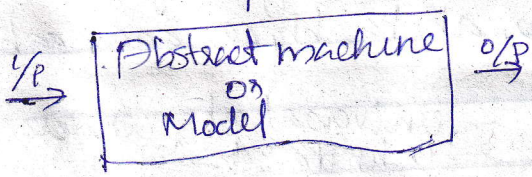
Theory of Computation

TOC helps to understand how CS is treated as a science. It tells what kind of things can really be computed mechanically.

Computation means processing that takes I/P & produces O/P. The basic operation is called Abstract machine or Model



The AM. or model is used to implement the algorithm efficiently. The low level implementation/abstract machine/model is called Automata.

Applications of TOC.

1) Design of Compilers. → Lexical Analyser
2) Pattern matching. Eg: Password.
3) Spell check

} % Valid or Invalid

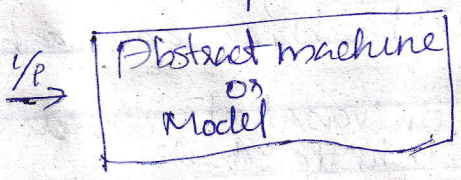
Automata Theory Covers 3 concepts.

- Automata used for develop a model. (or Algorithm)
- Computability (Whether the it is computable)
- Complexity (finds less complex solution)

Theory of Computation

TOC helps to understand how CS is treated as a science. It tells what kind of things can really be computed mechanically.

Computation means processing that takes I/P & produce O/P. The basic operation is called Abstract machine or Model



The AM. or model is used to implement the algorithm efficiently. The low level implementation/abstract machine/model is called Automata.

Applications of TOC.

1) Design of Compilers. → Lexical Analyser
2) Pattern matching. Eg: Password.
3) Spell check

} % Valid or Invalid

Automata Theory Covers 3 concepts.

* Automata used for develop a model. (or Algorithm)
* Computability (Whether the it is computable)
* Complexity (finds less complex solution)

Basic Concepts of Automata

Symbol ← a, b, c ..... , 0, 1, 2, 3 .....

Alphabets – Σ – collection of symbols.
Eg: $\Sigma = \{a, b\}$, $\Sigma = \{0, 1\}$.

String – Collection of symbols over alphabet
Eg: aab is a string over $\Sigma = \{a, b\}$.

Language – Collection of set of strings over alphabet
= { a, b, ab, ba, bb, aa, aaa, aab, ... }

Kleene star – Σ^* – universal set, universal language
– Infinite set.
$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots\ldots$
$\Sigma^0 = \in$ length of strings must be zero.
Null string / empty string
$\Sigma^1 = \{a, b\}$, $\Sigma^2 = \{ab, ba, aa, bb\}$...

Kleene closure – $\Sigma^+ = \Sigma^* - \Sigma^0$ or $\Sigma^* - \{\in\}$
$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots$

Eg: Write a language that accepts all strings with length '2' over $\Sigma = \{0, 1\}$.

L_2 = {00, 01, 10, 11}

If L_3 is considered then L_3 {000, 001, 010, 011, 100, 101, 110, 111}

So L_2 & L_3 are finite languages

L_4 – If all strings with length more than 2, then it is infinite language.

So all languages are the subset of Σ^*.

* w - represents a string.
  |w| - represents the length of string.

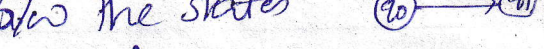
Finite Automata

A mathematical model or abstract machine build with finite number of states.

FA
|
DFA — Deterministic Finite Automata.
NFA — Non-deterministic F.A.

A finite automata is defined as a 5-tuple.

State — A circle with a state name — 

Transition — A directed edge b/w the states 

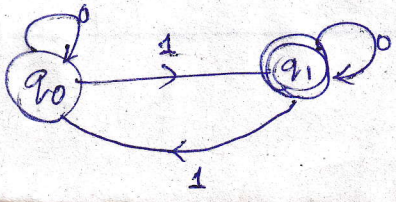
Initial state - state with an edge from no source.

Final state - A state represented as a double circle with state name - 

Alphabets — Σ

* FA ~~will~~ can have one initial state and multiple final states.
* FA is represented by
  - → Directed graph with states — Transition diagram
  - → Transition table.

Eg:



	0	1
q_0	q_0	q_1
q_1	q_1	q_0

FA – 5-tuple $(Q, \Sigma, \delta, q_0, F)$

Q – All states, Σ – i/p symbols.

δ – transition function

q_0 – initial state, F – set of final states

Transition function $\delta \rightarrow Q \times \Sigma \rightarrow Q$

DFA

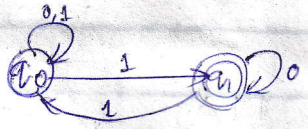
q_0 – 0 – q_0

q_0 – 1 – q_1

q_1 – 0 – q_1

q_1 – 1 – q_0

NFA ($\delta \rightarrow Q \times \Sigma \rightarrow 2^Q$ power set of Q)



$q_0 \rightarrow 0 \rightarrow q_1$

$q_0 \rightarrow 1 \rightarrow \{q_0, q_1\}$

DFA	NFA
① DFA = $(Q, \Sigma, \delta, q_0, F)$ $\delta = \Sigma \times Q \to Q$	NFA = $(Q, \Sigma, \delta, q_0, F)$ $\delta = \Sigma \times Q \to 2^Q$ (power set)
② Transition leads to unique state	Transition may lead to multiple state
③ More space required (many transitions)	Requires less space.
④ From every state every i/p must have a transition.	It is not mandatory.
⑤ Empty string transition (ϵ-move) are not supported	ϵ-transitions allowed.
⑥ Practical implementation ~~of~~ is feasible.	Not feasible, (needs to convert to DFA (is complex)
⑦ Construction is difficult in some case.	Simple/easy construction

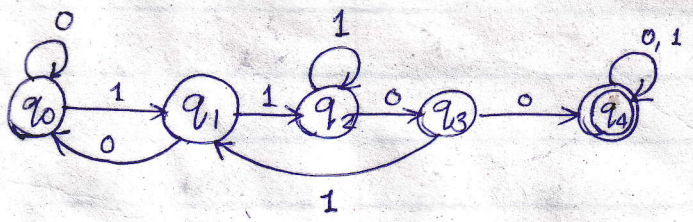
~~DFA~~

DFA

Exercises

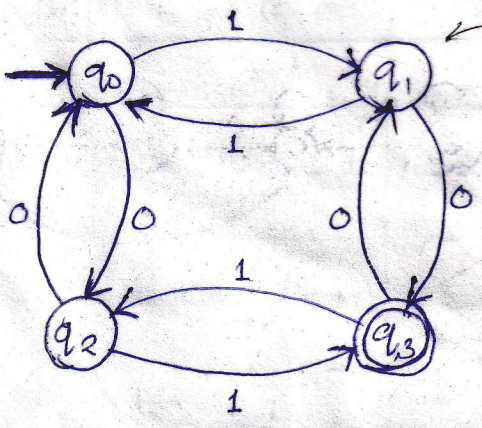
1. All strings contains 1100 as substring.

$(0+1)^* 1100 (0+1)^*$



2. (i) Odd no. of zeroes & odd no. of 1's

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



Eg: first: We can have 4 states odd 0 & odd 1 .... even 0, even 1 on reading 1 at odd 1 odd 0 we go to even 1 & odd zero so on...

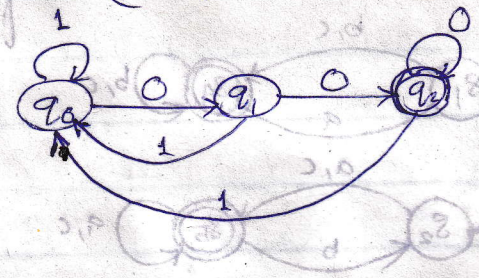
even 0's & even 1's — F.S — q_0

even 0's & odd 1's — F S — q_1

Odd 0's & even 1's — F S — q_2

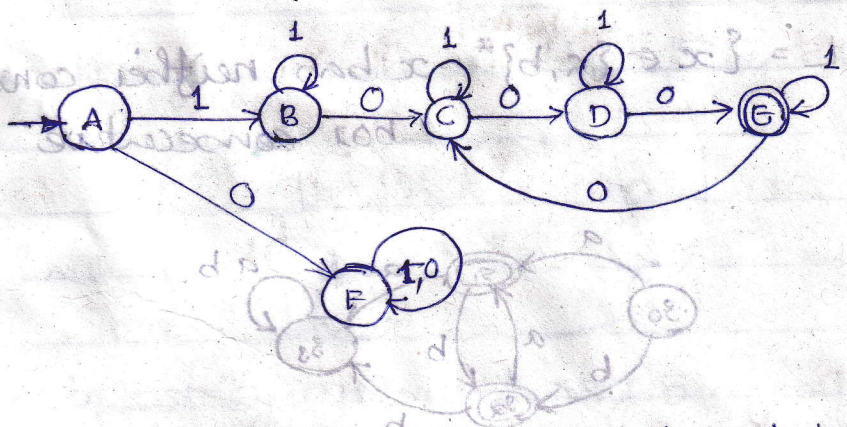
3) All strings ending with 00

$(0+1)^*00$

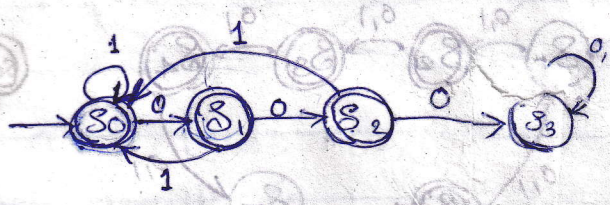


4) Set of all strings starting with 1 & no. of zeros is divisible by 3.

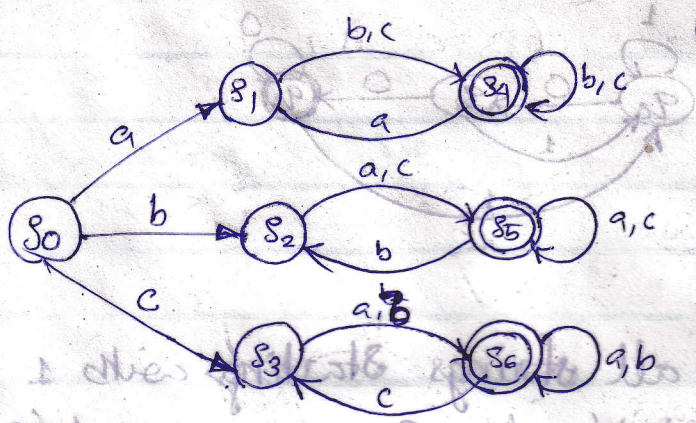
~~1 (000)~~ 1 $(01^*01^*01^*)$



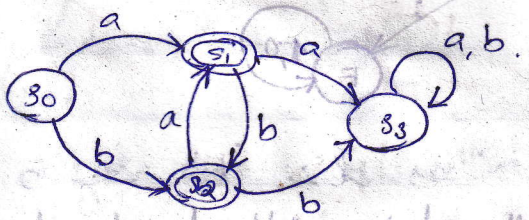
5) $\Sigma = \{0,1\}$ All strings that donot contain three consecutive 0's.



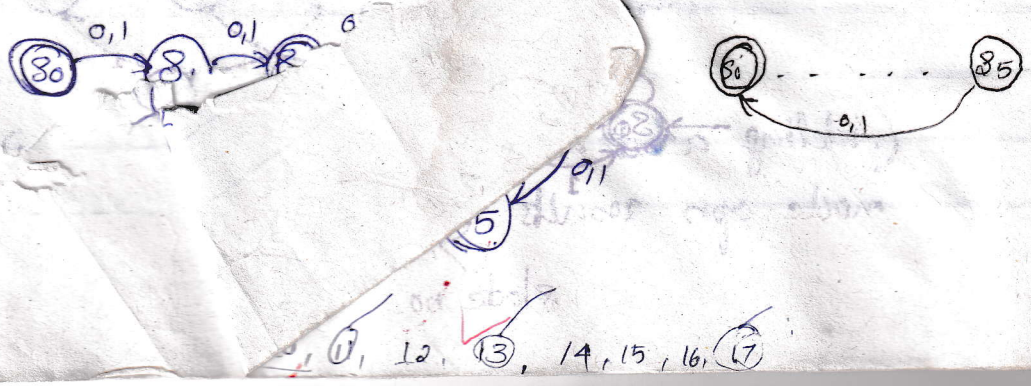
6] $\Sigma = \{a, b, c\}$ $L = \{x \in \{a, b, c\} \mid x$ begin and end with different symbols$\}$



7] $L = \{x \in \{a, b\}^* \mid x$ has neither consecutive a's nor consecutive b's$\}$

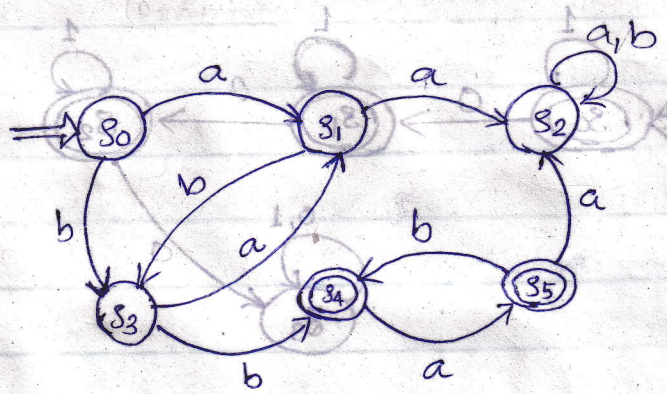


8] $L = \{x \in \{0, 1\} \mid |x|$ is multiple of 2 or 3.

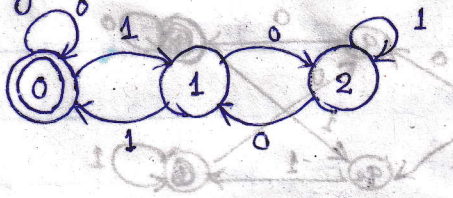


0, 1, 2, 3, 4, 5, 6, 7

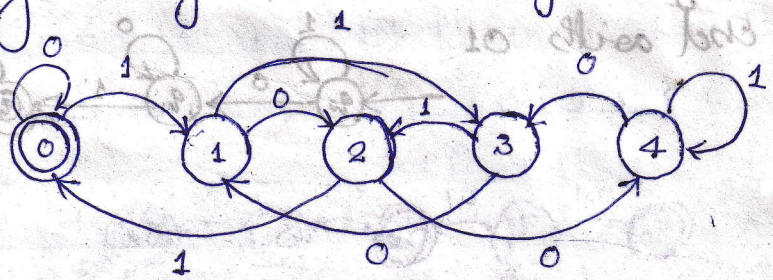
9] $L = \{x \in \{a,b\}^* :$ x contains atleast two consecutive b's and does not contain ~~two co~~ two consecutive a's$\}$.



10. Binary strings which are divisible by 3.



11. Binary strings divisible by '5'



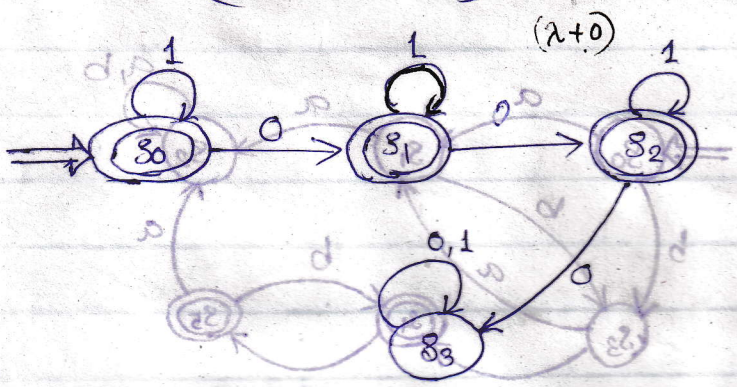
(Adding a bit at LSB means (Eg: 011 → 0111) $2n + a$ (added bit)

mode opn results $(2n) \bmod 3 + (a) \bmod 3$

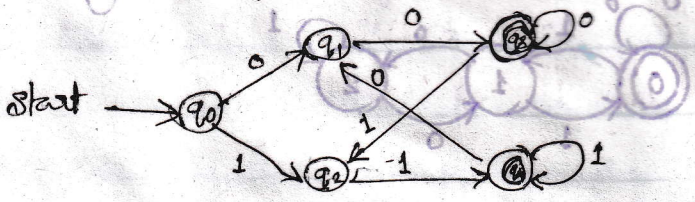
state no. added bit (0 or 1)

12. Strings containing atmost two consecutive zeroes.

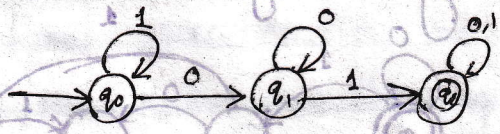
$$(1+01+001)^*(\lambda+0+00)$$



13. ~~Eg~~ DFA ~~Syntax~~ start with '00' or end with '11'



14. Design a DFA that accepts either start with 01 or end with 01



Extended Transition function.

Language of a DFA is a set of labels along all paths that lead from start state to any of accepting states.

Extended transition function describes what happens when we start in any state and follow any sequence of I/ps.

Denoted by $\hat{\delta}$

Basis: $\hat{\delta}(q, \epsilon) = q$

Induction: Suppose w is a string of form xa; 'a' is the last symbol of 'w' and 'x' is the string consisting of all symbols but the last symbol.

$$\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$$

~~$\hat{\delta}$ is a mapping from $S \times \Sigma^*$ into $P(S)$~~

Language of a DFA

DFA $A = (Q, \Sigma, \delta, q_0, F)$ the Language of A

$L(A) = \{ w / \hat{\delta}(q_0, w) \text{ is in } F \}$

→ If L is $L(A)$ for some DFA A, then we say 'L' is a regular language.

NFA

NON DETERMINISTIC FINITE AUTOMATA

It has the power to be in several states at once. NFA's accepts exactly regular languages as DFA's do.

1. In NFA, we can have several possible 'next states' for a given pair of current state and i/p symbol (ie Range of transition map is a subset of P(S).
2. A state can change into next state without having any i/p symbol.
3. A set $\delta(q,a)$ $a \in \Sigma$, $q \in Q$ may be empty. There may not be any transition defined on a specific pair of i/p symbol and current state.

An NFA 'A' is defined by

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

Q ← Finite set of states Σ - set of i/p symbols.
q_0 ← Member of Q; start state F ← subset of Q
* 'δ' ← A function that takes a state in 'q' and an i/p symbol in Σ and return a subset of 'Q'

Extended Transition function

Basis: $\hat{\delta}(q, \epsilon) = \{q\}$

Induction: Suppose w is a string of form $w = xa$ where 'a' is the final symbol and x is the rest in 'w'.

Also suppose that $\hat{\delta}(q, x) = \{p_1, p_2, \ldots, p_k\}$

Let $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \ldots, r_m\}$

Then $\hat{\delta}(q, w) = \{r_1, r_2, \ldots, r_m\}$.

Language of an NFA

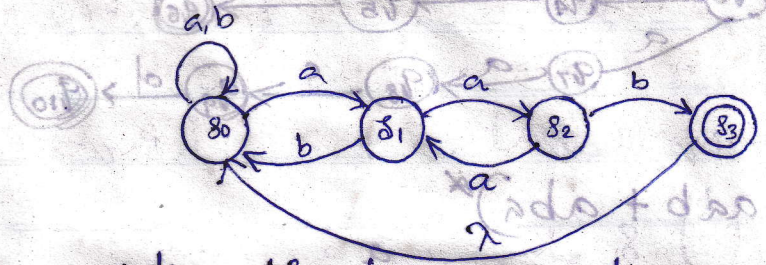
An NFA can make any choice of next state on reading a character of 'w'. Even if any one of the choice of states leads w to an accepting state; then w is a language of NFA.

Let $A = (Q, \Sigma, \delta, q_0, F)$ then

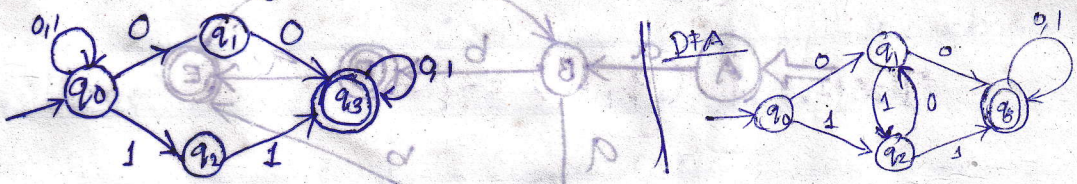
$$L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

1) Construct an NFA for language L

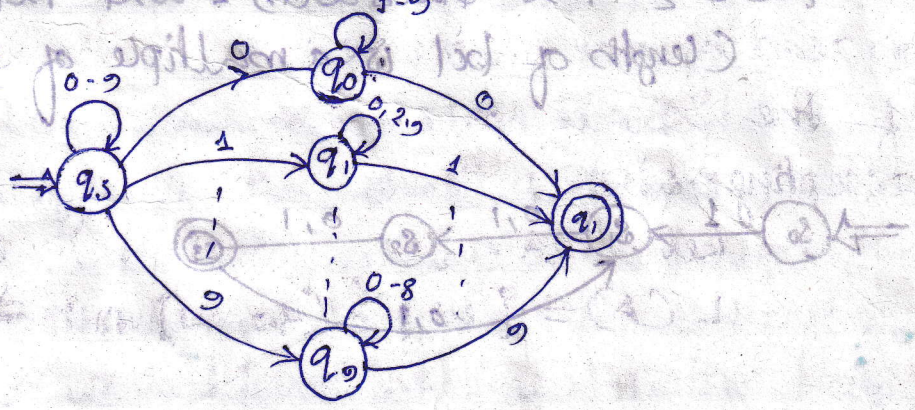
$L = \{x \in \{a,b\}^* \mid x \text{ ends with } aab\}$



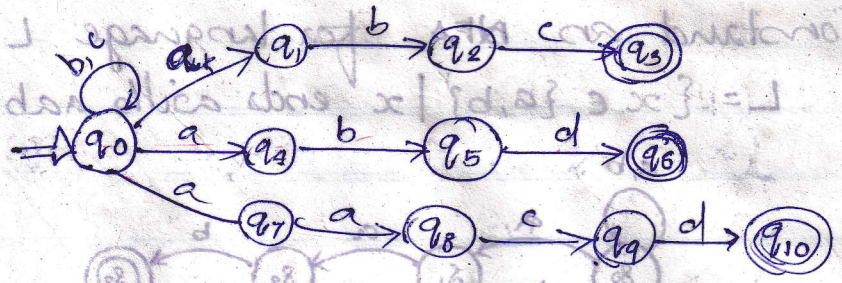
2) NFA with either two consecutive zeroes or 1's.



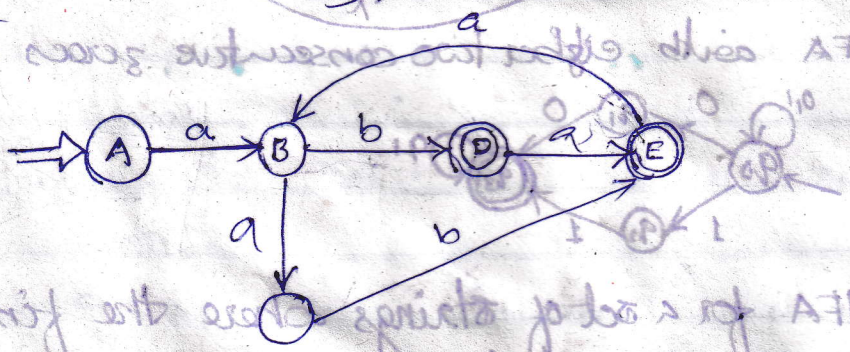
3) NFA for a set of strings where the final digit has appeared before. $\Sigma = \{0 \ldots 9\}$.



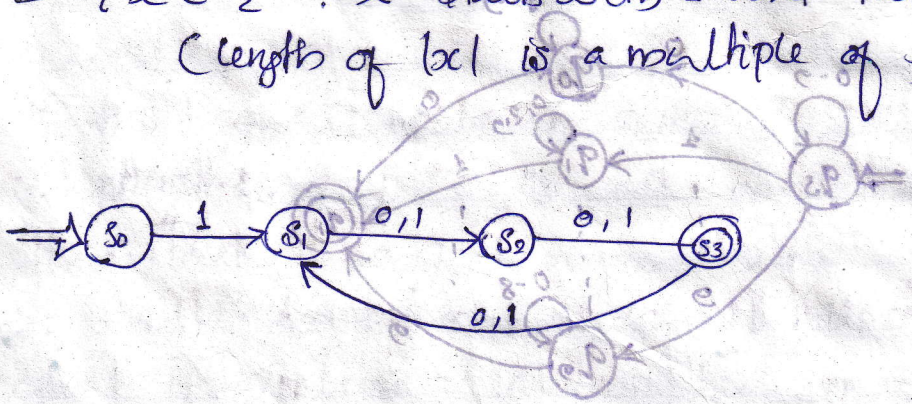
④ NFA that recognizes the following strings
abc, abd, aacd $\Sigma = \{a, b, c, d\}$



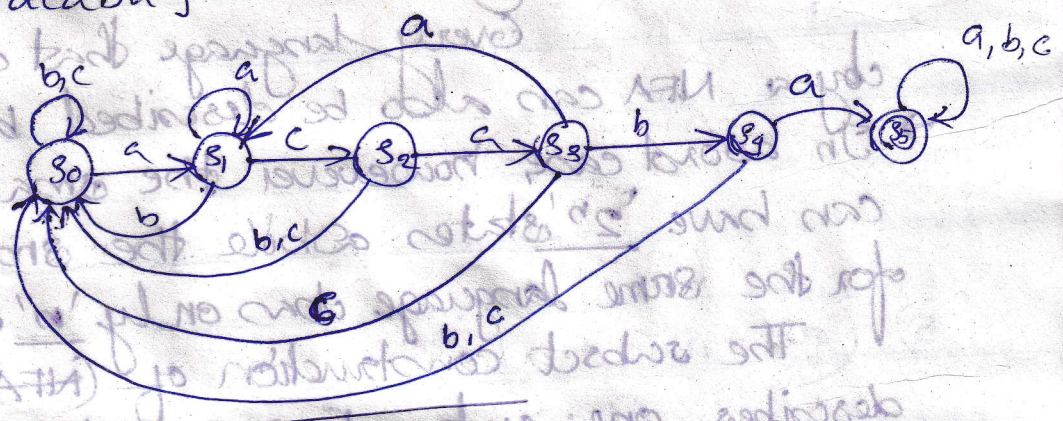
⑤ $(ab + aab + aba)^*$



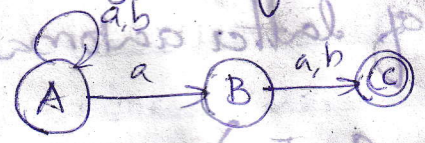
⑥ $L = \{x \in \Sigma^* : x \text{ starts with } 1 \text{ and } |x| = 0 \bmod 3\}$
(length of $|x|$ is a multiple of 3)



7) $L = \{x \in \{a,b,c\}^* : x$ contains substring acaba$\}$



8) Second symbol from last is 'a'

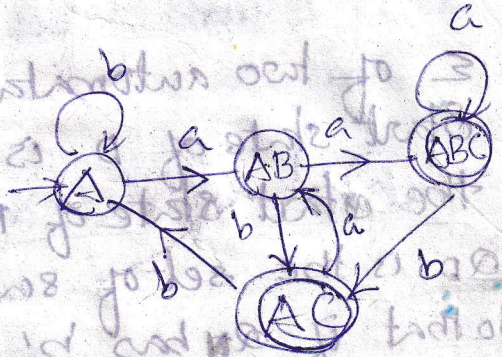


Transition table of NFA

	a	b
→A	{A,B}	{A}
B	{C}	{C}
C	{}	{}

converting to DFA

	a	b
→A	[A,B]	[A]
[A,B]	[A,B,C]	[A,C]
[A,C]	[A,B]	[A]
[A,B,C]	[A,B,C]	[A,C]



Equivalence of DFA and NFA.

Every language that can be described by a NFA can also be described by some DFA

→ In worst case however the smallest DFA can have 2^n states while the smallest NFA for the same language does only 'n' states

The subset construction of (~~NFA~~) states describes one automata (DFA) in terms of state and transmissions of other (NFA) without knowing the specifics of latter automata.

NFA = $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ It is used to describe a DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$

∴ $L(D) = L(N)$.

set containing only start state of N

→ Σ of two automata are same

→ Start state of D is the set containing only the start state of N.

→ Q_D is the set of subsets of N | Q_D is powerset of so that if Q_N has 'n' states then Q_D will have 2^n states. [of these 2^n states, a large no. of inacc states may be there.]

→ F_D is the sets of subsets S of Q_N such that $S \cap F_N \neq \emptyset$

ie F_D is all sets of N's states that include atleast one accepting state of N.

→ For each set $S \subseteq Q_N$ and for each i/p symbol a in Σ,

$$\delta_D(S,a) = \bigcup_{p \text{ in } S} \delta_N(p,a)$$

To compute $\delta_D(S,a)$ we look at all the states p in S and see what states N goes from p on input a, and take the union of all those states.

Theorem: If $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ is a DFA constructed from NFA $N = (Q_N, \Sigma, \delta_N, q_0$ then $L(D) = L(N)$

Proof: By induction on $|w|$, we have to prove $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$

Basis: Let $|w| = 0$ ie $w = \epsilon$, By basis definition of $\hat{\delta}$ for DFA's & NFA's both $\hat{\delta}_D(\{q_0\}, \epsilon)$ and $\hat{\delta}_N(q_0, \epsilon)$ are $\{q_0\}$.

Induction: Let w be of length $n+1$ and assume the statement for lengths 'n'. Break 'w' as $w = xa$, where 'a' is the final symbol of w.

By inductive hypothesis,

$$\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x)$$

Let both these sets of N's states be $\{P_1, P_2 \ldots P_k\}$

By inductive part of the definition of $\hat{\delta}$ of NFA

$$\hat{\delta}_N(q_0, w) = \bigcup_{i=1}^{k} \delta_N(P_i, a) \quad \text{——— (1)}$$

From the subset construction

$$\delta_D(\{P_1, P_2 \ldots P_k\}, a) = \bigcup_{i=1}^{k} \delta_N(P_i, a) \quad \text{——— (2)}$$

From – ② and the value $\hat{\delta}_D(\{q_0\}, x) = \{P_1, P_2, \ldots P_k\}$ is the inductive part of definition of $\hat{\delta}$ for DFA can be written as

$$\hat{\delta}_D(\{q_0\}, w) = \delta_D(\hat{\delta}_D(\{q_0\}, x), a)$$

$$= \delta_D(\{P_1, P_2 \ldots P_k\}, a)$$

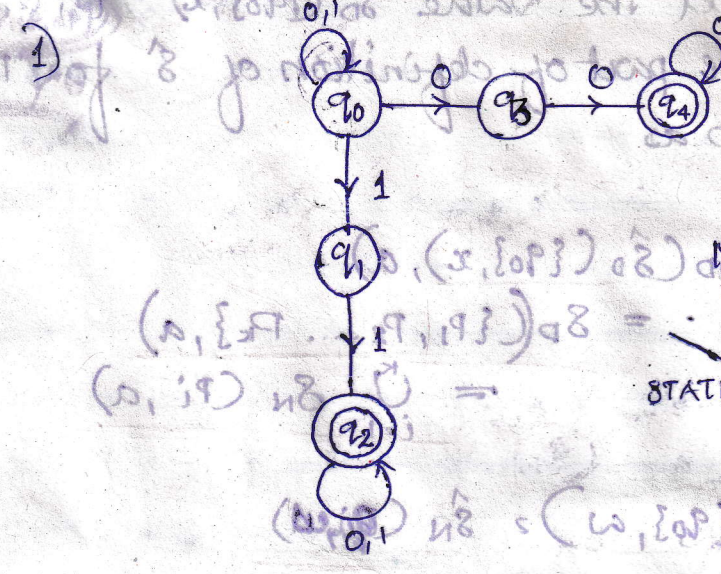
$$= \bigcup_{i=1}^{k} \delta_N(P_i, a)$$

So $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$

When we observe that D & N both accept w iff $\hat{\delta}_D(\{q_0\}, w)$ or $\hat{\delta}_N(q_0, w)$, respectively contain a state in F_N. Thus $L(D) = L(N)$

NFA To DFA

1)

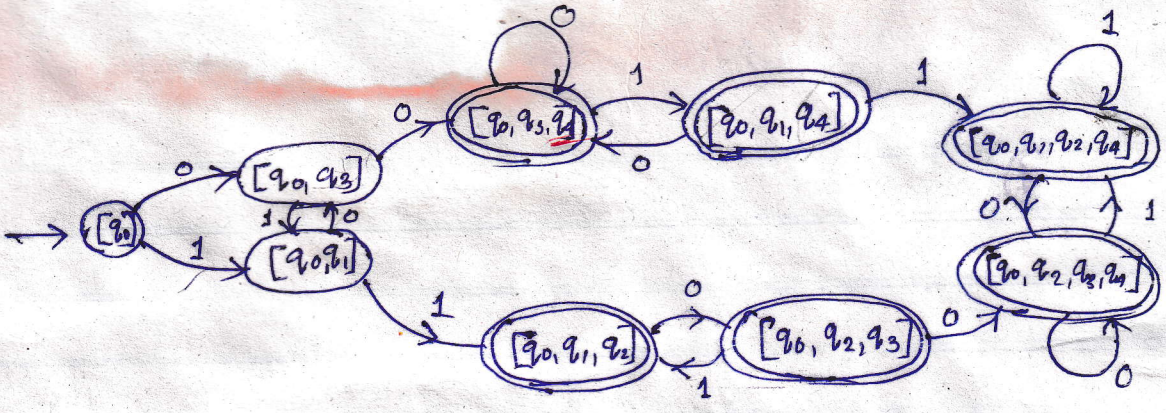


NFA transition table

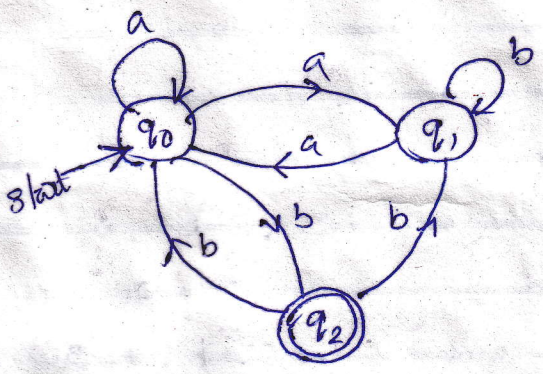
I/P / STATE	0	1
q_0	q_0, q_3	q_0, q_1
q_1	ϕ	q_2
q_2	q_2	q_2
q_3	q_4	ϕ
q_4	q_4	q_4

DFA transition table.

	0	1
$[q_0]$	$[q_0, q_3]$	$[q_0, q_1]$
$[q_0, q_3]$	$[q_0, q_3, q_4]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_3]$	$[q_0, q_1, q_2]$
$[q_0, q_3, q_4]$	$[q_0, q_3, q_4]$	$[q_0, q_1, q_4]$
$[q_0, q_1, q_2]$	$[q_0, q_3, q_2]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_4]$	$[q_0, q_3, q_4]$	$[q_0, q_1, q_2, q_4]$
$[q_0, q_2, q_3]$	$[q_0, q_2, q_3, q_4]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2, q_4]$	$[q_0, q_2, q_3, q_4]$	$[q_0, q_1, q_2, q_4]$
$[q_0, q_2, q_3, q_4]$	$[q_0, q_2, q_3, q_4]$	$[q_0, q_1, q_2, q_4]$



2)



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

$\delta[q_0, b] = \{q_2\}$

$\delta([q_0, q_1], a) = \delta(q_0, a) \cup \delta(q_1, a)$

$= \{q_0, q_1\} \cup \{q_0\}$

$= \{q_0, q_1\}$ //

FINITE AUTOMATA WITH ε-Transitions.

An NFA is allowed to make a transition spontaneously, without receiving an input symbol. Thatas called ε-transition.

This capability does not expand the classes of languages that can be accepted by a finite automata, but it gives some programming convenience. ε-NFAs are closely related to regular expression and useful in proving the equivalence b/w classes of languages accepted by F.A. & RE's.

ε-NFA

The representation of ε-NFA same as NFA but its transition function must include information about transitions on ε.

ε-NFA $A = (Q, \Sigma, \delta, q_0, F)$ with all components have same interpretation as for an NFA except δ has two arguments

1. a state in 'Q'

2. a member of $\Sigma \cup \{\epsilon\}$, that is, either an input symbol or the symbol ϵ.

Epsilon - Closure:

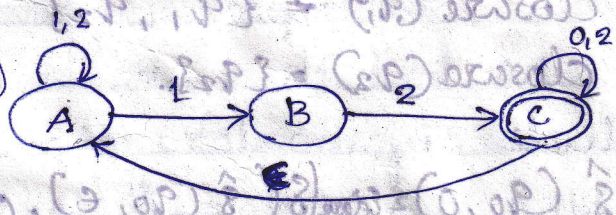
ϵ-closure of a state 'q' is found by following all the ~~states~~ transitions out of 'q' that are labelled ϵ. If we get to other states followed by ϵ, we follow the ϵ-transitions out of those states, and so on, eventually finding every state that can be reached from 'q' along any path whose arc labelled ϵ.

Reducing ϵ-NFA ~~to~~ to NFA without ϵ transitions

1. ϵ-closure
2. $\delta'(q,a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q), a))$
3. $F' = F \cup \{q_0\}$ if $\epsilon\text{-closure}(q_0)$ is having F as member.

NFA - $\{F, Q, \delta, \ldots\}$ | ϵ-NFA = $\{F', Q', \delta'\}$

$= F$ otherwise.



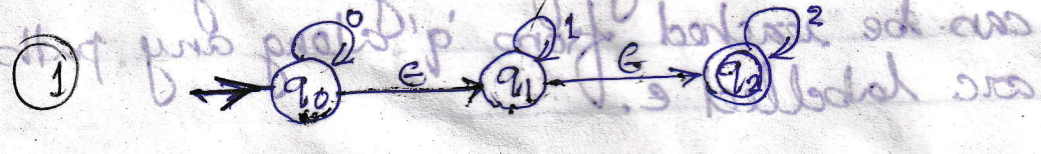
If 'ε' is to be removed, we must insert the transition to the states where we can reach with this ε- transition.

So first find all the states that we can reach with any i/p from state A.

Reduction logic: All the transitions from a particular state should be inserted to those states which are ε closed to that particular state.

So first find the ε-closure of a particular state and apply all i/ps to these ε-closed states of that particular states.

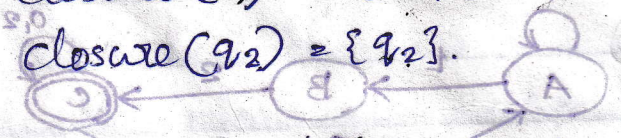
1] Convert it to NFA without ε-moves



ε-closure of q_0 = $\{q_0, q_1, q_2\}$

ε-closure (q_1) = $\{q_1, q_2\}$

ε-closure (q_2) = $\{q_2\}$.



$\hat{\delta}(q_0, 0) = \varepsilon\text{closure}(\delta(\hat{\delta}(q_0, \epsilon), 0))$

$\varepsilon\text{closure}(\delta(\{q_0, q_1, q_2\}, 0))$

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

$\hat{\delta}(q_0,1) = \epsilon\text{closure}(\delta(\hat{\delta}(q_0,\epsilon),1)$

$= \epsilon\text{closure}(\delta(\{q_0,q_1,q_2\},1)$

$= \epsilon\text{closure}(q_1) = \underline{\{q_1,q_2\}}$

$\hat{\delta}(q_0,2) = \epsilon\text{closure}(\delta(\hat{\delta}(q_0,\epsilon),2)$

$= \epsilon\text{closure}(\delta\{q_0,q_1,q_2\},2)$

$= \epsilon\text{closure}(q_2) = \underline{\{q_2\}}$

$\hat{\delta}(q_1,0) = \epsilon\text{closure}(\delta(\hat{\delta}(q_1,\epsilon),0)$

$= \epsilon\text{closure}($

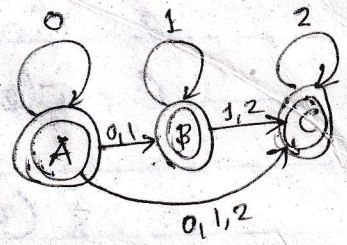
$= \phi$

$\hat{\delta}(q_1,1) =$

$= \{q_1,q_2\}$

$\hat{\delta}(q_1,2) =$

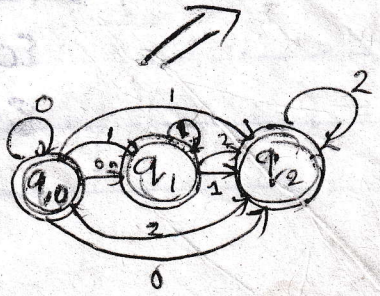
$= \{q_2\}$



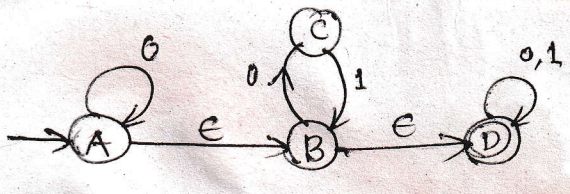
$\hat{\delta}(q_2,0) = q_2 = \{\phi\}$

$\hat{\delta}(q_2,1) = \phi$

$\hat{\delta}(q_2,2) = \{q_2\}$



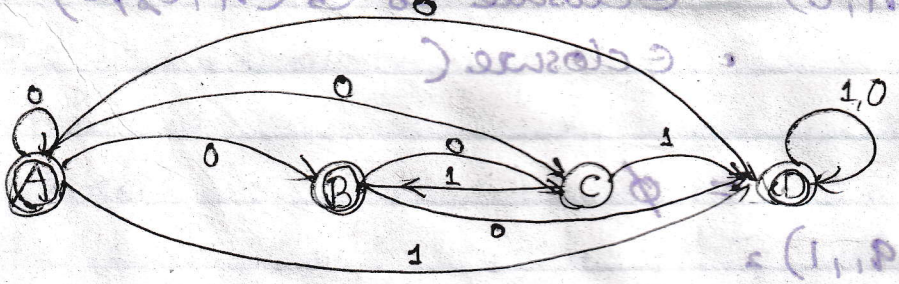
(2)

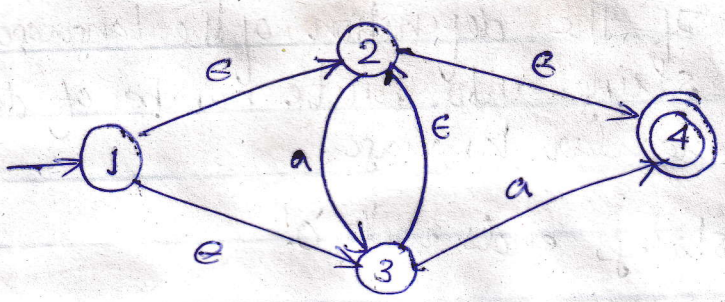


(* Both A & B are reachable to D only by ϵ. & all are final state)

Ans:

	0	1
→ A	{A, B, C, D}	{D}
B	{C, D}	{D}
C	{∅}	{B, D}
D	{D}	{D}



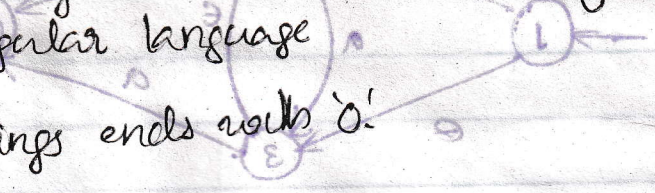

1
2
3
4
ε
ε
ε
ε
a
a

REDUCING NO. OF STATES IN DFA

MYHILL - NERODE'S THEOREM.

If the definition of the language L divides all strings into finite number of classes then L is a regular language

Eg: L = {all strings ends with 0}



two classes

strings ends with 0 | strings don't end with 0

If accepted L divides the string into 2 classes, The smallest finite Automata will have 2 states. Each state corresponds to each class.

$\equiv_L$ equivalence relation.

A language is regular iff $\equiv_L$ partition Σ^* into finite many equivalence classes

Relation: An ordered pair is a seq of two objects: (x, y)

A binary relation over two sets (A, B) is subset of A × B.

Let A = {Dave, Sara}, B = {cake, pie, nut}.

A × B = {(Dave, cake), (Dave, pie), (Dave, nut), (Sara, cake), (Sara, pie), (Sara, nut)}.

a relation "likes" is subset of A × B.

likes = {(dave, nuts), (sara, pie)}.

Let A = {a, b, c, d}.

Reflexivity: If a relation maps every element of set back on

$\forall x \in A \quad (x, x) \in R$. Eg: {(a,a), (b,b), (c,c), (d,d), (a,d), (b,a)}.

Symmetry: If an element $x \xrightarrow{\text{maps to}} y$ then we must also map y back on to x.

$\forall x, y \in A$, if $(x, y) \in R$ then $(y, x) \in R$

Transitive: $\forall x, y, z \in A$, If $(x, y) \in R$ & $(y, z) \in R$ then $(x, z) \in R$.

Equivalence relation

R is equivalence iff it is reflexive, Symmetric & transitive.

* Set of states can be divided into equivalence class. Say $\{1, 2, 3, 4, 5\}$ states can be divided into sets of equiv classes, as $\underbrace{\{1, 2, 5\}}_{\text{an equivalent class}} \{4\}, \{5\}$ like that...

Say equivalence relation of L $\quad \Sigma = \{a, b\}$

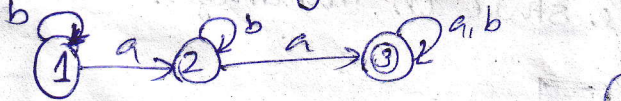
$L = \{w \in \Sigma^* : w \text{ has atleast two } a\text{'s}\}$.

hint: Equivalence classes are the set of all strings, no matter which string you have, if you put a particular i/p at the end of the string, They share the same feature such as part of L or no part of L.

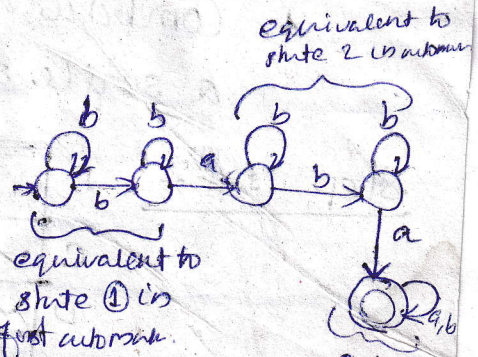
Eg: ①: $\{\epsilon, b, bb, bbb \ldots\}$ ← they all are those strings which required two a's to be in L.

at the same time ②: $\{a, ab, bab, bbba \ldots\}$ ← are equivalent as they are those which required a single 'a' to be in L

③: $\{aa, aab, bbaab, bbbaabbb \ldots\}$ ← strings which are already in L.



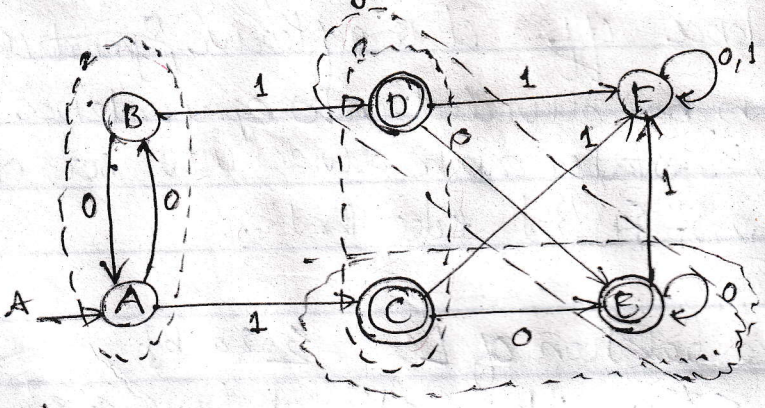
Suppose someone designed an automata



In the original automata class 1 and class 2 are not combinable. ① Because that feature are different

* ② An automata requires states equal to minimum no. of equivalence classes.

Minimization of DFA – Table Filling method (Myhill–Nerode Theorem)

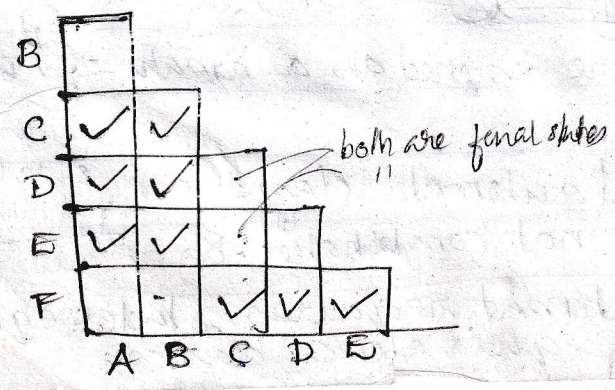


Steps

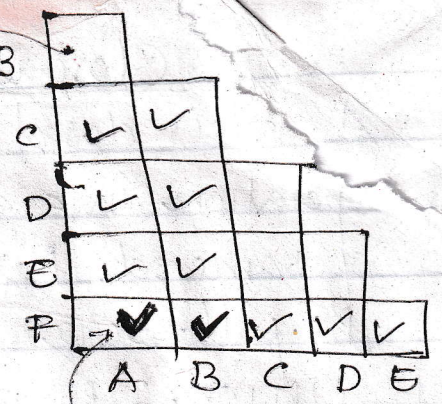
1. Draw table for all pairs of states (P, Q)
2. Mark all pairs (P, Q) where P ∈ F and Q ∉ F
3. If there are any unmarked pairs (P, Q) such that [δ(P, x), δ(Q, x)] is marked, then mark [P, Q] where x is i/p symbol.

   REPEAT THIS UNTIL NO MORE MARKINGS CAN BE MADE
4. Combine all unmarked pairs and make them a single state in minimized DFA.

Step 1 & 2:



step 3



(B,A): $\delta(B,0) = A$, $\delta(A,0) = B$ — not marked.
$\delta(B,1) = D$, $\delta(A,1) = C$

(D,C): $\delta(D,0) = E$, $\delta(C,0) = E$ — $\delta(D,1) = F$, $\delta(C,1) = F$

(E,C): $\delta(E,0) = E$, $\delta(C,0) = E$ — $\delta(E,1) = F$, $\delta(C,1) = F$

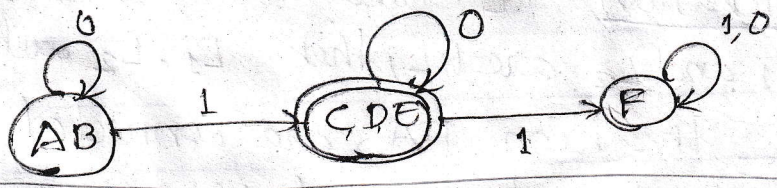
(E,D): $\delta(E,0) = E$, $\delta(D,0) = E$ — $\delta(E,1) = F$, $\delta(D,1) = F$ — ! MARKED so mark FA.

(F,A): $\delta(F,0) = F$, $\delta(A,0) = B$ — $\delta(F,1) = F$, $\delta(A,1) = C$

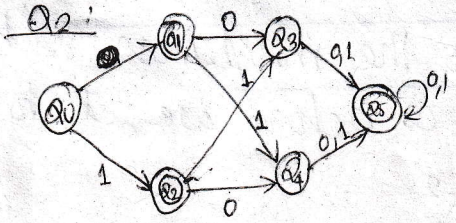
(F,B): $\delta(F,0) = F$, $\delta(B,0) = A$ — $\delta(F,1) = F$, $\delta(B,1) = D$

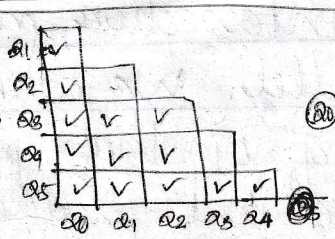
Repeat until no new markings can be made

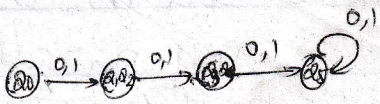
Unmarked pairs: (A,B), (C,D), (E,C), (E,D)



Q2:



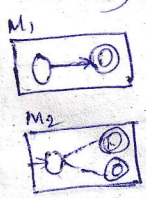




Closure Properties of Regular Languages.

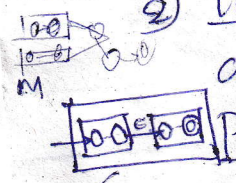
A set is closed under an operation if doing the operation on a given set always produces a member of same set. Regular languages are closed under the following opns.

1) Union: RL are closed under union. ie if L_1 and L_2 are RL, then $L_1 \cup L_2$ are also R.L



Proof: Since L_1 & L_2 are RL, there exists some F.A that describes them called M_1 & M_2. If we make 'M' from M_1 and M_2, the machine will accept 'w' if either M_1 or M_2 accepts 'w'. Since M simulates M_1 and M_2, M accepts the union of L_1 & L_2

2) Intersection: RL closed under intersection. If L_1 and L_2 are RL, then $L_1 \cap L_2$ will also be RL.



Proof: Just like union, construct M' that accepts by running M_1 & M_2 parallelly and accepts if both M_1 & M_2 accepts.

3) Concatenation: RL closed under concatenation. If L_1 and L_2 are RL, then $L_1 \cdot L_2$ will also be R.

Proof: In union, FAs are connected in parallel but in intersection machines M_1 & M_2 connected in series. We connect the machines such that all/any of L_1 strings produce before we get to machine that produces L_2.

(3.5) Reverse op: $L = L^R$ & L^R are RL

4) Kleene star: RL are closed under star operation. Because star is a repeated self concatenation, and we know that concatenation is closed.

5) Complement: R.L. are closed under complement. ie if L is RL then $\bar{L}$ is also RL.

Proof: change all accepting states of L to non-accepting states and vice-versa. The new machine will be accepting L's complement, ie $\bar{L}$ and $\bar{L}$ is a regular language.

6) Difference: $L_1 - L_2 = L_1 \cap \bar{L_2}$. Proof: Construct M such that the final states of M be the pairs, where M_1-state is final but M_2-state is not.

7) Homomorphism h(L) - substitution.

Converting one set to another, preserving properties.

$h(L) = \{h(w) / w \in L\}$ where $h: \Sigma$ to Γ^* is called homomorphism.

h(L): homomorphism of language L

h(w): homomorphic image of all strings in L

alphabets.

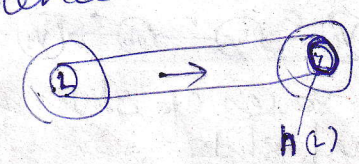
Eg: $\Sigma = \{0,1\}$ $\Gamma = \{a,b\}$

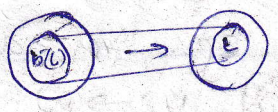
$h(0) = aa$ — homomorphic image of '0' is 'aa'. $h(1) = bb$

If $L\{00, 10\}$ then find $h(L) = ?$

$h(L) = \{aaaa, bbaabb\}$

Homomorphism is closed. ie R.L. is having a R.E. The homomorphic image of RE is another R.E. Hence RL closed under homomorphism. 'ε' is also like





Inverse homomorphism

Let $h(0) = a \quad h(1) = b \quad h(2) = ab$

$\Sigma = \{0, 1, 2\}$ (original) $\quad \Gamma = \{a, b\}$

$L = \{abaab\}$ has to be inversed

$h^{-1}(L) = \{0101, 22, 201, 012\}$

(1) $h(0) = aa, \quad h(1) = bb \quad \Sigma = \{0, 1\}, \quad \Gamma = \{a, b\}$

$h = \{aa, aabb, baaab, abaab\}$.

$h^{-1}(L) = \{0, 01\}$

Then $h(h^{-1}(L)) = \{aa, aabb\}$

ie $h(h^{-1}(L)) \subseteq L$

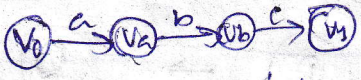
Equivalence of Regular grammar and DFA

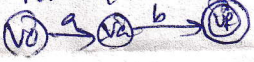
RG and DF are equivalent since we can conv

a R.L. or L.L, regular grammar to DFA & vice versa,

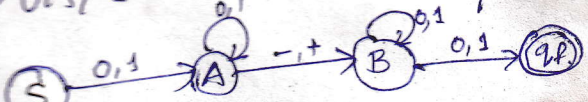
RG to DFA

$G = (V, T, P, S)$ can be converted to $M = (Q, \Sigma, \delta, q_0, F)$

1. Create a state for each vble
2. Convert each production rule into transition.
   (i) $V_i \to a V_j$, add $\delta(V_i, a) = V_j$
   (ii) $V_i = w V_j$, $(w \in T^*)$, create a series of
   states which derive 'w' and ends in V_j,
   eg: $V_0 \to abc V_1 \Rightarrow$ 

   (iii) $V_i \to w$ then create series of states which
   derive w and end on final state.
   eg: $V_0 \to ab$ 

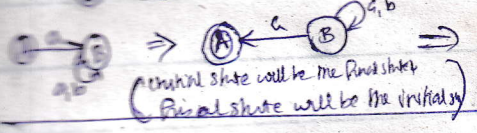
Eg: $S \to 0A/1A$ $\quad A \to 0A/1A/+B/-B \quad B \to 0B/1B/0/1$



~~Left linear grammar to~~ DFA to LL steps: FA $\xrightarrow{R\leftarrow \text{reverse}}$ FA $\to$ RLG $\xrightarrow{R\leftarrow \text{reverse}}$ LLG

(L) (L^R) (L^R) (L^R)^R

① Reverse the F.A ② Write the R.L Grammar ③ Rwerse R.L Grammar



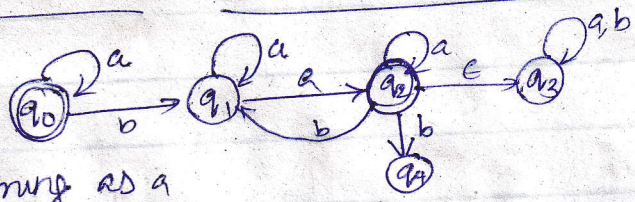
(original state will be the final state
final state will be the initial state)

$B \to aB/bB/aA \Rightarrow B \to Ba/Bb/Aa$

$A \to \epsilon$ (Since final state) $\quad A \to \epsilon$.

RG from F.A

①



Step 1: Give level number naming as a grammar vble. Eg: q_0 to S, q_1 to A, q_2 to B.... (avoid the trap states like q_3, q_4 in Example)

Step 2: Consider alphabets as terminals and Levels vbles as vbles

Convert $\delta(S,a) \to S$ as $S \to aS$

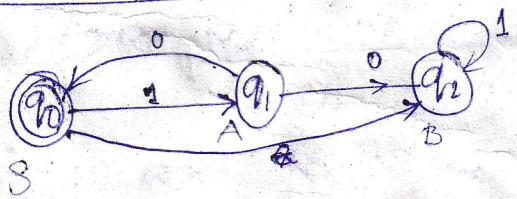
$\delta(S,b) \to A$ as $S \to bA$

If S is a final state, then give $S \to \epsilon$

$S \to aS|bA|\epsilon$ ~~$S \to bA$~~ $\quad A \to aA|aB$

$B \to aB|bA|\epsilon$

②



RLG.

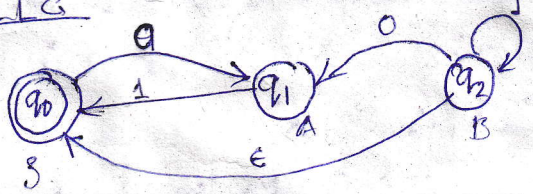
$S \to \epsilon/1A|B$

$A \to 0S/0B$

$B \to 1B$

LLG

Reverse



= Corresponding RLG.

$S \to \epsilon/1/0A$

$A \to 1S$

$B \to 1B/0A/S$

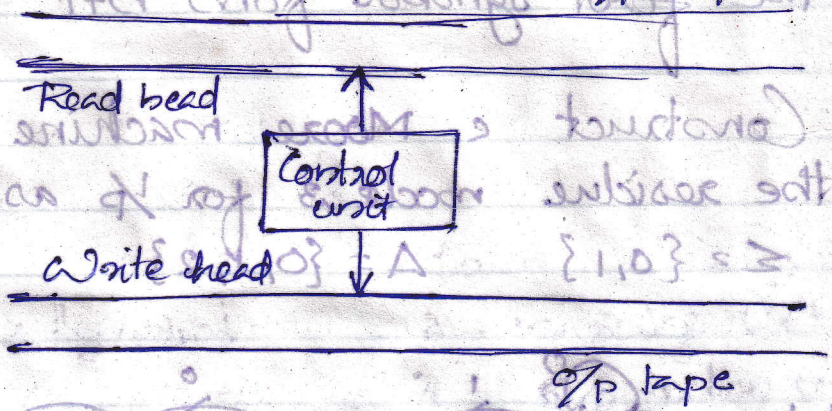
Convert to R.LG.

$S \to \epsilon/A0 \quad A \to S1$

$B \to B1/A0/S.$

FINITE AUTOMATA WITH O/P

→ It is an automata that accepts I/P string and produces an O/P string



It can model many kind of machines including vending machines, delay machines, binary adders etc

1. MOORE MACHINES

It is a machine with finite no. of states and for which the o/p symbol at 'q' depends only upon the present state of machine

A machine is a 6-tuple notation

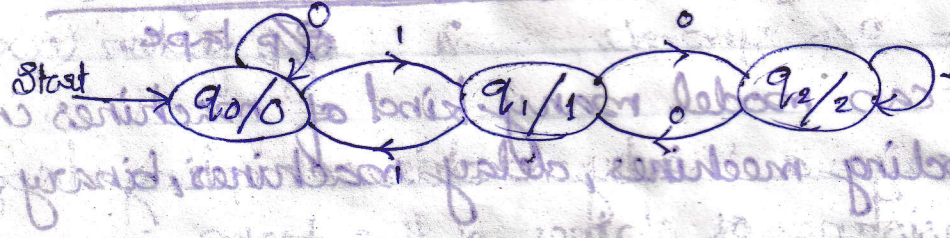
$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

Q - Finite set of states
Σ - I/P symbol
Δ - O/P alphabet
δ - mapping $(Q \times \Sigma \to Q)$
λ - O/P for mapping $(Q \to \Delta)$
q_0 - starting state.

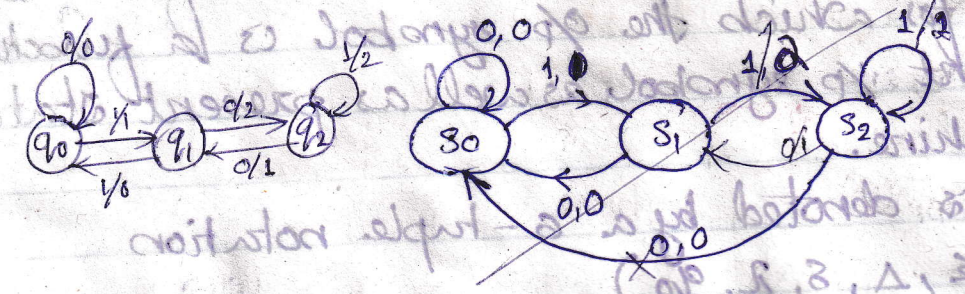
In Moore machine n+1 o/p symbols can be produced for n i/p symbols The actual o/p is the o/p with discarding the first symbol from n+1

② Construct a Moore machine to determine the residue mode 3 for i/p as binary no.s.

$\Sigma = \{0,1\}$ $\Delta = \{0,1,2\}$



b) Design a mealy machine to calculate mod 3



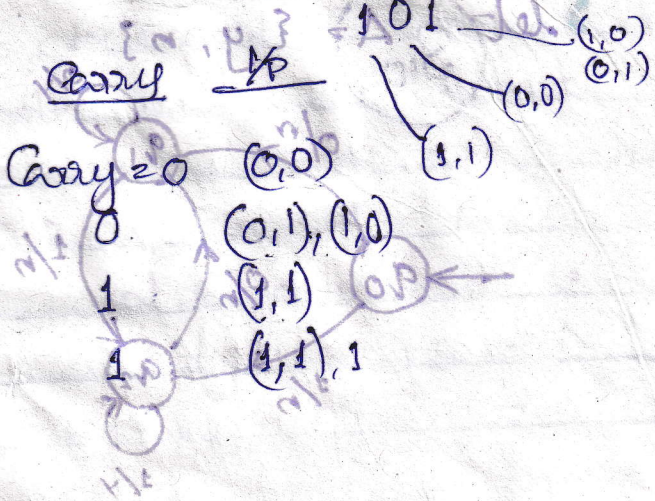
a) Design a Moore machine that will add 2 binary numbers.

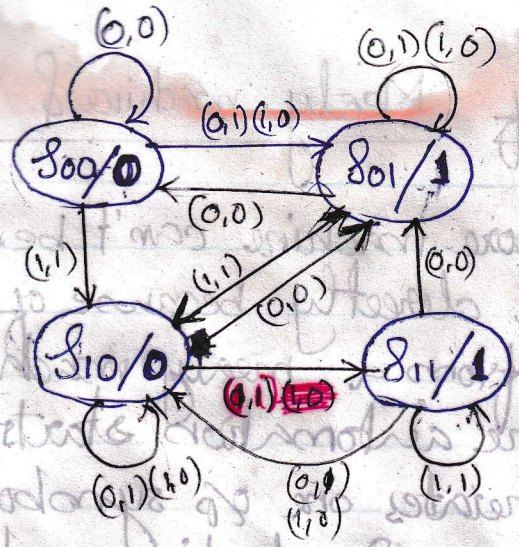
Solution:

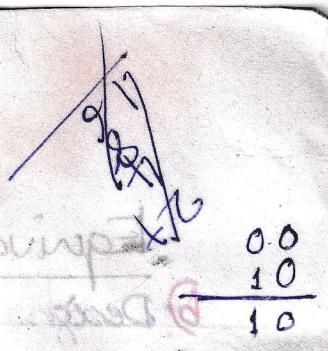
We can have 4 cases:

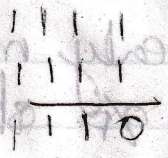
10101
001100
101

States	Sum	Carry	I/P
S_{00}	Sum = 0	Carry = 0	(0,0)
S_{01}	1	0	(0,1), (1,0)
S_{10}	0	1	(1,1)
S_{11}	1	1	(1,1), 1

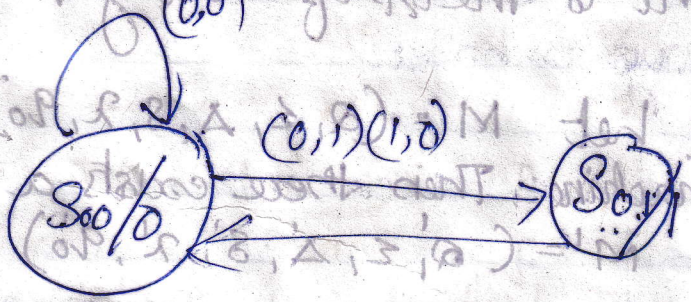


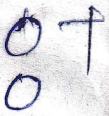


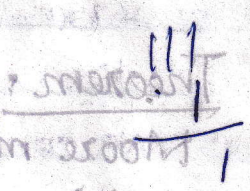


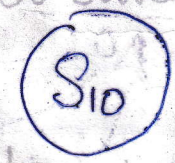


1010 +
1001
0 1 1









Equivalence of Mealy machine & Moore machine

Moore machine can't be equivalent Mealy machine directly because of the length of o/p string from a mealy machine. Moore machine automation starts from state q_0 and it generates one o/p symbol without depending on i/p. So starting symbol in the o/p moore machine is deleted to make it equivalent to the o/p of mealy machine

Theorem: Let $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is the Moore machine Then there exists a mealy machine $M' = (Q', \Sigma, \Delta', \delta', \lambda', q_0')$

Proof

Proof is based on the constructive algorithm

Define a mealy machine

$M' = (Q', \Sigma, \Delta', \delta', \lambda', q_0')$ where

$Q' = Q \quad \Delta' = \Delta, \quad q_0' = q_0$

The transitive map $\delta' : Q' \times \Sigma \to Q'$ as same as $\delta \; S \times \Sigma \to S$.

But the o/p map λ' from $Q' \times \Sigma \to \Delta'$ is defined by

$$\lambda'(q,a) = \lambda(\delta(q,a)) \quad \text{for all } a \in \Sigma \text{ and for all } q \in Q$$

Mealy

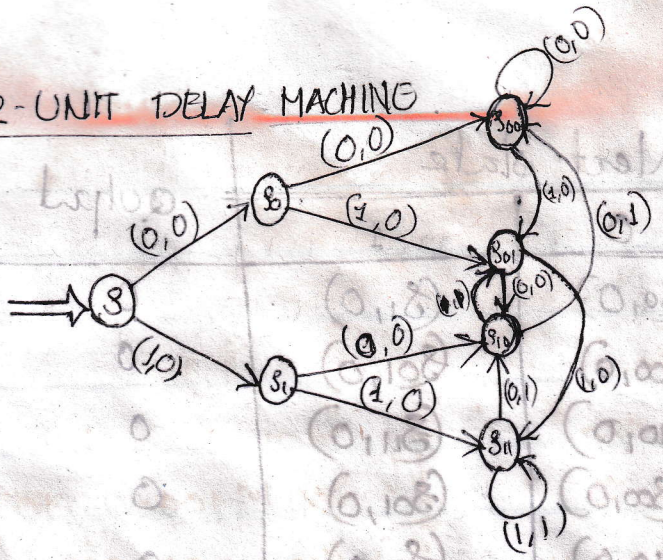
(* o/p equals the o/p of next state in moore

Theorem: If $M_1 = (Q^1, \Sigma, \Delta^1, q_0^1, \delta^1, \lambda^1)$ be a mealy machine, there exists an equivalent Moore Machine $M_2 = (Q^2, \Sigma, \Delta^2, q_0^2, \delta^2, \lambda^2)$ where

1. $Q^2 = Q^1 \times \Delta^1$
2. $\Delta^2 = \Delta^1$
3. $q_0^2 = (q_0, b)$ where 'b' is an arbitrary member of $\Delta^1 = \Delta^2$.
4. The transition map $\delta^2 = Q^2 \times \Sigma \to Q^2$ is defined by (one more step)
$\delta^2((q,c),a) = (\delta^1(q,a), \lambda^1(q,a))$ $\forall$ $q \in Q^1$ $c \in \Delta^1$ and $a \in \Sigma$
5. The output map $\lambda^2 : Q^2 \to \Delta^2$ is defined by
$\lambda^2((q,c)) = c$ $\forall$ $q \in Q^1$ and $c \in \Delta^1$

It can be checked by induction on the length of any arbitrary string from Σ^*, that M_2 is equivalent to M_1.

2-UNIT DELAY MACHINE



→ Produces same o/p but with delay of two i/p symbols

Eg: i/p : 0100110101 1
o/p : 00 0100110101 1

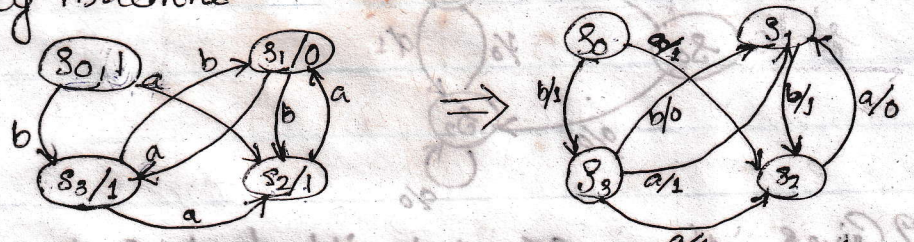
Transition table

Present state	next state / o/p	
	0	1
S	S_0 / 0	S_1 / 0
S_0	S_{00} / 0	S_{01} / 0
S_1	S_{10} / 0	S_{11} / 0
S_{00}	S_{00} / 0	S_{01} / 0
S_{01}	S_{10} / 0	S_{11} / 0
S_{10}	S_{00} / 1	S_{01} / 1
S_{11}	S_{10} / 1	S_{11} / 1

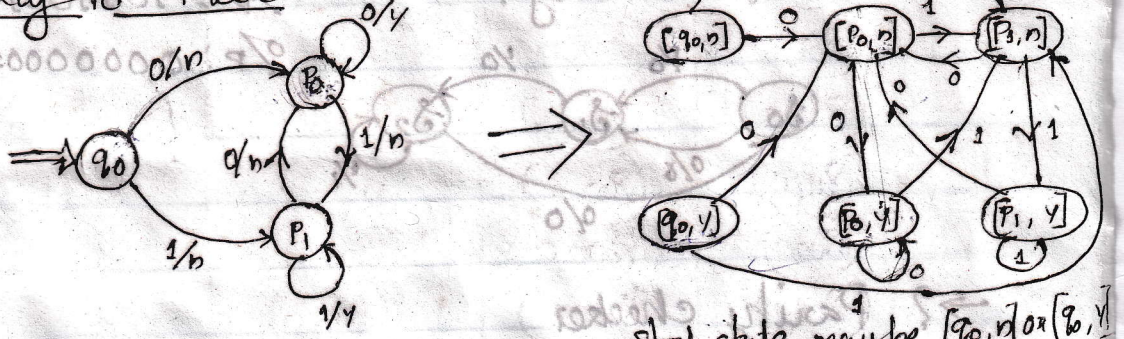
Present state	Next state		Output
	0	1	
⇒ $(S_0,0)$	$(S_0,0)$	$(S_1,0)$	0
$(S_0,0)$	$(S_{00},0)$	$(S_{01},0)$	0
$(S_1,0)$	$(S_{10},0)$	$(S_{11},0)$	0
$(S_{00},0)$	$(S_{00},0)$	$(S_{01},0)$	0
$(S_{01},0)$	$(S_{10},0)$	$(S_{11},0)$	0
$(S_{10},0)$	$(S_{00},1)$	$(S_{01},1)$	0
$(S_{11},0)$	$(S_{10},1)$	$(S_{11},1)$	0
$(S_{00},1)$	$(S_{00},0)$	$(S_{01},0)$	1
$(S_{00},1)$	$(S_{10},0)$	$(S_{11},0)$	1
$(S_{10},1)$	$(S_{00},1)$	$(S_{01},0)$	1
$(S_{11},1)$	$S_{10},1$	$(S_{11},0)$	1

Fig:

→ Convert the following Moore machine into it equivalent Mealy machine



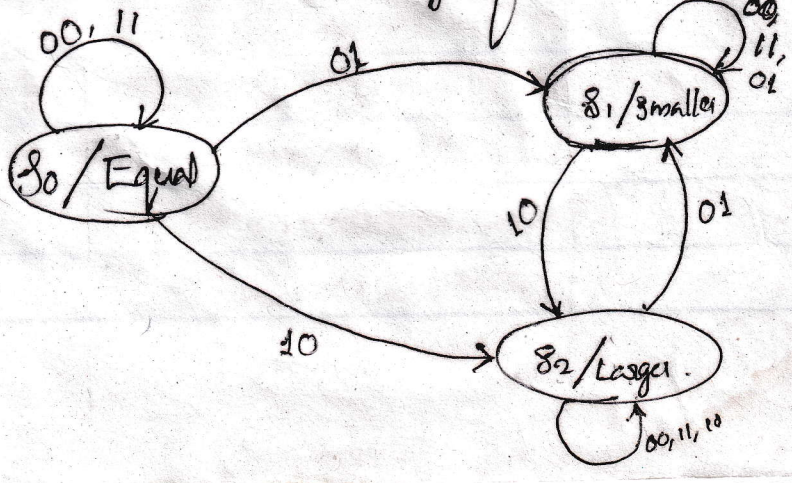
Mealy to Moore



Start state may be $[q_0, n]$ or $[q_0, y]$

→ Design a FSM that compares two binary no.s to determine whether they are equal and which of the two is larger.

Sol: Assume that digits of two no.s come one by one and ~~LSB~~ starting from LSB.



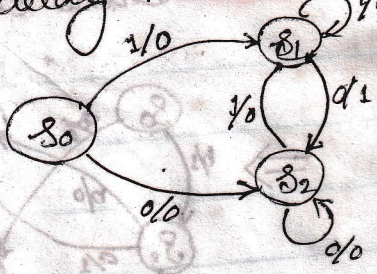
101101
101110

11
S_1 S_2

→ 2 bit delay machine

i/p : $a_1 a_2 a_3 \cdots a_k$

o/p : $0\ a_1\ a_2 \cdots a_{k-1}$

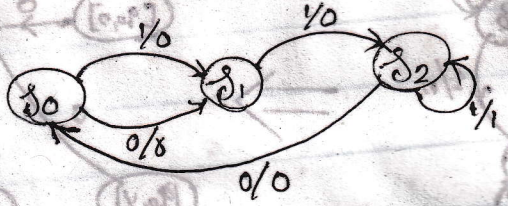


→ ? Gives '1' as o/p but iff last 3 i/p bits are all 1's

Eg :

i/p : 10110111011111

o/p : 00000001000011



→ ? Parity checker