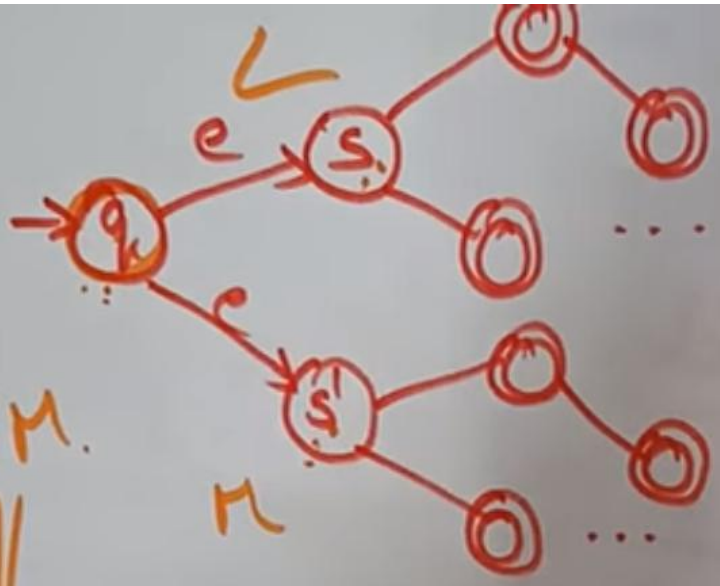
Mealy Machines

⇒ There are two types of finite state machines that generate output-

1. Mealy Machine
2. Moore machine

Mealy Machine :-

A mealy machine is a FSM whose output depends on the present state as well as the present input.



$L \cup M$.

M

→ let L & M be lang of R.E R & S.

Then R+S is a R.E whose l

① closure under concatenation & kleen

The same idea can

① clos anguage is L

e intersect

Let

⇒ R, S are regular expression whose language is L

⇒ R^* is a regular expression whose language is L^*

) closure under intersection:

If L & M are regular languages, so their intersec
is defined by:

$$L \cap M = \{ w : w \in L \text{ and } w \in M \}$$

using DeMorgan's law

$$L \cap M = (L^c \cup M^c)^c$$

Solution :-

Contradiction

① At first, we assume that L is regular and n is the no: of states.

② Let $w = a^n b^n$. Thus $|w| = 2n \geq n$

a.a b.b.b
2×3×3

③ By pumping lemma, Let $w = xyz$, where $|xy| \leq n$

④ Let,

$x = a^p$
$y = a^q$
$z = a^r b^n$

where $p+q+r = n$,

$p \neq 0,\ q \neq 0,\ r \neq 0$

Thus $|y| \neq 0$

$xy^n z \notin L$

#11.

FLAT

DFA

① $(Q, \Sigma, \delta, q_0, F)$

$\delta : Q \times \Sigma \to Q$

② Transition leads to one (unique state)

③ Back Tracking is acceptable

④ $Q = \{q_0, q_1\}$ $\Sigma = \{0, 1\}$ $q_0 \xrightarrow{0/1}$, $q_1 \xrightarrow{0/1}$

⑤ DFA Required more space

⑥ Empty string Transitions (ϵ-moves) not required

⑦ practical implementation is feasible

⑧ Transition Function $\delta(q,a) = \{q\}$ (s/s)

m of DFA is deterministic

NFA

① $(Q, \Sigma, \delta, q_0, F)$

② $\delta : \Sigma \times Q \to 2^Q$

Transition may leads to multiple states

③ Back Tracking may not required

④ $Q = \{q_0, q_1\}$ $\Sigma = \{0, 1\}$

$q_0 \to 1$, $q_1 \to 0$

⑤ It requires less space

⑥ ϵ – Transitions can be allowed

⑦ Not feasible

⑧ $\delta(q,a) = \{q_1, q_2 \dots\}$ (multiple states)

⑨ NFA construction is very simple.

Formal Definition :

A TM can be formally described as 7-tuples $(Q, X, \Sigma, \delta, T, B, F)$ where :-

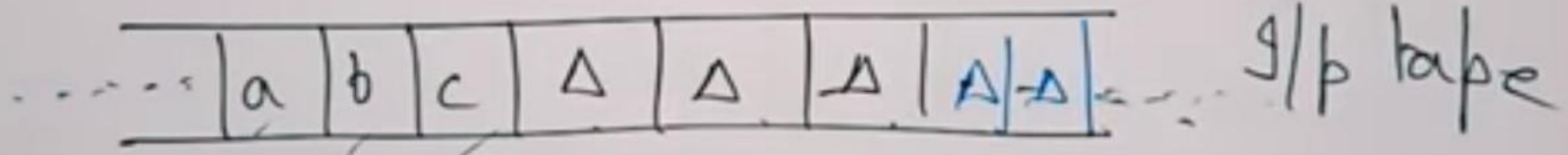
- Q is a finite set of states
- X is the tape alphabet
- Σ is the i/p alphabet
- δ is a transition function;

$$\delta : Q \times X \rightarrow Q \times X \times \{\text{left shift, Rig}$$

- q_0 is the initial state
- B is the Blank symbol

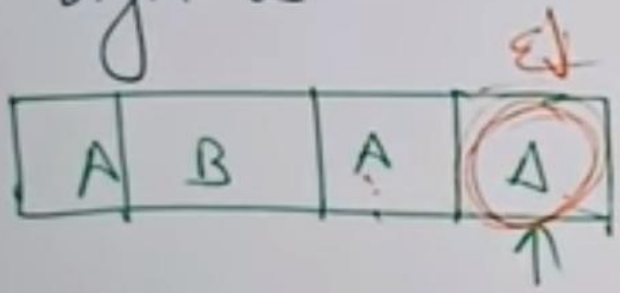
Basic Model of Turning Machine

① The i/p tape is having an infinite no: of cells, each cell containing one i/p symbol. The empty tape is filled by blank characters



② The finite control & the tape head which is responsible for reading the current i/p symbol. The tape head can move to left to right.

The move will be $\delta(q_2, a) = \delta(q_3, A, R)$ which means it will go to state q_3, replaced 'a' by 'A' and head will move to right as:

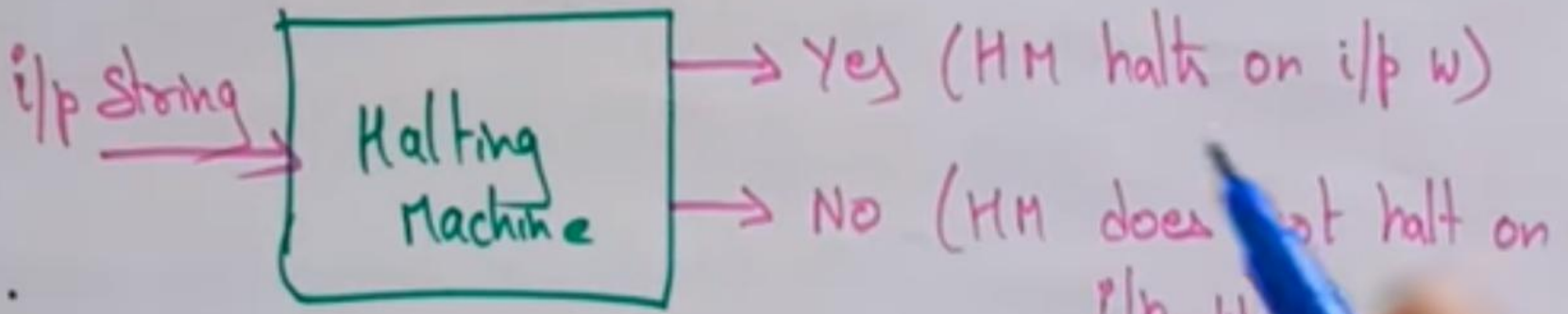


The move $\delta(q_3, \Delta) = (q_4, \Delta, S)$ which means it will go to state q_4 which is HALT state which is accept state for any I

We will call this T.M as a Halting m/c that produce a 'yes' or 'no' in a finite amount of time.

→ If the halting m/c finishes in a finite amount of time, o/p comes as 'yes' otherwise as 'no'.

The block dig of Halting m/c :-



#11.

FLAT

DFA

① $(Q, \Sigma, \delta, q_0, F)$

$\delta : Q \times \Sigma \to Q$

② Transition leads to one (unique state)

③ Back Tracking is acceptable

④ $Q = \{q_0, q_1\}$ $\Sigma = \{0, 1\}$ $q_0 \xrightarrow{0,1}$ $q_1 \xrightarrow{0,1}$

⑤ DFA Required more space

⑥ Empty string Transitions (ε-moves) not required

⑦ Practical implementation is feasible

⑧ Transition Function $\delta(q,a) = \{q\}$ (s/s)

⑨ Construction of DFA is difficult

NFA

① $(Q, \Sigma, \delta, q_0, F)$

② $\delta : \Sigma \times Q \to 2^Q$

Transition may leads to multiple states

③ Back Tracking may not required

④ $Q = \{q_0, q_1\}$ $\Sigma = \{0, 1\}$ $q_0 \to 1$, $q_1 \to 0$

⑤ It requires less space

⑥ ε-Transitions can be allowed

⑦ Not feasible

⑧ $\delta(q,a) = \{q_1, q_2 \ldots\}$ (multiple states)

⑨ NFA construction is very simple.

Equivalence of two Finite Automata

A ≠ B

Steps to identify equivalence :-

① For any pair of states (q_i, q_j) the transition for input $a \in \Sigma$ is ~~identi~~ 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 & other is final state

② If initial state is final state of one automata,

Minimization of DFA

The minimization of DFA means reducing the number of states from a given FA.

1: Remove all the states that are unreachable from initial state via any set of transition of DFA.

2: Draw the transition table for all pair of states.

3: Now split the transition table into two tables T_1 & T_2.

T_1 contains all final states

T_2 contains non-final states

Minimization of DFA

The minimization of DFA means reducing the number of states from a given FA.

1: Remove all the states that are unreachable from initial state via any set of transition of DFA.

2: Draw the transition table for all pair of states.

3: Now split the transition table into two tables T_1 & T_2.

T_1 contains all final states

T_2 contains non-final states

4. Find similar rows from T_1 such that:

$$\delta(q, a) = p$$
$$\delta(r, a) = p$$

i.e. find the two states which have same value of a and b and remove one of them.

5. Repeat step 3 until we find no similar rows available in T_1

6. Repeat step 3 & step 4 for table T_2 also.

7. Now combaine the reduced T_1 & T_2 tables.
i.e. the final transition table of minimized DFA.

Mealy Machines

⇒ There are two types of finite state machines that generate output-

1. Mealy Machine
2. Moore machine

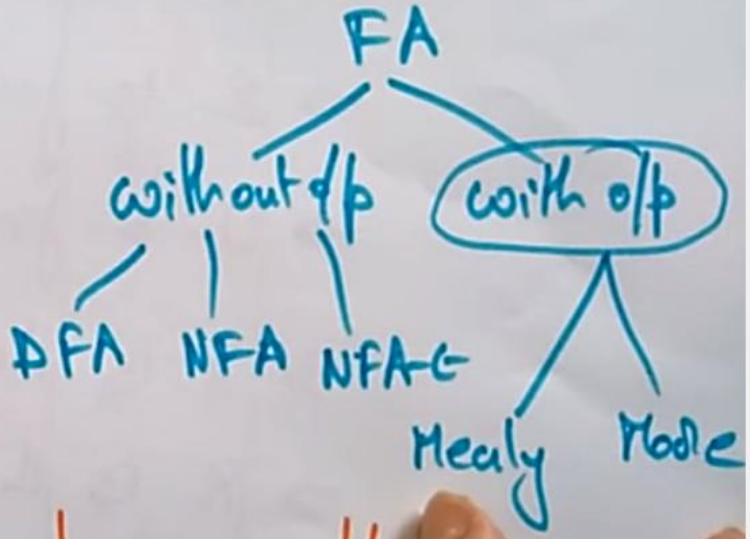
Mealy Machine :-

A mealy machine is a FSM whose output depends on the present state as well as the present input.

Mealy Machines

⇒ There are two types of finite state machines that generate output-

① Mealy Machine
② Moore machine



Mealy Machine :-

A mealy machine is a FSM whose outp
depends on the present state as well as the ent
input.

It can be described by 6 tuples $(Q, \Sigma, O, \delta, X, q_0)$ where -

→ Q is a finite set of states

→ Σ is finite set of symbols called i/p alphabet

→ O is finite set of symbols called o/p alphabet

→ δ is the i/p transition function where $\delta: Q \times \Sigma \to Q$

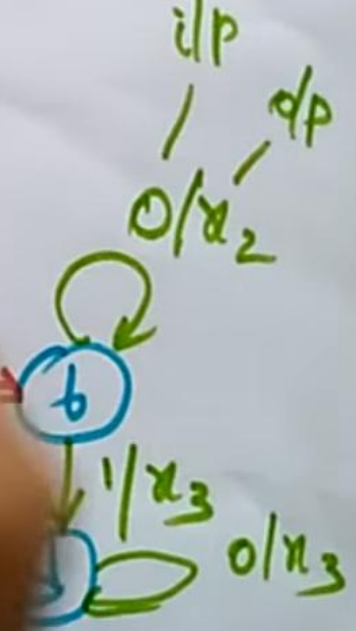
→ X is the o/p transition function where $X: Q \times \Sigma \to O$

→ q_0 is the initial state from where any input is processed ($q_0 \in Q$)

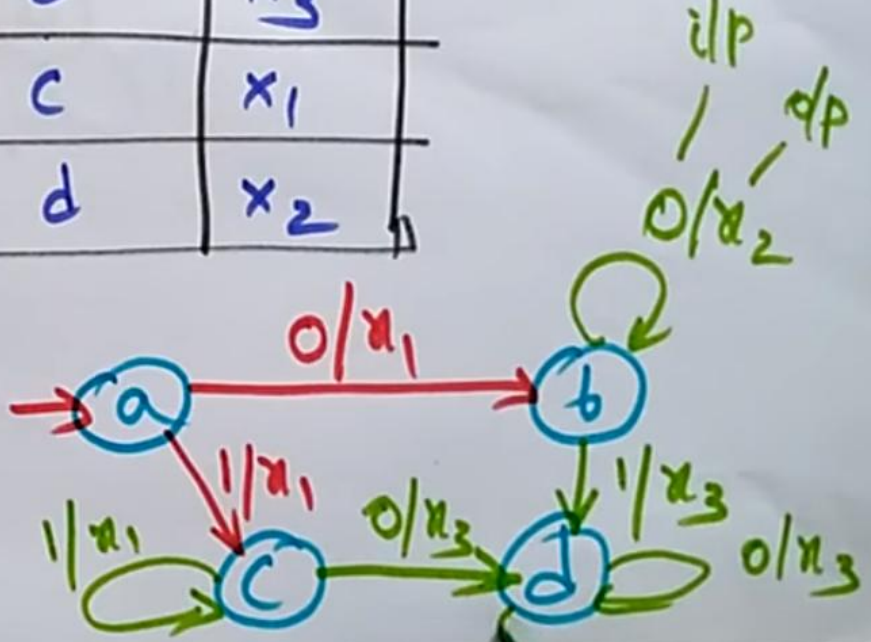
Example 1: State table of a Mealy Machine

Present state	Next state			
	input = 0		input = 1	
	state	o/p	state	o/p
→ a	b	x_1	c	x_1
b	b	x_2	d	x_3
c	d	x_3	c	x_1
d	d	x_3	d	x_2

Present state	Next state			
	input = 0		input = 1	
	state	o/p	state	o/p
→ a	b	x_1	c	x_1
b	b	x_2	d	x_3
c	d	x_3	c	x_1
d	d	x_3		



Present state	Next state			
	input = 0		input = 1	
	state	o/p	state	o/p
→ a	b	x_1	c	x_1
b	b	x_2	d	x_3
c	d	x_3	c	x_1
d	d	x_3	d	x_2



Mode Machine

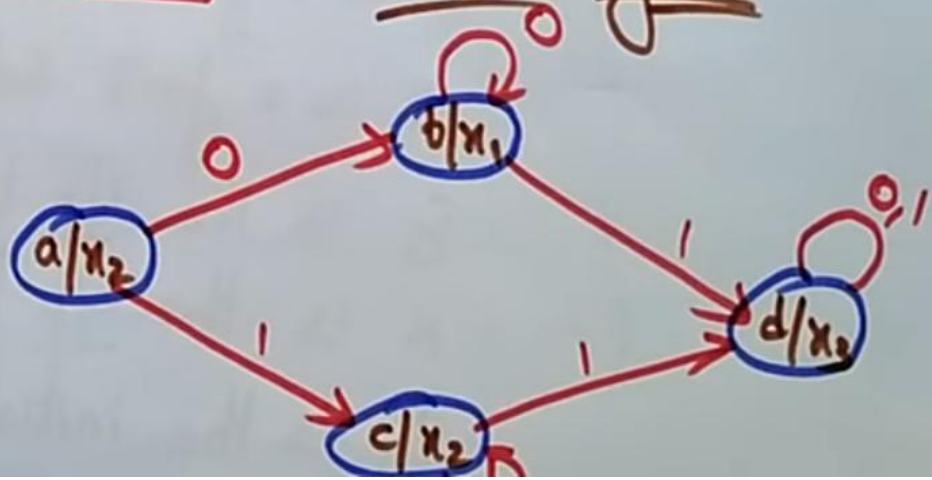
Mode Machine is a FSM whose output depend on only the present state. A Mode Machine can be described by a 6 tuple $(Q, \Sigma, O, \delta, x, q_0)$ where

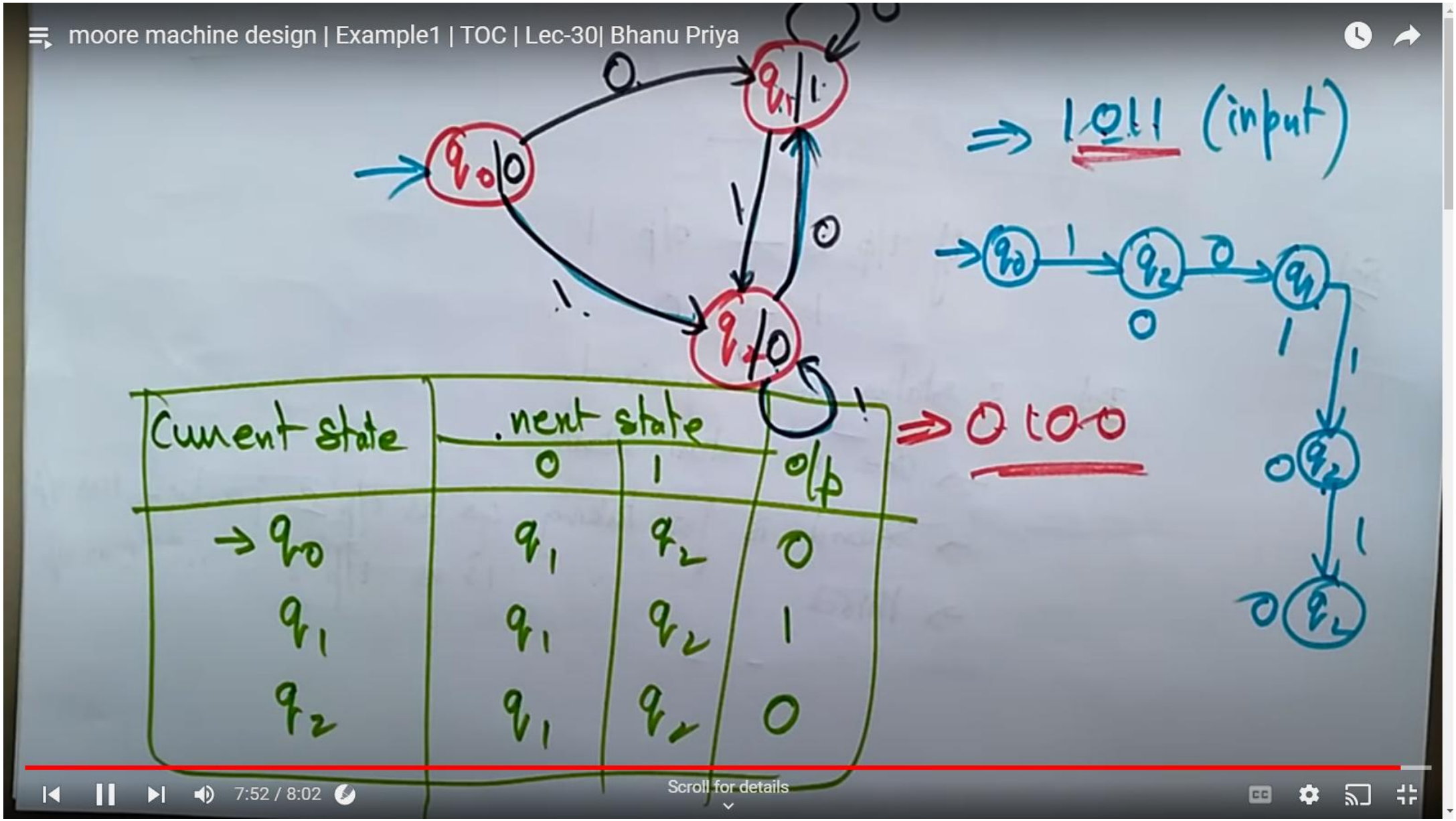
- Q is a finite set of states
- Σ is a finite set of symbols called i/p alphabet
- O is a finite set of symbols called o/p alphabet
- δ is the i/p transition function where $\delta: Q \times \Sigma \to Q$
- x is the o/p transition function where $x: Q \to O$
- q_0 is the initial state from where any i/p is

Eg:- State table of Moore M/c

Present state	Next state		output
	Input = 0	Input = 1	
→ a	b	c	x_2
b	b	d	x_1
c	c	d	x_2
d	d	d	x_3

State diagram




moore machine design | Example1 | TOC | Lec-30| Bhanu Priya
⇒ 1011 (input)
⇒ 0100
Current state
next state
0
1
o/p
→ q0 q1 q2 0
q1 q1 q2 1
q2 q1 q2 0
7:52 / 8:02
Scroll for details

Regular Expression

→ The language accepted by finite automata can easily described by simple expression called Regular Expression.
It is most effective way to represent any language.

→ the languages accepted by some regular Expression are refered to as Regular languages.

→ A regular expression can also be described as a sequenc of pattern that defines a string.

For Instance:

→ In R.E x^* means zero or more occurence of x.

It can generate $\{\epsilon, x, xx, xxx, \ldots\}$

→ In R.E x^+ means one or more occurence of x.

It can generate $\{x, xx, xxx, \ldots\}$

operations on Regular language:

① **Union**: If L & M are two regular lang their union L∪M

② Intersection : If L & M are two R.L & there intersection is also an intersection.

$L \cap M = \{ st \mid s \text{ is in } L \text{ \& } t \text{ is in } M \}$

$L \to s$
$M \to t$
$\Rightarrow L$

③ Kleen closure : If L is R.L then its kleen closure L^* also be R.L.

L^* = zero or more occurence of lang L.

Conversion of RE to FA

To Convert RE to FA, we use a method called Subset method.

Step 1 :- Design a transition dig for given R.E, usin
NFA with ϵ-moves.

Step 2 :- Convert NFA with ϵ to NFA without ϵ

Step 3 :- Convert the obtained NFA to Equivalent DFA.

Conversion of RE to FA

Convert RE to FA, we use a method called

RE → NFA-ϵ → NFA ≃ DFA

bset method.

Step 1 :- Design a transition dig for given R.E, using NFA with ϵ-moves.

Step 2 :- Convert NFA with ϵ to NFA without ϵ

Step 3 :- Convert the obtained NFA to Equivalent DFA.

Closure Properties of Regular Languages

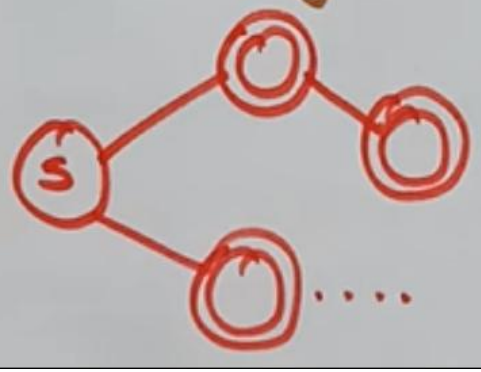
① closure under Union :-

If L & M are R.L, so their union is defined by :

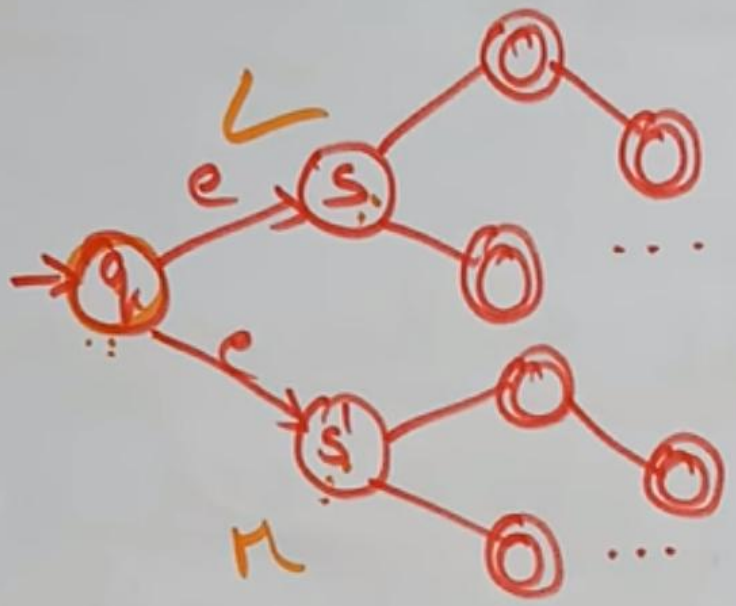
$$L \cup M = \{w : w \in L \text{ or } w \in M\}$$

Let A_L & A_M two finite automata accepting L & M respec

A_L:



A_M:



AL

AM

LUM

= ω / ωεL
ωεM.

→ Let L & M be lang of R.E R & S. respectively

Then (R+S) is a R.E whose language is (LUM.

② closure under concatenation & kleen closure :

⇒ R, S are regular expression whose language is L

⇒ R^* is a regular expression whose language is L^*

) closure under intersection:

If L & M are regular languages. So their intersection is defined by:

$$L \cap M = \{ w : w \in L \text{ and } w \in M \}$$

using DeMorgan's law

$$L \cap M = (L^c \cup M^c)^c$$

④ closure under Difference:

Let L & M be two languages. Then their difference is defined by:

$$L - M = \{ w : w \in L \ \& \ w \notin M \}$$

using Demorgan's laws:

$$(L - M) = L \cap M^c$$

Thus, there must be a finite automata for $(L - M)$

(5) Closure under ~~Concatenation~~ Complementation :-

The complement of a language L (with respect to an alphabet Σ such that Σ^* contains L) is $\Sigma^* - L$.

⇒ Since Σ^* is surely regular, the complement of a regular language is always regular.

⑥ Closure under Reversal :

Given lang L, LR is the set of strings whose reverse is in L.

$L = \{0, 01, 100\}$; $LR = \{0, 10, 001\}$

Basis : If E is a symbol a, ϵ, or ϕ, then $ER = E$.

Induction : If E is

→ $F + G$, then $ER = FR + GR$

→ FG, then $ER = GRFR$

→ F^*, then $ER = (FR)^*$

Pumping Lemma for Regular Grammars

Theorem:

Let (L) be a R.L. Then there exists a constant 'p' such that for every string ω in L -

$$|w| \geq p$$

We can break w into three strings, $\omega = xyz$ such that -

- $|y| > 0$
- $|xy| \leq p$
- For all $k \geq 0$ the string xy^k

Applications of Pumping Lemma

Pumping lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

- If L is regular, it satisfies pumping lemma
- If L does not satisfy pumping lemma, it is non regular.

To Poove that a language is not Regular using pumping lem
follow the steps

→ At first, we have to assume that L is regular
→ So, the pumping lemma should hold for L
→ It has to have a pumping length (say P)
→ All strings longer than P can be pumped $|w| \geq P$
→ Now find a string 'w' in L such that $|w| \geq P$
→ Divide w into xyz
→ Show that $xy^iz \notin L$ for some i

Follow the steps

→ At first, we have to assume that L is regular

→ So, the pumping lemma should hold for L

→ It has to have a pumping length (say P)

→ All strings longer than P can be pumped $|w| \geq P$

→ Now find a string 'w' in L such that $|w| \geq P$

→ Divide w into xyz

→ Show that $xy^iz \notin L$ for some i

$w = \underbrace{x}\underbrace{y}\underbrace{z}$

3 parts

$|xy^2z| \notin L$

⇒ Then consider all ways that W can be divide into xyz.

⇒ Show that none of these can satisfy all the 3 pumping conditions at the same time.

⇒ ~~b/s~~ W cannot be pumped == CONTRADICTION

Solution :-

Contradiction

① At first, we assume that L is regular and n is the no: of states.

② Let $w = a^n b^n$. Thus $|w| = 2n \geq n$

a.a b.b.b
2×3×3

③ By pumping lemma, Let $w = xyz$, where $|xy| \leq n$

④ Let,

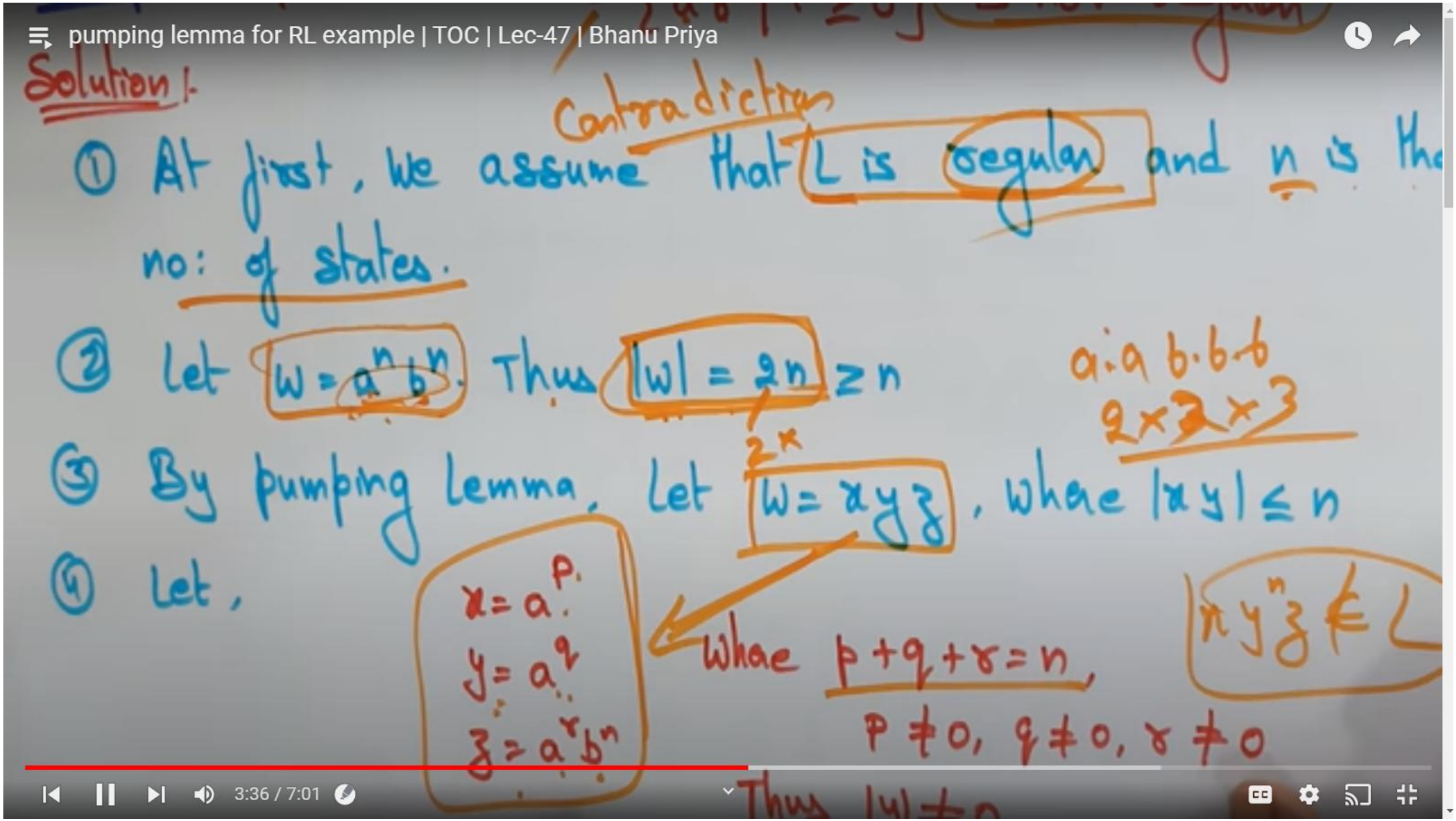
$x = a^p$
$y = a^q$
$z = a^r b^n$

where $p+q+r = n$,

$p \neq 0, q \neq 0, r \neq 0$

Thus $|y| \neq 0$

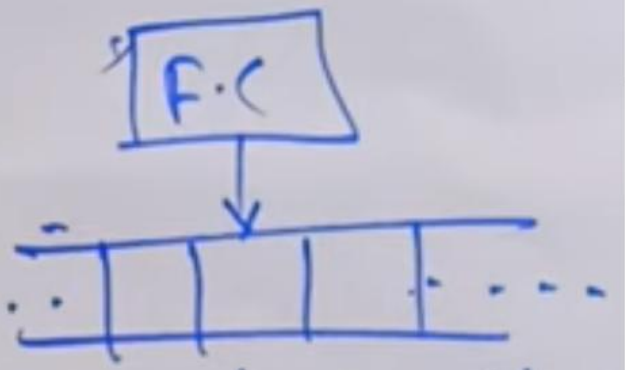
$xy^n z \notin L$


pumping lemma for RL example | TOC | Lec-47 | Bhanu Priya
Solution :-
Contradiction
① At first, we assume that L is regular and n is the no: of states.
② Let w = a^n b^n. Thus |w| = 2n ≥ n
a.a b.b.b
2×3×3
③ By pumping lemma, Let w = xyz, where |xy| ≤ n
2^k
④ Let,
x = a^p
y = a^q
z = a^r b^n
where p+q+r = n,
p ≠ 0, q ≠ 0, r ≠ 0
Thus |y| ≠ 0
xy^n z ∉ L
3:36 / 7:01

Recursive & Recursively Enumerable languages

→ The T.M may

- ✱ Halt & accept the input
- ✱ Halt & reject the input, or
- ✱ Never halt / loop



Recursively Enumerable lang :-

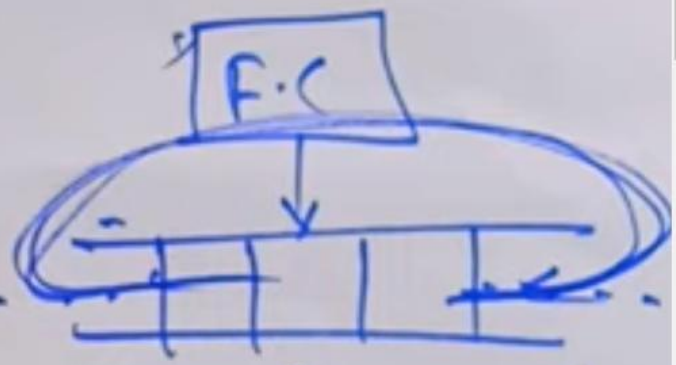
a T.M for a language which accep

Every st

Recursive & Recursively Enumerable language

→ The T.M may

- ★ Halt & accept the input
- ★ Halt & reject the input, or
- ★ Never halt / loop



"abc" ⇒ accept & Halt

Halt & reject

Recursively Enumerable lang :-

There is a T.M for a language which accept every string otherwise not.

Recursive language L

There is a T.M for a language which halt on Every string

Recursive language

There is a T.M for a language which halt on Every string

Recursive lang

Halt on every string

recursive enumerable lang

Accept every string not

⇒ Halting problem is undecidability. It is not a p...

asks question :- "is it possible to tell whether a given m/c will halt for some given i/p"

$\Sigma = \{\omega\}$

Eg: input: A T.M & i/p string (w)

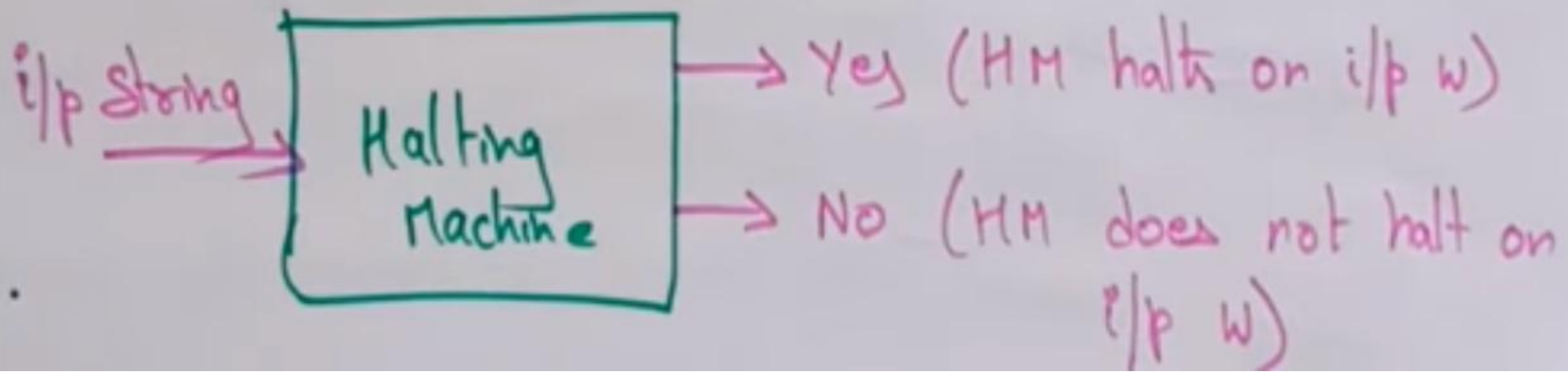
Problem: Does the TM finish computing of the string w in a finite no: of steps? The answer must be yes or no.

Proof:- Assum T.M exists to solve this problem & then we will show it is contradicting itse...

We will call this T.M as a Halting m/c that produce a 'yes' or 'no' in a finite amount of time.

→ If the halting m/c finishes in a finite amount of time, o/p comes as 'yes' otherwise as 'no'.

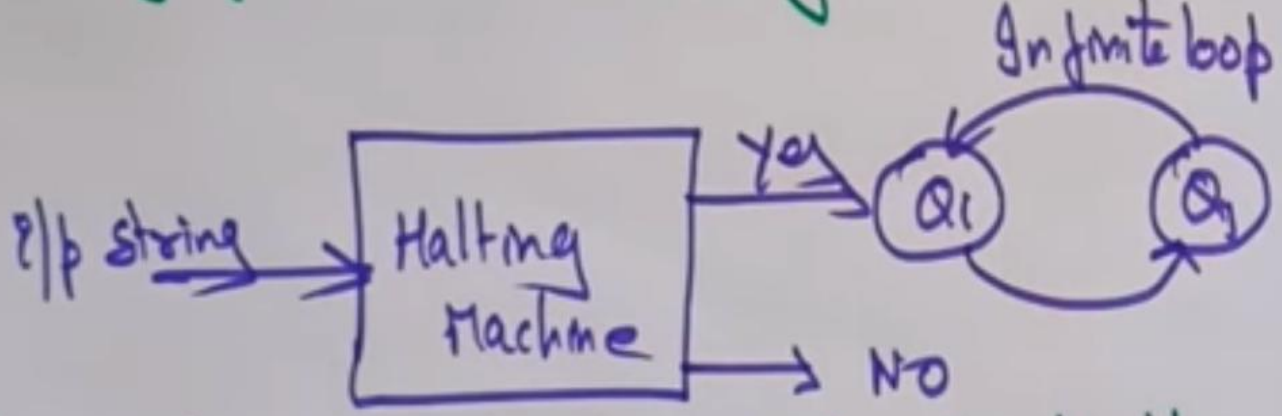
The block dig of Halting m/c :-



Now we will design an inverted halting machine as :-

- If H returns Yes, then loop forever
- If H returns No, then Halt.

The dfg for inverted Halting m/c :-



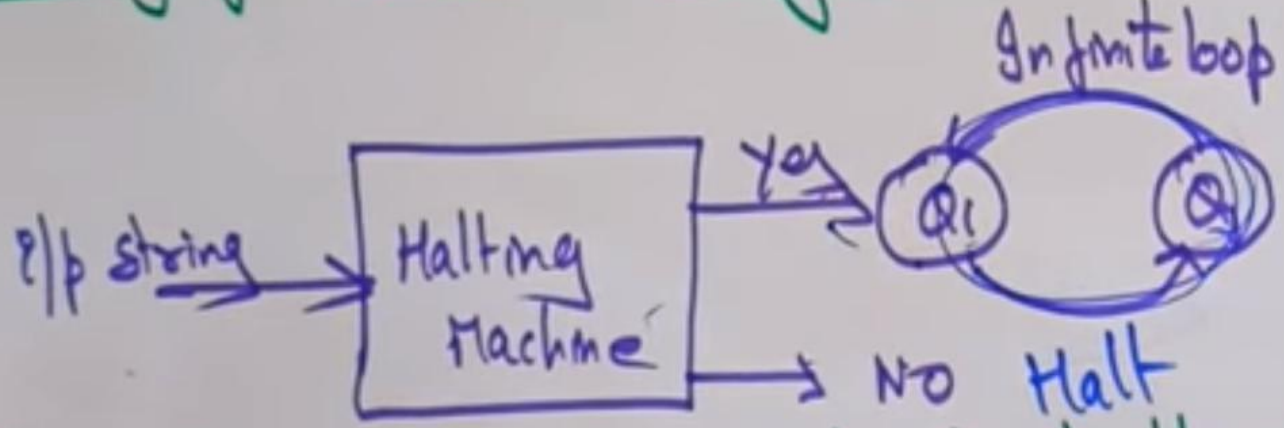
After that a m/c $(HM)_2$ which I/p itself constructed as :-

- If $(HM)_2$ halts on i/p, loop forever

Now we will design an inverted halting Machine as :-

- If H returns Yes, then loop forever
- If H returns No, then Halt.

The dig for inverted Halting m/c :-



After that a m/c $(HM)_2$ which i/p itself constructed as :-

- If $(HM)_2$ halts on i/p, loop forever

Complexity classes

Time

space i/p - "abc"

⇒ Time Complexity

- How long computation takes to Execute how long?
- In T.M this could be measured as no: of moves which are required to perform computation.

no: of m/c cycles

Complexity.

- How much strage is required for

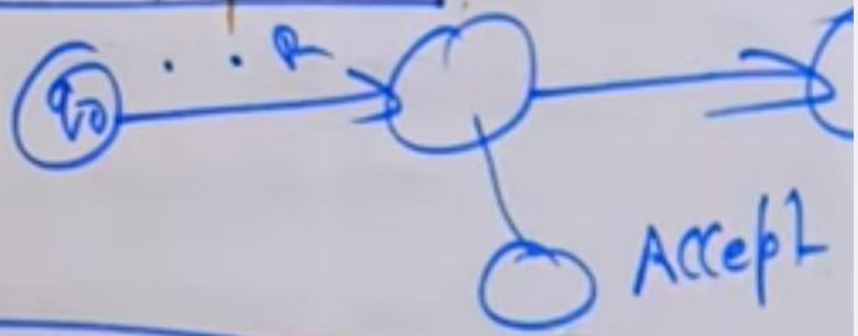
Time Complexity

- How long computation takes to Execute

how long !

- In T.M this could be measured as no: of moves which are required to perform computation.

no: of m/c cycles



Space complexity.

- How much Strage is required for computat
- In T.M, no: of cells are used
- no bytes used.

Turing Machine

⇒ Turing M/c has Infinite Size tape and it is used to accept Recursive Enumerable languages.

- ⇒ TM can move in both directions. Also it doesn't accept ϵ.
- ⇒ If the string inserted in not in lang, m/c will halt in non-final state.

⇒ TM is a mathematical model which consists of an infini length tape divided into cells on which i/p is given.

Turing M/c has infinite size tape and it is used to accept Recursive Enumerable languages.

⇒ TM can move in both directions. Also it doesn't accept ε.

⇒ If the string inserted in not in lang, m/c will halt in non-final state.

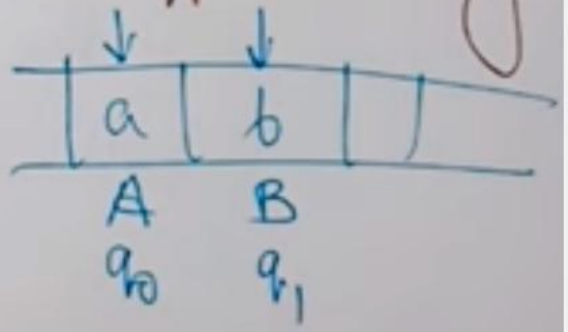
TM is a mathematical model which consists of an infinite length tape divided into cells on which i/p is given.

It consists of a head which reads the i/p tape.

⇒ A state reg stores the state of TM.

⇒ After reading an i/p symbol, it is replaced with another symbol, its internal state is changed & it moves from one cell to the right or left.

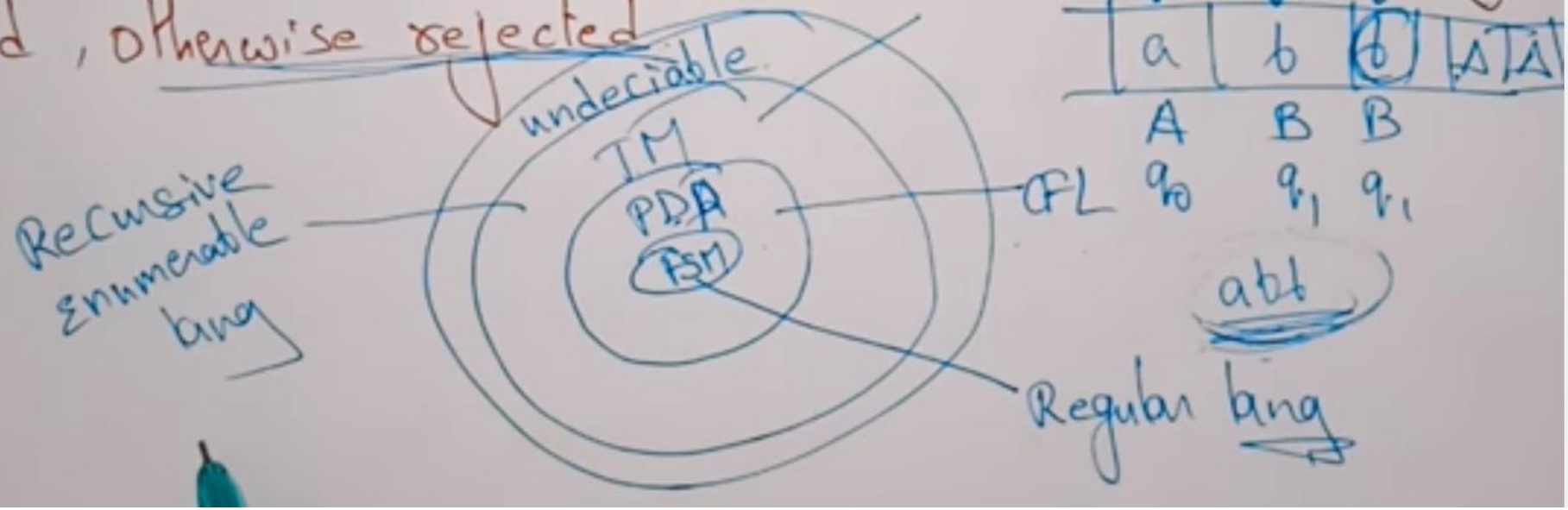
⇒ If the TM reaches the final state, the i/p string accepted, otherwise rejected



⇒ A state ... stores the ...

⇒ After reading an i/p symbol, it is replaced with another symbol, its internal state is changed & it moves from one cell to the right or left.

⇒ If the TM reaches the final state, the i/p string is accepted, otherwise rejected.



Formal Definition :

A TM can be formally described as 7-tuples $(Q, X, \Sigma, \underset{T}{\delta}, B, F)$ where :-

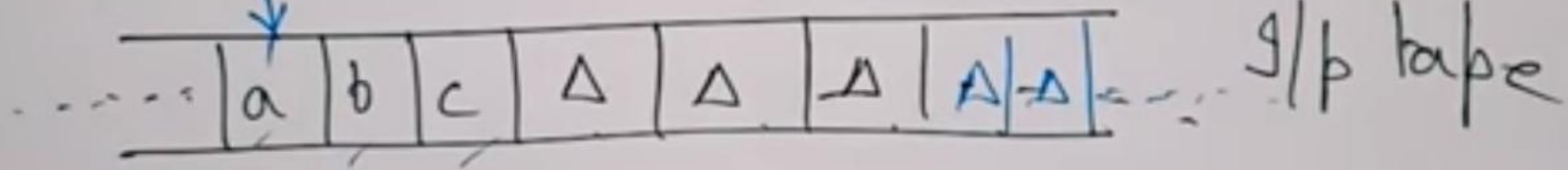
- Q is a finite set of states
- X is the tape alphabet
- Σ is the i/p alphabet
- δ is a transition function;

$$\delta : Q \times X \rightarrow Q \times X \times \{\text{left shift, Rig}$$

- q_0 is the initial state
- B is the Blank symbol

Basic Model of Turing Machine

① The i/p tape is having an infinite no: of cells, each cell containing one i/p symbol. The empty tape is filled by blank characters



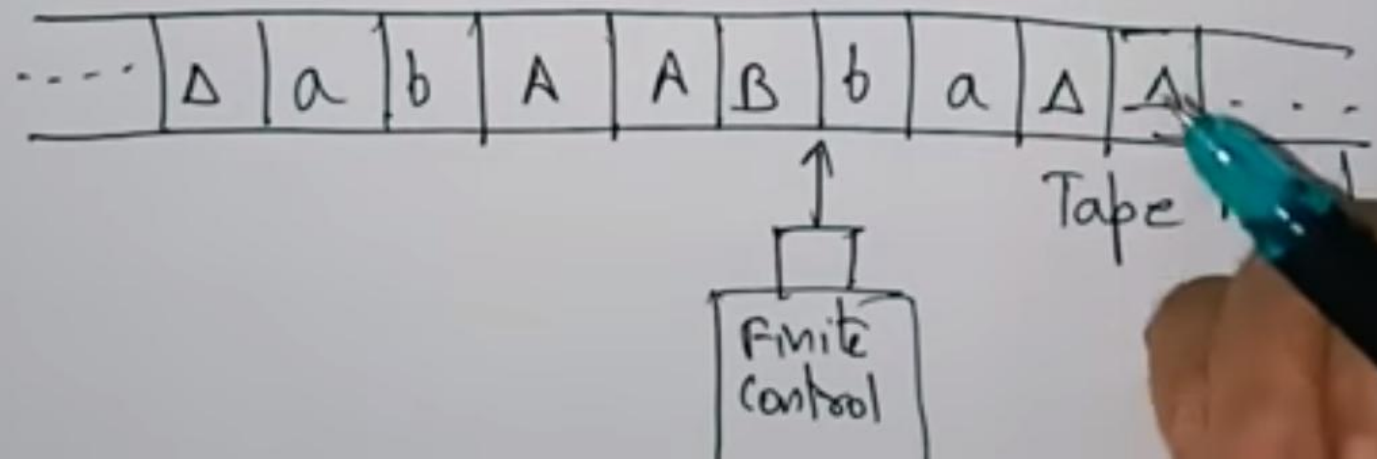
② The finite control & the tape head which is responsible for reading the current i/p symbol. The tape head can move to left to right.

3. A finite set of states through which m/c has to undergo

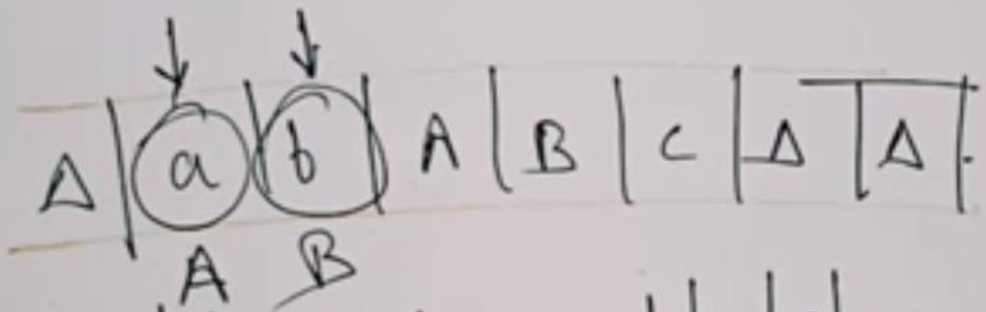
4. Finite set of symbols called external symbols which are used in building the logic of TM.

Δ - blank is a symbol

Δ is used to ... finite

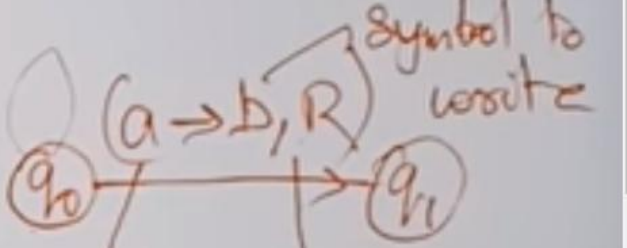


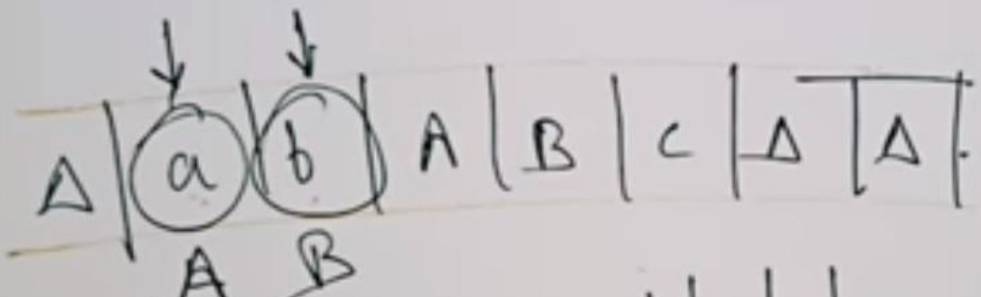
operations on the tape



1. Read / scan the symbol below tape head
2. Update / write a symbol below tape head
3. Move the tape head one step left
4. Move the tape head one step right

operations on the tape





1. Read / scan the symbol below tape head
2. Update / write a symbol below tape head
3. Move the tape head one step left
4. Move the tape head one step right

Turing's Thesis:

It states that any computation that can be carried by mechanical means can be performed by some T.M

The arguments for accepting this thesis are:

① Anything that can be done on existing digital co
Can also be done by TM

② No one has yet been able to suggest a problem sol
by what we consider an algorithm, for which a T.M

⇒ The language that is accepted by TM is Recursively Enumerable language.

A lang $L \subseteq \Sigma$ is said to be Recursively Enumerable if there exists a TM. that accepts it.

⇒ The language that is accepted by TM is

Recursively Enumerable language.

A lang $L \subseteq \Sigma$ is said to be Recursively Enumerable if there exists a TM. that accepts it.

replacing same set of rules for any no: of time

list of ele

Language accepted by Turing Machine

⇒ The T.M accepts all the lang even though they are recursive enumerable.

⇒ Recursive means repeating same set of rules for any no: of times.

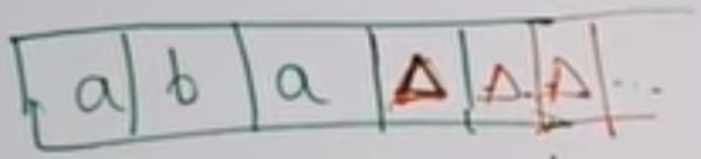
⇒ Enumerable means a list of elements

⇒ TM also accepts the Computable functions, such as addition, multiplication, subtraction, division, and many more.

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

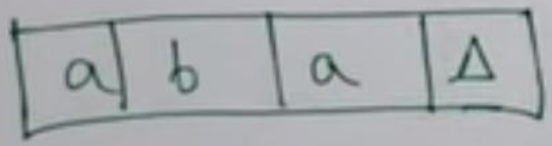
tape

Sol:- We will assume that an i/p tape the string 'aba' is placed

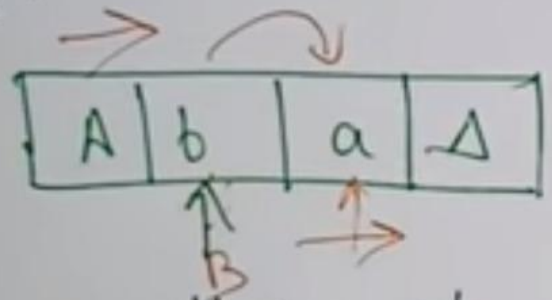


If the tape head is 'read out' 'aba' string then TM will halt after reading Δ.

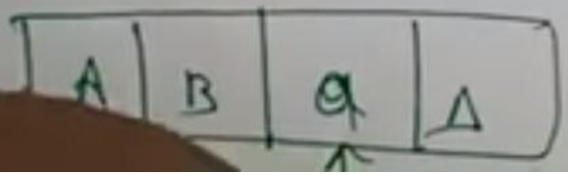
⇒ Initially, state is q_0 & head points to 'a' as:



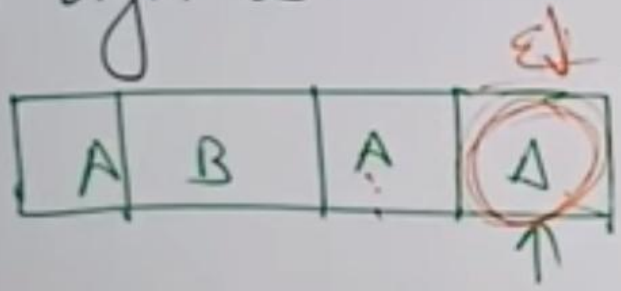
The move will be $\delta(q_0, a) = \delta(q_1, A, R)$ which means it will go to state q_1, replaced 'a' by 'A' & head will move to right as :



The move will be $\delta(q_1, b) = \delta(q_2, B, R)$ which means it will go to state q_2, replaced 'b' by 'B' and head will move to right as :



The move will be $\delta(q_2, a) = \delta(q_3, A, R)$ which means it will go to state q_3, replaced 'a' by 'A' and head will move to right as:



The move $\delta(q_3, \Delta) = (q_4, \Delta, S)$ which means it will go to state q_4 which is HALT state which is accept state for any I

It can be represented by Transition table

States	a	b	Δ
q_0	(q_1, A, R)	-	-
q_1	-	(q_2, B, R)	-
q_2	(q_3, A, R)	-	-
q_3	-	-	(q_4, Δ, S)
q_4	-	-	-

The same TM can be represented by Transition diagram