Dinesh Sharma

Theory of Automata & Formal Languages
(NCS 402)

UNIT – I

Languages: —

We all are familier with the notion of natural languages, such as ENGLISH & FRENCH. Dictionary defines the term Language informally as "a system suitable for expression of certain facts, ideas or concepts, including a set of symbols and rules for their manipulation".

While the above definition gives an idea of what a language is, it is not sufficient for the study of formal languages.

Alphabet (Σ): is a finite non empty set of symbols. From the individual symbols of alphabet we construct strings, which are finite sequence of symbols from the alphabet Σ.

e.g If $\Sigma = \{a, b\}$

Then abab and aaabbba are strings on Σ.

We use lower case letters a, b, c, ... for elements of Σ and u, v, w for string names.

e.g $w = abaaa$

w — string name; abaaa — string

Concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w,

For e.g if $w = a_1, a_2, \ldots, a_n$

$v = b_1, b_2, \ldots b_n$ then concatenation of w and v, denoted by wv is

$$wv = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_n$$

The **Reverse** of a string is obtained by writing the symbols in reverse order. If $w = a_1 a_2 \ldots a_n$

Then $w^R = a_n \ldots a_2 a_1$

The Length of a string w denoted by $|w|$ is the number of symbols in the string.

Empty string (λ) is a string with no symbol at all.

$$|\lambda| = 0$$
$$\lambda w = w\lambda = w$$

Any string of consecutive symbols in some string w is said to be substring of w.

If $w = uv$ Then u and v are said to be prefix and suffix of w

e.g. If $w = abbab$

Then $\{\lambda, a, ab, abb, abba, abbab\}$ is the set of all prefixes of w.

Kleene Closure: If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains λ.

To exclude the empty string, we define

(Positive Closure) $\Sigma^+ = \Sigma^* - \{\lambda\}$

while Σ is finite by assumption, Σ^* and Σ^+ are always infinite since there is no limit on the length of the strings in these sets.

example Let $\Sigma = \{a, b\}$ then

$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, bbb \ldots\}$.

The set $\{a, aa, aab\}$ is a language on Σ because it has a finite number of sentences, we call it a finite language. The set $L = \{a^n b^n : n > 0\}$ is also a language on Σ. The string aabb and aaaabbbb are in language L but the string abb is not in L. This language is infinite. Most interesting languages are infinite.

Language: A language is defined very generally as a subset of Σ^*. A string in a language L will be called a sentence of L

Since Languages are sets, the union, intersection, and difference of two languages are immediately defined. The compliment of a langugage is defined with respect to Σ^*

$$\text{Compliment of } L\ (\bar{L}) = \Sigma^* - L.$$

The reverse of a language is the set of all string reversals, i.e

$$L^R = \{ w^R : w \in L \}$$

The concatenation of two languages L_1 & L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2, specifically,

$$L_1 L_2 = \{ xy : x \in L_1, y \in L_2 \}.$$

star closure of Language

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

and positive closure is

$$L^+ = L^1 \cup L^2 \cup \ldots$$

Grammars: —

A grammar is defined as a quadruple

$$G = (V, T, P, S)$$

where $V \rightarrow$ Finite set of objects called variables
$T \rightarrow$ Finite set of objects called terminals
$P \rightarrow$ Finite set of productions
$S \rightarrow S \in V$ is a special symbol called start symbol.

* V and T are non empty and disjoint set

The production rules are the heart of grammer, they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar.

Production rules are of form

$$x \to y$$

where $x \in (V \cup T)^+$

$y \in (V \cup T)^*$

e.g

Consider $G = (\{S\}, \{a,b\}, S, P)$

with P given by

$S \to aSb$

$S \to \lambda$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

So we can write

$$S \overset{*}{\Rightarrow} aabb$$

The string aabb is a sentence in language generated by G, while aaSbb is a sentential form.

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit ~~clear~~ description of the language from the grammar. Here however, the answer is fairly clear.

$$L(G) = \{a^n b^n : n \geq 0\}$$

e.g Find a Grammar that generates $L = \{a^n b^{n+1} : n \geq 0\}$

Ans $G = (\{S, A\}, \{a,b\}, S, P)$

with productions

$S \to Ab$

$S \to aAb$

$A \to \lambda$

Ans

Equivalence of Grammars :—

Two grammars G_1 and G_2 are equivalent if they generate the same language i.e

$$L(G_1) = L(G_2)$$

e.g

$G_1 = (\{A, S\}, \{a, b\}, S, P_1)$

$S = aAb \mid \lambda$

$A = aAb \mid \lambda$

$G_2 = (\{S\}, \{a, b\}, S, P)$

$S \rightarrow aSb$

$S \rightarrow \lambda$

$$L(G_1) = L(G_2) = \{a^n b^n : n \geq 0\}.$$

Automaton :—

An Automaton is an abstract model of a digital computer. It has mechanism for reading input. It will be assumed that input is a string over a given alphabet, written on an input file, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. Input file is read from left to right, one symbol at a time. The input mechanism can detect the end of input string. The Automaton can produce the output in some form.

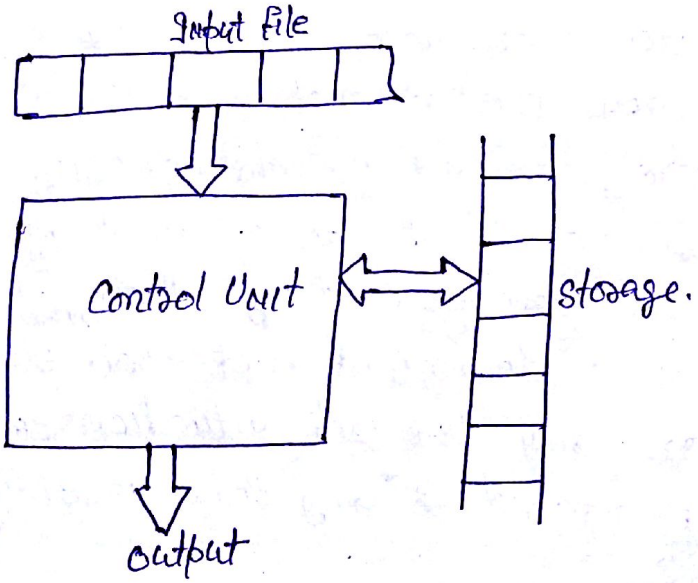


Fig: Automaton

Automaton has a temporary infinite storage which it can read and write. Finally Automaton has a control unit, which can be in any one of a finite number of internal states and which can change state in some defined manner.

Definition

A automaton is defined as a system where energy, materials and information are transformed, transmitted and used for performing some functions without direct participation of man. For e.g. automatic machine tools, automatic packing machines, and automatic photo printing machines.

The characteristics of automaton are:

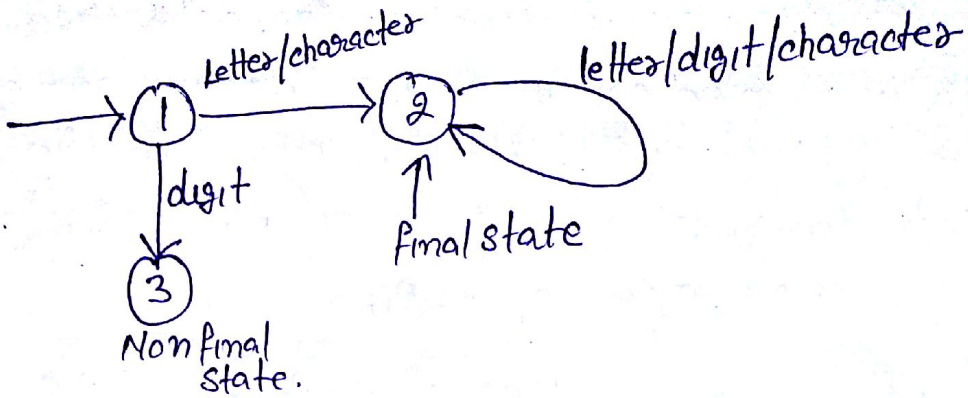
(1) Input
(2) Output
(3) states
(4) state Relation
(5) Output Relation.

Finite Automaton : —

Finite automaton is the simplest model. This is abstract model of digital computer. By being abstract, we mean that theoratically it is closest to the description of a computer system and it formally describes the behaviour of a computer system.

Finite automaton is also called finite acceptors. This type of automaton is characterized by having no temporary storage. Since an input file can not be re-written, a finite automaton is severely limited in its capacity to "remember" things during computation. A finite amount of information can be stored in the control unit by placing the unit into specific states. But since the number of such states is finite, a finite automata can only deal with situations in which the information to be stored at any time is strictly bounded.

e.g



Applications of Finite Automata :—

1. Design of digital circuit.
2. string matching
3. communication protocol for info^n interchange
4. Lexical Analysis phase of compiler

Deterministic Finite Automata (DFA) —

Definition : A DFA is defined by quintuple

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

where

Q is finite, non empty set of states

Σ is finite, non empty set of symbols, called input alphabet

$\delta : Q \times \Sigma \to Q$ is a total function called transition function

q_0 is initial state, $q_0 \in Q$

$F \subseteq Q$ is a set of final state

Working of DFA :—

At the initial time, DFA is assumed to be in initial state q_0, with its input mechanism on leftmost symbol on input string. During each move, the input mechanism moves one position to the right. When the end of input string is reached, the string is accepted if automata

is in one of its final states, otherwise string is rejected. The input mechanism can only move one symbol at a time from left to right. The transition function from internal state to another is governed by transition function δ

e.g $\delta(q_0, a) = q_1$

then if DFA is in state q_0 and input is a DFA will go to state q_1

Transition Graph :— To visualize and represent finite ~~automento~~ autometa we use transition graph in which vertices represent states and edges represent transition. The initial state will be identified by an incoming arrow (un labelled) not originating at any vertex. Final states are drawn with double circles.

Note If $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA

→ Its transition graph G_M has exactly $|Q|$ vertices, each labelled with $q_i \in Q$

→ For every transition rule $\delta(q_i, a) = q_j$, the graph has an edge (q_i, q_j) labelled with a

→ Vertex with name q_0 will be initial state

→ Vertex labelled $q_f \in F$ will be final state.

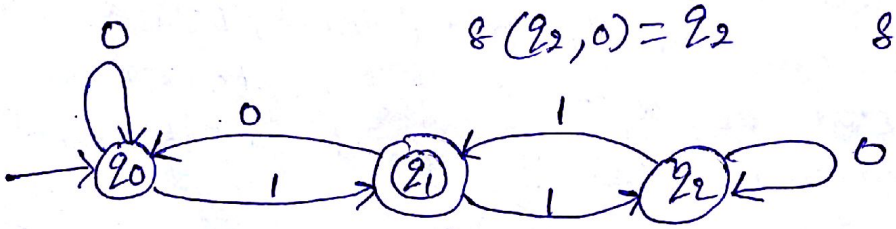
example $M = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$

where δ is

$\delta(q_0, 0) = q_0$ $\quad$ $\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_0$ $\quad$ $\delta(q_1, 1) = q_2$

$\delta(q_2, 0) = q_2$ $\quad$ $\delta(q_2, 1) = q_1$



Accepts 101, 0111, 11001

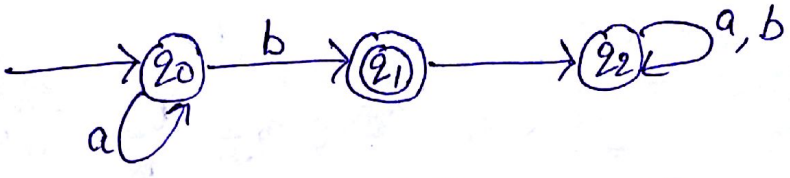
Rejects 100 or 1100

Language and DFA

Def^n The language accepted by DFA $M=(Q, \Sigma, \delta, q_0, F)$ is the set of all strings on Σ accepted by M

or $L(M) = \{ w \in \Sigma^* : \delta^*(q_0, w) \in F \}$

e.g



$L = \{ a^n b : n \geq 0 \}$

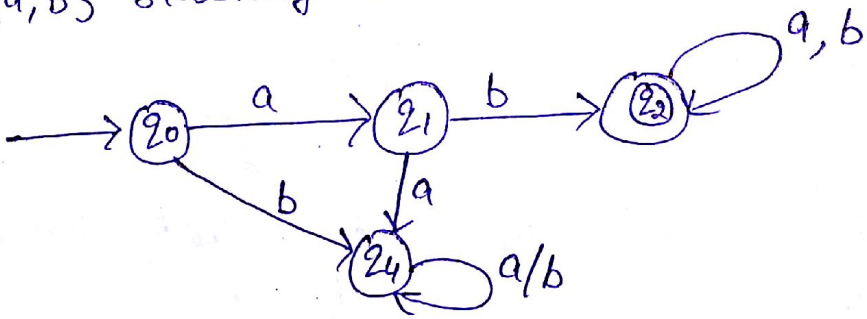
Accepts all strings consisting arbitrary no of a's followed by one b

State \ Input	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2

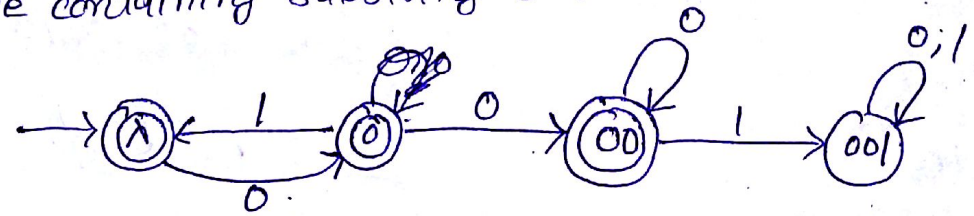
Transition Table.

Q Find a DFA that recognizes the set of all strings on $\Sigma = \{a, b\}$ starting with prefix ab

Ans



Q Find a DFA that accepts all strings on $\{0, 1\}$ except those containing substring 001



check for 100100 and 10101100

Regular Language —

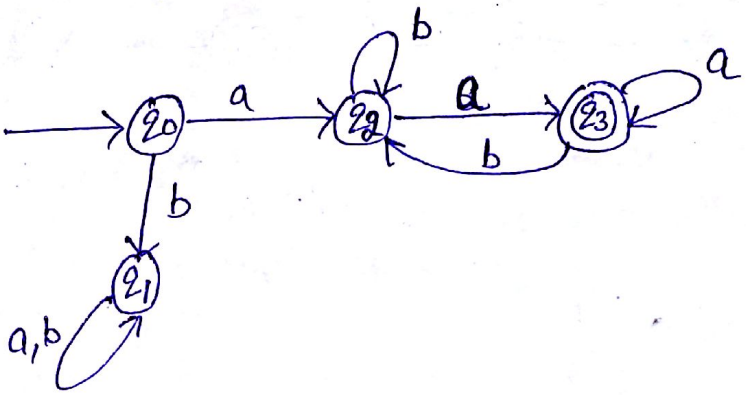
A language L is called regular if and only if there exist some DFA M such that

$$L = L(M)$$

e.g. show that the language

$$L = \{awa : w \in \{a,b\}^*\}$$ is regular

Sol^n



Yes.

Acceptability of String by FA —

e.g. M =

state	Input	
	0	1
→ (q0)	q2	q1
q1	q3	q0
q2	q0	q3
q3	q1	q2

Sol^n (a) 10110l

$\delta(q_0, 10110l)$

$\delta(q_1, 10110l)$

$\delta(q_3, 10110l)$

$\delta(q_2, 10110l)$

$\delta(q_3, 10110l)$

$\delta(q_1, 10110l)$

$\delta(q_0, 10110l)$

$\delta(q_0, \Lambda) = q_0$

Since q_0 is final state so string is accepted

Find which of the following string is accepted by M

a) 10110l b) 11111 c) 000000

Sol^n b Not accepted

Sol^n c Accepted.

Construction of DFA —

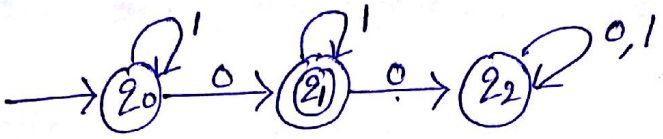
(1) Find a DFA that accept all strings of a's including empty string

Sol^n

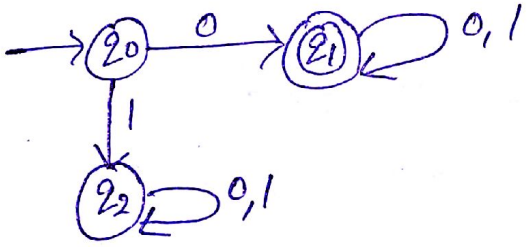
$M = (\{q_0\}, \{a\}, \delta, q_0, \{q_0\})$

(2) Find a DFA that accepts all strings that have exactly one 0 over input alphabet $\{0,1\}$

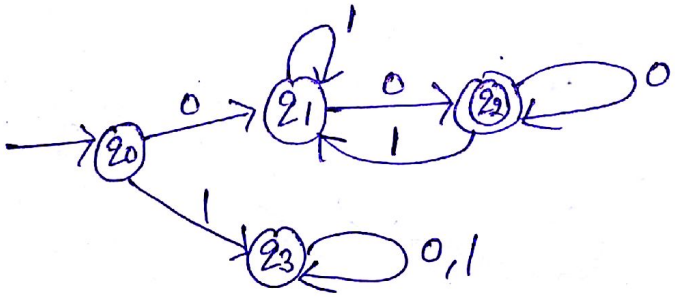
Sol^n



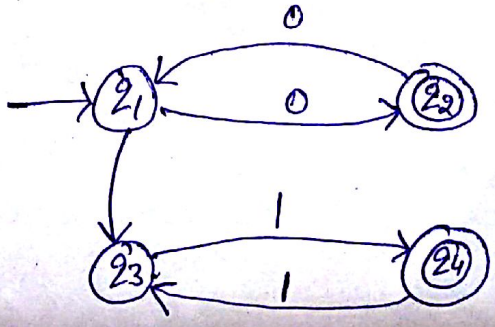
(3) Find a DFA that accepts all strings beginning with 0 over input alphabet $\Sigma = \{0,1\}$



(4) Find a DFA for the language $L = \{0w0 : w \in \{0,1\}^*\}$

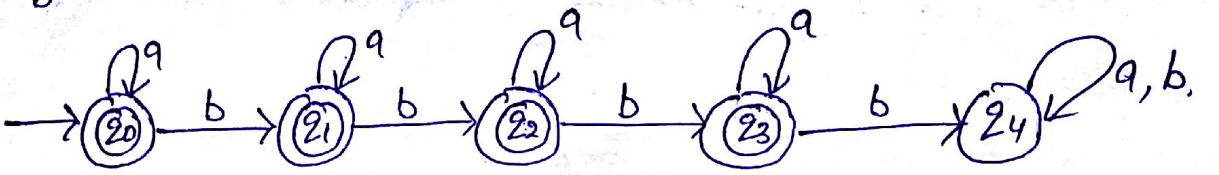


(5) Find a DFA which accepts the language which has either odd no of 0's or even no of 1's but not both together over $\Sigma = \{0,1\}$

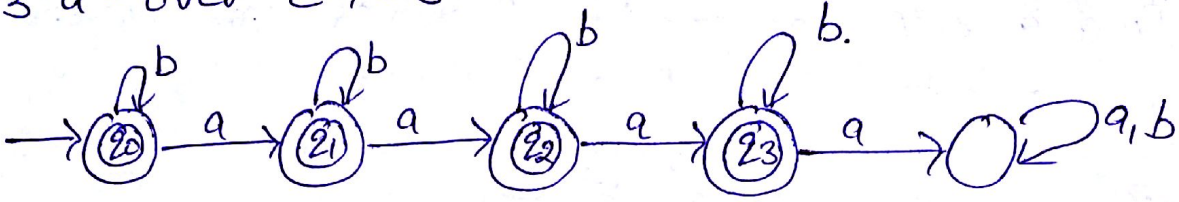


A1

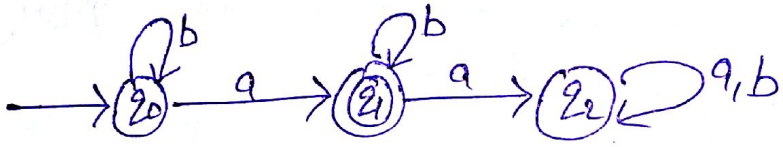
⑥ Find a DFA that accepts all strings with atmost 3 b's over $\Sigma = \{a, b\}$



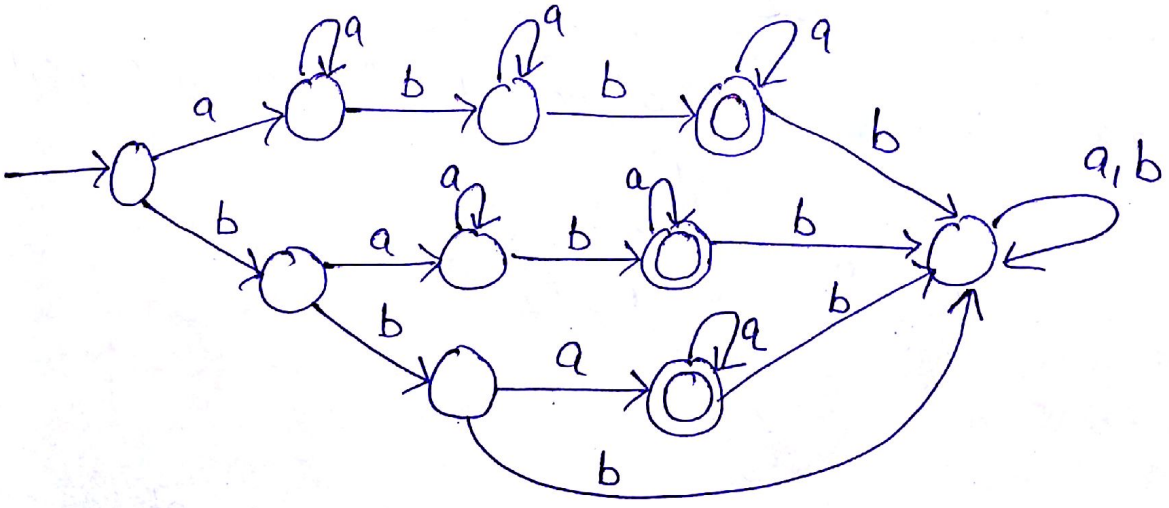
⑦ Find a DFA that accepts all strings with no more than 3 a's over $\{a, b\}$



⑧ Design a DFA with exactly one a over $\Sigma = \{a, b\}$



⑨ Design a DFA for all strings with atleast one a and exactly two b's.



Non Determinism

Non determinism means a choice of moves for an automaton. Rather than prescribing a unique move in each situation, we allow a set of possible moves. Finally we achieve this by defining a transition function so that its range is a set of possible states.

Def^n A non deterministic finite acceptor or NFA is defined by the quintuple

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

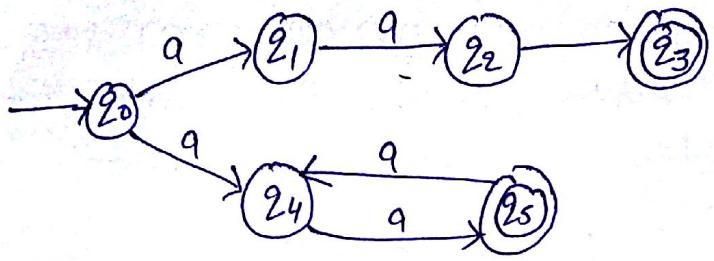
where Q, Σ, q_0 and F are defined as for DFA but $\delta : Q \times (\Sigma \cup \{\Lambda\}) \longrightarrow \{2^Q\}$

Differences b/w NFA & DFA $\longrightarrow$

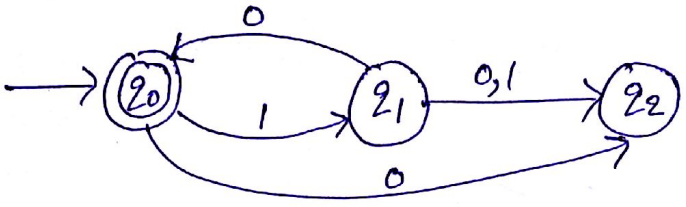
1. In an NFA the range of δ is in the power set 2^Q, so that its value is not a single element of Q, but a subset of it. This subset defines all the possible states that can be reached by transition.
2. NFA can make a transition without consuming an input ~~into~~ symbol.
3. Finally, in NFA, $\delta(q_i, a)$ can be empty meaning that no transition defined for this specific situation.

→ A string is accepted by NFA, if there is some sequence of possible moves that will put the machine in a final state at the end of the string.

e.g ① NFA



e.g ② NFA



$L = (10)^n : n \geq 0$

Note that $\delta(q_2, 0) = \phi$

strings 1010, 101010 are accepted.
but not 110 and 10100. Also note that for 10 there are two walks, one leading to q_0 other to q_2.

Def^n The Language L accepted by NFA $M = (Q, \Sigma, \delta, q_0, F)$ is defined as the set of all strings accepted by it.

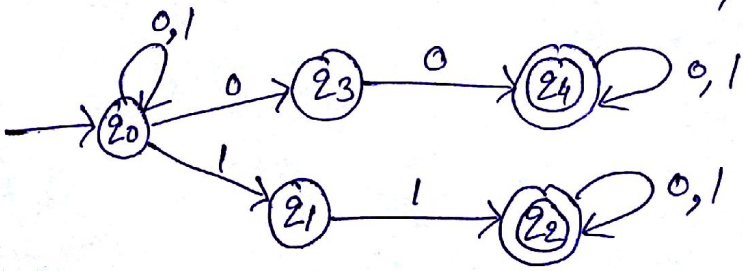
Formally

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \cap F \neq \phi\}$$

i.e the language consists of all strings w for which there is a walk labeled w from initial vertex of the transition graph to some final vertex.

Q Design NFA for L = all strings over {0,1} that have atleast two consecutive 0's or 1's. i.e 00, 11, 101000, 101100

Sol^n



Ans

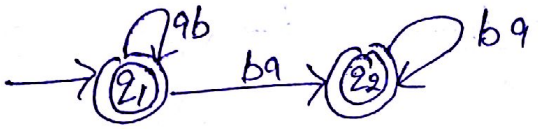
Q.2 Design NFA for $L=(ab)^*(ba)^* \cup aa^*$

Soln.

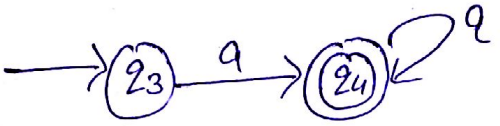
$L = L_1 \cup L_2$

$L_1 = (ab)^*(ba)^*$

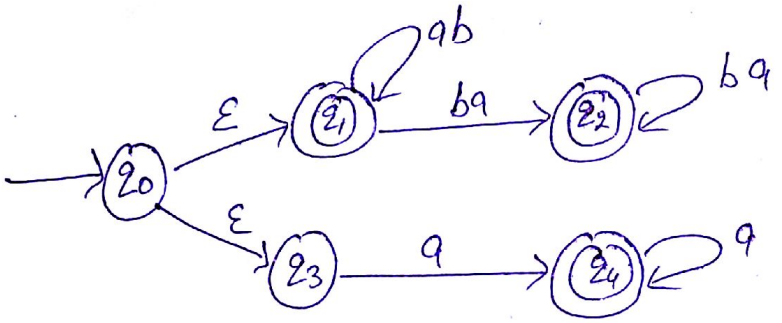
$L_2 = aa^*$



NFA for L_1



NFA for L2



NFA with ∈ moves

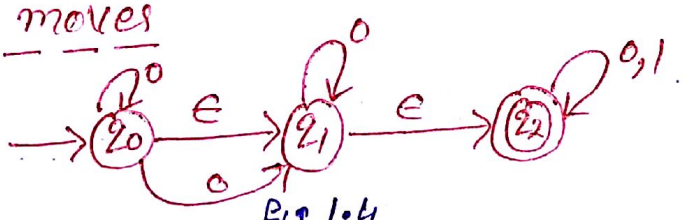


Fig 1.4

This is an NFA with ∈ moves because it is possible to make a transition from state q_0 to q_1 and from q_1 to q_2 without accepting any of the input symbol.

NFA with ∈ transition will also be defined as five tuples

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

Q is finite, non empty set of states

Σ finite set of input symbols called alphabet

$\delta : Q \times \Sigma \cup \epsilon \rightarrow 2^Q$

q_0 is initial state

F : set of final state.

Acceptance of string by NFA with ϵ moves → A string w in Σ^* will be accepted by NFA with ϵ moves if there exist atleast one path corresponding to w, which start in initial state and ends in one of the final state. This path may be formed by ϵ transitions or non ϵ transitions.

Note To find out whether w is accepted or not by NFA, we need to define a function ϵ-closure(q)

ϵ-closure(q) : The function ϵ-closure denotes the set of all states of automata which can be reached from q on path labelled ϵ i.e it is the set of states with distance 0 from state 'q'.

Method To find ϵ-closure : —

To find the set of ϵ-closure(q) say P we do the following (Refer to Fig 1.4 on previous page) :

① Add q to P

② Find all sets $\delta(q, \epsilon)$, for each $q \in P$ and add all P elements of these sets that are not already included in P. stop when this set does not change.

for e.g For NFA in Fig 1.4

To find the ϵ-closure(q_0), we proceed as follows From q_0, we always reach q_0 itself without any input.

So $q_0 \in \epsilon$-closure(q_0)

$\delta(q_0, \epsilon) = q_1$ so add q_1 to P

$\delta(q_1, \epsilon) = q_2$ add q_2

Hence ϵ-closure(q_0) = $\{q_0, q_1, q_2\}$

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

(q_2) = $\{q_2\}$ Ans

Construction of NFA without ε moves from NFA with ε →

Method Let $M=(Q,\Sigma,\delta,q_0,F)$ be an NFA with ε moves. In order to construct NFA without ε moves $M'=(Q,\Sigma,\delta',q_0,F')$ we follow the below steps:

i) $F'=F\cup\{q\}$
   If ε closure of q contains a member of F
   or $F=F'$

ii) $\delta'(q,a)=\varepsilon\text{-closure}(\delta(\hat{\delta}(q,\varepsilon),a))$

e.g Construct NFA without ε from the following NFA

Sol^n

$\varepsilon\text{-closure}(q_0)=q_0q_1q_2$
$\varepsilon\text{-closure}(q_1)=q_1,q_2$
$\varepsilon\text{ closure}(q_2)=q_2$.

$\delta'(q_0,0)=\varepsilon\text{-closure}(\delta(\hat{\delta}(q_0,\varepsilon),0))$
$=\varepsilon\text{ closure}(\delta(q_0q_1q_2,0))$
$=\varepsilon\text{ closure}(\delta(q_0,0)\cup\delta(q_1,0)\cup\delta(q_2,0))$
$=\varepsilon\text{ closure}(\{q_0\},\varepsilon,\phi,\varepsilon,\phi)$
$\delta'(q_0,0)=\{q_0q_1q_2\}$

Now $\delta'(q_0,1)=\varepsilon\text{ closure}(\delta(\hat{\delta}(q_0,\varepsilon),1))$
$=\varepsilon\text{ closure}(\delta(\{q_0q_1q_2\},1))$
$=\varepsilon\text{ closure}(q_1)$
$\delta'(q_0,1)=\{q_1,q_2\}$

Now $\delta'(q_0, 2) = \varepsilon\text{ closure}(\delta(\hat{\delta}(q_0, \varepsilon), 2))$

$= \varepsilon\text{ closure}(q_2)$

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

Similarly

$\delta'(q_1, 0) = \phi$

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

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

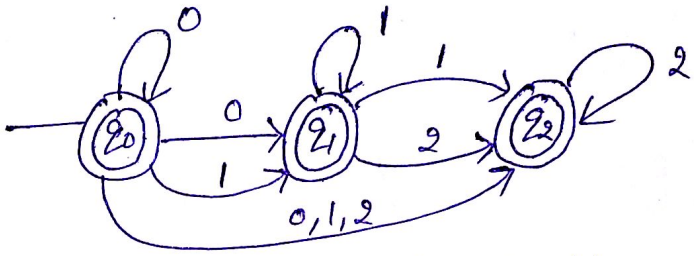
$\delta'(q_2, 0) = \phi$

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

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

State Transition Table

state	Input		
	0	1	2
→ 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



State Transition Graph.

NFA To DFA Construction —

For every NFA, there is a equivalent DFA. Since the DFA equivalent of NFA is to simulate the moves of NFA in parallel every state of DFA will be combination of ~~both~~ one or more states of NFA. Hence every state of DFA will be represented by some subset of states of NFA and therefore, the transformation from NFA to DFA is normally called subset construction.

Algorithm: NFA To DFA

① Create a graph G_D with Vertex $\{q_0\}$. Identify this vertex as the initial vertex.

② Repeat the following until no more edges are missing.

Take any vertex $\{q_1, q_2 \dots q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_1, a)$, $\delta^*(q_2, a) \dots \delta_N^*(q_k, a)$. If $\delta_n^*(q_1, a) \cup \delta_N^*(q_2, a) \cup \dots \cup \delta_N^*(q_k, a)$

$= \{q_e, q_m, \dots, q_n\}$

Create a vertex for G_D, labelled $\{q_e, q_m, \dots q_n\}$ if it does not already exist. Add to G_D and edge from $\{q_1, q_2 \dots q_k\}$ to $\{q_e q_m \dots q_n\}$ and label it with a.

③ Every state of G_D whoose label containing any $q_f \in F_n$ is identified as final state.

④ If M_N accept λ, the vertex q_0 in G_D is also made final vertex.

Example Convert the given NFA To DFA

NFA

$(q_0, 0) = q_0 q_1$

$(q_0, 1) = q_1$

$(q_0 q_1, 0) = q_0 q_1 q_2$

$(q_0 q_1, 1) = q_1 q_2$

$(q_1, 0) = q_2$

$(q_1, 1) = q_2$

$(q_0 q_1 q_2, 0) = q_0 q_1 q_2$

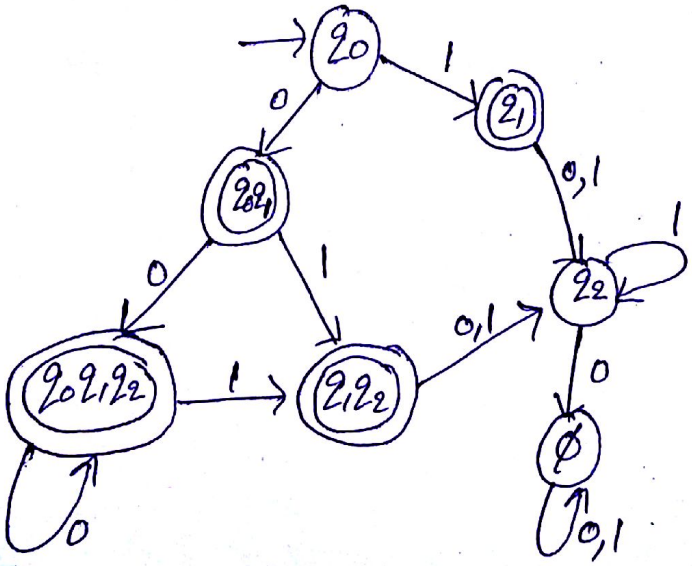
$(q_0 q_1 q_2, 1) = q_1 q_2$

$(q_1 q_2, 0) = q_2$

$(q_1 q_2, 1) = q_2$

$(q_2, 0) = \phi$

$(q_2, 1) = q_2$ As



DFA

Reduction in Numbe of states — Any DFA defines a unique language but the converse is not true. For a given language there may be many DFA that accept it. There can be considerable difference in the number of states in these equal automata. Theoratically all solutions are equally satisfactory but if the result to be applied in practical settings, there may be reasons for preferring one over the other

e.g Two equivalent DFA

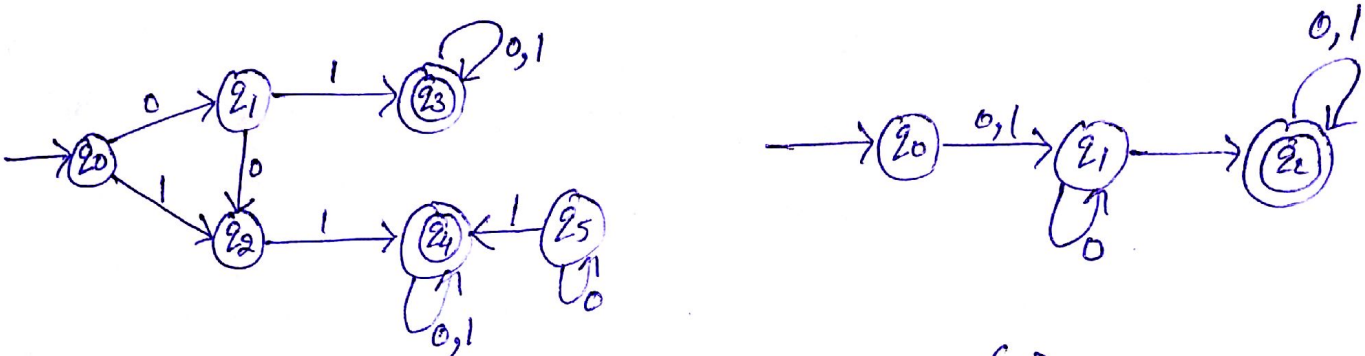


Fig: Two Equivalent DFA

Defn Two states P and Q of a DFA are called indistinguishable if

$\delta^*(P,w) \in F$ implies $\delta^*(Q,w) \in F$

and $\delta^*(P,w) \notin F$ implies $\delta^*(Q,w) \notin F$

For all $w \in \Sigma^*$. If on the other hand, there exist some string such that

$\delta^*(P,w) \in F$ and $\delta^*(Q,w) \notin F$ or vice versa, then the state P and Q are said to be distinguishable by a string w

Method To Reduce the Number of states:

Procedure Mark:

① Remove all inaccessible paths. This can be done by enumerating all the simple paths from initial state. Any state not part of some path is inaccessible.

② Consider all pairs of states (P, Q). If $P \in F$ and $Q \notin F$ or vice versa mark the pair (P, Q) as distinguishable.

③ Repeat the following step untill no previously unmarked pairs are marked.

— For all pairs (P, Q) and all a's $\in \Sigma$, Computer $\delta(P, a) = P_a$ and $\delta(Q, a) = Q_a$. If the pair (P_a, Q_a) is marked distinguishable, mark (P, Q) as distinguishable.

Procedure Reduce:

① Use mark to generate all equivalence classes, say $\{Q_i, Q_j, \ldots, Q_k\}$ as described.

② For each set $\{Q_i, Q_j \ldots, Q_k\}$ of such indistinguishable states, create a state labelled $ij \ldots k$ for $\hat{M}$.

③ For each transition rule of $\hat{M}$ of the form

$$\delta(Q_r, a) = Q_p$$

Find the set to which Q_r and Q_p belongs. If $Q_r \in \{Q_i Q_j \ldots Q_k\}$ and $Q_p \in \{Q_l Q_m \ldots Q_n\}$ add $\hat{\delta}$ a rule

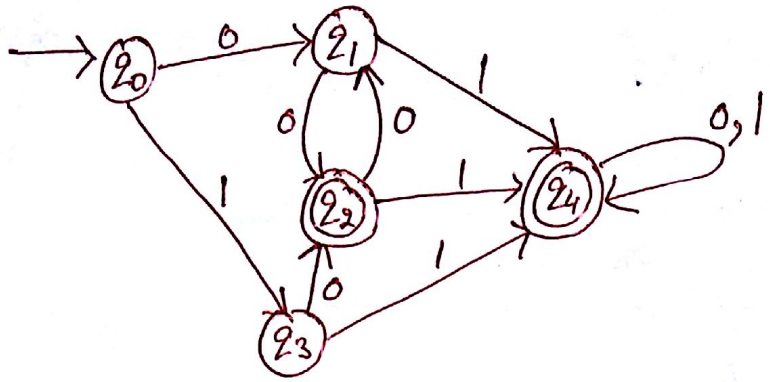
$$\hat{\delta}(ij \ldots k, a) = (l m \ldots n)$$

④ The initial state $\hat{Q}_0$ is that state of $\hat{M}$ whose label includes 0

⑤ $\hat{F}$ is the set of all the states whose label contains i such that $Q_i \in F$

Q-1

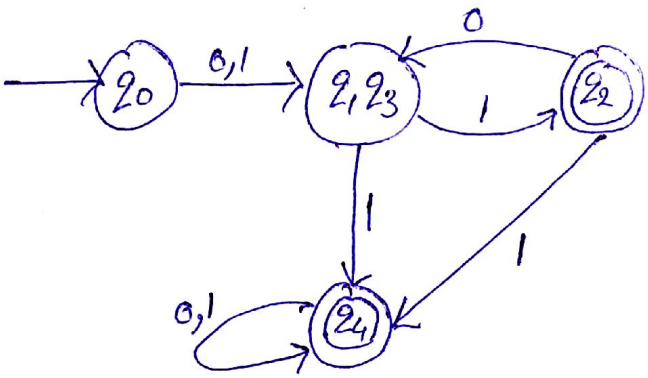
e.g Reduce the no of states from the DFA



Soln $[\{q_2, q_4\}\ \{q_0, q_1, q_3\}]$

$[\{q_2, q_4\}\ \{q_0\}\ \{q_1, q_3\}]$

$[\{q_2\}\ \{q_4\}\ \{q_0\}\ \{q_1, q_3\}]$



DF with minimum number of states.

Q-2 Construct a minimum state automata equivalent to given DFA

state	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_4	q_3
q_2	q_4	q_3
(q_3)	q_5	q_6
(q_4)	q_7	q_6
q_5	q_3	q_6
q_6	q_6	q_6
q_7	q_4	q_6

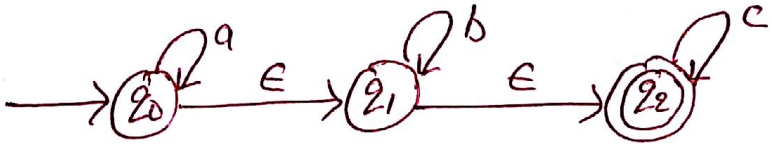
$(q_0 q_1 q_2 q_5 q_6 q_7)(q_3 q_4)$

Sol^n

state	a	b
$[q_0]$	$[q_1 q_2]$	$[q_1 q_2]$
$[q_1 q_2]$	$[q_3 q_4]$	$[q_3 q_4]$
$[q_3 q_4]$	$[q_5 q_7]$	$[q_6]$
$[q_5 q_7]$	$[q_3 q_4]$	$[q_0]$
$[q_6]$	$[q_6]$	$[q_6]$

NFA with ε moves to NFA without ε moves conversion

Q-1 Consider the NFA with ε moves and find out the equivalent NFA without ε.



Sol^n initial states of NFA with without $\varepsilon = \varepsilon$ closure(q_0)

$= \{q_0 q_1 q_2\}$

Rest of the states are

ε closure$(q_1) = \{q_1 q_2\}$

ε closure$(q_2) = \{q_2\}$

$\delta(q_0q_1q_2, a) = \varepsilon\text{ closure}(\delta(q_0q_1q_2, a))$

$= \varepsilon\text{ closure}(q_0)$

$= \{q_0q_1q_2\}$

$\delta(q_0q_1q_2, b) = \varepsilon\text{ closure}(\delta(q_0q_1q_2, b))$

$= \varepsilon\text{ closure}(q_1)$

$= \{q_1q_2\}$

$\delta(q_0q_1q_2, c) = \varepsilon\text{ closure}(\delta(q_0q_1q_2, c))$

$= \varepsilon\text{ closure}(q_2)$

$= \{q_2\}$

$\delta(q_1q_2, a) = \varepsilon\text{ closure}(\delta(q_1q_2, a))$

$= \varepsilon\text{ closure}(\phi)$

$= \phi$

$\delta(q_1q_2, b) = \varepsilon\text{ closure}(\delta(q_1q_2, b))$

$= \varepsilon\text{ closure}(q_1)$

$= \{q_1q_2\}$

$\delta(q_1q_2, c) = \varepsilon\text{ closure}(\delta(q_1q_2, c))$

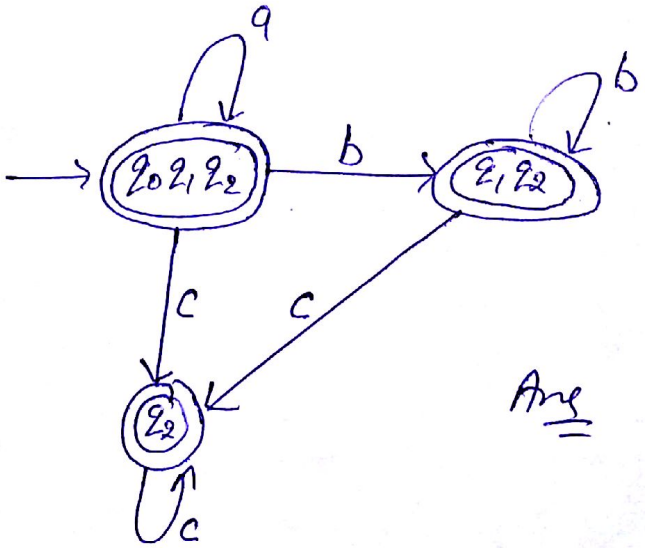
$= \varepsilon\text{ closure}(q_2)$

$= \{q_2\}$

$\delta(q_2, a) = \phi$

$\delta(q_2, b) = \phi$

$\delta(q_2, c) = q_2$



Ans

Comparision Method — (Testing Equivalence of Two DFA's)

Two DFA's M and M' are said to be equivalent if they accept the same language.

i.e $L(M) = L(M')$

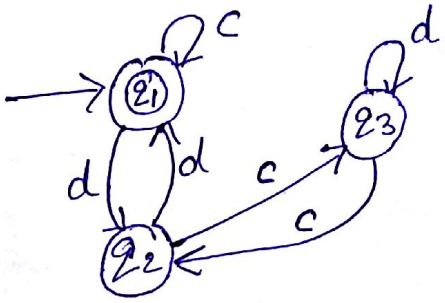
To Test the equivalence of Two DFA's M and M' we use the row wise constructor method as follows:

The rowwise construction is repeated. There are two cases

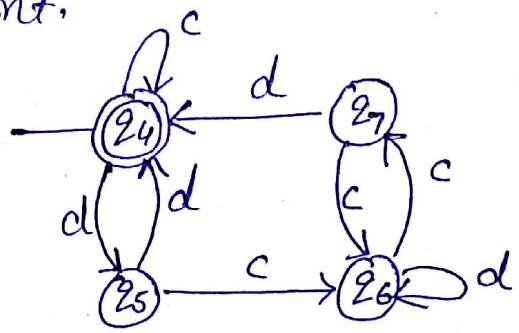
case 1: If we reach a pair (q, q') such that q is the final state of M and q' is non final state of M' or vice versa, we terminate the construction and conclude M and M' are not equivalent

case 2: Here the construction is terminated when no new element ~~is added~~ appears in the second and subsequent columns which are not in the first column. In this case we conclude that M and M' are equivalent.

e.g



M



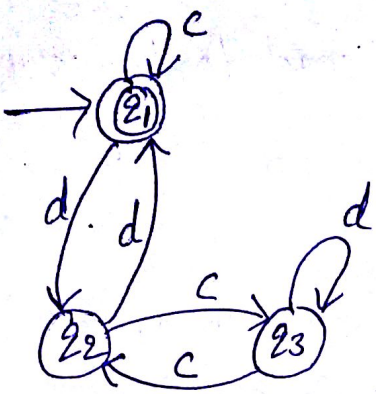
M'

Comparision Table

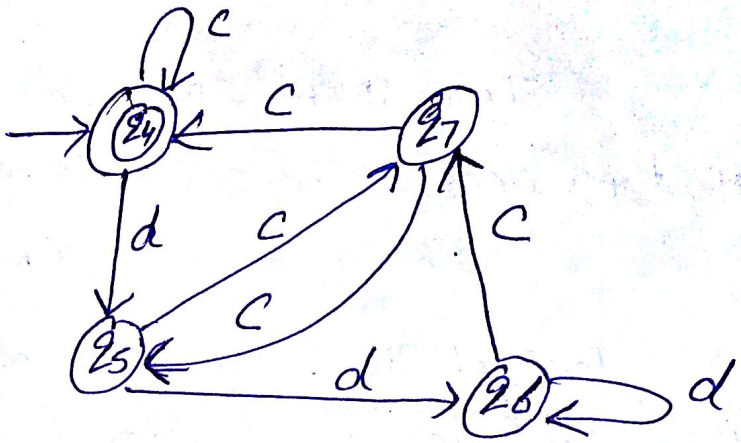
(q, q')	(q_c, q'_c)	(q_d, q'_d)
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

So $M = M'$

② show that M and M' are not equivalent



M



M'

Closure Properties of languages Accepted by DFA →

The class of languages accepted by DFA is closed under

a) Union b) Intersection c) Difference d) Compliment.

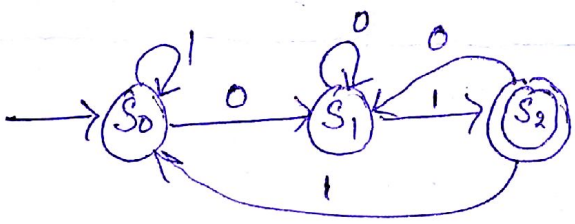
e.g. Suppose we have two m/c M_1 and M_2

M_1 : For accepting set of strings over $\Sigma = \{0,1\}$ ending in 01

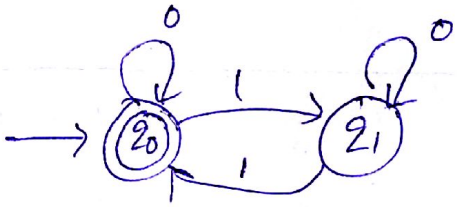
M_2 : For accepting set of strings over $\Sigma = \{0,1\}$ containing even no of 1's

Design DFA for following.

1) $L(M_1) \cup L(M_2)$
2) $L(M_1) \cap L(M_2)$
3) $L(M_1) - L(M_2)$
4) $L(M_1) - L(M_2)$



DFA for ending with 01



DFA for having even no of 1's.

Construction of Combined DFA M_c

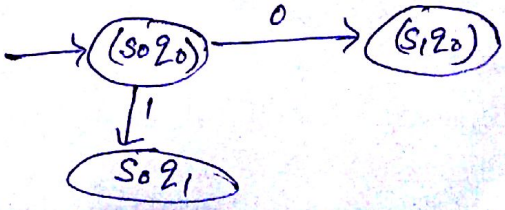
Step 1 We start with state (S_0, Q_0) where S_0 is initial state of M_1 and Q_0 is initial state of M_2.

$(S_0 Q_0)$ is expanded and necessary transitions added.

$\delta(S_0 Q_0, 0) = (\delta(S_0, 0)\ \delta(Q_0, 0))$

$= (S_1\ Q_0)$

$\delta(S_0 Q_0, 1) = (S_0\ Q_1)$



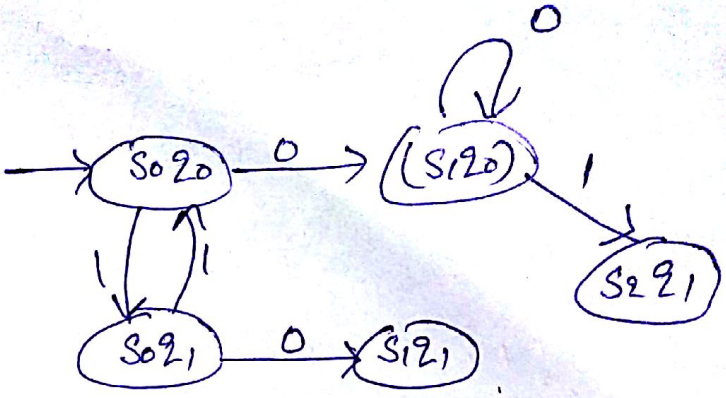
Step 2 state (S_1Q_0) and (S_0Q_1) are selected for expansion

$\delta(S_0Q_1, 0) = (S_1Q_1)$

$\delta(S_0Q_1, 1) = (S_0Q_0)$

$\delta(S_1Q_0, 0) = (S_1Q_0)$

$\delta(S_1Q_0, 1) = (S_2Q_1)$



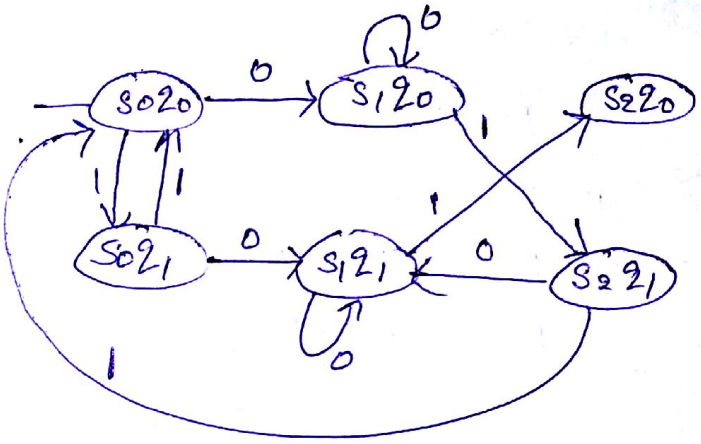
Step 3 state (S_1Q_1) and (S_2Q_1) are selected for expansion.

$\delta(S_1Q_1, 0) = (S_1Q_1)$

$\delta(S_1Q_1, 1) = (S_2Q_0)$

$\delta(S_2Q_1, 0) = (S_1Q_1)$

$\delta(S_2Q_1, 1) = (S_0Q_0)$

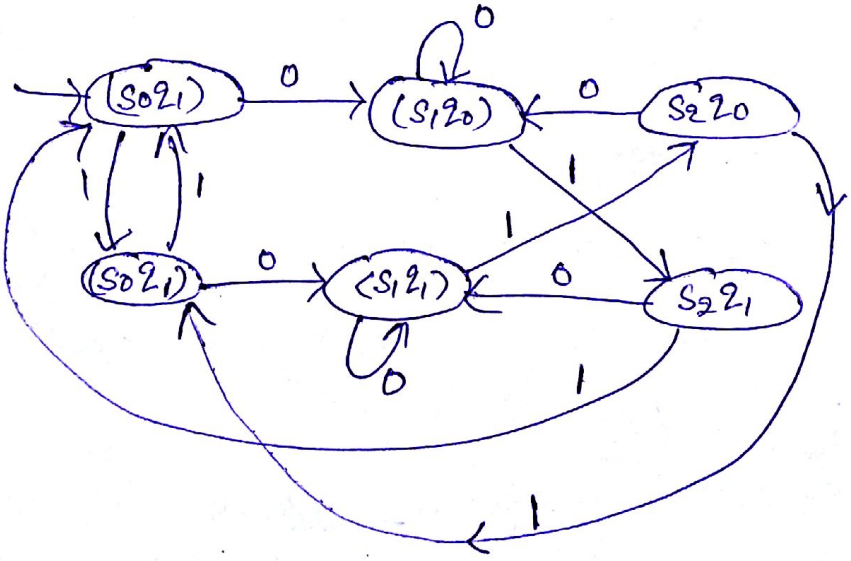


Step 4 state (S_2Q_0) is selected for expansion.

~~$\delta(S_2Q_0, 0) = (S_2Q_0)$~~

$\delta(S_2Q_0, 0) = (S_1Q_0)$

$\delta(S_2Q_0, 1) = (S_0Q_1)$



For Finding DFA for $L(M_1) \cap L(M_2)$:

The set of final state of the DFA is given by :

Every $(S_i\, q_j)$ such that $S_i \in$ final state of M_1 and $q_j \in$ final state of M_2

For Finding DFA for $L(M_1) \cup L(M_2)$:

The set of final state of required DFA is given by

Every $(S_i\, q_j)$ such that

$S_i \in$ Final state of M_1 OR $q_j \in$ Final state of M_2

For finding DFA $L(M_1) - L(M_2)$:

The set of final states of the required DFA is given by

every $(S_i\, q_j)$ such that $S_i \in$ final state of M_1 and $q_j \notin$ final state of M_2

Complimentation —

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an arbitrary DFA. To prove the closure property complementation, we must show that there

6. Design DFA which accepts all those strings of a's and b's in which number of a's is even and number of b's is divisible by 3