Unit-2
Introduction to Finite Automata:

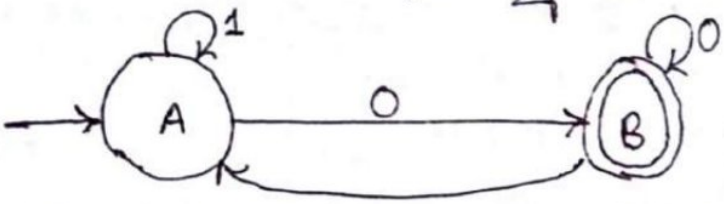
⊛. Finite Automata / Finite State Machine (FSM):

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 as automation

Example: Let we have following FSM:



Things I write inside circles are for understanding only

Now, five tuples for this FSM can be described as;

$Q = \{A, B\}$ ← set of all states

$\Sigma = \{0, 1\}$ ← set of all inputs

$q_0 = A$ ← whenever we see an arrow from nowhere pointing to a state then it is start state or initial state

$F = \{B\}$ ← set of final states, final states are denoted by double rings. Initial state is only one but final states can be more than one.

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

Other tuples are same but formula for δ will be different for DFA, NFA, ϵ-NFA. $\delta = Q \times \Sigma$ is for DFA

	0 ←	1 ←
A	B	A
B	B	A

we write inputs at top (i.e, at column)

we write states at left (i.e, at row)

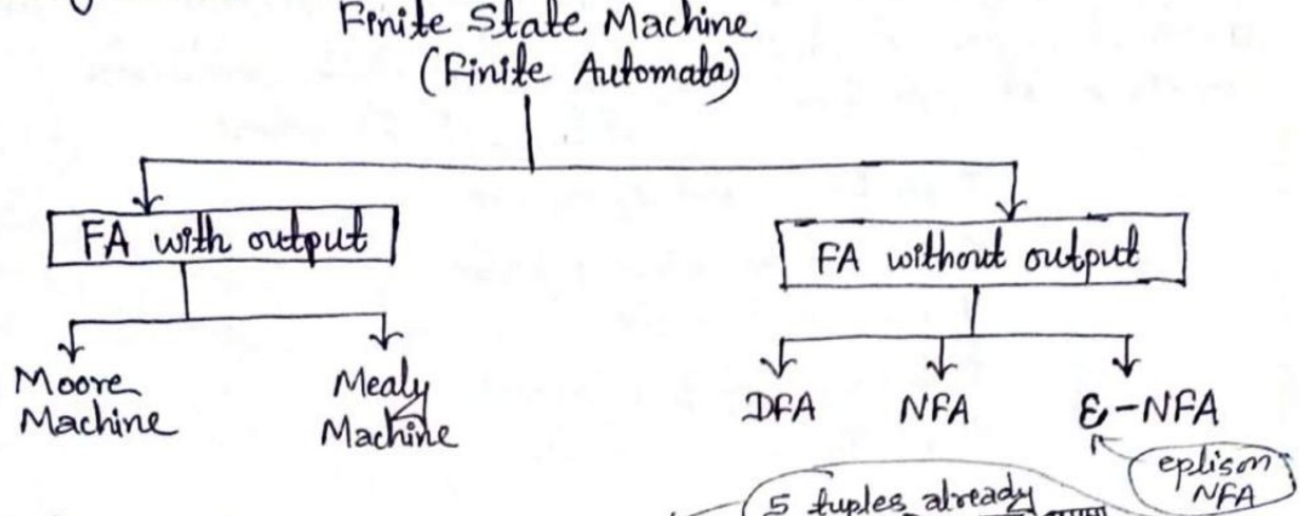
Initially we are at start state A. Now A on getting input 0 it goes to B. similarly for others

transition table (i.e, δ) for above diagram considering it as DFA.

Applications of FSM:

The finite state machines are used in applications in computer science and data networking. For example; finite-state machines are basis for programs for spell checking, indexing, grammar, searching large bodies of text, recognizing speech, transforming text using markup languages such as XML and HTML, and network protocols that specify how computers communicate.

Categories of FSM:

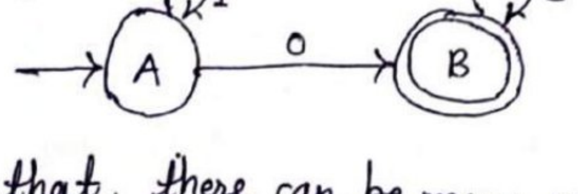


1) Deterministic Finite Automata (DFA):

(5 tuples already discuss गरिसकेका छन् यसमा exam मा आएर 5 tuples पनि discuss गर्नि / we did before)

It is the simplest model of computation which has a very limited memory. In a DFA, for a particular input character, the machine goes to one state only. A transition function is defined on every state for every input symbol. In DFA null (or ε) move is not allowed i.e, DFA cannot change state without any input character. DFA consists of 5 tuples $\{Q, \Sigma, q_0, F, \delta\}$ which we already discussed. Write same for all except δ which is defined as $\delta: Q \times \Sigma \rightarrow Q$.

For example: The below DFA with $\Sigma = \{0, 1\}$ accepts all strings ending with 0.



→ Note that, there can be many possible DFAs for a pattern. A DFA with minimum number of states is generally preferred.

⊛ Notations for DFA:

There are two preferred notations for describing DFA;

→ Transition table.

→ Transition Diagram.

Transition table:

Transition table is a conventional tabular representation of the transition function δ that makes the arguments from $Q \times \Sigma$ and returns a value which is one of the state of the automata. The row of the table corresponds to the states, while column of the table corresponds to the input symbol. The starting state of the table is represented by arrow (→) followed by the state (i.e, represented as: $\rightarrow q$) where; q is considered any initial state. The final state is represented as $*q$ for q being final state. The entry for a row corresponding to state q and the column corresponding to input a, is the state $\delta(q,a)$.

For example: Consider a DFA as;

$Q = \{q_0, q_1, q_2, q_3\}$

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

$q_0 = q_0$

$F = \{q_0\}$

$\delta = Q \times \Sigma \rightarrow Q$.

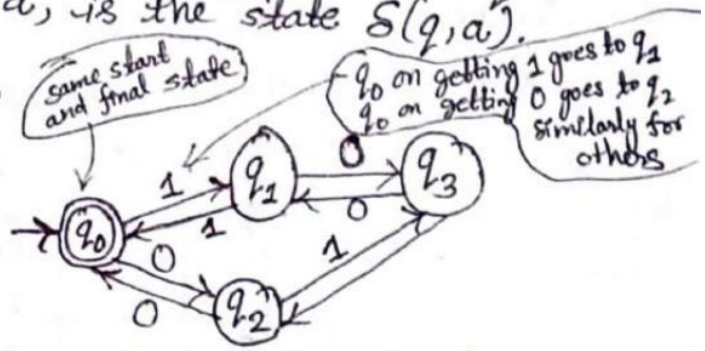


Fig: transition diagram.

Then, the transition table for above DFA is as follows:-

same start and final state

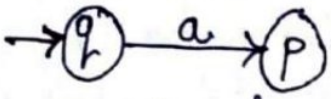
δ	0	1
$*\rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

look at figure q_0 on getting 0 goes to q_2 and q_0 on getting 1 goes to q_1 similarly for others

⇒ This DFA accepts strings having both an even number of 0's & even number of 1's.

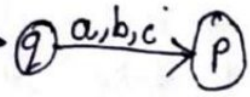
Transition diagram:-

A transition diagram of a DFA is a graphical representation or a graph. For each q in Q and each input 'a' in Σ, if $\delta(q,a)=p$ then there is an arc from node q to p labelled as a in the transition diagram, as below;



(i.e, q on getting a goes to p).

If more than one input symbol cause the transition from state q to p then arc from q to p is labelled by a list of those symbols.

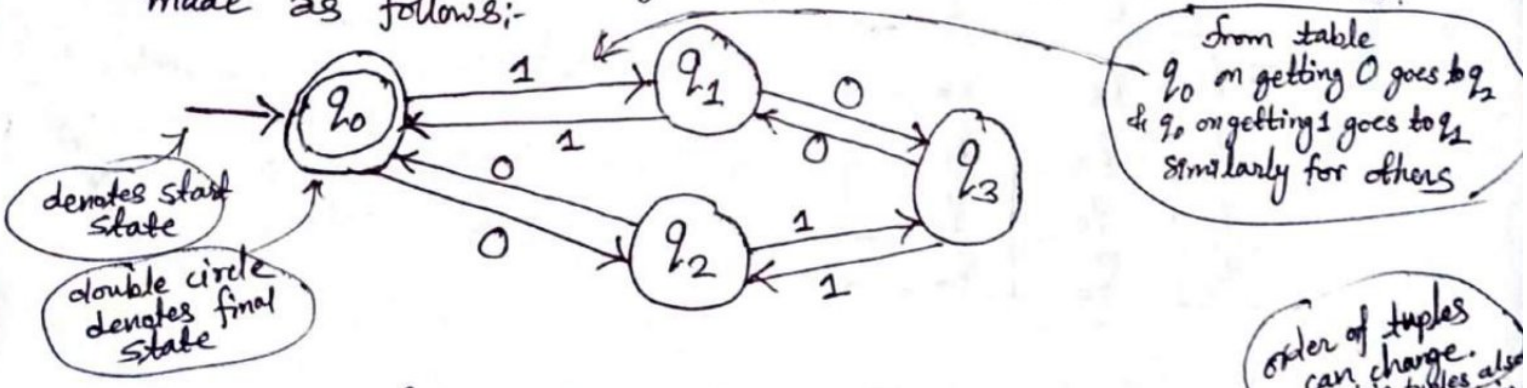
for e.g. 

(i.e, q on getting a or b or c goes to p).

In transition diagram, start state is denoted by labelled by an arrow comming from nowhere and the final state or accepting state is marked by double circle. Let we have Following transition table;

*	δ	0	1
→	q_0	q_2	q_1
	q_1	q_3	q_0
	q_2	q_0	q_3
	q_3	q_1	q_2

Now the transition diagram for this transition table can be made as follows;-



order of tuples can change. Explain tuples also if asked in exam

Language of DFA:- The language of DFA, $M=(Q,\Sigma,\delta,q_0,F)$ denoted by $L(M)$ is a set of strings over Σ^* that are accepted by M. Σ^* is the set of all the input symbols. This means the language of DFA is the set of all strings (i.e, sequence of input symbols) that take DFA from start state to one of accepting states.

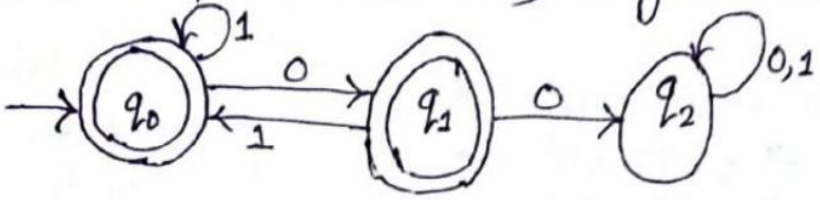
⊛ Extended Transition Function of DFA ($\hat{\delta}$):-

The extended transition function of DFA, denoted by $\hat{\delta}$ is a transition function that makes two arguments as input: one is the state q of Q and another is a string belonging to Σ^* and generates a state $p \in Q$. This state p is reached when the current state processes the sequence of inputs.

i.e, $\hat{\delta}(q, W) = p$

where, q = current state.
W = sequence of inputs.
P = generated or new reached state.

For Example: Compute $\hat{\delta}(q_0, 1001)$ using the DFA below:



Here,

$\hat{\delta}(q_0, 1001)$
$= \delta(\hat{\delta}(q_0, 100), 1)$
$= \delta(\delta(\hat{\delta}(q_0, 10), 0), 1)$
$= \delta(\delta(\delta(\hat{\delta}(q_0, 1), 0), 0), 1)$
$= \delta(\delta(\delta(\delta(\hat{\delta}(q_0, \varepsilon), 1), 0), 0), 1)$
$= \delta(\delta(\delta(\delta(q_0, 1), 0), 0), 1)$
$= \delta(\delta(\delta(q_0, 0), 0), 1)$
$= \delta(\delta(q_1, 0), 1)$
$= \delta(q_2, 1)$
$= q_2$, since q_2 is not final state, so the string 1001 is not accepted.

first expand until we get $\hat{\delta}(q_0, \varepsilon)$ empty

Now solve one by one. $\hat{\delta}(q_0, \varepsilon)$ is always q_0 since ε is empty

Now we solved $\delta(q_0, 1)$. q_0 on getting 1 (in the fig.) goes to q_0 itself so $\delta(q_0, 1) = q_0$. Similarly we proceed rest

q_2 को सट्टा finally हामिले q_0 वा q_1 पाको नभए string 1001 accept हुने थियो किनकि q_0 र q_1 final state हुन् जसलाई double circle ले denote गरिएको छ।

Understanding what are string and other terms

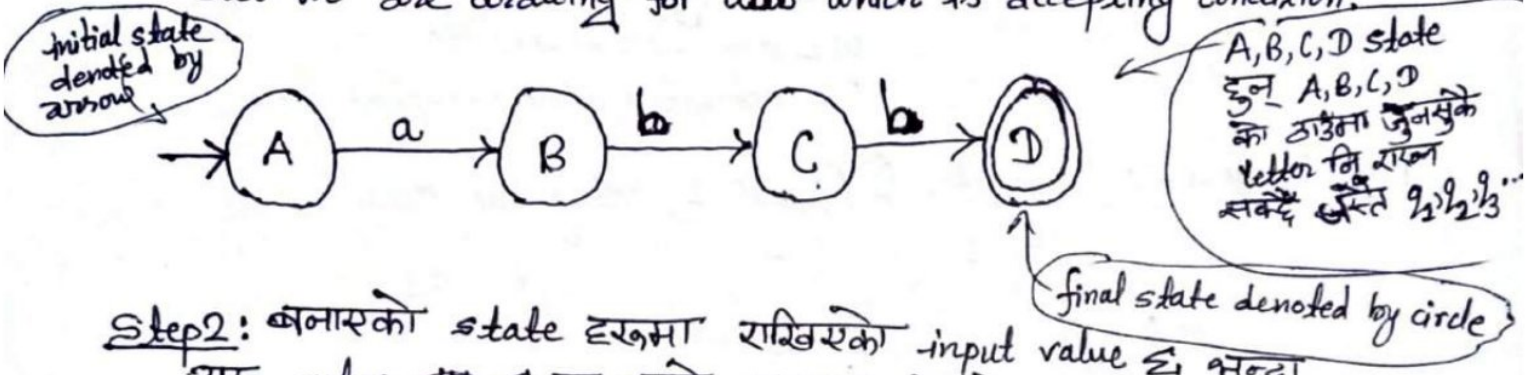
Symbol → a, b, c, 0, 1, 2, 3 ...
Alphabet → Collection of symbols
e.g. {a, b}, {0, 1, 2} ...
String → Sequence of symbols
e.g. a, b, 0, 1, aa, bb, 01, aaa, 111 ...
Language → Set of strings e.g. {0, 1}, {aa, bb} ...

⊛. Constructing DFA with examples:-/Examples of DFA:

(for understanding only)

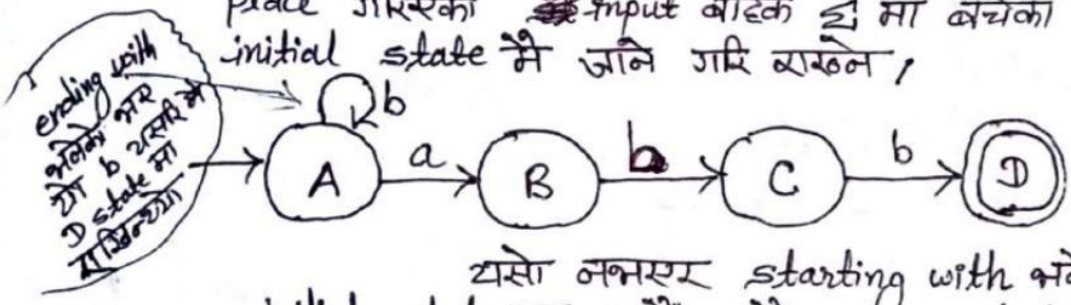
Method/Steps: Let we are going to construct a DFA, that accepts all strings over $\Sigma = \{a, b\}$ ~~that accepts~~ ending with abb.

Step1: Question मा दिइएको accept गर्ने condition अनुसार, सुरुमा त्यो condition accept गर्ने diagram मात्र बनाउने initial state देखि final state सम्म / According to above question taken ~~let~~ we are drawing for abb which is accepting condition.



Step2: बनाएको state हरूमा राखिएको input value Σ भन्दा अरु value ~~Σ~~ Σ मा भएको सम्बन्धा दिएको condition लाई असर नगर्ने गरि मिलाउने जसमा तलका sub-step काम लाउने ।

i) Question मा ending with भनेको छ भने start state मा already place गरिएको ~~input~~ input बाहेक Σ मा बचेका अरु input त्यहि initial state मै जाने गरि राख्ने ।

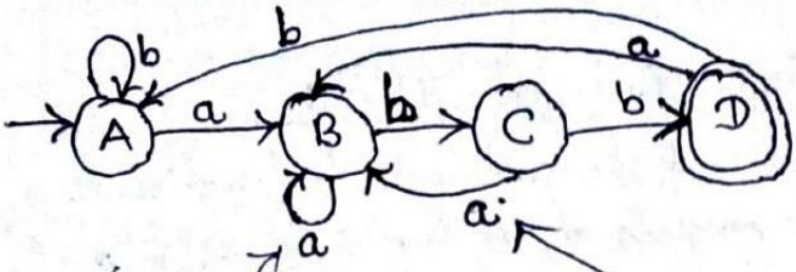


यसो नभएर starting with भनेको नर जसरी initial state मा राखेथें त्यसै गरि initial state मा नभएर final state मा राखिन्छ। starting with, ending with केहि भनेको छैन र containing string, having string भनेको छ भने initial र final दुबैमा राखिन्छ ।

ii) बचेका state मा input गर्न बाँकी string राख्ने र सम्बन्धा transition (arrow) कुन state मा जान्छ analyse गर्ने, यदि दिइएको condition ~~किमिले~~ बिगारेन भने त्यसै current state मा रहन्छ ।

बिगारेन वा ~~नमिले~~ चाहिएको string फेरि पाउने सम्भावना छ भने

→ ~~छ~~ त्यसो नभएर यदि दिएको condition मा भनेको string बिगार्ने word आउँछ भने input मा (for eg. abb sequence बिगार्ने according to example) त्यतिबेला त्यो word first ~~छ~~ चोटि पाएर कुन state मा पुगेका थियें त्यहि state मा arrow पुऱ्याउने ।



(3) Similarly D state मा कुनै पनि input दिन गरिसको so, a र b दुबैको लागि check गर्ने

(2) Similarly B state को जस्तै state C मा already b input भएर D मा गएको छ / So, यहाँ बाट बचेको अब a मात्र हो / abb condition मा state C मा a आउदा abab हुन्छ जस्ले दिएको condition abb को sequence बिगार्छ / abb पाउने सम्भावना छैन त्यसैले पहिलो चोटि a पाएर जुन state मा (state B) मा पुगेका थियौं त्यहि state मा जान्छ / (According to step2 (ii) 2nd part.)

(1) State B मा b already input थियो अब a को लागि check गर्ने / यसमध्येको सुरुको word a पाएर already B मा छ / अब पेरि a आर b state मा string aabb type को हुन्छ जसमा aabb ending with abb देख्न सक्दो condition अनुसार पाउने सम्भावना छ त्यसैले यस्तो sequence बिगार्छैन त्यसैले step 2 (ii) को अनुसार त्यहि state मा बस्यो /

Now we get Complete DFA according to the question.

Step3: Note that कुनै word कुनै पनि state मा जान सकेन यो condition अनुसार बनेर त्यो नया state dead state मा जान्छ र dead state मा सबै input value त्यहि रहन्छन् / Question हरु जस्मा restrictions हुन्छन् (जस्तै having length 2 only) यस्तो type मा dead state हुन्छ / ~~Note: Constructing NFA~~ Restriction नभएको case मा dead state हुँदैन जस्तो अहिले गरिको step सँगै को example मा /.

⇒ Question मा does not भनेर आयो भने त्यसलाई पहिला positive बनाएर solve गर्ने (for e.g. does not contain लाई contains बनाउने) र finally; ~~final~~ non-final state जति सबलाई final बनाउने र non-final जति सबै state लाई final बनाउने /

There is another method to construct DFA. As we know constructing NFA is easy so first construct NFA then convert it to DFA. Conversion is discussed later.

Practice DFA questions as much as you can. If you understand DFA then it is easy to understand whole chapter.

2) Non-Deterministic Finite Automata (NFA):

A non-deterministic finite automata/automation 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 symbols (finite)

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

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

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

all are same as DFA except δ definition

There are two main things that we should keep in mind:

i) NFA does not contain any dead state.

ii) Unlike DFA, a transition function in NFA takes the NFA from one state to several states just with a single input.

For Example1:

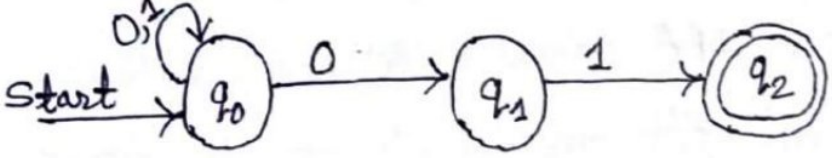
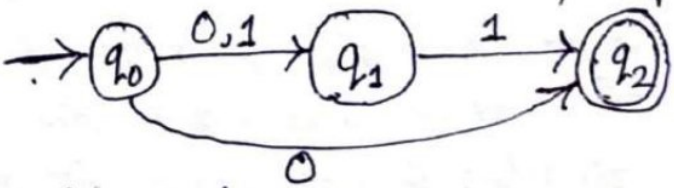


fig: NFA accepting all strings that end in 01.

Here, q_0 state on getting 0 goes to q_0 itself as well as q_1. But in DFA on getting particular input it goes to one particular state only.

Example2: NFA over {0,1} accepting strings {0, 01, 11}.



transition table:

δ	0	1
$\rightarrow q_0$	$\{q_0, q_2\}$	q_1
q_1	$\emptyset$	q_2
$*q_2$	$\emptyset$	$\emptyset$

* shows final state

$\emptyset \rightarrow$ denotes it goes no where i.e. empty

Constructing NFA is easier than DFA. No need to draw dead state. Draw only what it accepts. With one input may go to more than one state.

DFA without Dead state is NFA which can go to many states on single input

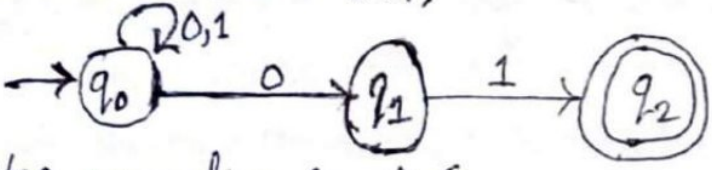
⊛. **Notations for NFA:** Notations for NFA are also same as DFA. [Theory write same only change example by writing example of NFA].

⊛. **Language of NFA:** Language for NFA is also same as DFA except definition of $\delta = Q \times \Sigma \to 2^Q$. In DFA $\delta = Q \times \Sigma \to Q$. [Write same theory as we wrote in DFA explaning 5 tuples].

⊛ **Extended Transition function of NFA:** ← Theory similar to DFA only change example

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: Consider a NFA:



Now computing for $\hat{\delta}(q_0, 01101)$

Solution:

$\hat{\delta}(q_0, 01101)$

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

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

$\hat{\delta}(q_0, 01) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

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

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

$\hat{\delta}(q_0, 01101) = \delta(q_0, 1) \cup \delta(q_1, 1)$

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

$= \{q_0, q_2\}$

$=$ Accepted.

Since $(q_0, 0) = \{q_0, q_1\}$ in above step
so, $((q_0, 0), 1) = (q_0, 1) \cup (q_1, 1)$

$(q_0, 01) = \{q_0, q_2\}$
$((q_0, 01), 1) = \{q_0, 1\} \cup \{q_2, 1\}$

$(q_0, 011) = \{q_0\}$
$((q_0, 011), 0) = \{q_0, 0\}$
q_0 on getting 0 goes to q_0 & q_1

finally find गरेको set मा कुनै एउटा state पनि final state देखिए accept हुन्द नत्र reject

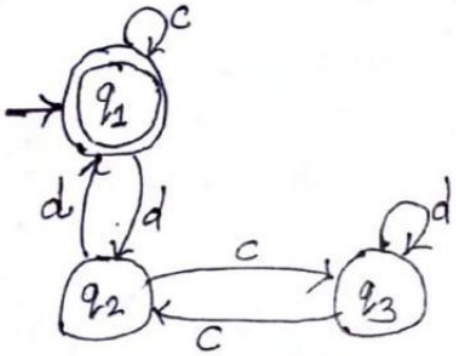
⊛ Equivalence of NFA & DFA:-

The equivalence of two finite automata determines whether the two automatas are equal or not. i.e, the two automatas perform the same function. The languages they accept will be same. So, in order to identify two automatas are equivalent or not there are certain steps we need to follow:-

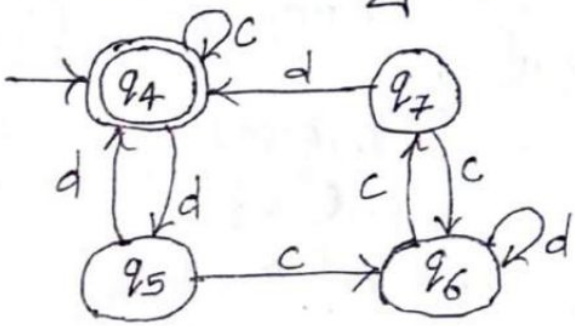
1) If initial state is final state of one automation, then in second automation also initial state must be final state for them to be equivalent.

2) For any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$. The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is intermediate state and the other is final state.

Example: Let we have following two automatom namely A & B.



Automaton (A)



Automaton (B)

⇒ In automation A, q_1 is the initial state as well as final state and In automation B also q_4 is the initial state as well as the final state so it satisfies our first condition.

⇒ Now we check second condition, for that let we first construct pair of states from two automaton and check transition in the table as follows:-

States	Inputs	
	c	d
(q_1, q_4)	(q_1, q_4) FS FS	(q_2, q_5) IS IS
(q_2, q_5)	(q_3, q_6) IS IS	(q_1, q_4) FS FS
(q_3, q_6)	(q_2, q_7) IS IS	(q_3, q_6) IS IS
(q_2, q_7)	(q_3, q_6) IS IS	(q_1, q_4) FS FS

IS → Intermediate state
FS → Final state

Now we can see that all the pairs checked have transition in c and d both are either initial states or final states. This satisfies the second condition.

⇒ Hence both automation A and B are equivalent.

Note: While making or checking transition if we found two pairs are not same (i.e, both are not intermediate states or final states) then we stop there and conclude as not equivalent.

⊛ Conversion of NFA to DFA: (using subset construction method):

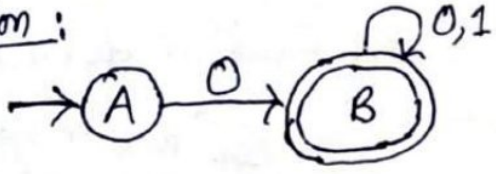
⇒ Remember that every DFA is an NFA but not vice versa. But there is an equivalent DFA for every NFA.

language

Example 1: Construct NFA for L = {Set of all strings over (0,1) that starts with '0'} then convert it to equivalent DFA.

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

Solution:



δ	0	1
→A	B	∅
*B	B	B

empty state A on getting 1 goes nowhere

This is the required DFA with transition table for given question.

Now, to convert to equivalent DFA first we construct transition table for DFA looking at (or using) transition table of NFA.

δ	0	1
→A	B	C
*B	B	B
C	C	C

In NFA A on getting goes to ∅ but in DFA transition can not go to ∅. So, we made new state C for transition A on getting 1. Now we proceed on the basis of new states.

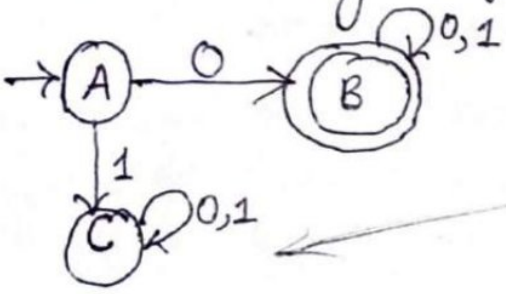
For this next step we look in first row and states that are not input till now are used as input. B is not taken input so we take B.

Analysing table for NFA we know that B on getting 0 and 1 both goes to B

Now look before rows A, B. Here, C is not taken as input till now. So, we took C as input

∅ was changed to C. So, C is dead state, C on getting any input goes itself to C. Now A, B, C all are used as input no state remains so we stop there.

Now, based on this transition table for DFA we can construct DFA easily as follows:-



Here, C is dead state or trap state

Example 2: Find the equivalent DFA for the NFA given by $M=[\{A,B,C\},(a,b),\delta,A,\{C\}]$ where, δ is given by;

δ	a	b
→A	A,B	C
B	A	B
*C	∅	A,B

these are 5 tuples where, [States, input, δ, Initial state, final state]

Solution:-

For converting into equivalent DFA first we draw transition table as follows:-

δ	a	b
→A	AB	C
→AB	AB	BC
*C	D	AB
*BC	A	AB
D	D	D

New state from first row, We combined AB two states as one so AB on getting a, the union operation of A and B on getting a in transition table of NFA is done. (i.e, (A,B) ∪ A = A,B = AB)

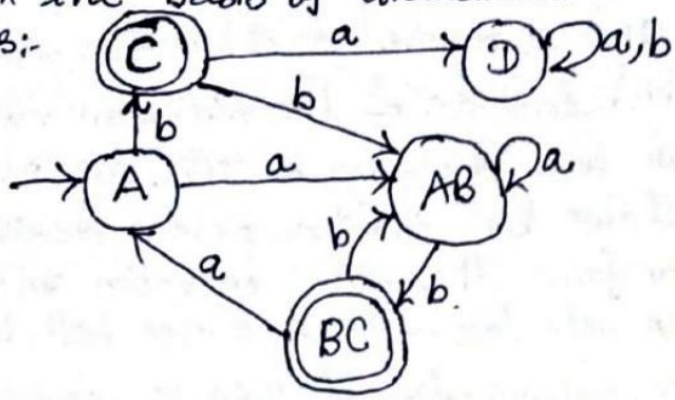
In NFA on single input it can go to more than one state but in DFA on single input goes to single state only so we combine A and B as AB as one state

∅ changed to new state

Since C state uses final state in NFA so, C containing all states are final here.

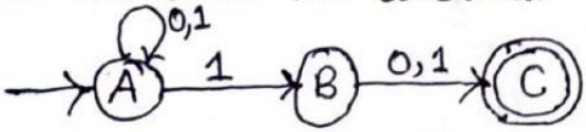
D is dead state so on getting any input goes to itself

Now on the basis of transition table equivalent DFA is as follows:-



Example3: Design an NFA for a language that accepts all strings over {0,1} in which the second last symbol is always '1'. Then convert it to its equivalent DFA.

Solution: Let first we construct NFA with its transition table:



δ	0	1
→A	A	A,B
B	C	C
*C	∅	∅

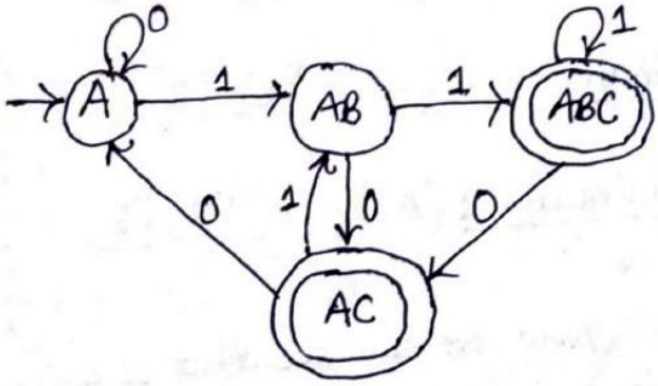
Now to convert into equivalent DFA, let we construct transition table for DFA using transition table of NFA.

δ	0	1
→A	A	AB
AB	AC	ABC
*AC	A	AB
*ABC	AC	ABC

Since DFA can not go more than one state so combined as one

Union operation of A and B on getting 0 from NFA table above. Similarly for ABC

AC and ABC are final states because they contain C which is final state in NFA



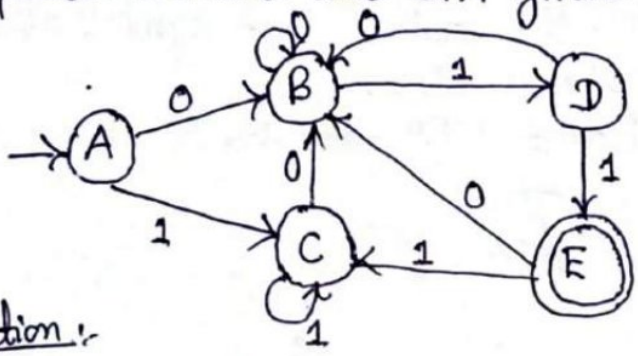
Note: Question मा NFA को figure दिएको भए DFA मा convert गर्न अगाडि पहिला दिएको figure अनुसार NFA को transition table बनाउने अनिमात्र convert गर्ने अर्को तरिका।

(*) Minimization of DFA:

Not in micro-syllabus but maybe imp.

Minimization of DFA is required to obtain the minimal version of any DFA which consists of the minimum number of states possible. Let we are designing a DFA, one person designed it with 5 states but another person designed with 4 states. But both perform the same operation and are equivalent. Obviously DFA with less states 4 states will be better with performance. Hence, minimization of DFA is required.

Example1: Minimize the DFA given below:



Solution:-

Let we construct transition table for given DFA.

δ	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C

Now based on transition table we construct equivalences as;

for constructing 0 equivalence we construct 2 seperate sets. one set for all non-final states and another for all final states.

0 Equivalence: {A,B,C,D} {E}

1 Equivalence: {A,B,C} {D} {E}

for 1 equivalence take sets more than one elements from 0 equivalence then check if they are 1 equiva--- to one another or not. If equivalent keep in same set otherwise seperate set. for eg. checking for A and B, A and B on getting 0 goes to B and B which are on same set in 0 equivalence. & A and B on getting 1 goes to C and D are which are also inside same set in 0 equivalence. So they can stay in same set. Similarly proceed for all except single sets.

Once we know A&B are on same set now for checking C can be done with any of them either A or B

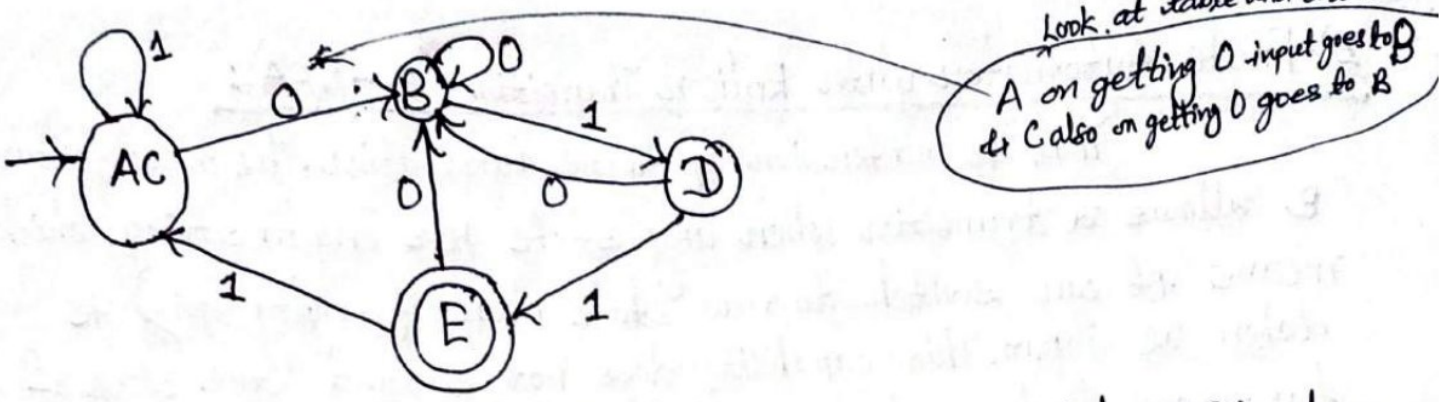
using 1 equivalence row

2 Equivalence: {A,C} {B} {D} {E}

3 Equivalence: {A,C} {B} {D} {E}

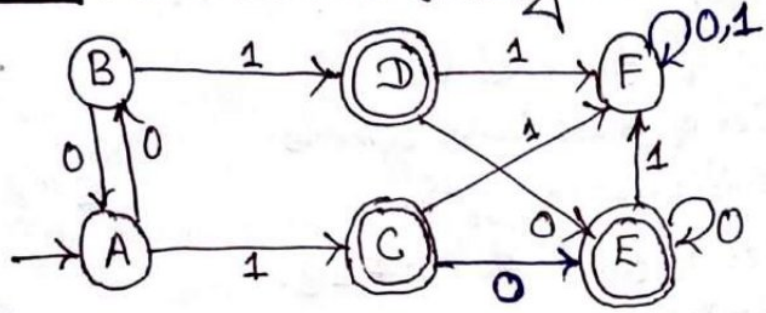
using 2 equivalence row

Now we can see that 2 equivalence and 3 equivalence are giving same results so we stop process and now we can construct minimized DFA combining states A & C as AC one.



Now we minimized 5 states DFA to 4 states DFA where both perform same operation.

Example 2: Minimize the following DFA:



Solution:-

S	0	1
→A	B	C
B	A	D
*C	E	F
*D	E	F
*E	E	F
F	F	F

Non-final states

final states

0 Equivalence: {A,B,F} {C,D,E}

1 Equivalence: {A,B} {F} {C,D,E}

2 Equivalence: {A,B} {F} {C,D,E}

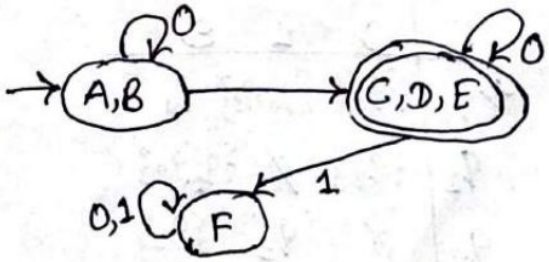
Now, we construct transition table for minimized DFA for easier.

S	0	1
→{A,B}	{A,B}	{C,D,E}
{F}	{F}	{F}
*{C,D,E}	{C,D,E}	{F}

A & B are combined now as one as AB one state. So, AB on getting 0 goes to B∪A = AB = {A,B}

A & B on getting 1 goes to CD, CD as new state as {C,D,E}

C,D,E is now one state



Note:- If there are any unreachable states (i.e, having no incoming arrow only outgoing arrow) then first we eliminate that state then we proceed as usual.

3) Finite Automaton with Epsilon Transition (ε-NFA):

This is extension of finite automation. The new feature ε allows a transition when any state gets empty string, which means it can switch to new state even if empty string is taken as input. This capability does not expand the class of languages that can be accepted by finite automata, but it does give some added "programming convenience".

Definition: A NFA with ε-transition is defined by five tuples $(Q, \Sigma, \delta, q_0, F)$ where;

Q = set of finite states

Σ = Set of finite input symbols

q_0 = Initial state, $q_0 \in Q$.

F = Set of final states; $F \subseteq Q$.

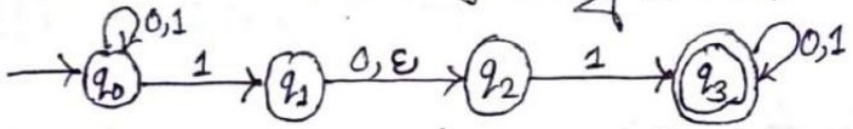
$\delta = Q \times \Sigma \cup \{\varepsilon\} \to 2^Q$, which is a transition function with two arguments.

→ a state Q

→ A member of $\Sigma \cup \{\varepsilon\}$ that is either an input symbol, or the symbol ε.

δ only different others are same as usual

Example: - Consider the following ε-NFA.



If at least one ε symbol is seen in diagram then we have to know that it is ε-NFA

Now,

$Q = \{q_0, q_1, q_2, q_3\}$

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

$q_0 = q_0$

$F = \{q_3\}$

& δ is given as;

δ	0	1	ε
→q_0	$\{q_0\}$	$\{q_0, q_1\}$	q_0
q_1	$\{q_2\}$	ϕ	$\{q_1, q_2\}$
q_2	ϕ	$\{q_3\}$	$\{q_2\}$
*q_3	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$

keep in mind that all states contain ε and every state on getting ε as input goes to itself.

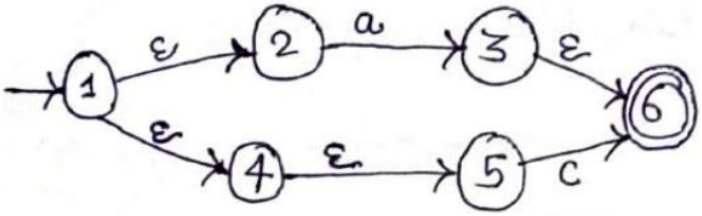
(ε नदिएको बेला यदि state मा जाने / ε दिएको ε भने दुवै state र ε आउदा जाने state दुवै मा जाने। for e.g. in fig. q_1 on getting ε goes to q_1 itself and q_2.

Notations for ε-NFA:- Notations for ε-NFA are also same as notations for DFA. [i.e, describing transition table and transition diagram taking examples of ε-NFA].

Epsilon Closure of a state:-(ε*):

Epsilon closure is the set of states that can be reached without reading any input symbol (i.e, only reading ε) from a particular state.

Example: Consider the ε-NFA below.



Then,

ε-closure(1) = {1, 2, 4, 5}

ε-closure(2) = {2}

ε-closure(3) = {3, 6}

ε-closure(4) = {4, 5}

ε-closure(5) = {5}

ε-closure(6) = {6}.

current state बाट ε मोर्दे हेर्दा कुन कुन state मा पुग्न सकिन्छ (यसको list) 1 on getting ε goes to itself so, 1 also counted

goes to itself. No other outgoing arrow on getting ε. So, only 2 itself.

Extended transition function of ε-NFA:-

The extended transition function of ε-NFA denoted by $\hat{\delta}$ is defined by;

i) Basis Step:

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

ii) Induction step: Let $w = xa$ be a string, where x is substring of w without last symbol a and $a \in \Sigma$ but $a \neq \varepsilon$.

Let $\hat{\delta}(q, x) = \{p_1, p_2, \ldots p_k\}$ i.e, p_i's are the states that can be reached from q following labeled x which can end with many ε & it can have many ε.

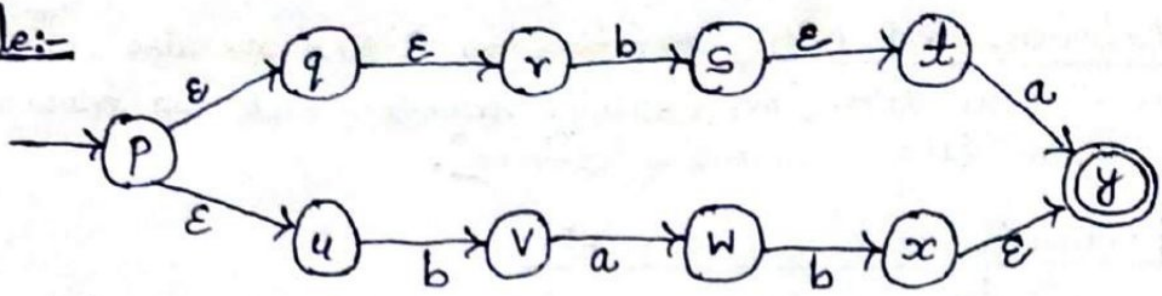
Also let,

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

i.e, union of $\delta(p_i, a)$ from $i=1$ to k

Then, $\delta(q, x) = \bigcup_{j=1}^{m} \varepsilon\text{-closure}(r_j)$.

Example:-



Now, let we compute for string ba.

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

(States that can be reached with input ε from state p)

2) Now we compute for b as;

$\delta(p,b) \cup \delta(q,b) \cup \delta(r,b) \cup \delta(u,b) = \{s, v\}$

(b is given input to all ε-closures(p) i.e p, q, r, u. Then we know δ(p, b) means state p on getting input b goes to nowhere. Similarly checking for others we get {s, v}.)

$\varepsilon\text{-closure}(s) \cup \varepsilon\text{-closure}(v) = \{s, t, v\}$

(i.e, states that can be reached from state s using input ε only. i.e, we get s, t) (Union) (states that can be reached using ε from v. only itself so v only.)

Similarly we compute for a as;

$\delta(s,a) \cup \delta(t,a) \cup \delta(v,a) = \{y, w\}$

(latest states s, t, v are used now for input a)

$\varepsilon\text{-closure}(y) \cup \varepsilon\text{-closure}(w) = \{y, w\}$

(final state is y now. यहाँ पर final state का set नहीं reject होगा)

⇒ Now, the final result set i.e {y, w} contains the one of the final state so, the string is accepted.

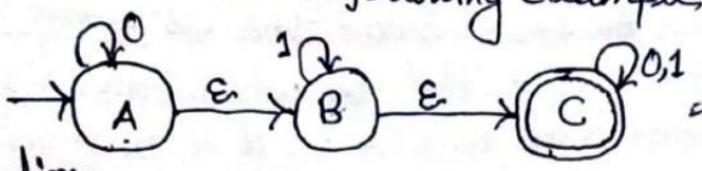
(ε* → means epsilon closure)

⊛ Removing Epsilon Transition using concept of Epsilon Closure OR Converting ε-NFA to its equivalent NFA:

The procedure for converting any ε-NFA to its equivalent NFA is as follows:-

1) For each state we have to check that where does the state goes on ε*. [ε* is the set of all states that can be reached from a particular state only by seeing the ε symbol].
2) Now set of states that we got in step (1), have to be checked on which state they go on getting a particular input.
3) Now the set of states that we got in step (2) are again checked on to which state do they go on ε* again.

Example: Let we take following example, to understand better:



This is an example for converting ε-NFA to NFA

Solution:

We have, For NFA:
Start state = A
Input = {0,1}

Since there is no ε in NFA so ε as input is eliminated in NFA

Now we follow process for each state and input as follows:-

$\delta_N(A,0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(A),0))$
$= \varepsilon\text{-closure}(\delta(\{A,B,C\},0))$
$= \varepsilon\text{-closure}(\{A,C\})$
$= \{A,B,C\}$

δ_N or simply δ.

for state A with input 0

Process step 1 completed in this line since ε-closure(A) is found & states A is found & states on getting ε-input are listed.

now A, B, C are checked where they go on getting input zero. A → A, B → ∅, C → C i.e, {A,C}

finally ε* for C done
ε*(A) = A,B,C.
ε*(C) = C
therefore ε*(A,C) = {A,B,C}

$\delta_N(A,1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(A),1))$
$= \varepsilon\text{-closure}(\delta(\{A,B,C\},1))$
$= \varepsilon\text{-closure}(\{B,C\})$
$= \{B,C\}$

for state A with input 1

$\delta_N(B,0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(B),0))$
$= \varepsilon\text{-closure}(\delta(\{B,C\},0))$
$= \varepsilon\text{-closure}(\{C\})$
$= \{C\}$

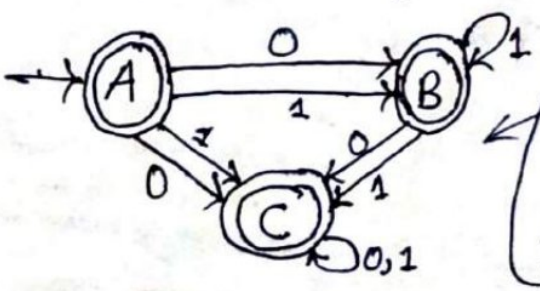
$\delta_N(B,1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(B),1))$
$= \varepsilon\text{-closure}(\delta(\{B,C\},1))$
$= \varepsilon\text{-closure}(\{B,C\})$
$= \{B,C\}$

$\delta_N(C,0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(C),0))$
$= \varepsilon\text{-closure}(\delta(\{C\},0))$
$= \varepsilon\text{-closure}(\{C\})$
$= \{C\}$

$\delta_N(C,1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(C),1))$
$= \varepsilon\text{-closure}(\delta(\{C\},1))$
$= \varepsilon\text{-closure}(\{C\})$
$= \{C\}$

⇒ Now, we can easily draw transition table and diagram for NFA as follows:-

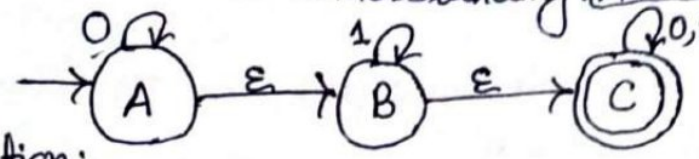
δ	0	1
*→A	{A,B,C}	{B,C}
*B	{C}	{B,C}
*C	{C}	{C}



Here A, B, C all are final states because. In NFA the final states are the states that can reach to final state of ε-NFA only by input ε

Converting ε-NFA to its equivalent DFA:

Converting ε-NFA process to its equivalent DFA is almost similar to converting equivalent DFA except that, while converting ε-NFA to NFA we used same start state 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. Let we take example below for clear understanding. Also step (i) is eliminated in this only step (ii) and (iii) applied.



Solution:

We have for DFA:

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

Input = {0, 1}

(δ' or δ_D can also be written as notation)

different here is start state, also only process (ii) and (iii) done of checking new state each time.

Now,

$\delta'(\{A,B,C\}, 0) = \varepsilon\text{-closure}(\delta(\{A,B,C\}, 0))$
$= \varepsilon\text{-closure}(\{A, C\})$
$= \{A, B, C\}$

$\delta'(\{A,B,C\}, 1) = \varepsilon\text{-closure}(\delta(\{A,B,C\}, 1))$
$= \varepsilon\text{-closure}(\{B, C\})$
$= \{B, C\}$.

new states after checking state {A,B,C} are {A,B,C} & {B,C}. Since {A,B,C} already checked so, {B,C} now.

$\delta(\{B,C\}, 0) = \varepsilon\text{-closure}(\delta(\{B,C\}, 0))$
$= \varepsilon\text{-closure}(\{C\})$
$= \{C\}$

$\delta(\{B,C\}, 1) = \varepsilon\text{-closure}(\delta(\{B,C\}, 1))$
$= \varepsilon\text{-closure}(\{B, C\})$
$= \{B, C\}$

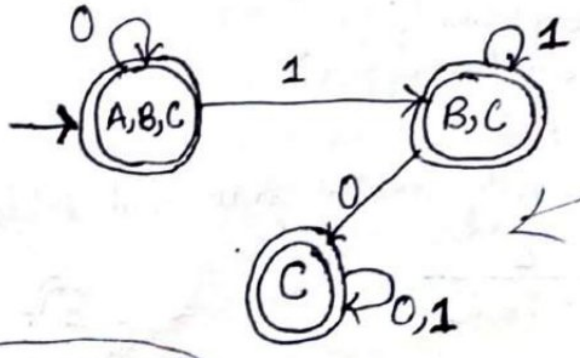
{C} of {B,C} are new states, but {B,C} already checked so remaining {C} now.

$\delta(C, 0) = \varepsilon\text{-closure}(\delta(\{C\}, 0))$
$= \varepsilon\text{-closure}(\{C\})$
$= \{C\}$

$\delta(C, 1) = \varepsilon\text{-closure}(\delta(\{C\}, 1))$
$= \varepsilon\text{-closure}(\{C\})$
$= \{C\}$

No new states to be checked, all are checked so we stop here.
Now we can draw transition table and diagram as follows:-

δ	0	1
*→{A,B,C}	{A,B,C}	{B,C}
*{B,C}	{C}	{B,C}
*{C}	{C}	{C}



If ∅ comes in transition table also draw it as a state. Since ∅ means dead state in DFA

C was final state in ε-NFA. So, all states containg C are final states in DFA

Finite State Machines with output:

1) Mealy Machine:

Mealy machine is defined by the six tuples $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where;

Q = finite set of states.

Σ = finite non-empty set of input alphabets.

Δ = The set of output alphabets.

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

λ = Output function: $\Sigma \times Q \to \Delta$

q_0 = Initial state/start state.

read as del

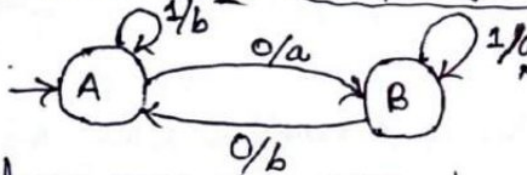
read as delta

this is output which is new here

This λ is also different here, Inplace of final state F here we have out put function λ.

This output function Δ is associated to input Σ & current state Q.

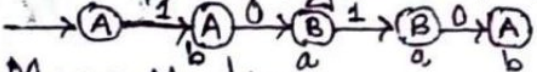
Example:



This means state A on getting input 1 goes to A itself producing output b.

Mealy machine मा output यसरि transition मा input संग slash गरि राखिन्छ र output depend गर्छ input र current state दुवैमा

#Let we pass string 1010 and see output.



. Here we can see that on passing string 1010 we get output as baab [i.e, for n input we get n outputs]

2) Moore Machine:

Moore machine is also defined by six tuples $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where;

Q = finite set of states.

Σ = finite non-empty set of input alphabets.

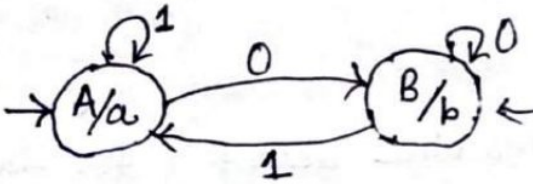
Δ = The set of output alphabets.

δ = Transition function: $Q \to \Delta$

q_0 = Initial state/Start state.

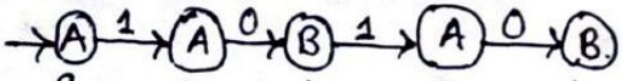
यसमा यो मात्र फरक हो अरु same. This means output function Δ is associated with current state Q only.

Example:



Moore machine मा output यसरि State संग slash गरि राखिन्छ र output current state मा मात्र depend गर्छ, यति फरक हो

Let we pass string 1010 and see output.



moore machine depends on current state only so without any input it can produce output on going in any state.

Here we can see that on passing string 1010 we get output as aabab. [i.e, for n inputs we get n+1 outputs in moore machine].

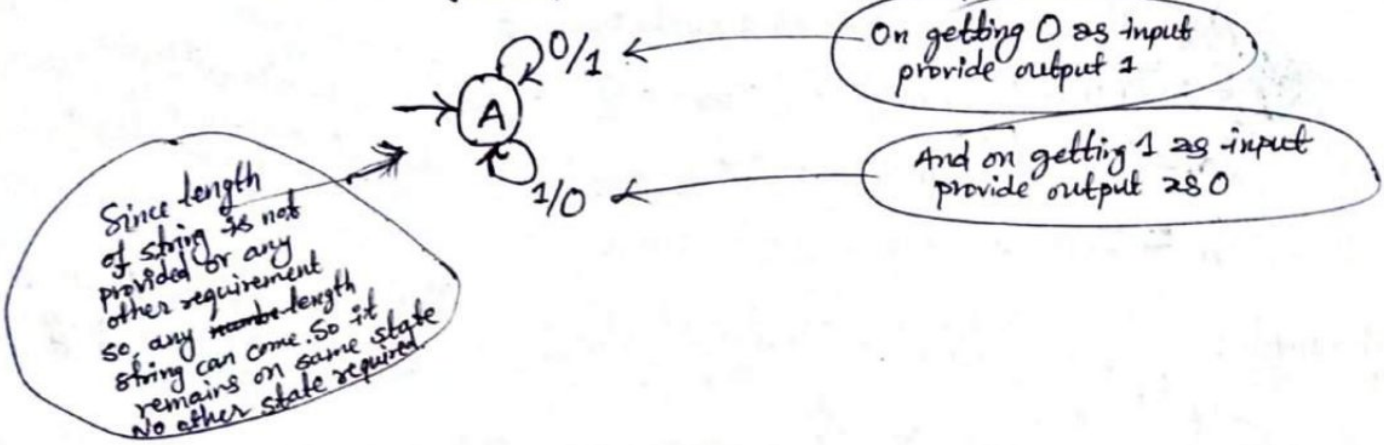
⊛Construction of Mealy Machines [Examples]:-

Remember that there are no any final states in melay machine

Example 1: Construct a Mealy Machine that produces the 1's complement of any binary input string.

Solution:-

We know that 1's complement of any binary string is converting 1's to 0's and 0's to 1's. So, the required Mealy Machine is as follows;



Example 2: Construct a Melay Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

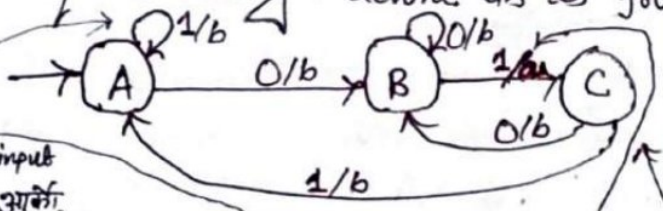
Solution:

$\Sigma = \{0, 1\}$ — inputs

$\Delta = \{a, b\}$ — outputs

b taken for all outputs other than a when 01 is not encountered

The required Melay Machine is as follows:



① first draw 3 states for given input sequence 01. then complete it as we did for DFA.

② A मा already 0 input गियो / 1 आउदा आर्को state मा जस्ट string लाई (ie, 01 sequence) अस्त गर्दै So त्यसै state मा रहन्छ।

③ Similarly C मा 0 र 1 आउदा already 01 पाइसकेपछि So, first चोटि 0 र 1 input आएर जुन state मा गयो त्यहि state मा जाने

④ B input मा B पुग्यो र 1 input मा C पुग्यो So, 01 sequence मिलेको ले question अनुसार C मा पुग्दा 'a' output गरिरहेको र बाकि सबैमा b

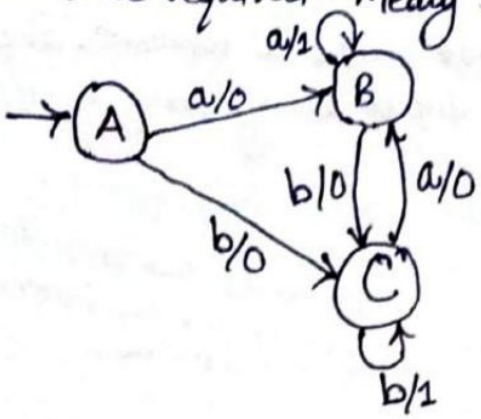
Example 3:- Design a Mealy Machine accepting the language consisting of strings from Σ^*, where $\Sigma = \{a, b\}$ and the strings should end with either aa or bb.

Solution

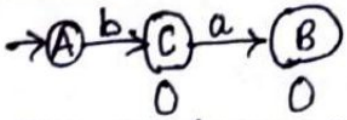
Σ^* means all any length of strings from $\Sigma = \{a, b\}$.

Our string should end with aa or bb. So, let if we get sequence aa or bb then we print output as 1. otherwise 0.

Now the required Mealy Machine is as follows:-

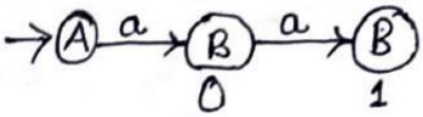


Let we check for string ba



Here outputs are 00 so no any 1's appeared. This means sequence aa or bb is not encountered.

Let similarly we check for string aa



Here outputs are 01 so, the 1 in it shows once the sequence aa or bb is encountered. [If we see more than 1 times let n then it means sequence encountered n times.]

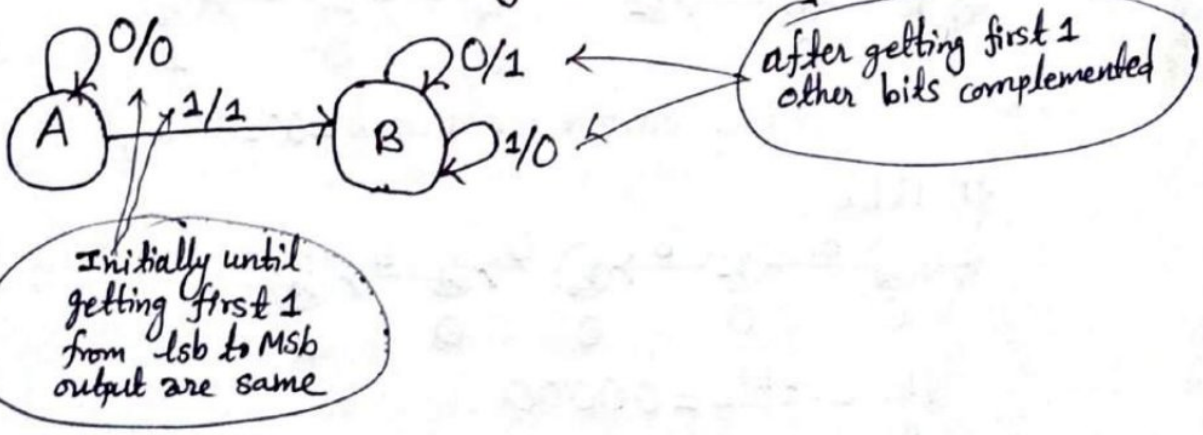
Example 4: Construct a Mealy Machine that gives 2's complement of any binary input. (Assume that the last carry bit is neglected).

Solution: We know that 2's complement = 1's complement + 1.

Let we have 10100 then 2's complement = (MSB ← LSB) 01011 + 1

2's = 01100

Now, if we look in this example. we found that on going from LSB towards MSB of 2's complement (i.e, 01100) and given string (i.e, 10100) we can see that until we get first 1 all digits are same (i.e, 100) but after that 1 is changed to 0 and 0 is changed to 1 (i.e, rest bits are complemented). This gives us idea to build our mealy machine.



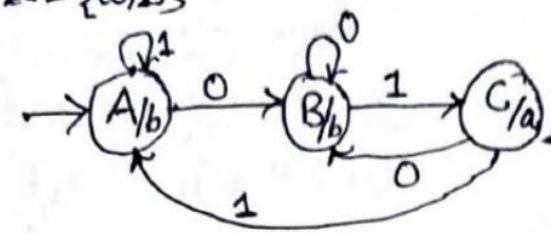
Construction of Moore Machine:

Example 1: Construct a Moore Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

Solution:-

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

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



Outputs are associated to states in moore machine.

prints a only on getting sequence 01 otherwise prints b.

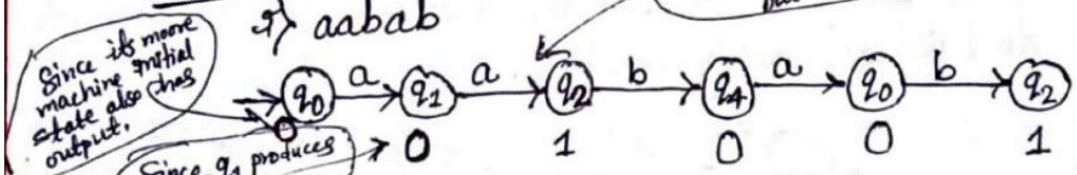
Example 2: For the following Moore Machine the input alphabet is $\Sigma = \{a,b\}$ and the output alphabet is $\Delta = \{0,1\}$. Run the following input sequences and find the respective outputs.

i) aabab ii) abbb iii) ababb.

States	a	b	Outputs
→ q_0	q_1	q_2	0
q_1	q_2	q_3	0
q_2	q_3	q_4	1
q_3	q_4	q_4	0
q_4	q_0	q_0	0

Solution:-

i) aabab



q_2 on getting a goes to q_2 but output of q_2 is 1

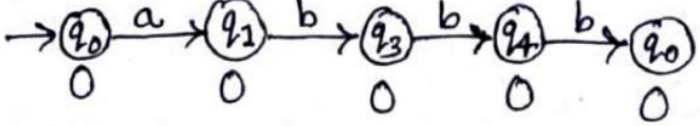
Since its moore machine initial state also has output.

Since q_1 produces output 0

Hence, output for aabab = 001001

If it was mealy machine then 0 output for initial state would not be here.

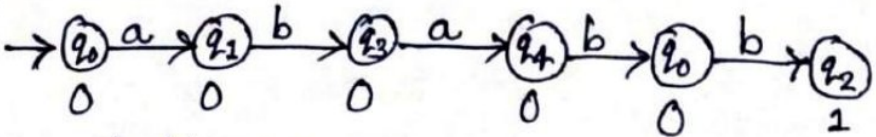
ii) abbb



Hence, abbb = 00000.

for moore machine with n inputs there are n+1 outputs

iii) ababb.



Hence, ababb = 000001.

Mealy Machine vs. Moore Machine

Mealy Machine	Moore Machine.
i) Output depends upon both the present state and present input.	i) Output depends upon only the present state.
ii) Generally it has fewer states than Moore Machine.	ii) Generally it has more states than Mealy Machine.
iii) For mealy machine with n inputs there are n outputs.	iii) For moore machine with n inputs there are n+1 outputs.
iv) The value of the output function is a function of transitions and the changes when the input logic on the present state is done.	iv) The value of output function is a function of the current state and the changes at the clock edges whenever state changes occur.
v) Mealy machines react faster to inputs since they react the same no. of clock cycles.	v) Moore machines react slower to inputs as they react generally one clock cycle later.

Language → Set of strings. Eg. $\Sigma = \{0,1\}$

L_1 = set of all strings of length 2.
= $\{00, 01, 10, 11\}$ (i.e, finite).

L_2 = set of all strings of length 3.
= $\{000, 001, 010, 011, 100, 101, 110, 111\}$ (i.e, finite).

L_3 = set of all strings that begin with 0.
= $\{0, 00, 01, 000, 001, 0000, \ldots\}$ (i.e, infinite).

Powers of Σ: Let $\Sigma = \{0,1\}$

Σ^0 = Set of all strings of length 0 : $\Sigma^0 = \{\epsilon\}$ (epsilon. that means empty)
Σ^1 = set of all strings of length 1 : $\Sigma^1 = \{0,1\}$
Σ^2 = set of all strings of length 2: $\Sigma^2 = \{00, 01, 10, 11\}$.
⋮
Σ^n = set of all strings of length n.

& Σ^* = set of all strings of any length that can be formed using Σ.