagram:



Note:

While converting NFA to DFA

(1) No change in the initial state.

(2) Number of final states may be changed

(3) Number of states may be changed.

(4) Transition function is not changed.

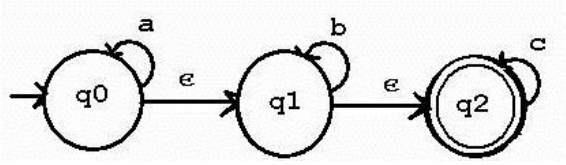
(5) If NFA contains 'n' states, then the equivalent DFA contains maximum 2^n states. These state names are also subset of given 'n' states.

3. Finite Automation with ∈ - Moves (Epsilon Transitions)

- An NFA is allowed to make a transition spontaneously, without receiving an input symbol. These ∈ - NFA's can be converted to DFA's accepting the same language.
- Finally NFA with ∈ - **moves** can be defined to be a 5-Tuple.

  M= (Q, ∑, δ , q0,F) with δ mapping from Q x (∑ ∪ {∈}) → 2^Q.

∈ - closure(q): ∈ - closure(q) is the set of all vertices p such that there is a path from q to p on ∈ - alone.



∴ In the figure,

∈ - closure(q_0)={q_0,q_1,q_2}

∈ - closure(q_1)={ q_1,q_2}

∈ - closure(q_2)={ q_2}

3.1 Conversion of an ∈- NFA to NFA without ∈- moves:

Convert a transition system with ∈- moves into an equivalent transition system without ∈- moves.

Suppose we want to replace an ∈- moves from vertex V_1 to vertex V_2

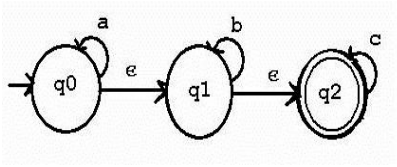
(1) Find all edges starting from V_2

(2) Duplicate all these edges starting from V_1, without changing the edge labels.

(3) If V_1 is an initial state, make V_2 also an initial state.

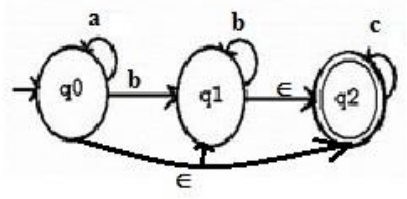
(4) If V_2 is a final state make V_1 also a final state.

Example:

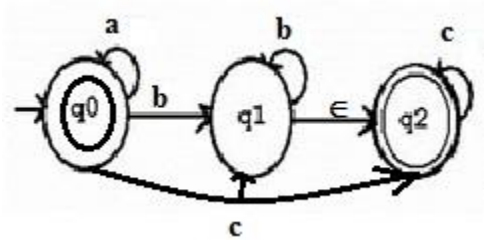


Solution:

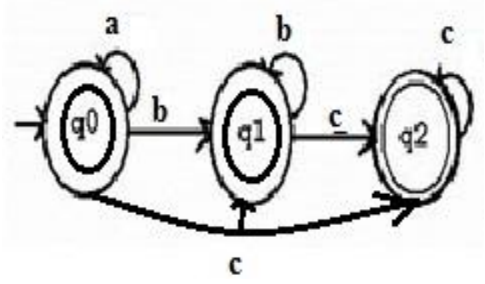
We first eliminate the ∈- moves from q_0 to q_1



Then we eliminate the ∈- moves from q_0 to q_2



Next eliminate the ∈- moves from q_1,q_2.



4. Moore Machine

- A Moore machine is a FA in which the output is associated with the state.
- A Moore machine is a 6- Tuple $(Q, \sum, \Delta, \delta, \lambda, q_0)$

Where

Q: finite set of states.

∑: finite set of input symbols

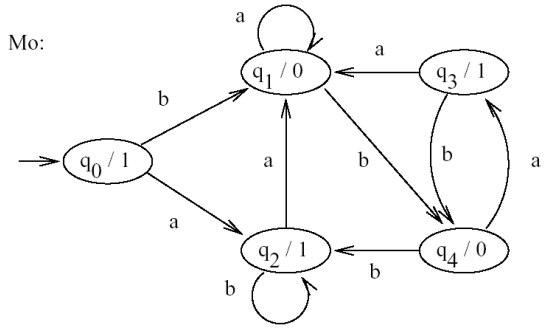
Δ: finite set of output alphabet

δ: Transition function i.e. QX∑→Q

λ (output function): Q→ Δ (λ is a function from Q to Δ)

q_0: initial state

Example:



Note:

- Without taking an input, a Moore machine produces the output.
- If the length of the input string is 'n' the Moore machine produces the output string of length 'n+1'.

5. Mealy machine

- In Mealy machine output is associated with each transition, output will be dependent on present state and present input symbol.
- A mealy machine is a 6-Tuple $(Q, \sum, \Delta, \delta, \lambda, q_0)$

Where

Q: finite set of states.

∑: finite set of input symbols

Δ: finite set of output alphabet

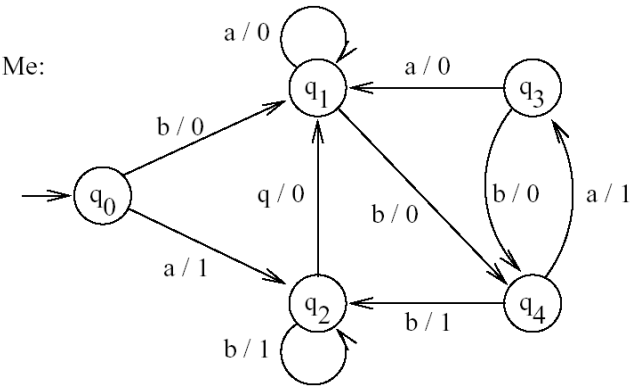
δ: Transition function i.e. QX∑→Q

λ (output function): QX∑→ Δ (i.e. λ (q,a) gives the output associated with the transition from state q on input a)

q_0: initial state

Example:

Me:



Note:

- Without giving any input the Mealy machine doesn't generate output.
- If the length of the input string is 'n' the Mealy machine produces the output string of length 'n'.

6. Equivalence of Moore and Mealy machines

6.1 Mealy machine equivalent to Moore machine:

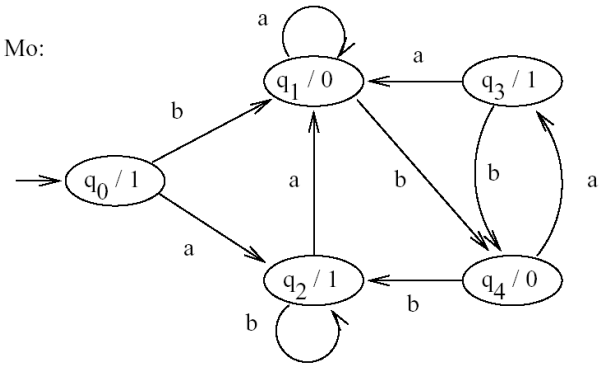
Theorem: If $M_1 = (Q, \sum, \Delta, \delta, \lambda, q0)$ is a Moore machine, then there is a Mealy M_2 equivalent to M_1.

Proof :

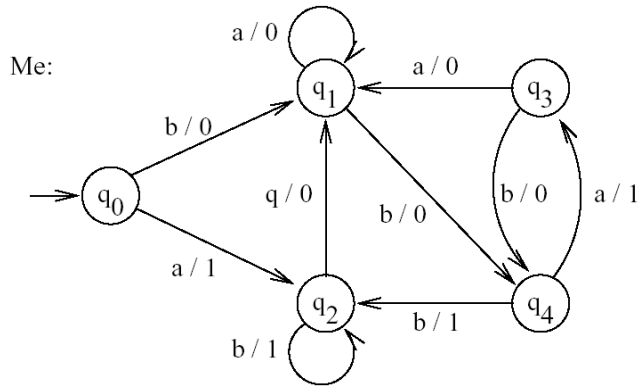
Let $M_2 = (Q, \sum, \Delta, \delta, \lambda^1, q_0)$ and define $\lambda^1(q,a)$ to be $\lambda(\delta(q,a))$ for all states q and input symbol 'a'.

Then $M_1$and M_2enter the same sequence of states on the same input, and with each transition M_2 emits the o/p that M_1 associates with the state entered.

Example: Construct an equivalent an Mealy machine for the following Moore machine.



Solution:



6.2 Moore machine equivalent to Mealy machine:

Theorem: If $M_1 = (Q, \sum, \Delta, \delta, \lambda, q_0)$ be a Mealy machine, then there is a Moore machine M_2 equivalent to M_1.

Proof :

Let $M_1 = (Q \times \Delta, \sum, \Delta, \delta^1, \lambda^1, [q_0, b_0])$ where b is arbitrary selected member of Δ. That is ,the states of M_2 are pairs [q,b] consisting of a state of M_1 and o/p symbol.

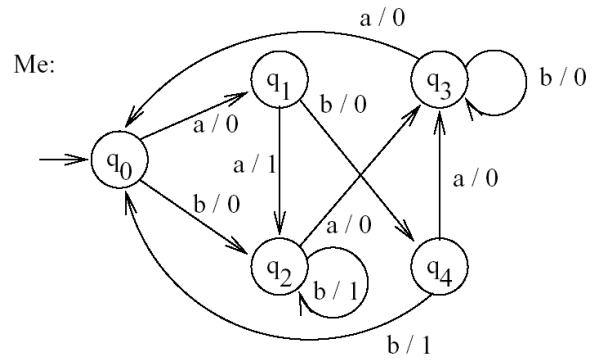
Define $\delta^1([q,b],a) = [\delta(q,a), \lambda(q,a)$ and $\lambda^1([q,b]) = b$.

The second component of state [q,b] of M_2 is the output made by M_1 on some transition into state q.

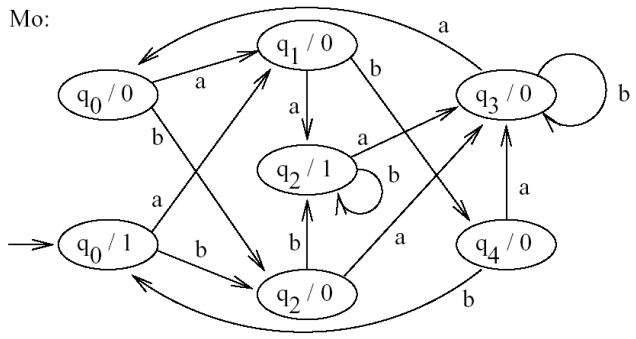
Only the first components of M_2 's states determine the moves made by M_2 .

Every induction on 'n' shows that if M_1 enters states $q_0, q_1 \ldots q_n$ .on input $a_1, a_2, \ldots a_N$ and emits output $b_1, b_2 \ldots\ldots b_n$ then M_2 enters states $[q_0,b_0]$, $[q_1,b_1]$---, $[q_n,b_n]$ and emits outputs $b_0, b_1 .. b_n$.

Example: Transform the following Mealy machine into its equivalent Moore machine.



Solution:

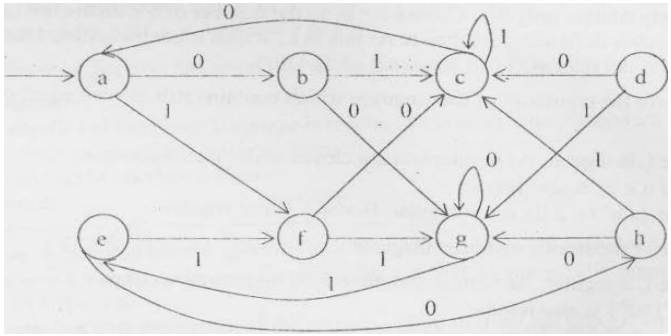


7. MINIMIZATION OF FINITE AUTOMATA

We say that a state p is distinguishable from state q if there exist a string x such that δ(p,x) is in F and δ (q .x) is not. or vice versa. Otherwise they are said to be equivalent.

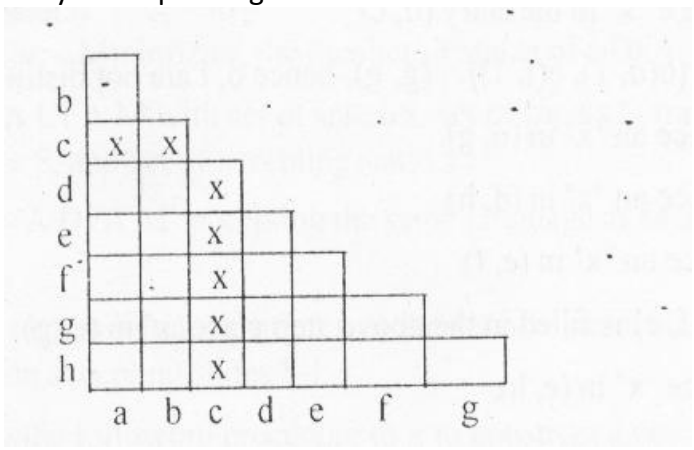
Example:

Minimization can be explained easily with an example. Consider the following DFA and minimize it.



Solution:

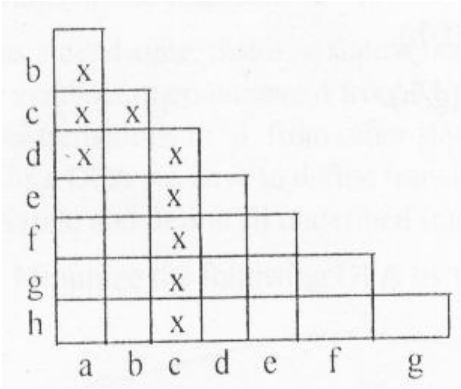
Construct a table as shown below and place an ·x· in the table each time we discover a pair of states that cannot be equivalent, that is they arc distinguishable. Initially an ·x' is placed in each entry corresponding to one final state and one non final state.



Next for each pair of states p and q that are not already known l0 be distinguishable we construct the pairs of states r = δ (p. a) and s = δ (q, a) for each input symbol a. If states r and s have been shown to be distinguishable by some string x. then p and q arc distinguishable by string x. Thus if the entry (r, s) in the table has an ·x', and 'x' is also placed at the entry (p, q). If the entry (r, s) does not yet have an 'x', then the pair (p, q) is placed on a list associated with the (r, s) entry. Al some future time if the (r. s) entry receives an 'x', then each pair on the list associated with the (r, s)-entry also receives an 'x'.

Continuing with the example, we place an 'x' in the entry (a. b) since the entry (δ (b. 1), δ (a, 1)) = (c. f) already has an 'x'. Similarly. the (a, d)- entry receives an 'x' since the entry (8(a, 0), 8(d, 0) = (b, c) has an':<.".

Now the table is



(δ (a, 1), δ (e, 1))= (f, f), on input '1' both are going to same state. So no string starting with '1' can distinguish states a and c. Now try on input '0'.

(δ (a, 0), δ (e, 0)) = (b, h), as (b, h) is not filled, so associate (a, e) with (b, h)-list.

-> (δ (a, 0), (f. 0)) = (b, c), as (b,c) already has an 'x' place 'x' in the entry (a, f).

-> (δ (a, 0), (g. 0)) = (b, g), as (b, g) is not filled associate (a. g) with (b, g)-list

-> ((a, 1), δ(h, 1)) = (f, c), as (f, c) already has an 'x' place an 'x' in the entry (a, h) also.

-> (δ (b, 0), (d, 0)) =(g, c). Place ·x' in the entry (b, d).

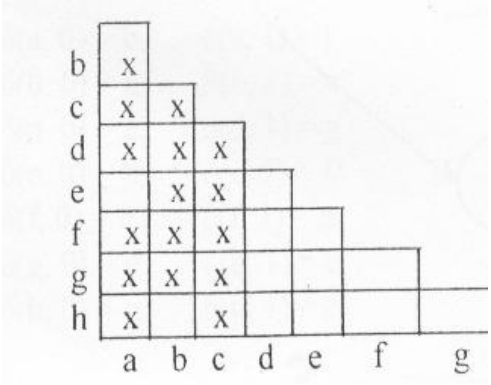
-> (δ (b, 1), (c. 1)) = (c. 1), place 'x' in the entry (b, e)

-> (δ (b, 1), (f. 1)) = (c, g), place 'x' in the entry (b. f).

-> (δ (b, 1), δ (g, 1)) = (c, e), place 'x' in the entry (b, g). As (a, g) is associated with (b. g) place and 'x' in {a, g) also.

-> ((b. 1), δ(h,1)) = (c, c) and (δ(b. 0), δ(h, 0))=(g, g). On each input symbol slates b and h are going to the same state. Hence they are not distinguishable.

Now the table is

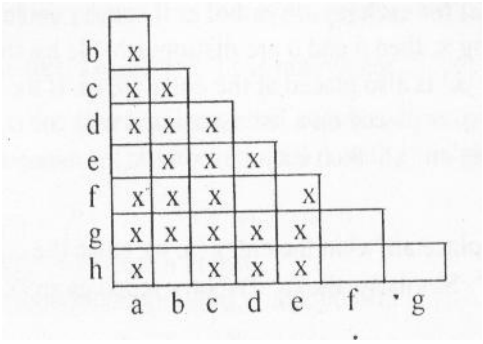


((d, 0), δ(e, 0))= (c, h), place 'x' in the entry (d. c).

(δ (d, 0), (f, 0)) = (c, c) and ((d,1) , δ(f, 1)) = (g, g). hence d, fare not distinguishable.

-> (δ (d, 0), (g, 0)) = (c, g), place an 'x' in(d, g).
-> (δ (d, 0). (h, 0)) = (c, g), place an 'x' in (d, h).
-> (δ (c, 0), (f, 0)) = (h, c), place an 'x' in (e, f).
-> (δ (c, 1), (g, 1))= (f. e), as (f, e) is filled in the above step place 'x ' in (c, g).
-> ((e, 1),δ(h,1)) = (f, c), place 'x' in (e, h).

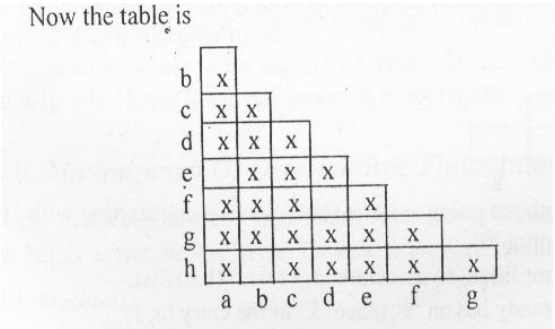
Now the table is



-> (δ (f, 0), δ (g, 0)) = (c, g), place an 'x' in (f. g).

-> (δ (f, 0), δ (h, 0)) = (c, g), place an 'x' in (f, h).

-> (δ (g, 1), δ (h, 1)) = (e, c), place an 'x' in (g , h).



from the above table, the equivalent states(that arc not filled with 'x') are (a, e), (b, h) and (d, f).

The minimum-state finite automaton is