Unit-2

Introduction to Finite Automata

Date:
Page:

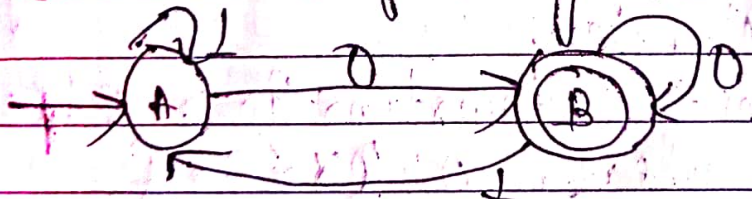
(*) Finite Automata / Finite State Machine (FSM) ⇒ Finite Automata (FA) is the simplest machine to recognize patterns. The finite state machine is an abstract machine which has five elements or tuple. It has a set of states and rules for moving from one state to another but it depends on applied input symbol. Basically, it is an abstract model of digital computer. A finite automata consists of five tuples: $(Q, \Sigma, q_0, F, \delta)$ where:

- $Q \Rightarrow$ finite set of states
- $\Sigma \Rightarrow$ set of input symbols.
- $q_0 \Rightarrow$ Initial state
- $F \Rightarrow$ set of final states
- $\delta \Rightarrow$ Transition function

Automata is also called automation.

(#) Example:

⇒ Let we have following FSM:



So, five tuples for this FSM can be described as,

$Q = \{A, B\}$

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

$q_0 = A$

$F = \{B\}$

$\delta = Q \times \Sigma$ which can be described by the table below

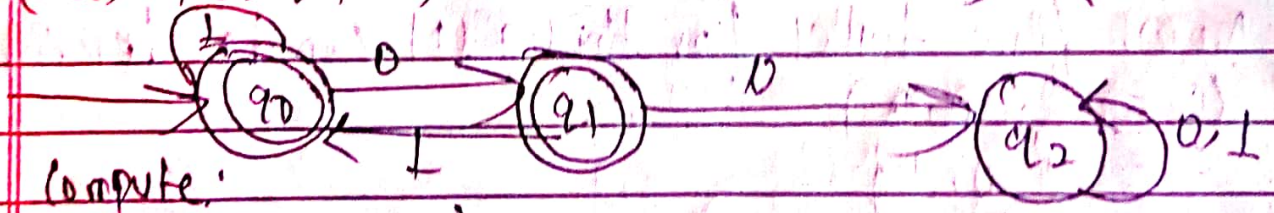
δ	0	1
A	B	A
B	B	A

Categories of FSM:
Finite State Machine
(Finite Automata)

- FA with output
  - Moore Machine
  - Mealy Machine
- FA without output
  - DFA
  - NFA
  - ε-NFA

(1) Deterministic Finite Automata (DFA) → It is the simplest model of computation which has every limited memory. In a DFA, for a particular input character the machine goes to one state only. In DFA null or ε move is not allowed i.e. DFA cannot change state without any input character.

Definition: A finite automation is a 5-tuple M = (Q, Σ, δ, q0, F) where, δ: Q × Σ → Q



Compute:

$\hat{\delta}(q_0, 1001)$

$= \delta(\hat{\delta}(q_0, 100), 1)$

$= \delta(\delta(\hat{\delta}(q_0, 10), 0), 1)$

$= \delta(\delta(\delta(\hat{\delta}(q_0, 1), 0), 0), 1)$

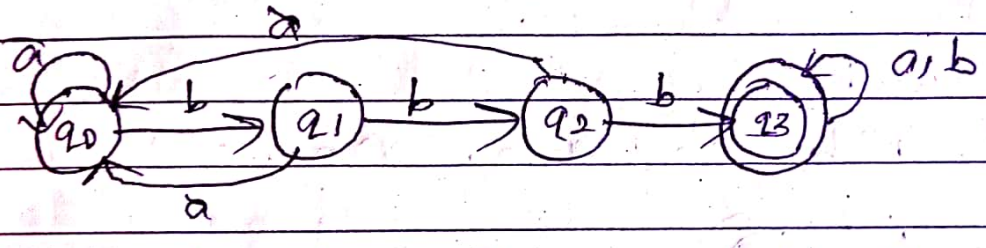
$= \delta(\delta(\delta(\delta(\hat{\delta}(q_0, \varepsilon), 1), 0), 0), 1)$

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

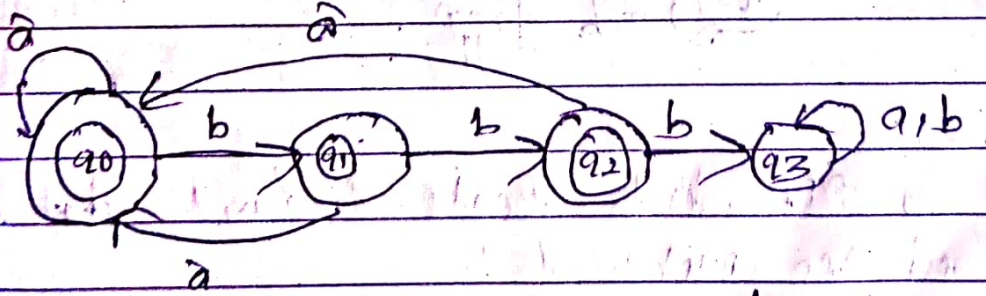
Date:
Page:

$= \delta(\delta(\delta(q_0, 0), 0), 1)$
$= \delta(\delta(q_1, 0), 1)$
$= \delta(q_2, 1)$
$= q_2$
Is not accepted #

(#) Construct DFA which accepts the language that doesnot contains 3 consecutive
⇒ Contains 3 cons. b's.
$\Sigma = \{a, b\}$
$L = \{bbb$

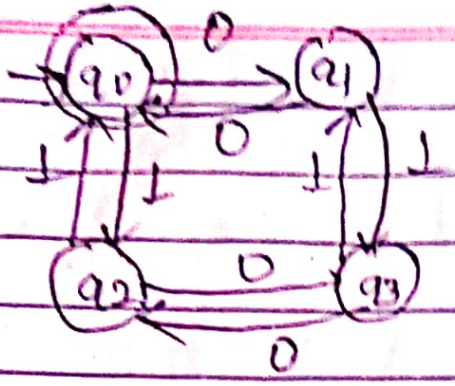


For Doesn't contain 3 cons. b's.
Final states ⇒ Non-final
Non-final ⇒ final states



(#) Construct DFA which accepts the $\Sigma = \{0, 1\}$ that has even no. of 0 and even no. of 1.
$L = \{0011, 0101,$

$a^n b^n$ $m, n > 0$
$a^0 b^m$



Homework

(i) Odd number of 0 & even no. 1

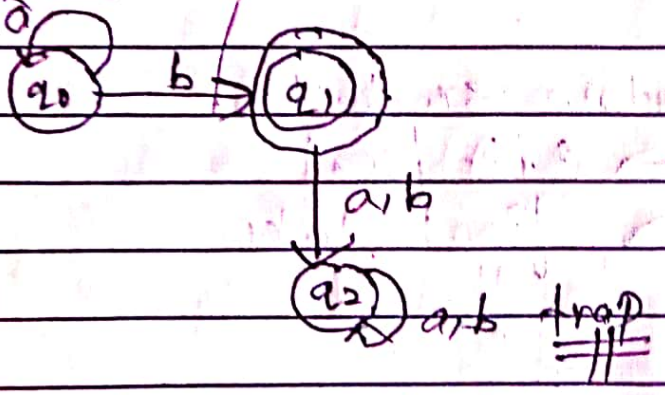
(ii) even number of 0 & odd no. 1

(iii) odd number of 0 & odd no. 1

(iv) Construct DFA which accepts the $L = a^n b$. $(n > 0)$

⇒ $\Sigma = \{a, b\}$

$L = \{ a^0 b, ab, a^2 b, a^3 b, \ldots \}$



Homework:

(i) Construct a DFA which accepts $\Sigma = \{0, 1\}$ that has no. of 0 and even number of 1.

⇒ Here,

$L = \{011, 0101, 0\ldots\ldots, 00011\ldots\ldots\}$

Date:
Page:

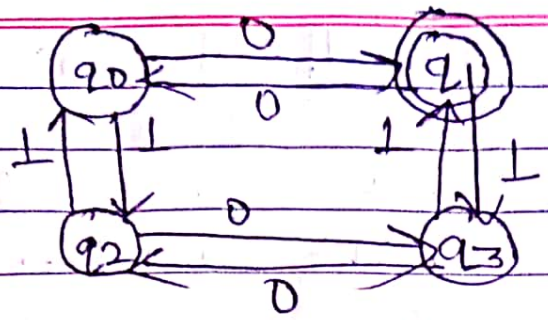
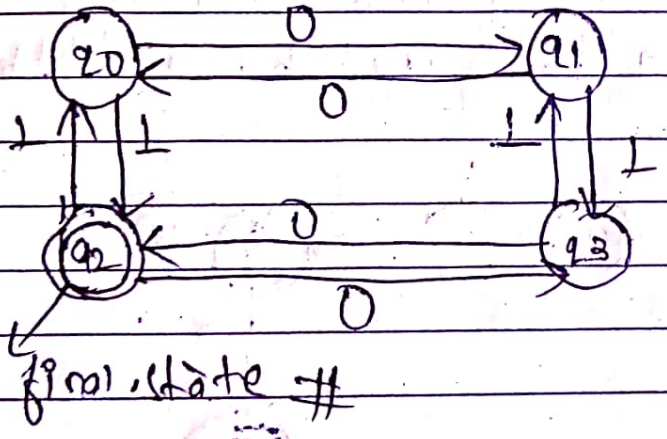


fig: DFA which accepts odd number of 0's and even number of 1. where, q_1 is final state.

(ii) Construct a DFA which accepts $\Sigma = \{0, 1\}$ that has even number of 0 and odd number of 1.

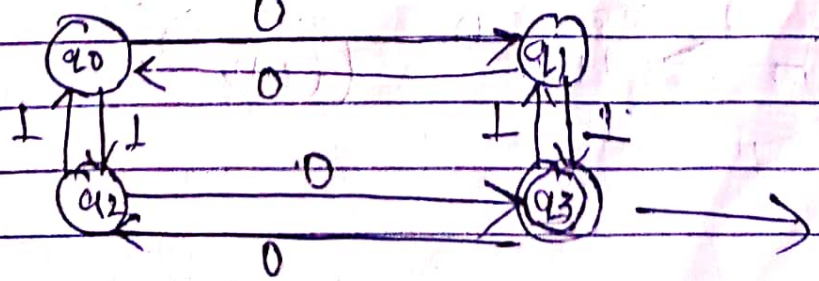
⇒ $L = \{001, 010, 00001, 00001111 \ldots\}$



(iii) Construct a DFA which accepts $\Sigma = \{0, 1\}$ that has odd number 0 & odd no. 1.

⇒ $\Sigma = \{0, 1\}$

$L = \{01, 000111, 0111, 10, 1110 \ldots\}$

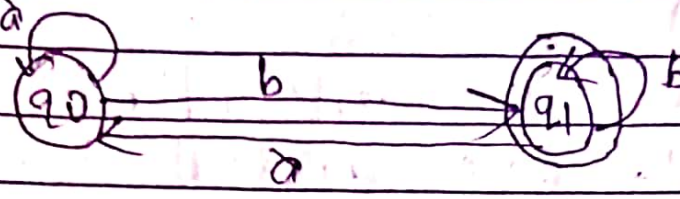


(#) Construct DFA which accepts the $L = a^n b^n$, $n \geq 0$

⇒ Here,

$L = \{ a^1b^1, a^2b^2, a^3b^3, \ldots \}$

$= \{ ab, aabb, aaabbb, aaaabbbb \}$



Another question

(#) Construct DFA which accepts the $L = a^2 b^m$, $m \geq 0$

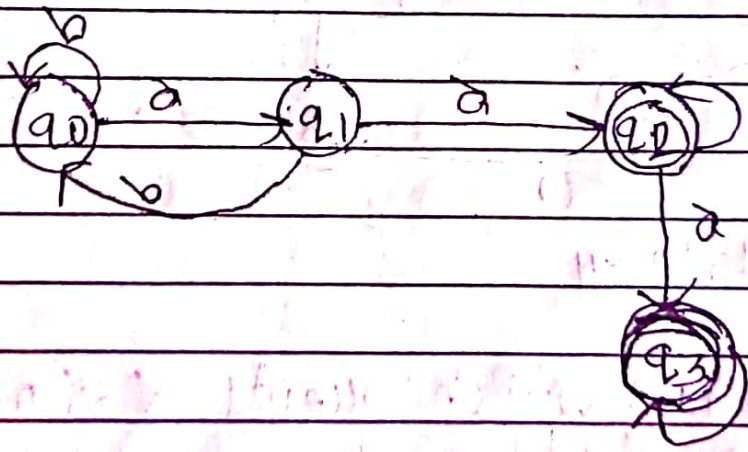
⇒ Here,

$\Sigma = \{a, b\}$

$L = \{ a^2b^1, a^2b^2, a^2b^3, a^2b^4 \ldots \}$

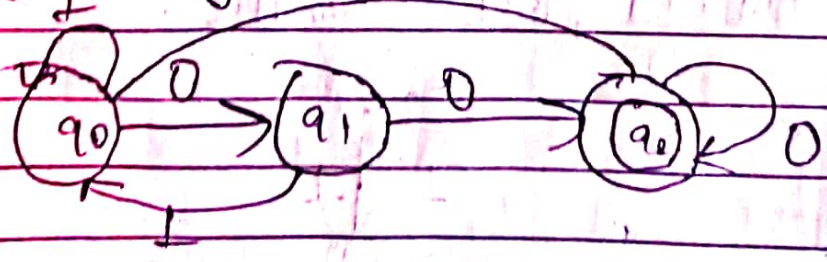
$= \{ aab, aabb, aabbb, aabbbb \ldots \}$

Now,



Classwork

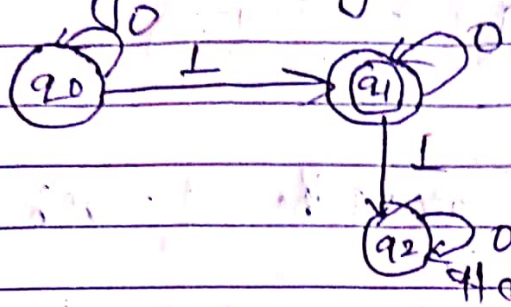
(#) Every string end in 00



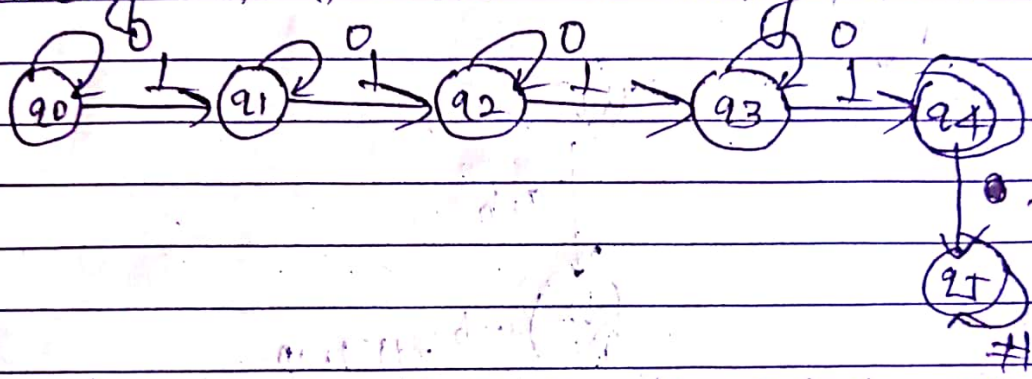
$b^m a b^n$

Date:
Page:

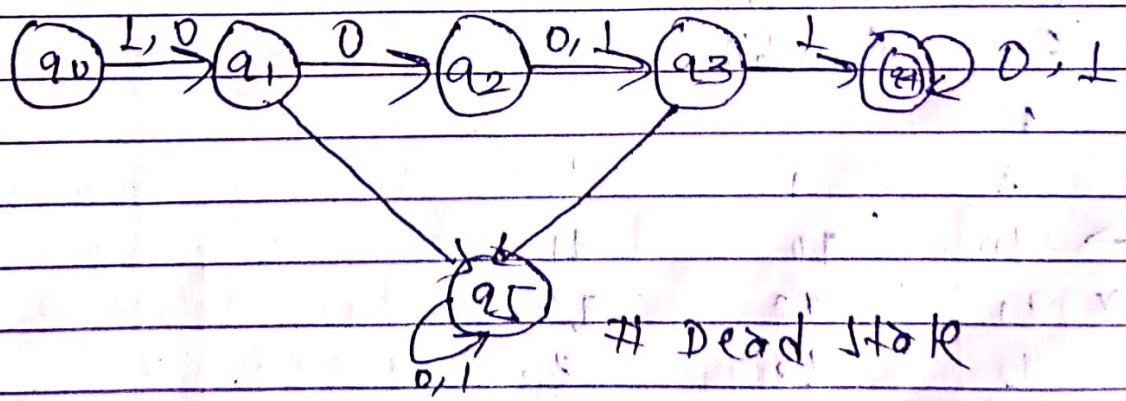
(#) containing exactly 1's in every string



(#) Exactly 4 1's in a string (011101)

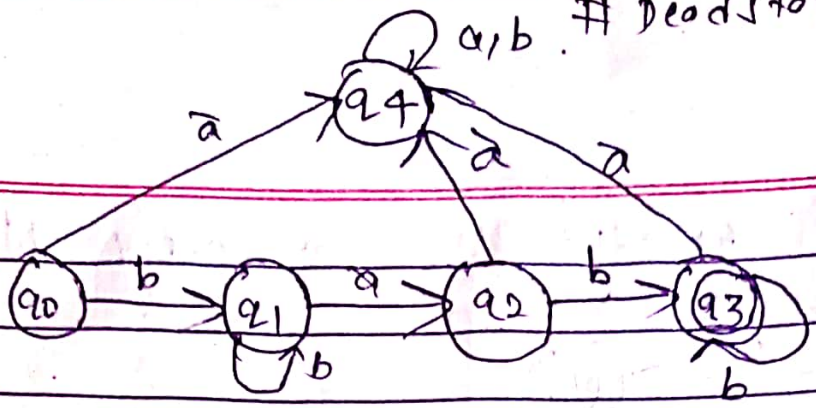


(#) second symbol of w: 1, 0 and fourth is 1
(10, 01, 00)



(#) $L = b^m a b^n \quad : m, n > 0$

$L = \{$ ~~bab~~, bab, babb, ..., $bab^n\}$



(*) Construct DFA which accepts the $L = a^n b \quad (n \geq 0)$

⇒ $\Sigma = \{a, b\}$

$L = \{ a^0 b, a^1 b, a^2 b \quad a^3 b \}$

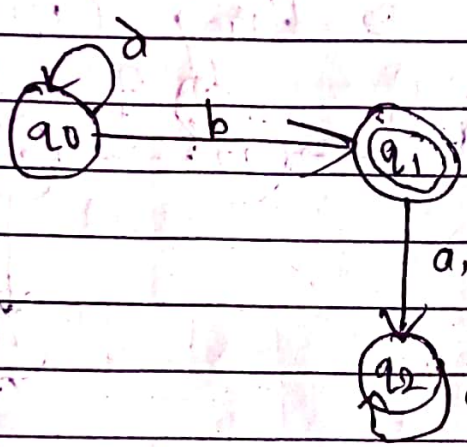


Fig: DFA which accepts the $L = a^n b$.

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

(i) $Q = \{q_0, q_1, q_2\}$

(ii) $\Sigma = \{a, b\}$

(iii) $\delta = ?$

δ	a	b
→ q_0	q_0	q_1
* q_1	q_2	q_2
q_2	q_2	q_2

$\delta(q_0, a) \Rightarrow q_0$

$\delta(q_0, b) \Rightarrow q_1$

$\delta(q_1, a) \Rightarrow q_2$

$\delta(q_1, b) \Rightarrow q_2$

Date:
Page:

(#) Non-deterministic finite automata ⇒ NFA is an automation machine in which for a particular input symbol. the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. A non deterministic finite automata is a mathematical model that consists of 5-tuples $(Q, \Sigma, \delta, q_0, F)$ where,

$Q \Rightarrow$ set of states (finite)

$\Sigma \Rightarrow$ A set of input symbol (finite)

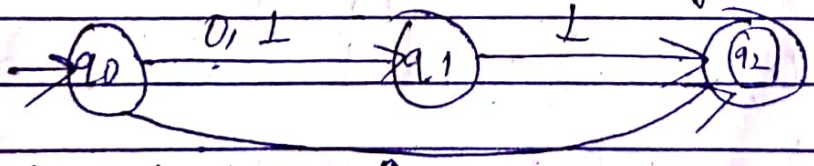
$\delta = Q \times \Sigma \Rightarrow 2^Q$, which is a transition function that maps state symbol pair to set of states.

$q_0 \Rightarrow q_0 \in Q$, which is initial or starting state.

$F \Rightarrow$ set of final states, where, $F \subseteq Q$.

- NFA doesn't contain any dead state.
- Unlike DFA, a transition function in NFA takes the NFA from one state to several states just with a single input.

Example: NFA over {0,1} accepting strings {0, 01, 11}



Transition table:

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

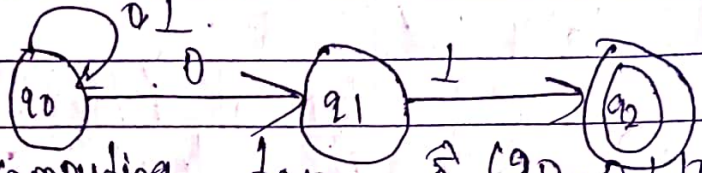
Date:
Page:

(#) Extended Transition function of NFA.

⇒ The extended transition function $\hat{\delta}$ is a function that takes a state q and a string of input symbol w and returns the set of states. In DFA it returns a single state but NFA can return more than one state.

Example:

Considered a NFA.



Now, computing for $\hat{\delta}(q_0, 0110)$

Solution:

$\hat{\delta}(q_0, 0110)$

$\hat{\delta}(q_0, \varepsilon) = \{q_0\}$

$\hat{\delta}(q_0, 0) = \{q_0, q_1\}$

$\Rightarrow \hat{\delta}(q_0, 01) = \delta(q_0, 1) \cup \delta(q_1, 1)$

$= \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

$\Rightarrow \hat{\delta}(q_0, 011) = \delta(q_0, 1) \cup \delta(q_2, 1)$

$= \{q_0\} \cup \{\phi\}$

$= \{q_0\}$

$\Rightarrow \hat{\delta}(q_0, 0110) = \delta(q_0, 0) = \{q_0, q_1\}$

$\Rightarrow \hat{\delta}(q_0, 01101) = \delta(q_0, 1) \cup \delta(q_0, 1)$

$= \{q_0\} \cup \{q_2\}$

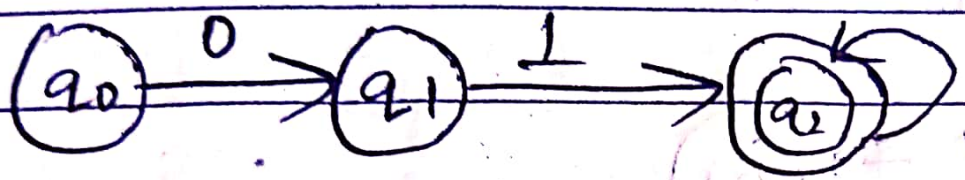
$= \{q_0, q_2\}$

= Accepted

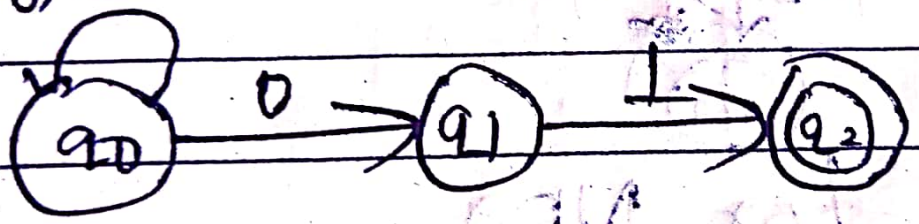
#

(#) NFA

(*) Construct NFA that start with 01



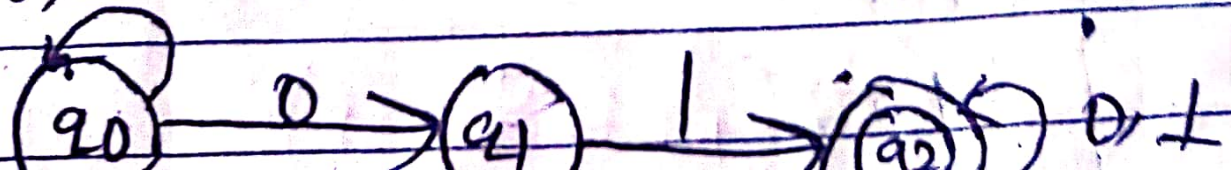
(*) Construct NFA that ends with 01



(बीचको)

(*) Construct NFA whole substring '01'

$L = \{01, 001, 0011, 0101, 0111, \ldots\}$



Date
Page

2.3 Equivalence of DFA and NFA, Subset-Construction:

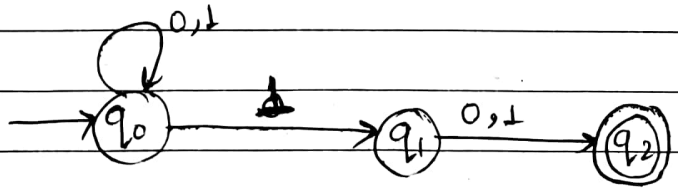
→ DFA and NFA recognise the same class of languages. That is; non-determinism does not make a finite automation more powerful.

To show DFA and NFA recognise the same class of languag we will describe the method of converting an arbitrary NFA into its equivalent DFA. This method is known as subset construction method.

Example

Conversion of NFA to DFA using {subset Construction Method}:

(a)



Here,

Transition Table of NFA is given below:

δ/Σ	0	1
$\rightarrow q_0$	q_0	q_0q_1
q_1	q_2	q_2
$*q_2$	ϕ	ϕ

Conversion of NFA to DFA is:

δ/Σ	0	1
$\rightarrow q_0$	q_0	q_0q_1
q_0q_1	q_0q_2	$q_0q_1q_2$
q_0q_2	q_0	$q_0q_1q_2$
$q_0q_1q_2$	q_0q_2	$q_0q_1q_2$

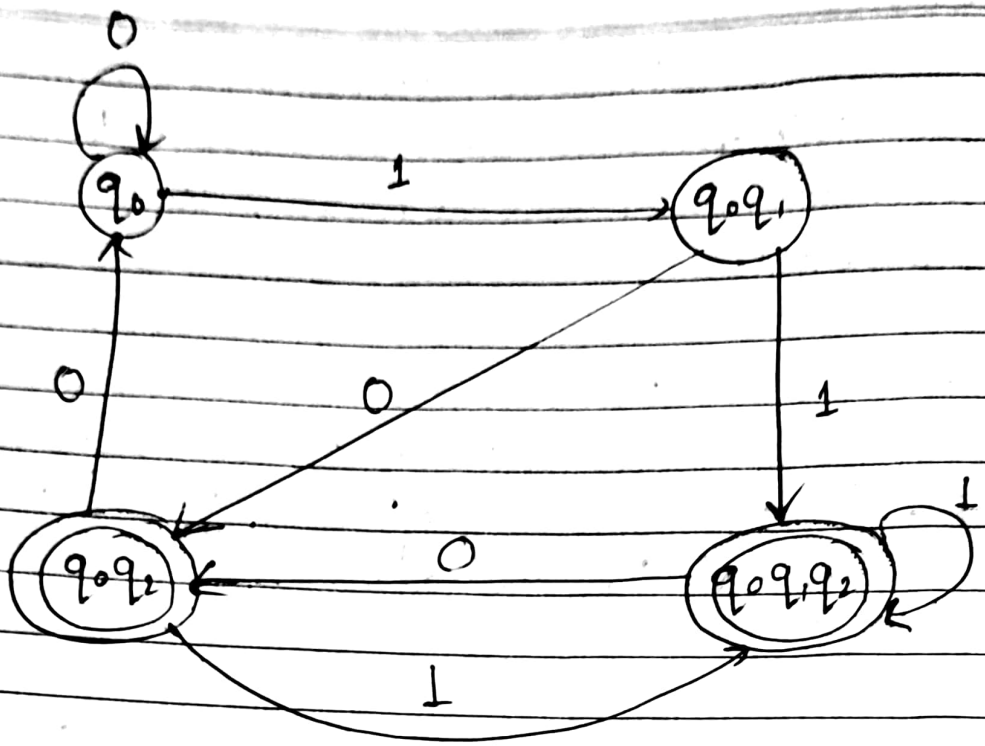
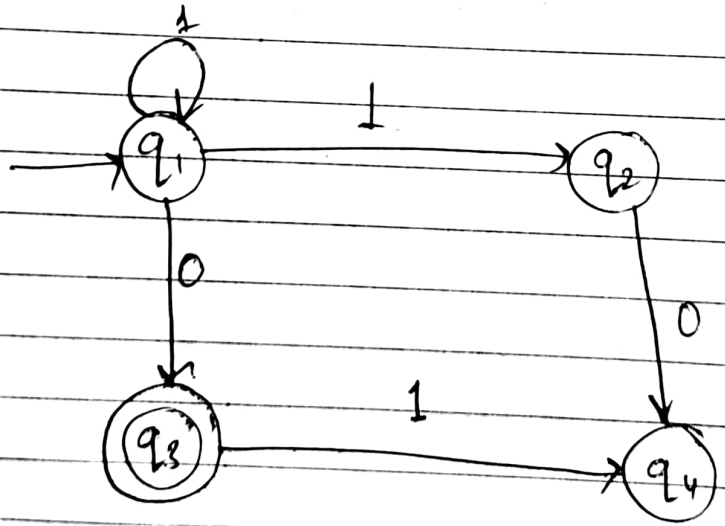


fig: DFA

Note: Maximum subsets are $2^{|Q|} = 2^3 = 8$
Here, there are 4 subsets.

b)



Solution,

Transition Table of NFA is,

$\delta \backslash \Sigma$	0	1
$\rightarrow q_1$	q_3	$q_1 q_2$
q_2	q_4	ϕ
$*q_3$	ϕ	q_4
q_4	ϕ	ϕ

And, converting NFA to DFA,

$\delta \backslash \Sigma$	0	1
$\rightarrow q_1$	q_3	$q_1 q_2$
$q_1 q_2$	$q_3 q_4$	$q_1 q_2$
$*q_3 q_4$	$\phi (q_5)$	$q_1 q_2 q_4$
$q_1 q_2 q_4$	$q_3 q_4$	$q_1 q_2$
$*q_3$	$\phi (q_6)$	q_4
q_4	$\phi (q_7)$	$\phi (q_7)$

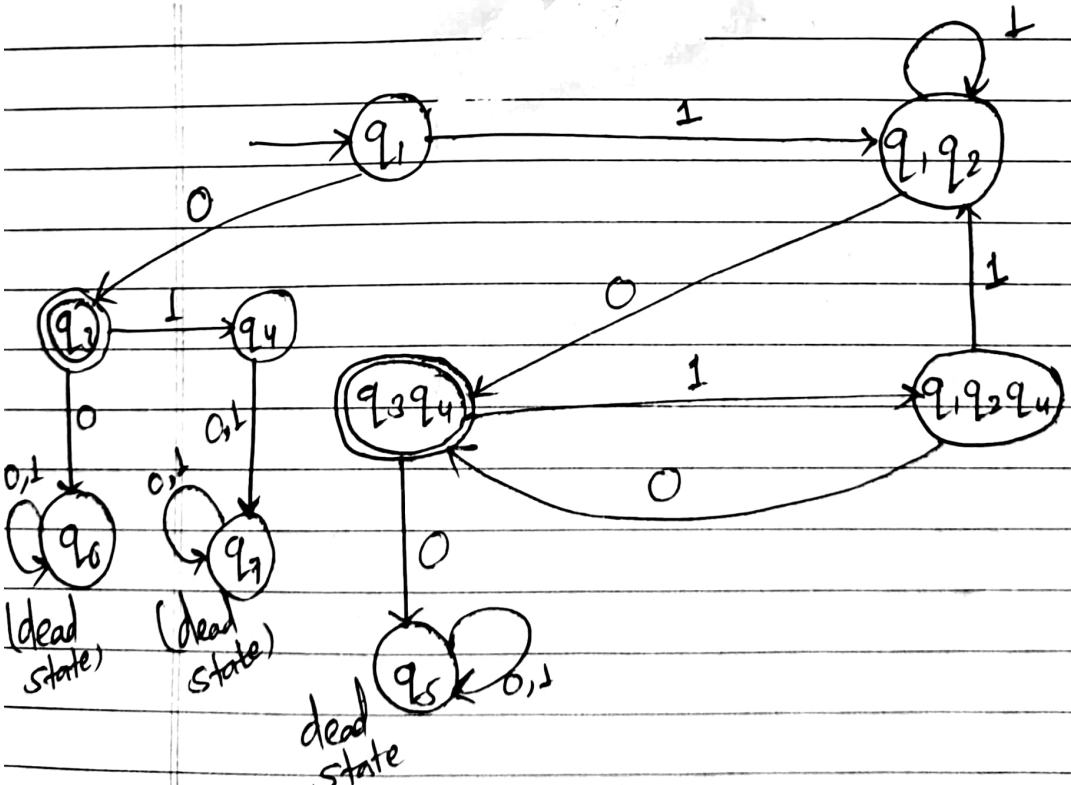
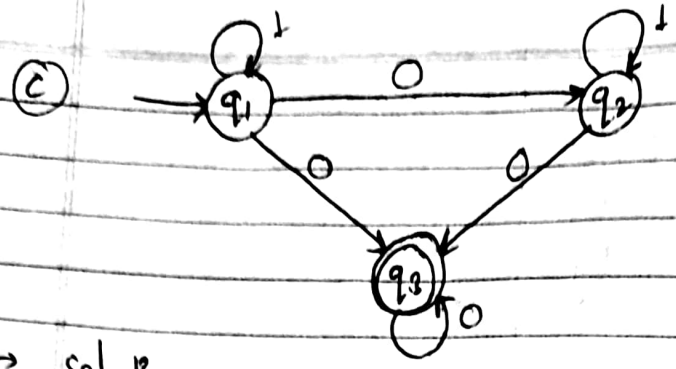


fig : DFA

Date:
Page:



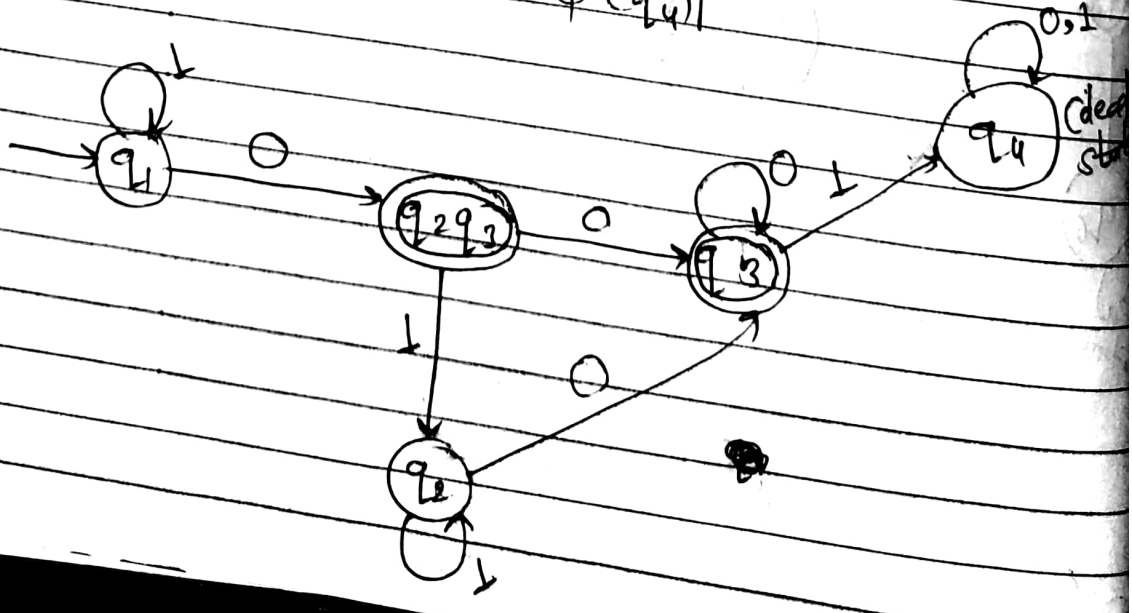
→ solution,

Transition Table of NFA is,

δ/Σ	0	1
→q_1	q_2q_3	q_1
q_2	q_3	q_2
*q_3	q_3	ϕ

Conversion of NFA to DFA is,

δ/Σ	0	1
→q_1	q_2q_3	q_1
*q_2q_3	q_2	q_2
q_2	q_3	q_2
* q_3	q_3	ϕ (q_4)



2.4 later NFA to DFA => done

2.5 Finite Automation with Epsilon Transition (ε-NFA), Notations for ε-NFA, Epsilon Closure of a state, Extended Transition Function of ε-NFA, Removing Epsilon Transition using the concept of Epsilon Closure, Equivalence of NFA and ε-NFA, Equivalence of DFA and ε-NFA

* Epsilon NFA (ε-NFA)

→ An epsilon NFA is defined as a mathematical model that consists of 5 tuple.

$E = (Q, \Sigma, \delta, q_0, F)$ where,

Q = finite set states

Σ = finite alphabet

q_0 = Initial state, $q_0 \in Q$

F = set of final states, $F \subseteq Q$

δ = transition function defined as $|Q|$

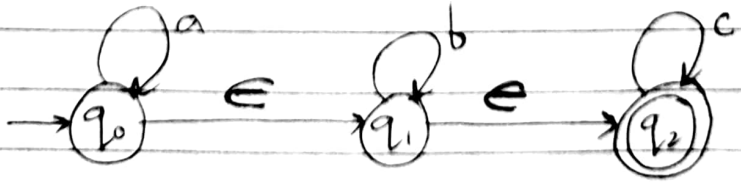
i.e. $\boxed{\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^{|Q|}}$

Eg:

Q. A ε-NFA that accepts following language:

L = set of strings consisting of a's followed by b's followed by c's

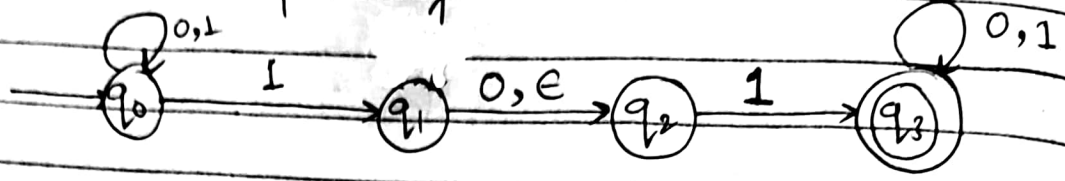
→ Solution,



$L = \{a, b, c, aa, ac, abc, aaabbbccc, \ldots\}$

Date:
Page:

Q. Consider the following ϵ-NFA.



Here,

i) $Q = \{q_0, q_1, q_2, q_3\}$ { Note: every state on ϵ goes itself.

ii) $\Sigma = \{0, 1\}$

iii) $q_0 = q_0$

iv) $F = \{q_3\}$

v) δ is defined as,

$\delta \backslash \Sigma$	0	1	ϵ
$\rightarrow q_0$	q_0	$q_0 q_1$	ϕ
q_1	q_2	ϕ	q_2
q_2	ϕ	q_3	ϕ
$*q_3$	q_3	q_3	ϕ

* Epsilon Closure of a State :

(Informal definition) : All the states reachable from a state q on reading epsilon (ϵ) only form epsilon closure of the state q.

In above example,

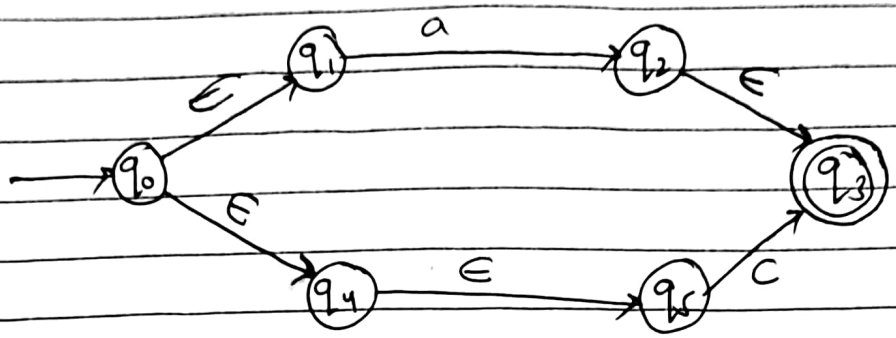
ϵ-closure of state q_0 is

ϵ-closure$(q_0) = \{q_0, q_1, q_2\}$

ϵ-closure$(q_1) = \{q_1, q_2\}$

ϵ-closure$(q_2) = \{q_2\}$

eg. Consider the ϵ-NFA below:



ϵ-closure $(q_0) = \{q_0, q_1, q_4, q_5\}$
ϵ-closure $(q_1) = \{q_1\}$
ϵ-closure $(q_2) = \{q_2, q_3\}$
ϵ-closure $(q_3) = \{q_3\}$
ϵ-closure $(q_4) = \{q_4, q_5\}$
ϵ-closure $(q_5) = \{q_5\}$

Not so important

Extended transition function of ϵ-NFA $(\hat{\delta})$:

→ Given, an ϵ-NFA $E = (Q, \Sigma, \delta, q_0, F)$, the extended transition function for E is defined as follows to reflect what happens on a sequence of inputs.

BASIS: $\hat{\delta}(q, \epsilon) = \epsilon$-closure$(q)$ i.e. if the label of path is ϵ, then we can follow only ϵ-labeled arcs extending from state q, that is exactly what ϵ-closure does.

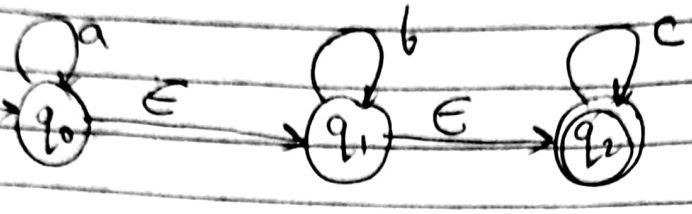
INDUCTION: Suppose w is of form xa, where a is the last symbol of w and a is a member of Σ & $a \neq \epsilon$
Then, we compute $\hat{\delta}(q, w)$ as,

1. Let $\{p_1, p_2, \ldots, p_k\}$ be $\hat{\delta}(q, x)$
2. Let $\bigcup_{p=1}^{k} \delta(p_i, a)$ be the set $\{r_1, r_2, \ldots, r_m\}$

Then, $\hat{\delta}(q, w) = \bigcup_{i=1}^{m} \epsilon$-Closure$(r_j)$

In above example,

Q.1



Soln

Let us use extended transition function of ε-NFA to analyze the acceptance of string "aabc".

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

$\hat{\delta}(q_0, a) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), a))$
$= \epsilon\text{-closure}(\delta(q_0, q_1, q_2), a)$
$= \epsilon\text{-closure}(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a))$
$= \epsilon\text{-closure}(\{q_0\})$
$= \{q_0, q_1, q_2\}$

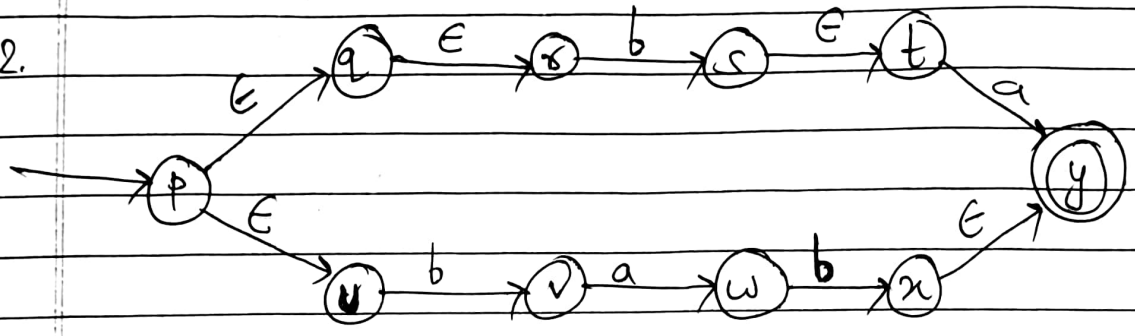
$\hat{\delta}(q_0, aa) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, a), a))$
$= \epsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}), a)$
$= \epsilon\text{-closure}(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a))$
$= \epsilon\text{-closure}(\{q_0\})$
$= \{q_0, q_1, q_2\}$

$\hat{\delta}(q_0, aab) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, aa), b))$
$= \epsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}), b)$
$= \epsilon\text{-closure}(\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b))$
$= \epsilon\text{-closure}(\{q_1\})$
$= \{q_1, q_2\}$

$\hat{\delta}(q_0, aabc) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, aab), c))$
$= \epsilon\text{-closure}(\delta(\{q_1, q_2\}), c)$
$= \epsilon\text{-closure}(\delta(q_1, c) \cup \delta(q_2, c))$
$= \epsilon\text{-closure}(\{q_2\})$
$= \{q_2\}$

Since, q_2 is the accepting state. So, we accept the string aabc.

Q.2.



Solution,

Let's check the string ba

$\hat{\delta}(p, \epsilon) = \epsilon\text{-closure}(p) = \{p, q, r, u\}$

$\hat{\delta}(p, ba) = \epsilon\text{-closure}(\delta(\hat{\delta}(p, \epsilon), b))$
$= \epsilon\text{-closure}(\delta(\{p, q, r, u\}, b))$
$= \epsilon\text{-closure}(\delta(p, b) \cup \delta(q, b) \cup \delta(r, b) \cup \delta(u, b))$
$= \epsilon\text{-closure}(\{s, v\})$
$= \{s, t, v\}$

$\hat{\delta}(p, ba) = \epsilon\text{-closure}(\delta(\hat{\delta}(p, b), a))$
$= \epsilon\text{-closure}(\delta(\{s, t, v\}), a))$
$= \epsilon\text{-closure}(\delta(s, a) \cup \delta(t, a) \cup \delta(v, a))$
$= \epsilon\text{-closure}(\{y, w\})$
$= \{y, w\}$

Here, the final result set i.e, {y, w} contains one of the final state y. So, the string ba is accepted.

* Removing Epsilon Transition using the concept of Epsilon closure.

(Converting ϵ-NFA to its equivalent NFA)

→ To construct NFA from ϵ-NFA, we have to eliminate the ϵ-transitions somehow so that the resulting NFA will have no more ϵ-transitions. for this, we do as below:

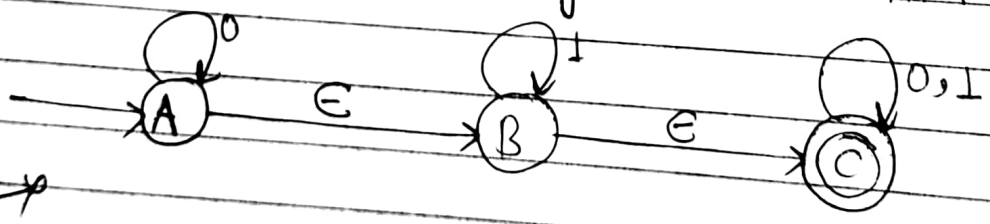
i) for each state, we find the ϵ-closure.

ii) Now set of states that we get in (i) have to be checked on which state they make transition on getting a particular input.

iii) Now, again find the ϵ-closure of all the states we get in (ii).

Eg:→ Convert the following ϵ-NFA to NFA.

(a)



→ Soln →

Let's find ϵ-closure of all states,

ϵ-closure (A) = {A, B, C}
ϵ-closure (B) = {B, C}
ϵ-closure (C) = {C}

It's transition table is,

Date:
Page:

$\delta \backslash \Sigma$	0	1	ϵ
$\rightarrow A$	A	ϕ	B
B	ϕ	B	C
* C	C	C	ϕ

VV.I **Note**: final states of NFA are those, that can reach final state only by reading ϵ.

So, NFA transition table is,

$\delta \backslash \Sigma$	0	1
$* \rightarrow$ A	{A, B, C}	{B, C}
* B	{C}	{B, C}
* C	{C}	{C}

Equivalent NFA is:

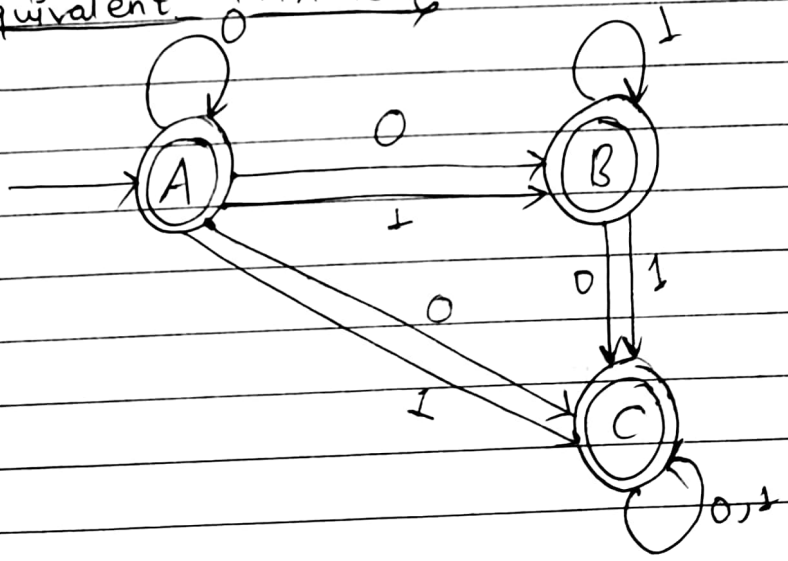
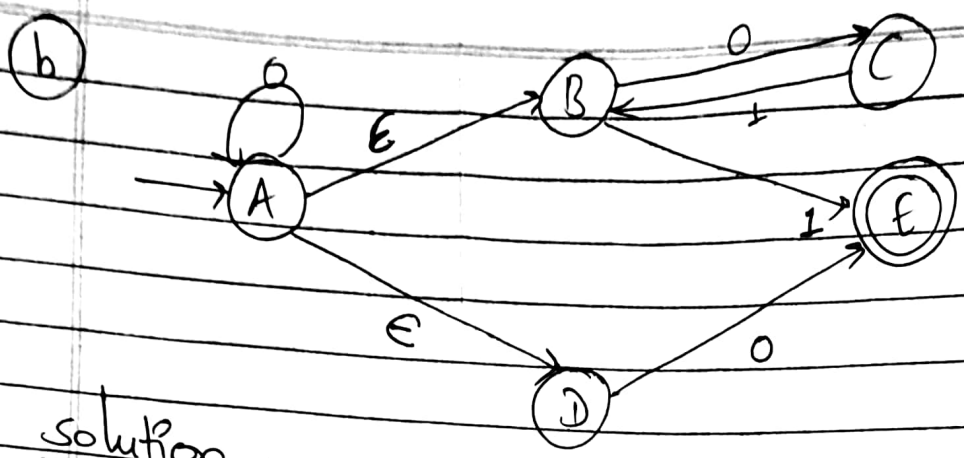


fig: NFA

Date:
Page:



solution,

Here, The ε-closure for all states is,

ε-closure of A = {A, B, D}
ε-closure of B = {B}
ε-closure of C = {C}
ε-closure of D = {D}
ε-closure of E = {E}

Transition table for ε-NFA is given as,

δ/ε	0	1	ε
→ A	{A}	ϕ	{B,D}
B	{C}	{E}	ϕ
C	ϕ	{B}	ϕ
D	{E}	ϕ	ϕ
* E	ϕ	ϕ	ϕ

Converting it into NFA,

we know,

A $\xrightarrow{\epsilon}$ B
B $\xrightarrow{0}$ C
A $\xrightarrow{\epsilon}$ D
D $\xrightarrow{0}$ E

⇒ final states मानेको ϵ-move भी final state सम्म पुग्ने states हरु हुन्छ।

δ/ε	0	1
→ A	{A, B, C, D, E}	{F}
B	{C}	{E}
C	ϕ	{B}
D	{E}	ϕ
*E	ϕ	ϕ

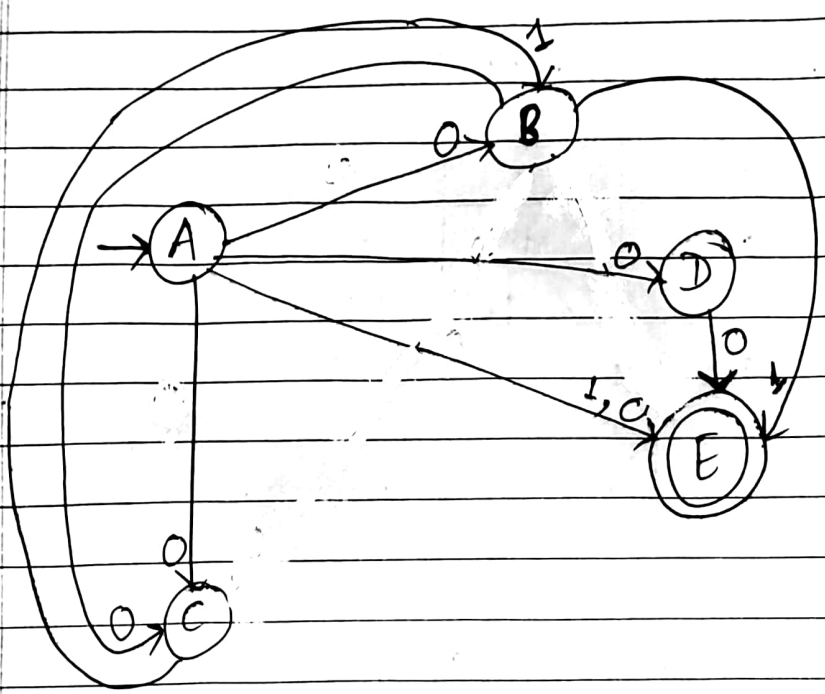


fig: NFA

* Converting ϵ-NFA to DFA (Almost same as NFA to DFA, but we write ϵ-closure of final result,

→ ~~NFA to DFA~~

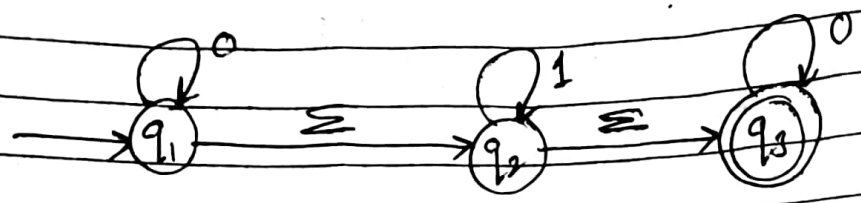
while converting ϵ-NFA to NFA, we used same start in NFA as it is in ϵ-NFA but now for converting ϵ-NFA to DFA, we will use ϵ-closure of start state in ϵ-NFA as a start state in DFA.

Example:

Note: The ϵ-closure of all states are the new states for DFA.

Date:
Page:

Q.1) Convert ε-NFA to DFA.



→ Solution,

Here,

Transition Table for ε-NFA is,

δ/Σ	0	1	ε
→ q_1	q_1	ϕ	q_2
q_2	ϕ	q_2	q_3
* q_3	q_3	ϕ	ϕ

So, the new states of DFA are ε-closure of all states.

Here,

ε-closure of q_1 = $q_1q_2q_3$

ε-closure of q_2 = q_2q_3

ε-closure of q_3 = q_3

1) for $(q_1q_2q_3, 0)$ = ε-closure $(\delta(q_1,0) \cup \delta(q_2,0) \cup \delta(q_3,0))$

= ε-closure $(q_1 \cup \phi \cup q_3)$

= ε-closure $(\{q_1, q_3\})$

= $\{q_1, q_2q_3\}$

for $(q_1q_2q_3, 1)$ = ε-closure $(\delta(q_1,1) \cup \delta(q_2,1) \cup \delta(q_3,1))$

= ε-closure $(\phi \cup q_2 \cup \phi)$

= ε-closure $(\{q_2\})$

= $\{q_2, q_3\}$

Date:
Page:

ii) for $(q_2q_3, 0)$ = ϵ-closure $(\delta(q_2,0) \cup \delta(q_3,0))$
= ϵ-closure $(\phi \cup q_3)$
= ϵ-closure $(\{q_3\})$
= $\{q_3\}$

for $(q_2q_3, 1)$ = ϵ-closure $(\delta(q_2,1) \cup \delta(q_3,1))$
= ϵ-closure $(q_2 \cup \phi)$
= ϵ-closure $(\{q_2\})$
= $\{q_2q_3\}$

iii) for $(q_3, 0)$ = ϵ-closure $(\delta(q_3,0))$
= ϵ-closure $(\{q_3\})$
= $\{q_3\}$

for $(q_3, 1)$ = ϵ-closure $(\delta(q_3,1))$
= ϵ-closure $(\{\phi\})$
= $\{\phi\}$ or q_4

The Transition Table for DFA will then be as follows:

δ/ε	0	1
→ *$q_1q_2q_3$	$q_1q_2q_3$	q_2q_3
*q_2q_3	q_3	q_2q_3
*q_3	q_3	ϕ or q_4
q_4	q_4	q_4

Note: It's start state is the ϵ-closure of state of ϵ-NFA and final state is all the states containing final state of ϵ-NFA.

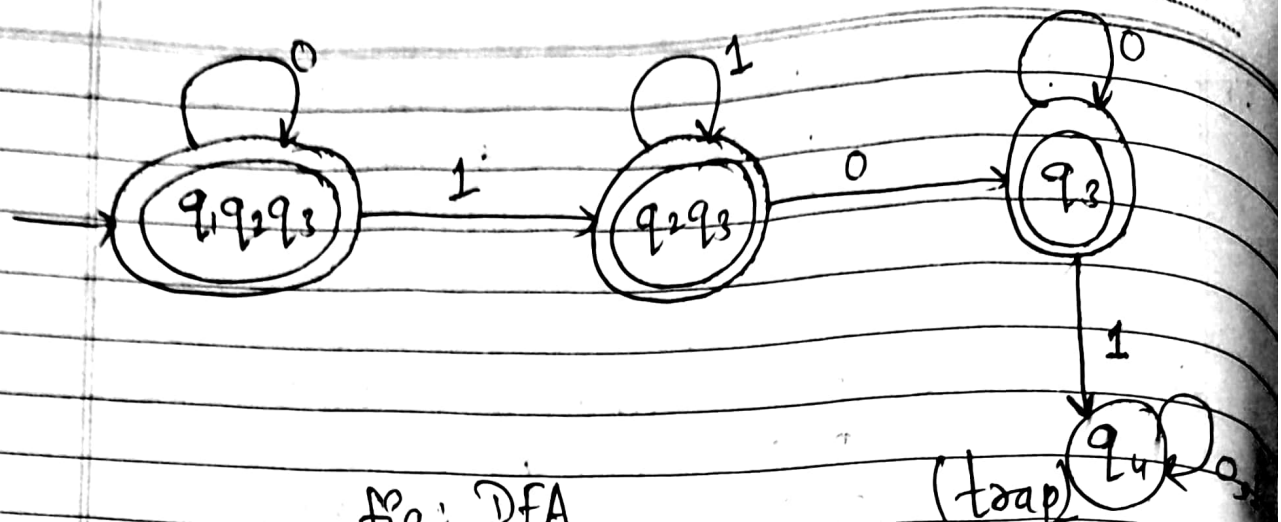
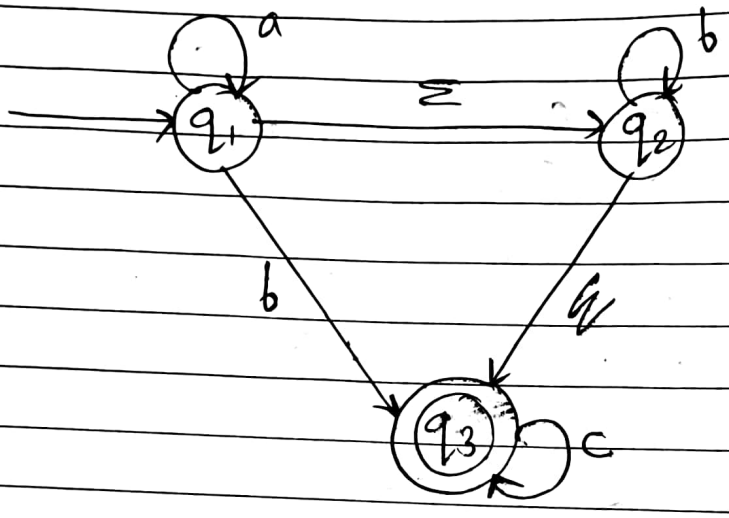


Fig: DFA

Q.2.



Solution,

Transition table of ϵ-NFA is,

δ/Σ	a	b	c	ϵ
$\rightarrow q_1$	q_1	q_3	ϕ	q_2
q_2	ϕ	q_2	ϕ	q_3
$*q_3$	ϕ	ϕ	q_3	ϕ

Now, the DFA states will be the ϵ-closure of all states.

ϵ-closure of q_1 = $\{q_1, q_2 q_3\}$
ϵ-closure of q_2 = $\{q_2 q_3\}$
ϵ-closure of q_3 = $\{q_3\}$

i) for $(q_1 q_2 q_3, a)$ = ϵ-closure $(\delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a))$
= ϵ-closure $(q_1 \cup \phi \cup \phi)$
= ϵ-closure $(\{q_1\})$
= $\{q_1 q_2 q_3\}$

for $(q_1 q_2 q_3, b)$ = ϵ-closure $(\delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_3, b))$
= ϵ-closure $(q_3 \cup q_2 \cup \phi)$
= ϵ-closure $(\{q_3, q_2\})$
= $\{q_2 q_3\}$

for $(q_1 q_2 q_3, c)$ = ϵ-closure $(\delta(q_1, c) \cup \delta(q_2, c) \cup \delta(q_3, c))$
= ϵ-closure $(\phi \cup \phi \cup q_3)$
= ϵ-closure $(\{q_3\})$
= $\{q_3\}$

ii) for $(q_2 q_3, a)$ = ϵ-closure $(\delta(q_2, a) \cup \delta(q_3, a))$
= ϵ-closure $(\phi \cup \phi)$
= ϵ-closure $(\{\phi\})$
= $\{\phi\}$ or q_4

for $(q_2 q_3, b)$ = ϵ-closure $(\delta(q_2, b) \cup \delta(q_3, b))$
= ϵ-closure $(q_2 \cup \phi)$
= ϵ-closure $(\{q_2\})$
= $\{q_2 q_3\}$

for $(q_2 q_3, c)$ = ϵ-closure $(\delta(q_2, c) \cup \delta(q_3, c))$
= ϵ-closure $(\phi \cup q_3)$
= $\{q_3\}$

Date:
Page:

for $(q_3, a) = \epsilon\text{-closure}(\delta(q_3, a))$
$= \epsilon\text{-closure}(\{\phi\})$
$= \{\phi\}$ or q_4

for $(q_3, b) = \epsilon\text{-closure}(\delta(q_3, b))$
$= \epsilon\text{-closure}(\{\phi\})$
$= \{\phi\}$ or q_4

for $(q_3, c) = \epsilon\text{-closure}(\delta(q_3, c))$
$= \epsilon\text{-closure}(\{q_3\})$
$= \{q_3\}$

Transition Table for DFA is,

δ/Σ	a	b	c
$\rightarrow * q_1q_2q_3$	$q_1q_2q_3$	q_2q_3	q_3
$* q_2q_3$	ϕ or q_4	q_2q_3	q_3
$* q_3$	ϕ or q_4	ϕ or q_4	q_3
q_4	q_4	q_4	q_4

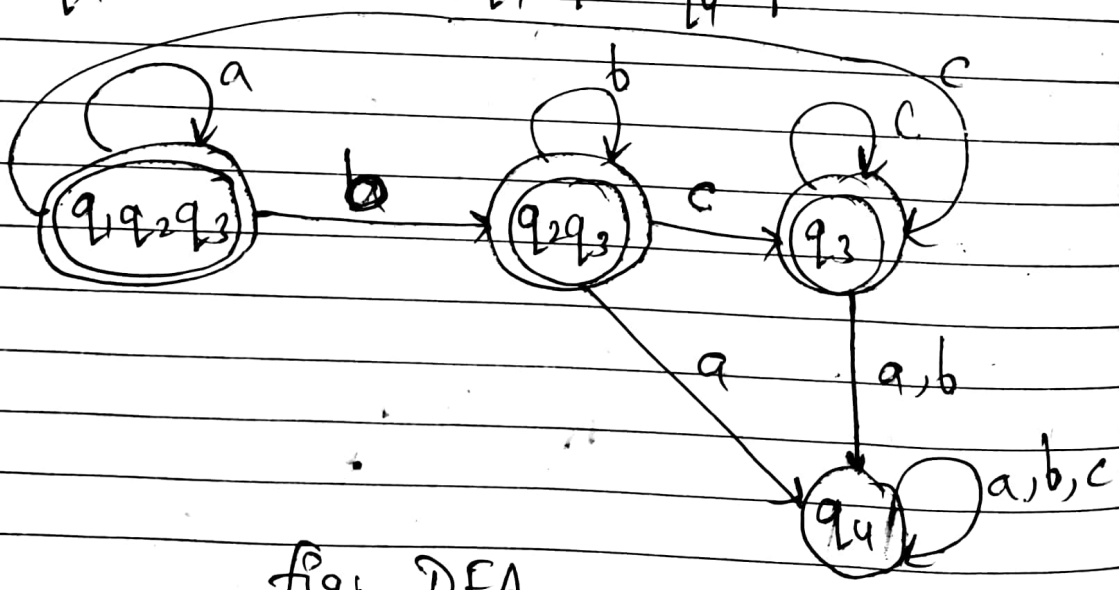


fig: DFA

Date:
Page:

2.6 Finite State Machines with output : Moore Machine and Mealy Machines, Illustration of the Moore and Mealy Machines.

* Moore Machine

→ Moore Machine is an finite state machine whose outputs depend on only the present state. It can be described by a 6 tuple,

$M_o = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$

where, Q = finite set of states

Σ = Input symbol (alphabet)

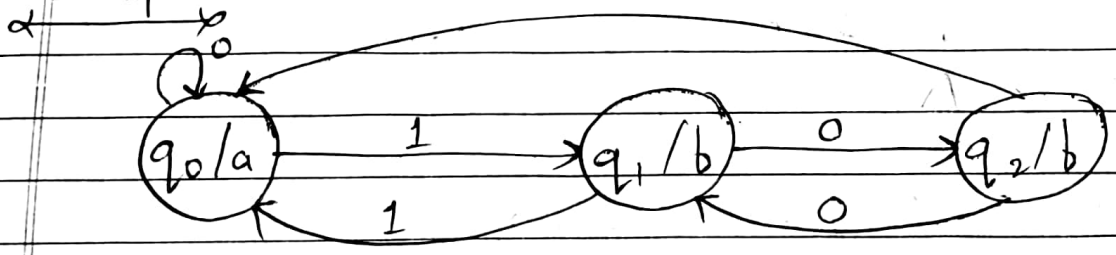
δ = Transition function $\delta : Q \times \Sigma \to Q$

q_0 = start state

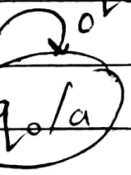
Δ = output symbol

λ = output function, $\lambda : Q \to \Delta$

Example



Here, $Q = \{q_0, q_1, q_2\}$



Note: (1) q_0/a means, state q_0 on getting input 0 goes to itself with output a.

$\lambda : Q \to \Delta \Rightarrow q_0 \to a$

$q_1 \to b$

$q_2 \to b$

Date:
Page:

(2) If you are giving input of n length string, it gives output of n+1 length.

(3) There is no concept of final states in finite state machines with known output.

Transition Table :-

Current state	Next state		Output
	0	1	
q_0	q_0	q_1	a
q_1	q_2	q_0	b
q_2	q_0	q_1	b

* Mealy Machine

→ A mealy Machine is a finite state machine whose output depends on the present state as well as the present input. It can be described by a 6 tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$.

$M_e = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$

where, Q = finite set of states

Σ = Input alphabet

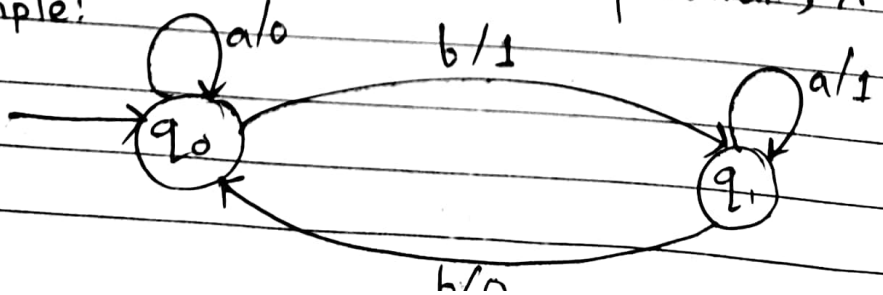
δ = Transition function, $\delta : Q \times \Sigma \to Q$

q_0 = Initial state

Δ = Output symbol

λ = Output function, $\lambda : Q \to \Sigma \to \Delta$

Example:



Date:
Page:

Here, (a, b) ⇒ Input symbol
(0, 1) ⇒ output symbol

λ : $q_0, a \rightarrow 0$
$q_0, b \rightarrow 1$
$q_1, a \rightarrow 1$
$q_1, b \rightarrow 0$

Note : If you give input string of length n, it gives output of same length n.

Transition Table :

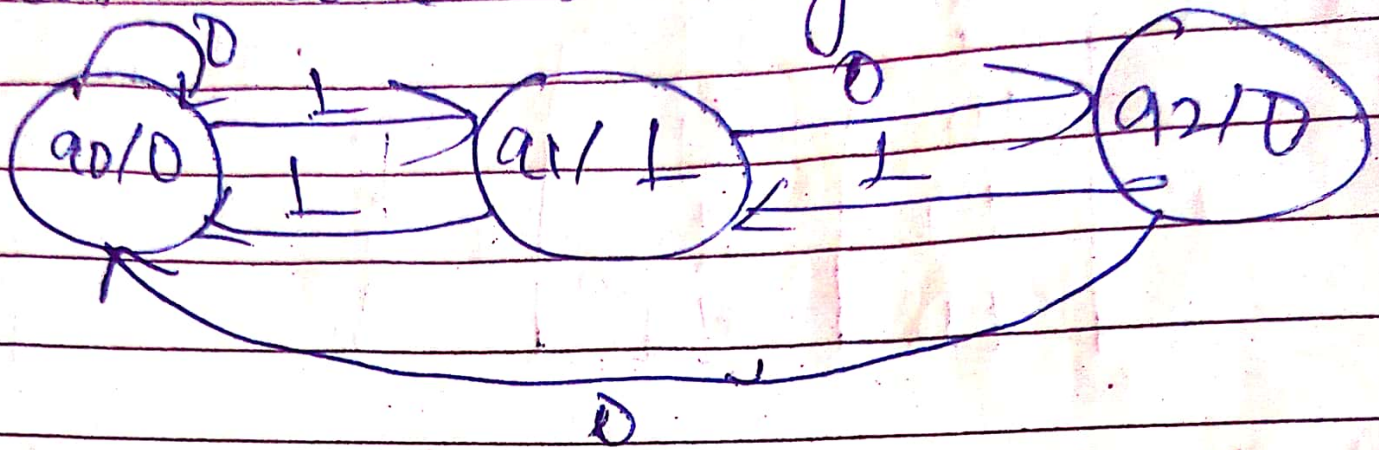
Current state	Next state			
	Input (a)		Input (b)	
	State	output	State	output
$\rightarrow q_0$	q_0	0	q_1	1
q_1	q_1	1	q_0	0

* Difference between Moore and Mealy Machine

Moore Machine	Mealy Machine
i) Output only depends on present state.	i) Output depends on present state and present input.
ii) Input string length 'n' output string 'n+1'.	ii) Input string of length 'n' and output is also 'n' length.
iii) Output is synchronous with clock.	iii) Output is asynchronous.
iv) Generally, it has more states than mealy.	iv) Generally, it has fewer states than Moore machine.

Refer Page no: 11 #

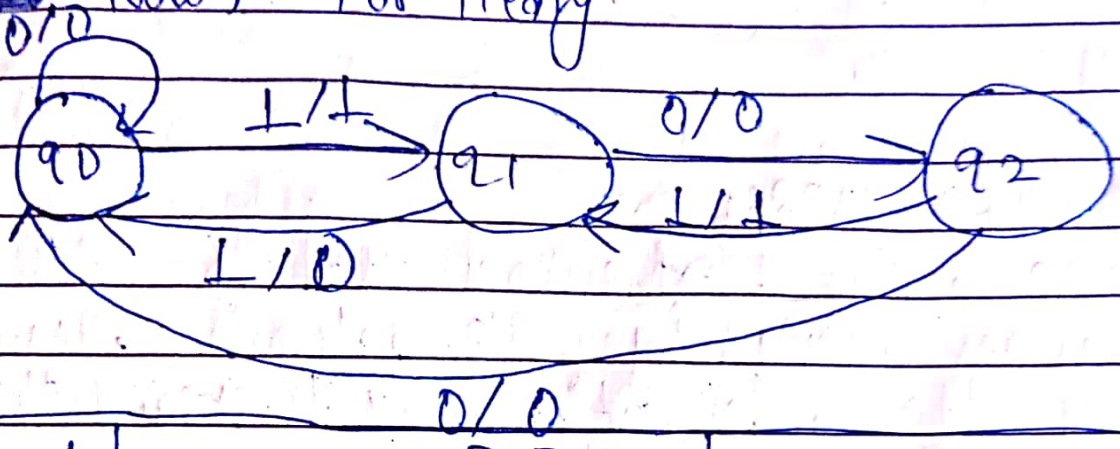
(#) Convert Moore to Mealy Machine



Transition table:

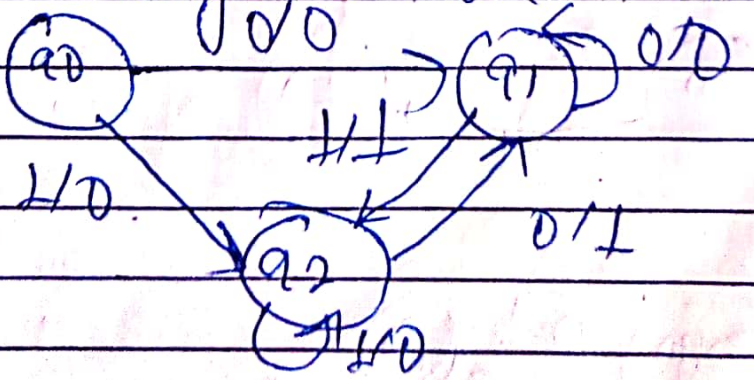
Current State	NEXT STATE		OUTPUT
	0	1	
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_0	q_1	0

Now, For Mealy



Current	NEXT STATE			
	State	output	State	output
q_0	q_0	0	q_1	1
q_1	q_2	0	q_0	0
q_2	q_0	0	q_1	1

(*) Convert Mealy to Moore Machine



Date:

Page: