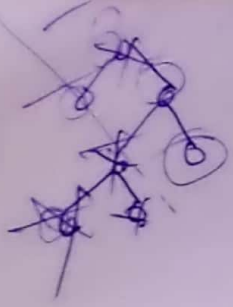


(Module 1, Part of module 3) → First Mid Sem 30%
(Module 2, part of module 3) → End Sem 50%
module 4. → Surprise quiz 20%

4 surprise tests.

Reference books:-

① Peter Linz : An intro to formal langgg.

② M Sipser. An intro to theory of computation. Thomson 2001.

Module 1, 2.

Module 3, 4

Central Area of theory of computation

→ Automata
→ Computability
→ Complexity

CHOMSKY HIERARCHY

class	Languages	Grammars	Automata (Machines for a particular language)
Type 3	Regular	Regular	Finite
Type 2	Context free	Context free	Pushdown
Type 1	Context sensitive	context sensitive	Linear Bounded.
Type 0	Recursive [Recursively enumerable]	unrestricted	Turing machine

Type 3: Automata is Yes/No output type (for finite & random mem)

Ex. i) a word is part of the language or not (accept) | (reject)
yes No.

ii) H/W design

iii) in compiler 1st stage.

Q What are fundamental capabilities and limitations of computers?

Q What makes some problems computationally hard & others easy?

SYMBOL: Are fundamental building blocks for a language.

Ex. {a, b, A} for english.

ALPHABET (Σ) is a finite set of symbols.

Ex - Σ = {a, b} (taken from {a, b, A}).

STRING: A string over Σ (alphabet) is a finite set of symbols.

Suppose Σ = {a, b}, Strings are = {a, b, ab, ba, aab, ... any combination of a&b}

Now consider a string x = aab,
|x| = |aab| = length of string.

$n_a(x)$ = No of occurrences of a in x.

Λ or ∈ = Null String.

Σ^* = (Universal set or star closure of all strings including NULL over given alphabet)

Ex: $\Sigma = \{a, b\}$, $\Sigma^* = \{NULL, a, b, aa, ab, \ldots\}$

Σ^k = All strings of length k. (eg $|x| = k$)

$= \{x \in \Sigma^* \mid |x| = k\}$

$\Sigma = \{a, b\}$ | $\Sigma^1 = \{a, b\}$ | $\Sigma^2 = \{aa, bb, ba, ab\}$.

$\rightarrow |\Sigma^k| = |\Sigma|^k$

POSITIVE Clos

$\Sigma^+ = \Sigma^* - \{\epsilon\}$ → Null.

LANGUAGE: A language over Σ is a subset of Σ^*.

Ex: $L1$ = set of all strings over $\{a, b\}$ of length 2.

→ this is informal description of language.

$L1 = \{aa, bb, ab, ba\}$.

* A no. of small languages can be used to define a bigger language.

$L1 = \{x \in \{a, b\}^* ; |x| = 2\} = \Sigma^2$

→ formal description of language.

Few real life languages

→ Language of legal JAVA Identifiers
→ The language of memory addresses
→ The language of legal JAVA program.

FINITE AUTOMATA

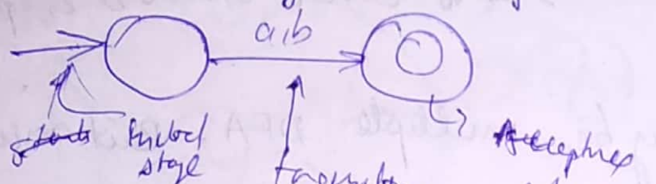
→ Only role is to accept or reject
→ Doesn't have any memory.

(FA) → Accept / Reject

- FA without output
  - DFA (Deterministic FA)
  - NFA (Nondeterministic FA)
  - ENFA (epsilon-NFA)
- FA with output
  - mealy machine
  - moore machine

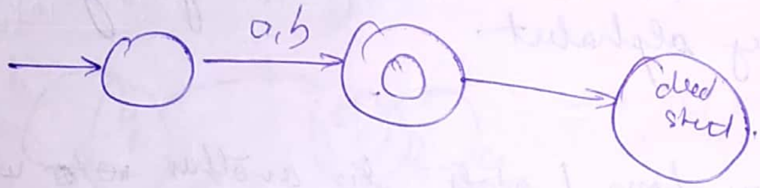
FA :→ finite no of symbols & finite no of states

Ex: Set of strings containing a & b, given $\Sigma = \{a, b\}$, recognize a or b is I/L.



(NFA)

NFA as final state has no o/p.



In DFA all the states have an output

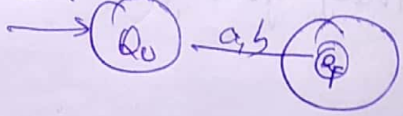
DFA can be defined by 5 tuples. $(Q, Q_0, Q_F, \Sigma, \delta)$

Q = Set of finite states

Q_0 = Initial state $(Q_0 \in Q)$

Q_F = (Set of final states)

$Q_F \subseteq Q$



Σ = alphabet

δ = transition function

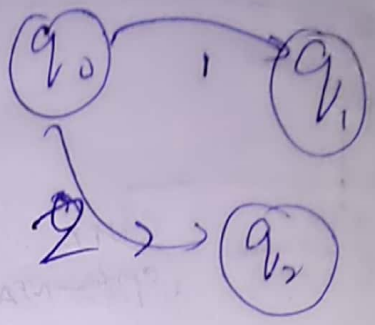
$\delta : (Q, \Sigma) \to Q$

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

$(\delta(q_i, a) \to q_j)$

DFA

(1) For a particular input symbol, machine (m/c) goes to 1 state only.



(2) for transition fn $\delta(Q, \Sigma) \to Q$

(3) Null (ϵ) move is not allowed.

(4) For every finite language, we can design a finite automata. However, for ∞ language we have to check whether it is possible or not.

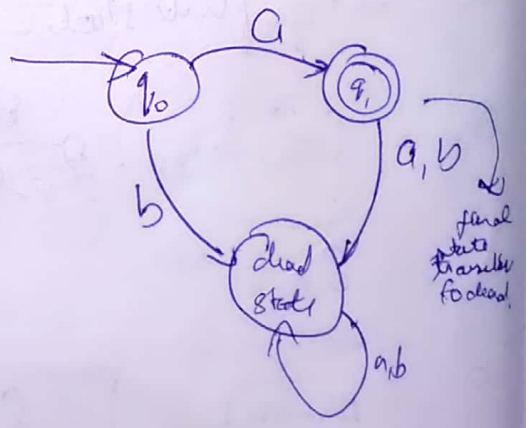
(5) For a language, there may be multiple DFA. But minimum DFA will be only one.

(6) from every state m/c must have outgoing transitions for every alphabet.

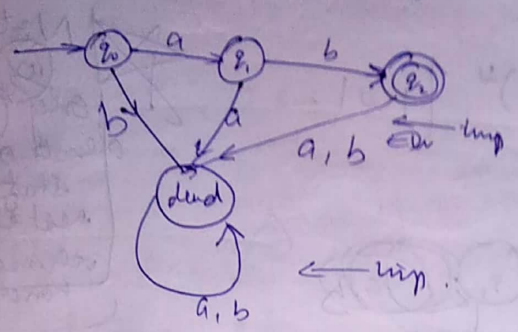
Null move ⇒ m/c goes from 1 state to another ~~state~~ without any I/P.

Ex: Design DFA for language containing 'a' only over $\{a, b\}$.

all states must have transition & hence they go ~~to~~ to dead state;



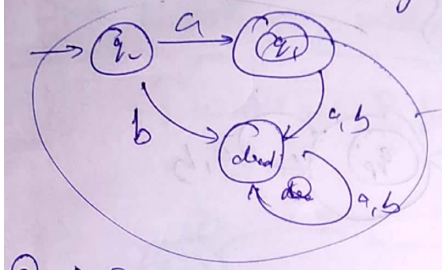
If lang has ab only, over $\Sigma = \{a, b\}$



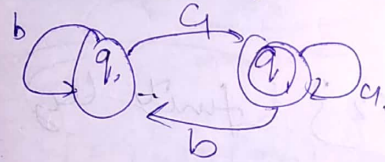
If string has aab it reaches q_2. & then whatever goes go to dead state

trap — Also in DFA all states must have transition from it for each alphabet, even dead state.

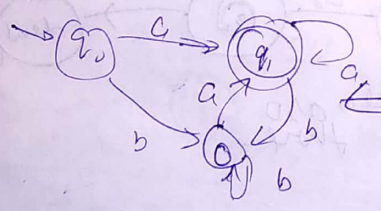
Q. Design DFA for a lang containg a string aab



Q. DFA for a lang that end with a.



→ no dead state here.



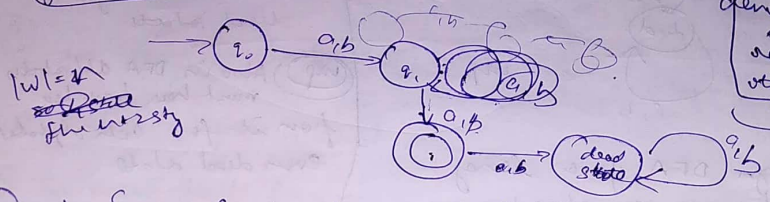
Wrong!

∵ dead state can only have self loop i.e, no transition from

If we are not bothered about an alphabet coming at a state, just put it as self loop.

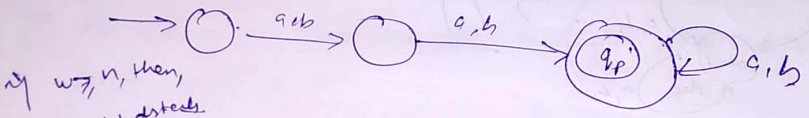
Q2 DFA based on length of string

(1) $L = \{ w \in (a,b)^* \mid |w| = 2 \}$ finite lang



$|w| = n$ finite lang

(2) $L = \{ w \in (a,b) \mid |w| \geq 2 \}$ ∞ lang



if $w \geq n$, then $n+1$ states.

(3) ∞ lang.

$L = \{ w \in (a,b)^* \mid |w| \leq 2 \}$ finite lang

$L = \{ \epsilon, a, b, ab, ba, aa, bb \}$ ∴ null is also final.

$w \leq n$

we require $n+2$ states.

DFA

1) L =

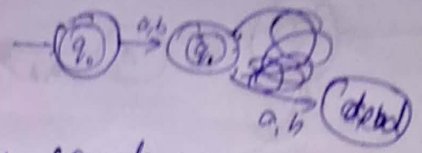
2)

3)

Note

(17.) $L = \{ w \in (a,b)^* \mid |w| < 2 \}$

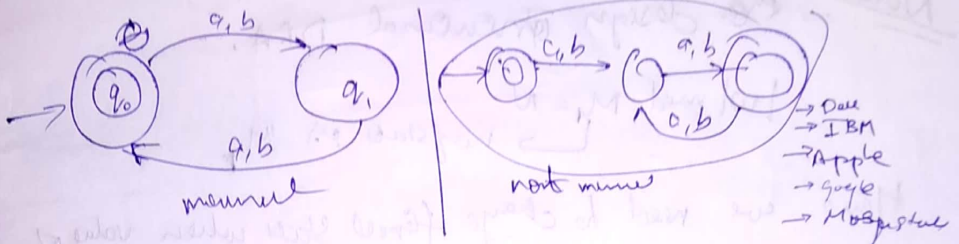
$L = \{\epsilon, a, b\}$



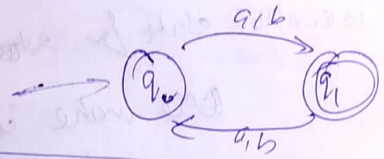
✡ Note that (2) & (4) are complement each other & are literally same, just, the empty state become regular & regular became empty.

DFA based on modulus.

1) $L = \{ w \in \{a,b\}^* \mid |w| \bmod 2 = 0 \}$ → any lang



2) $L = \{ w \in (a,b)^* \mid |w| \bmod 2 = 1 \}$ → any lang



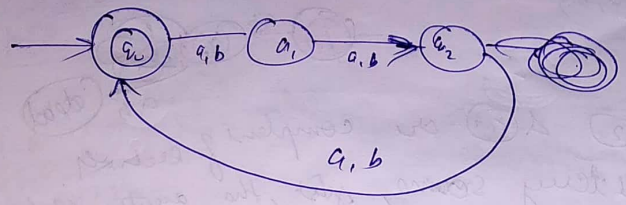
3) $|w| \bmod 2 > 0$, same as 1.

✡ In DFA finite means no. of states & symbols are finite

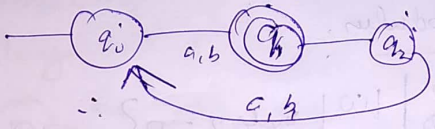
If not mentioned in question to design min. DFA, then anything will do, but if mentioned in question about minimal DFA, then do the following

Q. $L = \{w \in \{a,b\} \mid |w| \bmod 3 = 0\}$

$L = \{\in, aaa, abb, \ldots\}$



Q. $|w| \bmod 2 = 1$



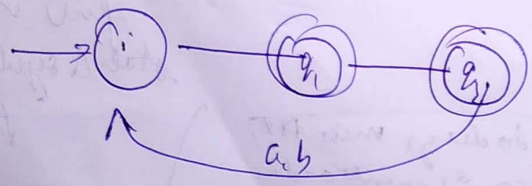
Note: To design universal DFA.

$|w| \bmod M_N = N$

↳ no of states in M_N

Here we need to change final state when value of N is changed. When N = 1, then q_1 will be final state provided q_0 is omitted state & ~~state~~ ~~state~~ are make in sequence.

Q $|w|_1 \bmod 3 > 0$



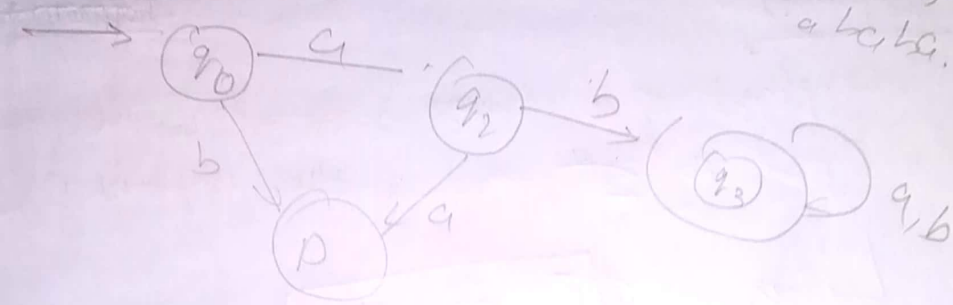
Prefix / Substring / suffix

$w = abaaa$

Design DFA

$L = \{ w \in (a,b)^* \mid w \text{ has prefix } ab \}$

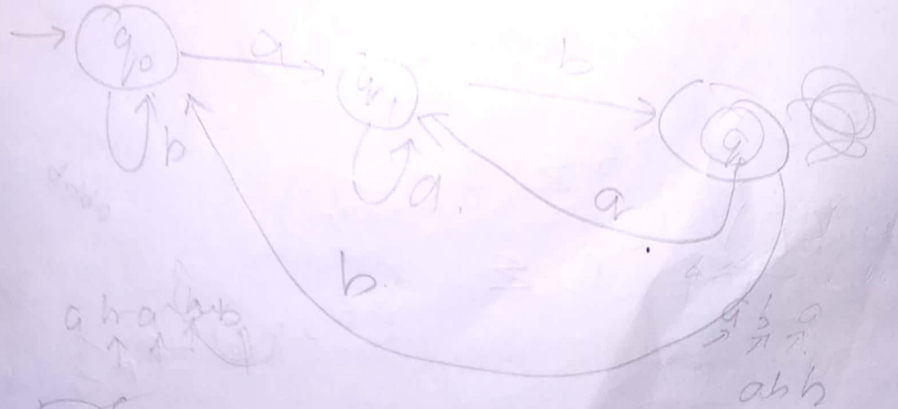
Basic string $= ab$ $L = \{ab, abaaa, abbbb, abab, aba\}$



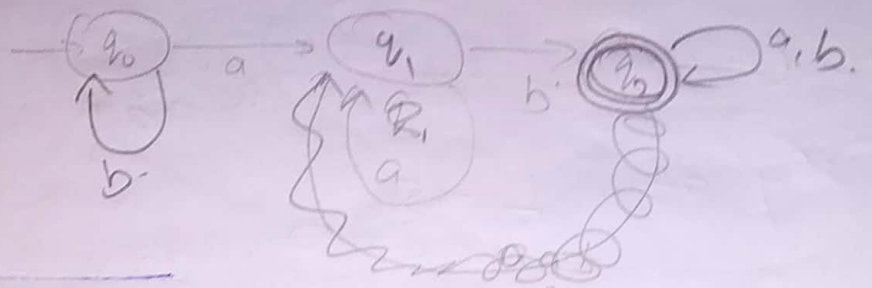
$L = \{ab, \ldots\}$

$L = ab\Sigma^*$

Q. $L = w \in (a,b)^+$ ... suffix ab.



$L = \{ w \in (a,b)^* \mid w \text{ has atleast one } a \}$



Complement

Complement of a language is defined with respect to Σ^*, that is the complement of L is

$$\boxed{\bar{L} = \Sigma^* - L}$$

Eg. $L = \{a^n \mid n > 3\}$

$L = \{a^n \mid n \leq 3\}$

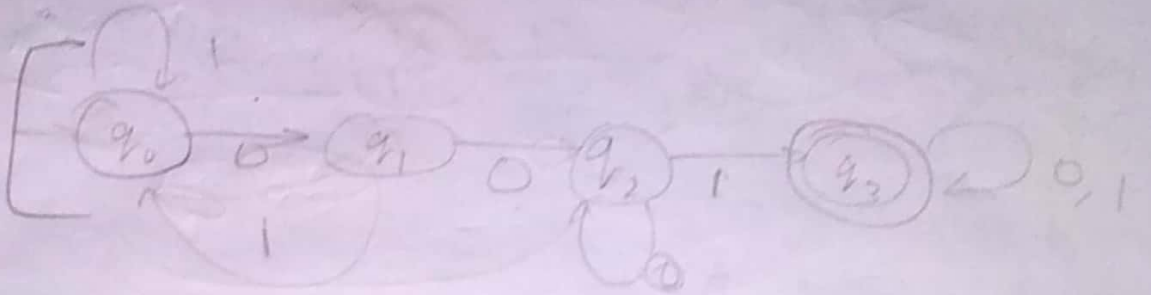
$M(L) = \{Q, \Sigma, \delta, q_0, Q_F\}$

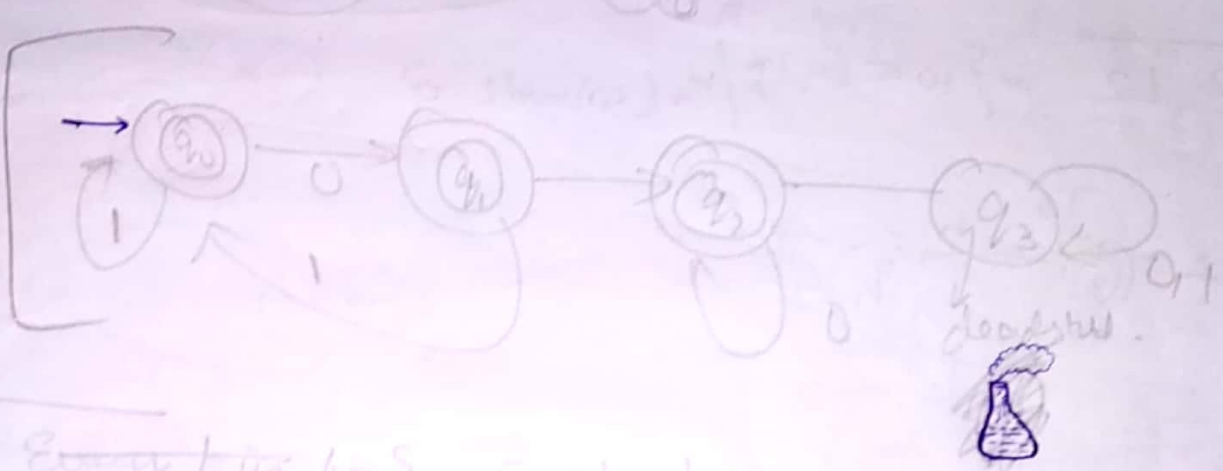
$M(\bar{L}) = \{Q, \Sigma, \delta, q_0, Q - Q_F\}$

abbb

Q Design a DFA that accepts all strings over {0,1} except those containing 0 0 1 as a substring

Ans





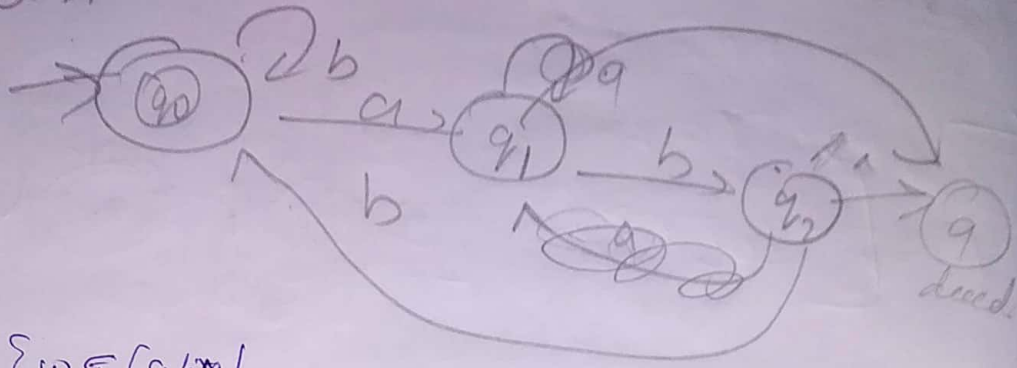
Ans

Q: Lang $L = \{w \mid w \in (a+b)^* \}$ every a in w is followed by bb

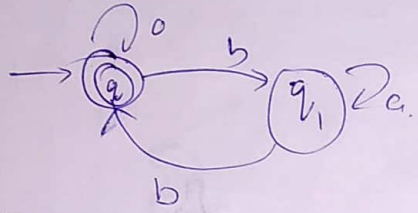
So, we will take complement,

$L \cap \bar{L}$ should not have any common words.

Pumped in

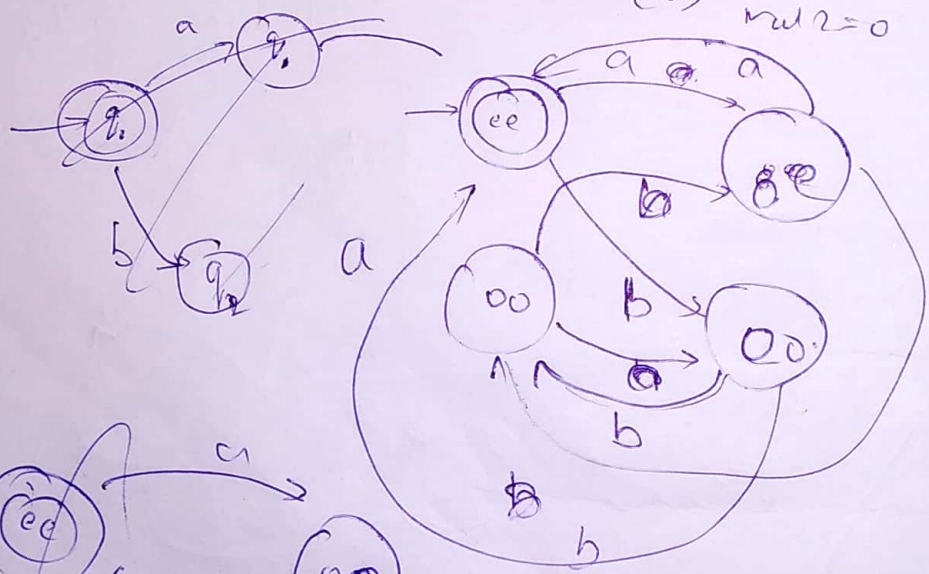


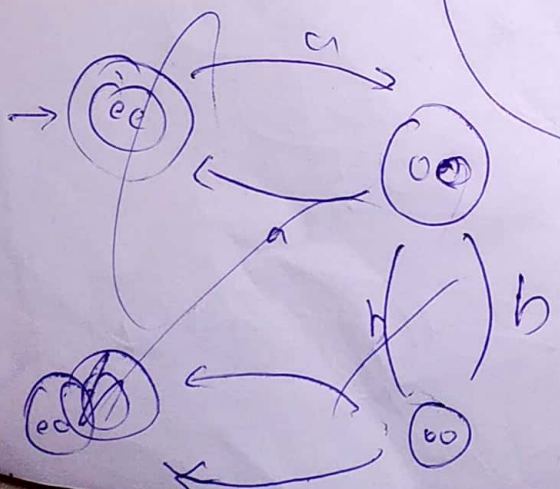
Q $L2 = \{ w \in (a,b)^* / n_b(w) \bmod 2 = 0 \}$



DFA
→ length
→ mod
→ substr
→ prefix/suffix
→ complement

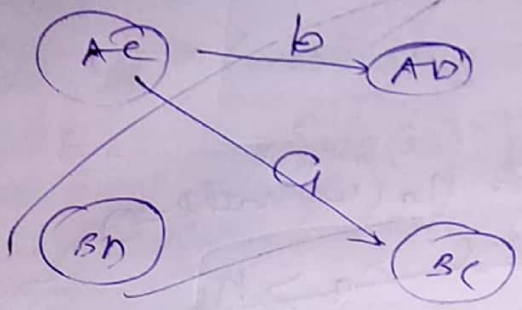
Q $L3 = \{ w \in (a,b)^* \mid n_a(w) \bmod 2 = 0 \text{ \& } n_a(w) \bmod 2 = 0 \}$





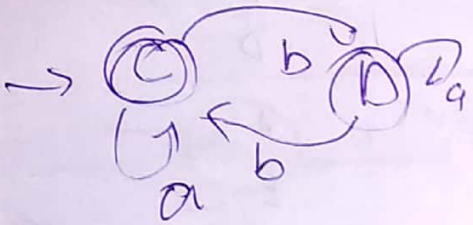
Cross product method

$\{A, B\} \times \{C, D\} = \{AC, AD, BC, BD\}$ → So total 4 states required,

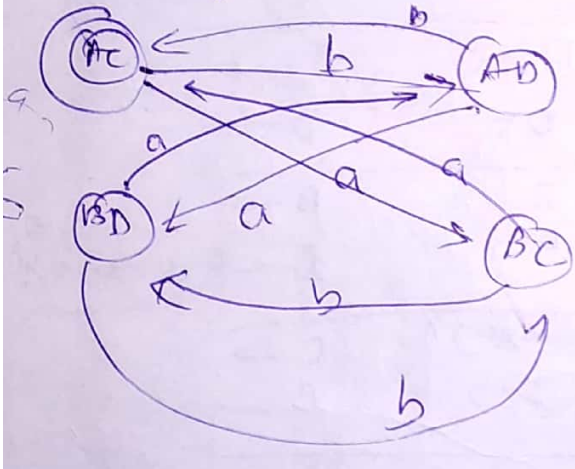


a	b
$A \to B$	$A \to A$
$C \to C$	$C \to D$

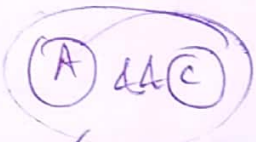
$A \to B$	$A \to A$
$D \to D$	$D \to C$



$B \to A$	$B \to B$
$C \to C$	$C \to D$

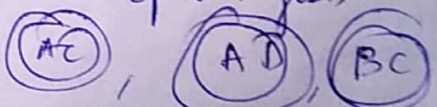


$B \to A$	$B \to B$
$D \to D$	$D \to C$



↳ accept state ∵ AC

likely for OR, we, we perform some operation like



we call all 3 to accept state.

$L_4 = n_a(\omega) \% 2 = 0$ & $n_b(\omega) \% 2 = 0$

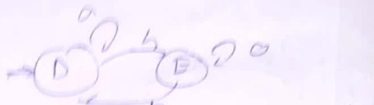
(A)

Q Design minimal DFA.

$L = \{ \omega \in (a,b)^* \mid n_a(\omega) \% 3 = 0 \text{ and } n_b(\omega) \bmod 2 = 0 \}$

~~& $n_a \geq n_b$~~

(B)



AD AE BD BE
CD CE

	b
~~A B~~	
A → B	A →
D →	D →
A → B	A →
E →	E →
B → C	B →
D →	D →
B → C	B →
E →	E →
~~C → A~~	C →
D →	D →
C → A	C →
B →	E →

22/08/2019.

Concatenation of 2 language L_1 & L_2 is the set of all strings obtained by concatenating element of L_1 with any element of L_2. Specifically $L_1L_2 = \{xy;\ x \in L_1,\ y \in L_2\}$

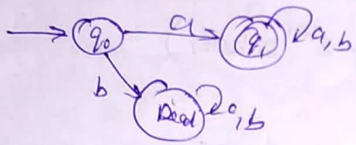
Note: We can club final state of L_1 & initial state of L_2 provided L_1 has only one final state. And we have to additionally take care of clubbing point.

★→ Concatenation is very useful concept in NFA.

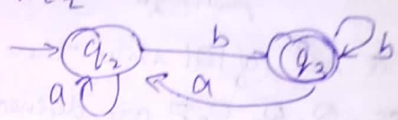
Ex. $L_1 = \{w \in (a,b)^* \mid w \text{ starts with } a\}$
$L_2 = \{w \in (a,b)^* \mid w \text{ end with } b\}$

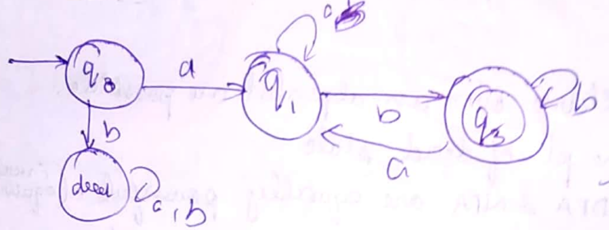
$L_1 = \{a, ab, aa \ldots\}$ | $L_2 = \{b, ab, aab \ldots\}$

DFA (L_1)



DFA L_2



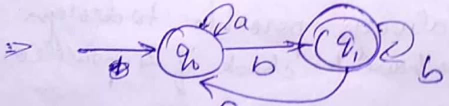


Reversal

$L = \{w \in (a,b)^* \mid w \text{ end with } b\}$
$= \{b, ab, abb, \ldots\}$
$L^R = \{b, ba, bba, bbaa, \ldots\}$

$DFA^R = \begin{cases} DFA \\ \text{or} \\ NFA \end{cases}$

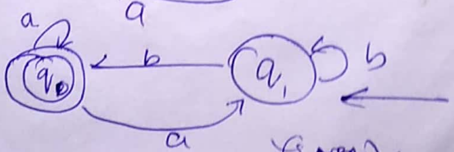
DFA(L) ⇒



(DFA)

★ change dir of arrow & make final accept state as initial state & initial state as final state

DFA (L^R)



(NFA).

NON DETERMINISTIC FINITE AUTOMATA (NFA/NDFA)

A NFA is defined by $M = (Q, \Sigma, \delta, Q_0, Q_F)$ where,

δ for NFA

$\delta : Q \times \Sigma \rightarrow 2^{|Q|}$ (power set of Q)

δ for ϵ-NFA

$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^{|Q|}$

(For a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets)

Hence, Q, Q_0, Q_F, Σ are defined as DFA.

Note:-

→ Multiple transitions over an alphabet are possible.

→ There is no concept of dead state

→ The classes of DFA & NFA are equally powerful. (Every NFA has equivalent NFA/DFA)

→ NFA for respective language must accept all strings in the language, and reject all which are not part of the language.

→ Avoid unnecessary transition. (Not reaching any acceptance state)

(here trapping will not occur as in non-deterministic TM as if string pass it reaches to halting) NFA is more of an acceptor than rejector

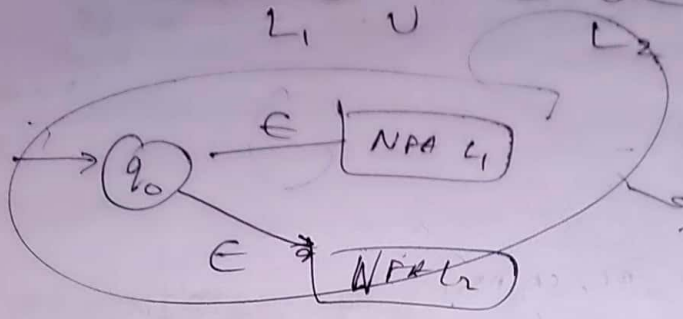
★ In ϵ NFA even without input, we can change states.

★ For a finite language always possible to design NFA or DFA. For an infinite language we have to check if its possible or not.

L_4: Design ϵ-NFA for

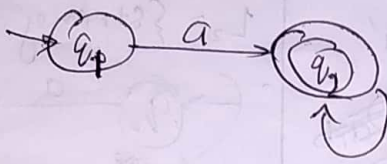
$L = \{a^n : n \geq 1\} \cup \{b^m a : m \geq 1\}$

$L_1 \cup L_2$

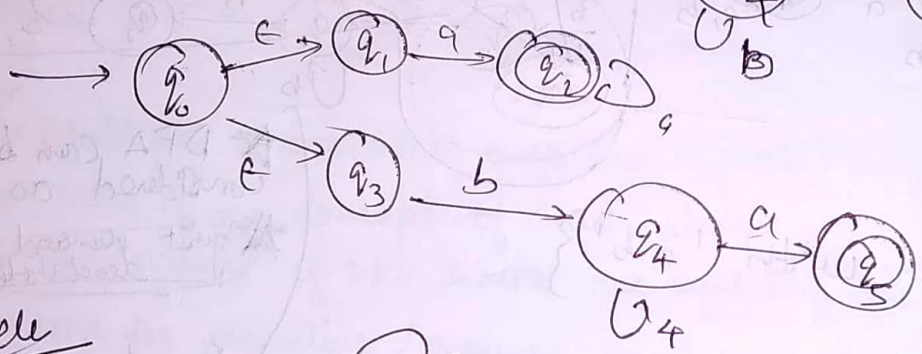


Union of L_1, L_2 are regular languages, then their union is also regular by closer of union.

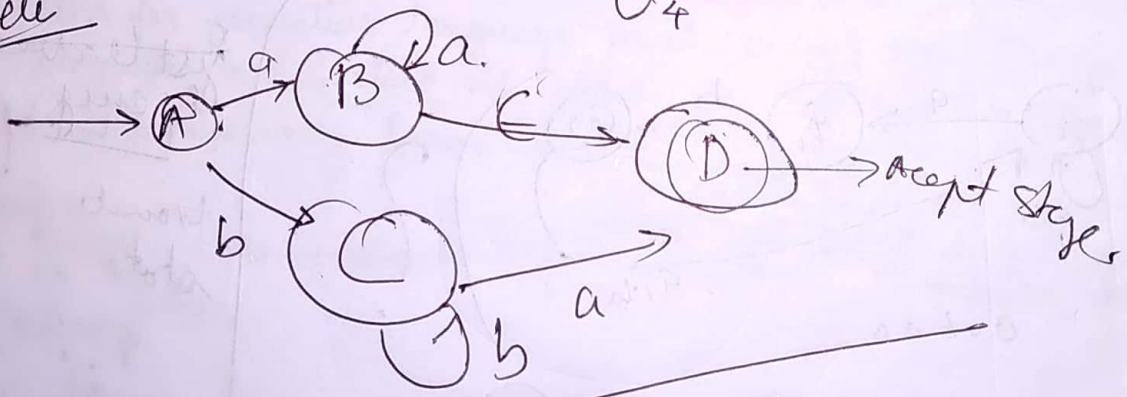
for L_1,



for L_2:

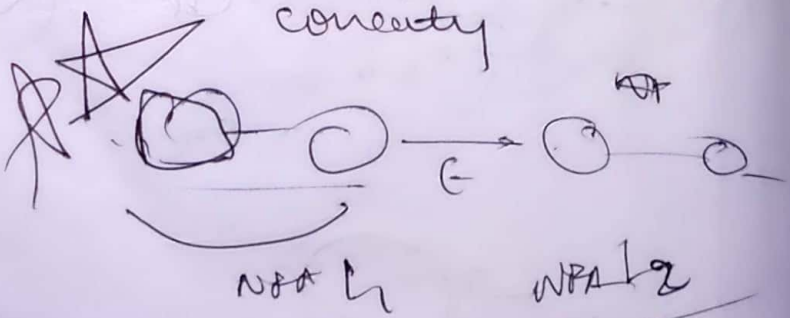


Alle



ϵ-NFA
↓ convert
NFA
↓ convert
DFA

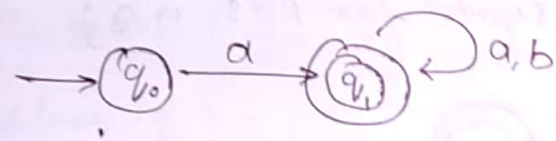
In union of L_1 L_2 concatenation



PROCEDURE TO CONVERT NFA TO DFA

1. Identify initial state of NFA as initial state of equivalent DFA.
2. Repeat following steps until no more edges are missing.
   Take any vertex $\{q_i, q_j, \ldots q_k\}$ in equivalent DFA that has no outgoing edge for some $a \in \Sigma$ & compute $\delta^*(q_i, a) \cup \delta^*(q_j, a) \cup \ldots \cup \delta^*(q_k, a) = \{q_l, q_m, q_n\}$
   make $\{q_l, q_m, q_n\}$ as new state if it doesn't exist in equivalent DFA. Then complete outgoing transition over Σ.
3. Every state in equivalent DFA whose state contains any $q_f \in F$ (final state of NFA) is identified as final state.
4. If NFA accept ϵ, then make initial state of equivalent DFA as a final state.

Ex



NFA

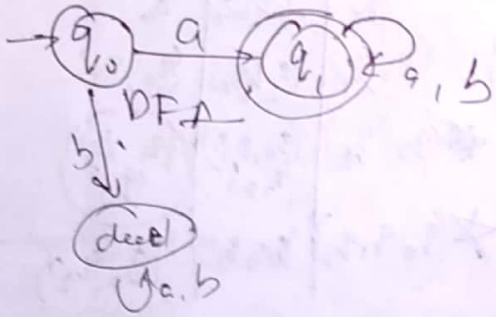
equiv DFA

St	a	b
q_0	q_1	{Dead}
q_1	q_1	q_1
{Dead}	{Dead}	{Dead}

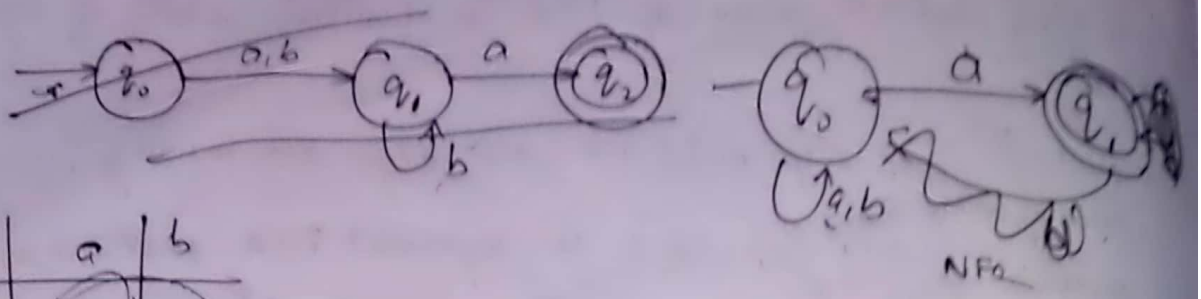
Now forget every NFA just take care of DFA.

No new states so we are done.

$L = \{\epsilon, aa, aba, \ldots\}$
Remove $\{L\}$
$L = \{a, ba, bba\}$
complement of $L = \Sigma^* - L$



Ex Make NFA for L = {w/w 'a' is second to last}



	a	b
q_0	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1\}$	q_0

new states

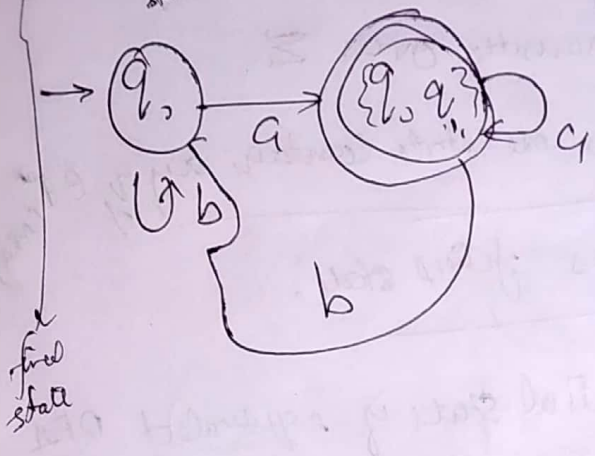
old states

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

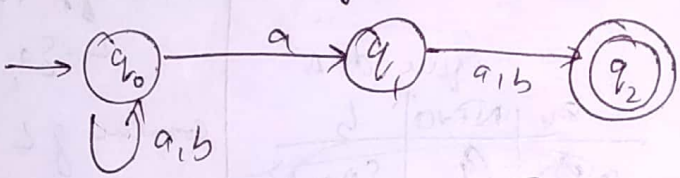
$= \delta(q_0, a) \cup \delta(q_1, a)$

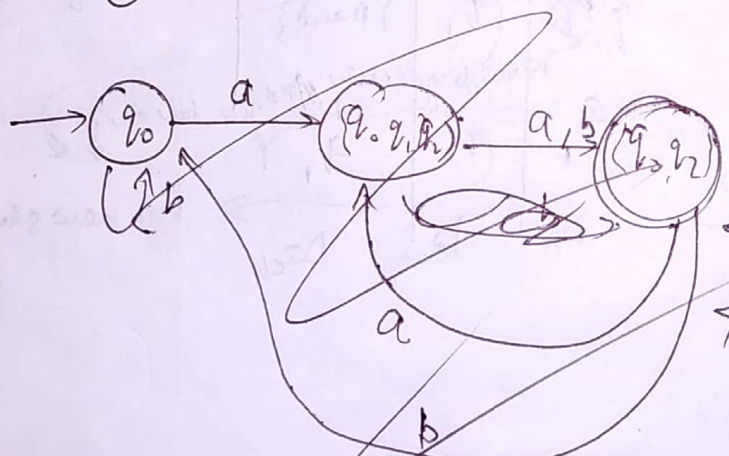
$= \{q_0, q_1\}$

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



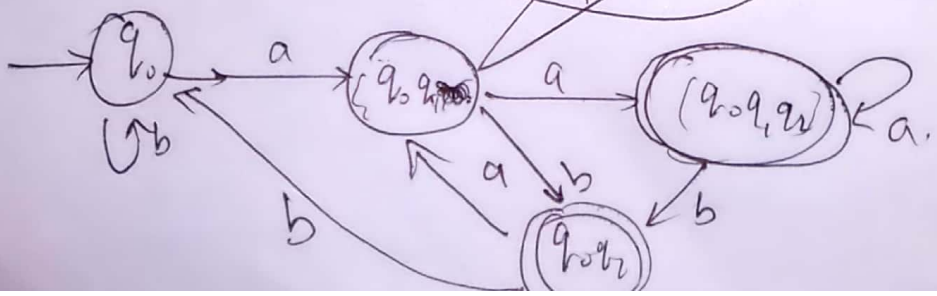
Ex L = {string containg second symbol from RHS is a}





a b a a b

state	a	b
q_0	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
* $\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$
* $\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$

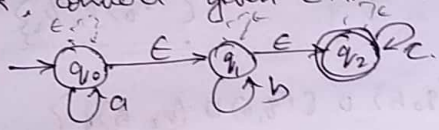


ϵ-NFA to NFA

function : ϵ-closure

ϵ-closure(q) ⇒ Set of all those states of NFA, (without ϵ transition), which can be reached from q on a path labelled by ϵ (i.e., without consuming any input symbol).

Ex, convert given ϵ-NFA to NFA



δ / Σ	a	b	c	ϵ
q_0	q_0	∅	∅	q_1
q_1	∅	q_1	∅	q_2
q_2	∅	∅	q_2	∅

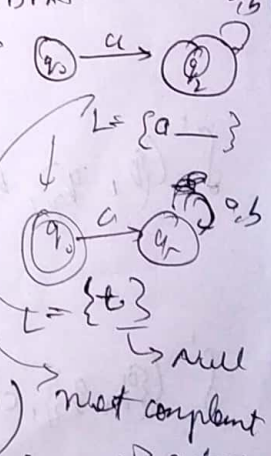
closure(q_0) = ~~$\{q_1, q_2\}$~~ $\{q_0, q_1, q_2\}$

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

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

✱ Note a vertex ϵ path is there as self loop at each node.

✱ Note ⇒ Completeness in NFA can be done in the same way as in DFA.



→ Initial State of NFA (w/o ϵ closure) will be as before (Here q_0 is initial state for ϵ-NFA)

ϵ-closure(q_0) = $\{q_0, q_1, q_2\}$

↳ initial state.

Rest states of NFA are :-

ϵ-closure(q_1) = $\{q_1, q_2\}$

ϵ-closure(q_2) = $\{q_2\}$

→ 2 new states of equivalent NFA (other than initial state)

→ Final state of NFA are all those new states which contain final state of ϵ-NFA, as a member.

So (q_0, q_1, q_2) & (q_1, q_2) (q_2) all 3 are final states

→ do the following transition for each alphabet for each state (δ' is the transition function for NFA and δ is transition function for ϵ-NFA)

PTO.

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

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

$= \epsilon\text{closure}\{q_0 \cup \phi \cup \phi\} = \epsilon\text{closure}\{q_0\}$

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

similarly,

$\delta'(\{q_0, q_1, q_2\}, b) = \epsilon\text{closure}\{\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)\}$

$= \epsilon\text{closure}(\phi \cup q_1 \cup \phi)$

$= \{q_1, q_2\}$

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

$\epsilon\text{closure}(\phi \cup \phi)$

$= \{\phi\}$

$\delta'(\{q_0, q_1, q_2\}, c) = \epsilon\text{closure}(\phi\ \phi\ q_2)$

$= \{q_2\}$

$\delta'(\{q_1, q_2\}, b) = \epsilon\text{closure}(q_1 \cup \phi) = \epsilon\text{closure}(q_1)$

$= \{q_1, q_2\}$

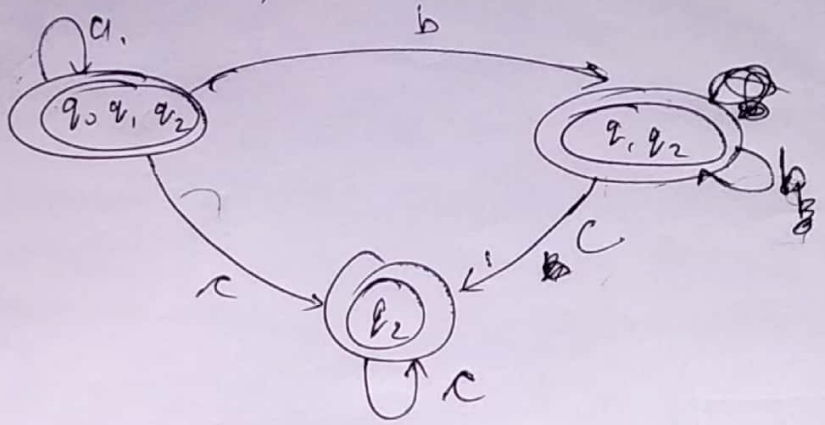
$\delta'(\{q_1, q_2\}, c) = \epsilon\text{closure}(\phi, q_2\} = \{q_2\}$

$\delta'(\{q_2\}, a) = \epsilon\text{closure}(\phi) = \{\phi\}$

" $b = \epsilon\text{closure}(\phi) = \phi$

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

So Our NFA



TOC

→ Answers questions like, "wh... are problems that can be solved using a machine?"
[Given a problem can it be solved using a computer] → Computational theory

Ex → add 2 nos (solvable)
→ ~~is it going to rain~~ who is going to win match (non solvable)
→ given a mathematical statement, check if its true or false (non solvable)

1932 – Alan Turing → Turing machine: A mathematical model of a computer.

A problem can be solved using a computer ⇒ It can be solved using a turing machine

with steps that can be carried out on a machine, one by one.

★ We assume turing machine has ∞ memory. (it is most complex machine).

→ What are different models of machines that can solve problems?
Ex. Calc (simple machine) < Computer (Turing machine)

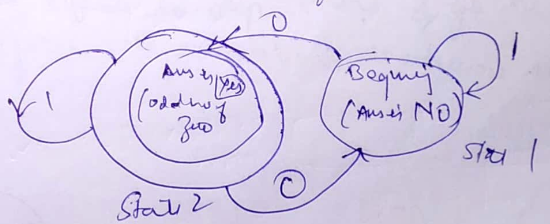
⇐ TOC → Complexity theory.
Computability theory
Automata theory → deal with different types of mathematical models of machines.

Input | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | Q: Does the string have an odd no. of zeros?

★ here output is Yes/No.
& we can think of it as deciding if the string belongs to a lang.
(here lang. will be the strings having odd no. of zeros)

→ bit by bit input
→ if empty string, here Ans is No. (Begin, Ans is no.)
→ ~~[struck]~~ State will change whenever a zero is encountered.
→ If input ends then last state is final ans.

Language
0 . 01
000
0010
0100



→ Called finite Automata or finite state machine

★ Restrictions
→ finite states
→ ... states
?

Lang. A

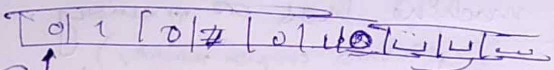
w (String input) → [box] → yes (accept) / → No (reject)

Q. String contains 0, 1 or # & check if it's of the form w # w ? (i.e., # is b/w 2 same wor..)

→ here multiple passes are allowed.

- Here we need to "remember" "stuff", that may be long (how long depends on input).

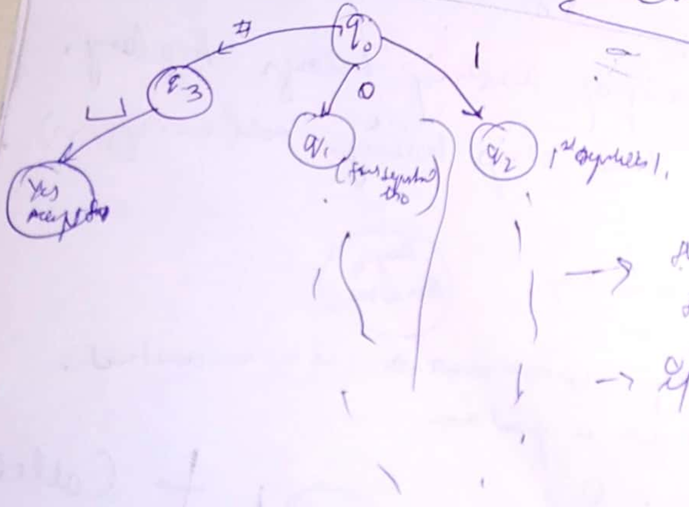
→ We assume a turing machine has tape.



→ Entirely tape contains input, followed by blank symbols but can be written in b/w

→ point moves one step at a time & can move back & forth.

Check if one # only using finite state automata then



→ then compare with digit after # then if not equal output No (reject)

→ if equal then replace thereby x & go next & continue

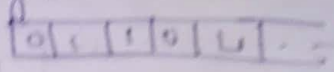
Sipser

Turing m/c has:

i) An input tape: An input string followed by blank symbols (␣)

ii) tape head: initially points to beginning of tape.

iii) At a time tape head moves left or right by 1



iv) Input alphabet Σ (here, 0, 1).

v) Tape alphabet Γ here $\{0, 1, ␣\}$ $\quad \Sigma \subseteq \Gamma$ & $␣ \notin \Sigma$

vi) Finite set of states (works for any I/P)

vii) $q_1 \rightarrow$ initial state
$q_{accept} \rightarrow$ accept "
$q_{reject} \rightarrow$ reject "
$\} \in Q \quad q_{accept} \neq q_{reject}$

viii) transition fn (or "moves"/"steps")

$\delta(q, a) \longrightarrow (r, b, \alpha)$ → decides next move

q → current state; a → current symbol; r → next state; b → new symbol; α → direction

Q.

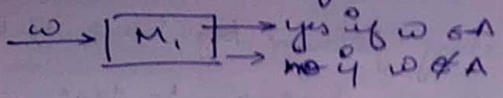
Q. $Q = q_0, q_1, q_2, q_{acc}, q_{rej}$

$\Gamma = \{0, 1, X, ␣\}$ → symbol that can be seen on tape

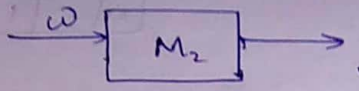
$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times D$

we need to define transition for each state & symbol if not it's assume to stop. then & go to q_{rej}.
(once rejected; tape symbol & direction not important).

M_1 decides lang A.



Now, in some cases this type of m/c may not be designed even if its computable. For those cases:



yes if $w \in A$.
it doesn't say 'yes' if $w \notin A$.
(i.e., it doesn't say no.

M_2 recognises lang A.

i.e., if $w \notin A$, it may enter ∞ loop i.e., doesn't stop.

⇒ Lang A is Turing decidable if ∃ M which decides A (recursive).

& lang A is turing recognizable if ∃ M' which recognizes lang A (recursively enumerable)

Set of turing decidable lang ⊆ set of Turing recognizable lang.

---

(1) $w\#w$
(2) $0^n 1^n$
(3) 0^{2n}

$w \in \{0, 1, .\}$
design m/c that checks these lang, decidable if

we say a turing machine accepts a lang if it reaches q_{accept} at end of tape.

---

m/c stops at q_{reject} / q_{accept} i.e., no moves from q_{rej} / q_{acept}

not be
cases:

$\omega \notin A$.
no.

donot stop

decides A

which

)

mzable
g.

g.

if a transition is not defined, it usually moves to q_{reject}.

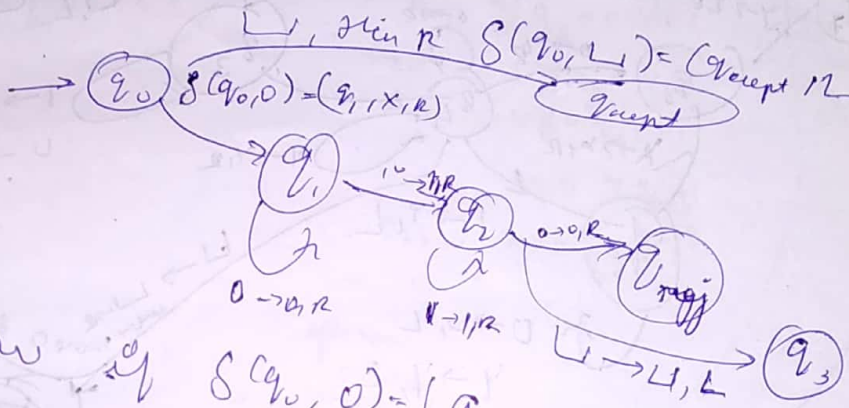
(2) $0^N 1^N = A$

a.) scan from left → right

~~b.)~~ if ∄ 0 after 1st 1 reject

b.) come back to left & cross off 0 and go right & cross of corresponding 1.
if they both end up at same time ~~reject~~ accept.

a.)



Now if $\delta(q_0, 0) = (q_{0,}, 0, R)$ then if i/p is

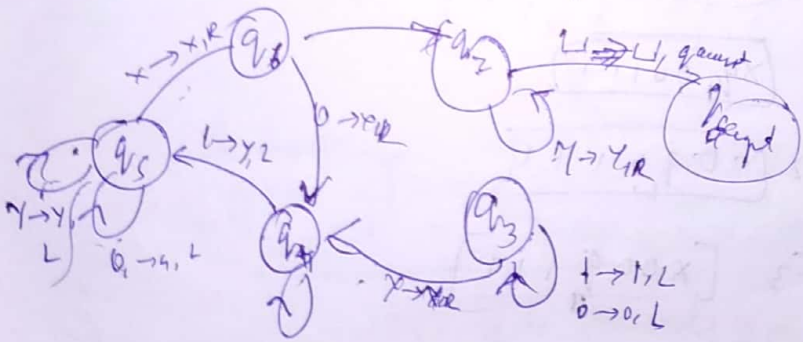
0 0 ␣ it accepts.

change 0 → x to know type head

Undefined transition → q_{reject}

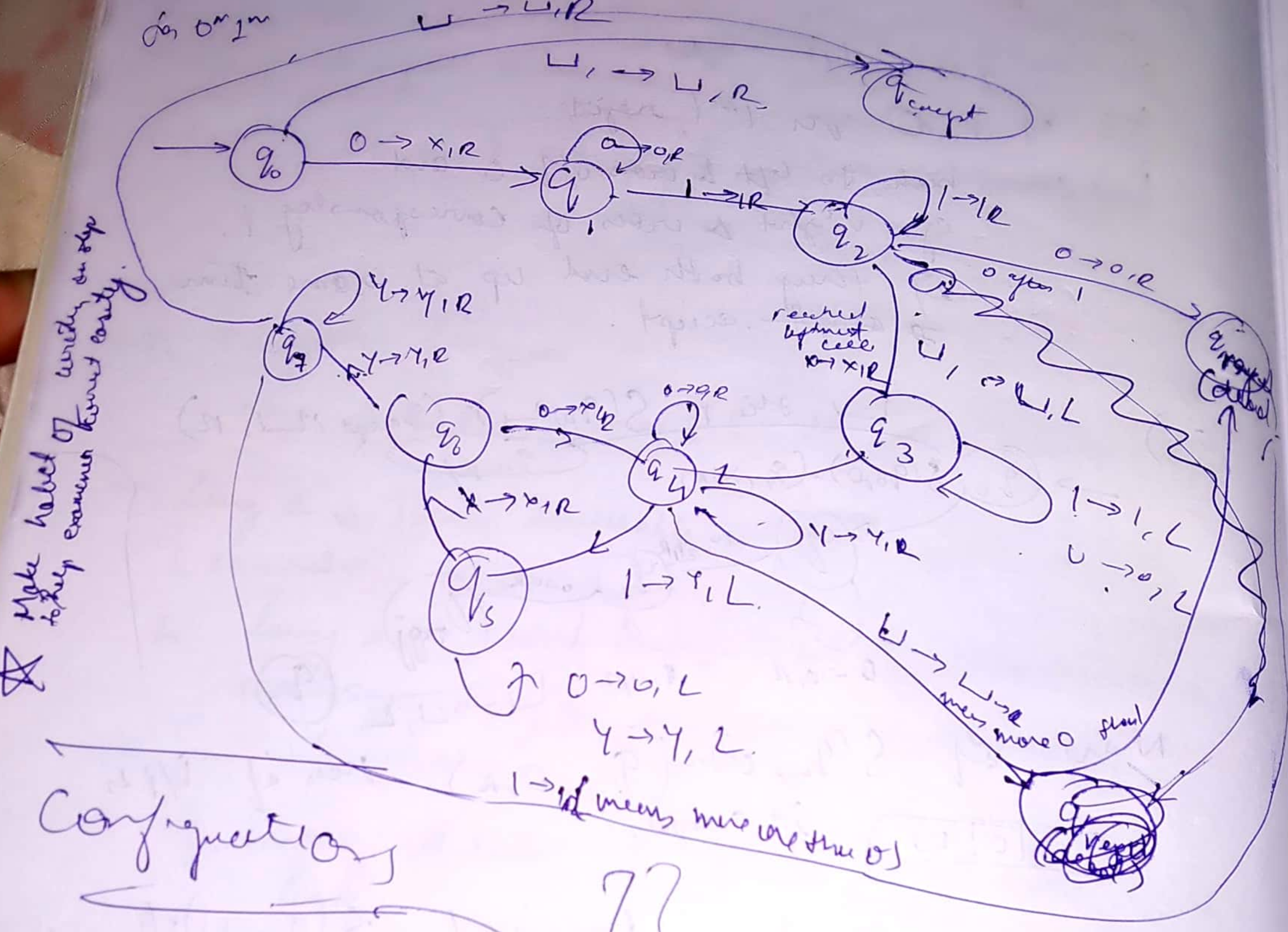
$\delta(q_0, ␣) = (q_{accept}, 0, R)$
$\delta(q_0, 1) = (q_1, X, R)$
$\delta(q_1, 1) = (q_2, 1, R)$
$\delta(q_1, 0) = (q_1, 0, R)$
$\delta(q_2, 1) = (q_2, 1, L)$
$\delta(q_2, 0) = (q_{rej}, 0, R)$

Now, once it reaches ␣ in q_2, we have to move back & follow step b

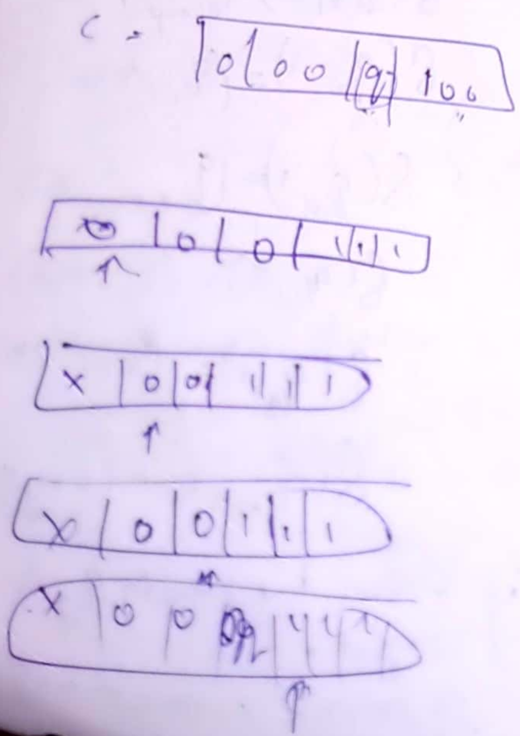


if extra 1, reach q_3 & reject.
if extra 0, reject at q_{rej}

In exam even if 2-3 extra states it is fine ... question is specifically for minimizing.



Configurations



$\alpha_1 \, q \, \alpha_2$

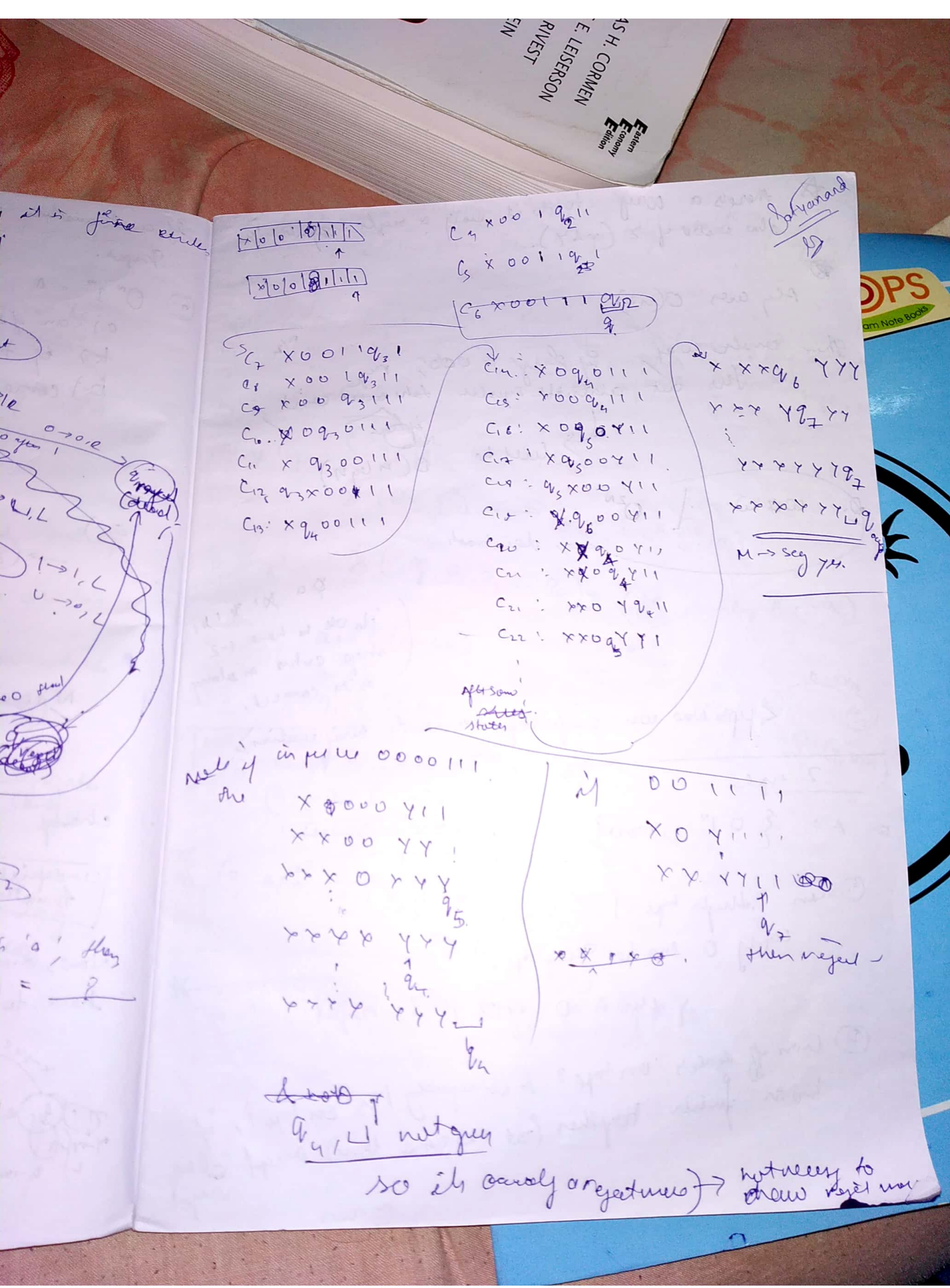
if 1st char of α_2 is 'a', then $\delta(q, a) = ?$

C_0 $q_0 000111$

C_1 $xq_1 00111$

C_2 $x0q_1 0111$

C_3 $x00q_1 111$



Scanned by CamScanner

☆ there's a way to do it using a single symbol also instead of 2 (x & y).

⊛ Alg uses $O(n^2)$.

There's another way, if string is odd, reject; then cut alternate symbols (take current coin) ($n/2$)

↳ will be $O(n \log n)$ (iteration steps)

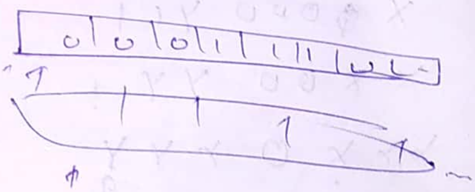
Q. $w \# w$ | 0^{2N} → do in book.

⊗ 0 ⊗ 1 ... ⊗ 1
(it's ok to have 1-2 steps extra as long as it's correct.)

< Upto Now was single tape deterministic turing machine >.

2 tape

Ex $A = \{ 0^N 1^N , N \geq 0 \}$



① Scan through tape 1, copy 0 on to second tape.

if there is 0 after 1^{st} 1, reject.

② Cross off zeroes on tape 2 & comparing 1's on tape 1, if both finish together (at the same time) accept else reject.

$O(n)$

So it was a multitape turing machine

In general we can have k-tape turing machine (k → control independent of input)

tape1: contains input entirely, followed by blanks.
tape2: <blank>
:
tapek: <blank>

& But state is only one and not separate for individual tapes.

single tape: $\delta(q,a) = (r,b,D)$

$\delta : (Q \times \Gamma) \longrightarrow Q \times \Gamma \times \{L,R\}$

k tape: $\delta(q, a_1, a_2 \dots a_k) = (r, b_1 \dots b_k, D_1 \dots D_k)$

$a_i \in \Gamma \ (i = 1 \text{ to } k)$

$b_i \in \Gamma$ "

$D_i \in \{L,R\}$

new state

$\delta(q_1, b) = (q_2, b, R)$

M:

$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L,R\}^k$

$(a_1, a_2 \dots a_k) \in \underbrace{\Gamma \times \Gamma \times \dots \Gamma}_{k \text{ times}}$

Eg: $\delta(q, a_1, a_2) = (r, (b_1, D_1), (b_2, D_2))$

tape 1 tape 2

~~Is Multi~~ M.W. Draw state diag. for 2 tape ——

Q. Are multi-tape turing machines better in terms of computational power?
(i.e, are there problems which can be solved using k-tape T.M. but not on S-tape T.M.?)

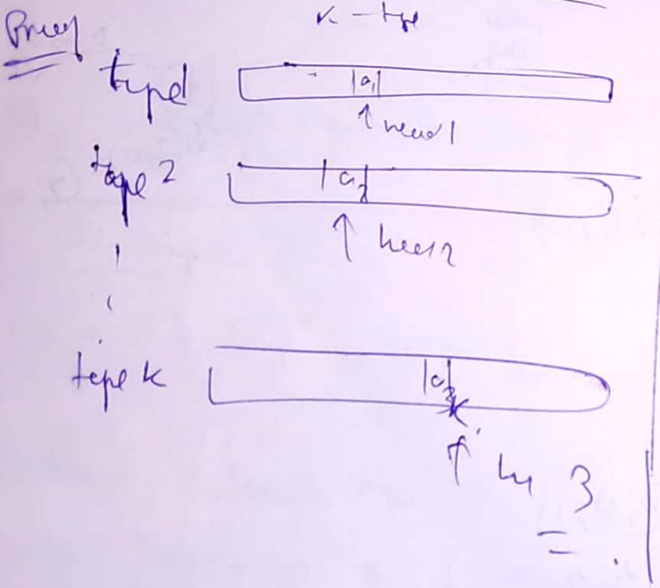
{not bothered about time complexity}.

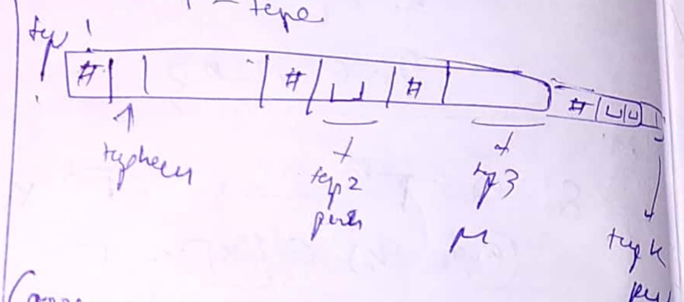
Ans No.! Anything that can be solved with k tapes (for any fixed k), can be solved with a single tape turing machine.

Theorem: If a language A is decidable (recognizable) with a k tape turing machine, then A is " " " single tape T.M.
& vice versa also.

Proof of vice versa is easy ∵ 1-tape is also a k-tape T.M.

Proof





Carries info. of k tapes, separated by a delimiter.

& to know pos. of head in each part, use dotted symbol like $\dot{0}$ if 0, $\dot{1}$ if 1 ... $\dot{\gamma}$ if γ.

- To identify "current symbol" on each tape, we use "dotted" symbols

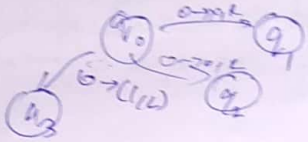
Ex: $\dot{0}, \dot{1}, \dot{x}, \dot{y}, \dot{\#}, \dot{\sqcup}$

Scan the tape once, to identify current symbols $a_1, a_2 ... a_k$
(the k^{th} dotted symbols).

$\delta(q_i, a_1, \ldots a_k) = (r, b_1, \ldots b_k, P_1, \ldots P_k)$

- Making changes in all k tapes, it done in one step.
  - this might involve multiple moves.
  - ↳ may involve shifting of all symbols to left or right by 1 power of each write (many moves on the single tape machine S).

- Deterministic T-M.
- Non deterministic T-M.

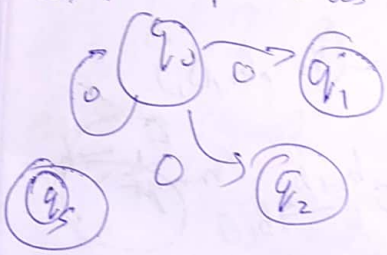


D. F. A.

Exactly one choice of move for any given $q \& a$

NFA

→ Multiple choices of moves are available for some given $q \& a$.

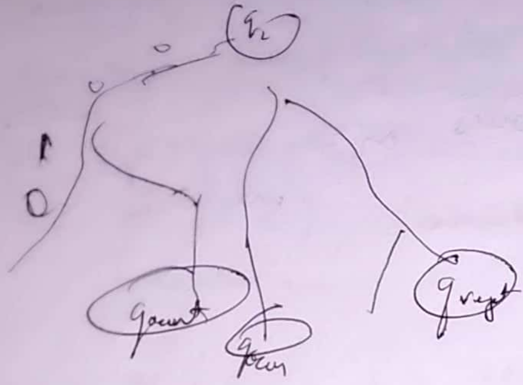


0100 take you to q_0, q_1 or q_3 depending on the choice of moves.

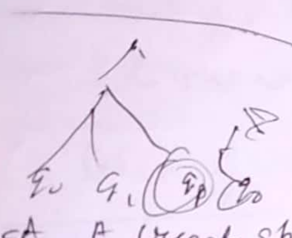
So we will say $\therefore q_3$ is accepted.

Non Determinste T.M.

$\delta(q_1, 0) = \{(q_1, 0, R), (q_2, x, R), (q_2, x, L)\}$



- We say a non determinist TM accepts a string w if any one of the execution path lead to accept.
- When all path lead to qreject, then rejected that string.



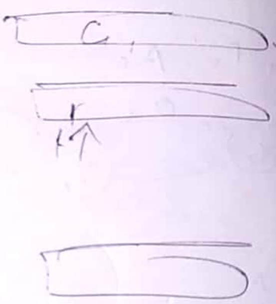
NFA A accepts string w if A is in a final state in any one of paths at the end of reading w.

Q1 . state q_1 tape 1

Q2 q_2 tape 2

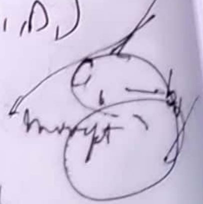
⋮

Q_k q_k tape k



$\delta_1(q_1, a_1) = (r_1, b_1, D_1)$

$\delta_2 = (q_2 a_2) = (r_2, b_2, D_2)$



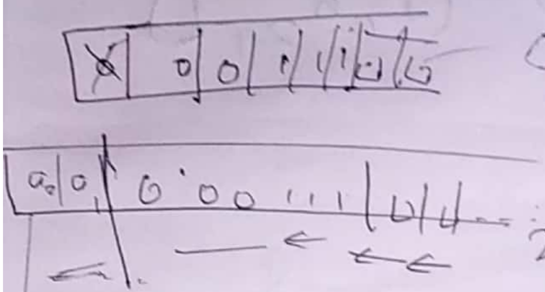
move left / Right

State $q_1 \to r_1$

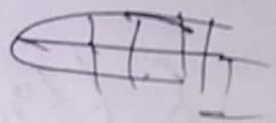
δ

?

?



Some authors use 2 way tape ... but it is equivalent



2 ways tape turing machine

→ ... on a single tape, the two way tape can be wrapped around to make a 2-track tape, then can carry 2 symbols Encoding

Encoding. Set of tape symbols Γ, new $\Gamma' = \Gamma \times \Gamma$

$(a_1, a_2) \in \Gamma \times \Gamma$

How to modify ① Set of states (if needed)
② Transition function (δ).

$\delta(q', a_3, a_1) =$

Case I: original move was in right half
Direction: R □ □

~~$\delta(q, q_1) = (r, b, R)$~~

$\delta(q, a_1) = (r, b, R)$

$Q' = Q \times \{0, 1\}$ | ~~$\delta'((q, 0), \binom{a_1}{a_2})$~~ $\delta'\left((q, 0), \binom{a_1}{a_2}\right) = \left((r, 0), \binom{b_1}{a_2}, R\right)$

$(q, 0) \to$ right half
$q, \_ \to$ S state q left half

Case II: On original machine we are on left half, direction is left. $\delta(q_1, a_2) = (r, b_2, L)$

$\delta'(\langle q_1, 1\rangle) = (r, b_2, L)$

$\delta'\left(\langle q_1, 1\rangle, \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}\right) = \left(\langle r, 1\rangle, \begin{pmatrix} a_1 \\ b_2 \end{pmatrix}, R\right)$

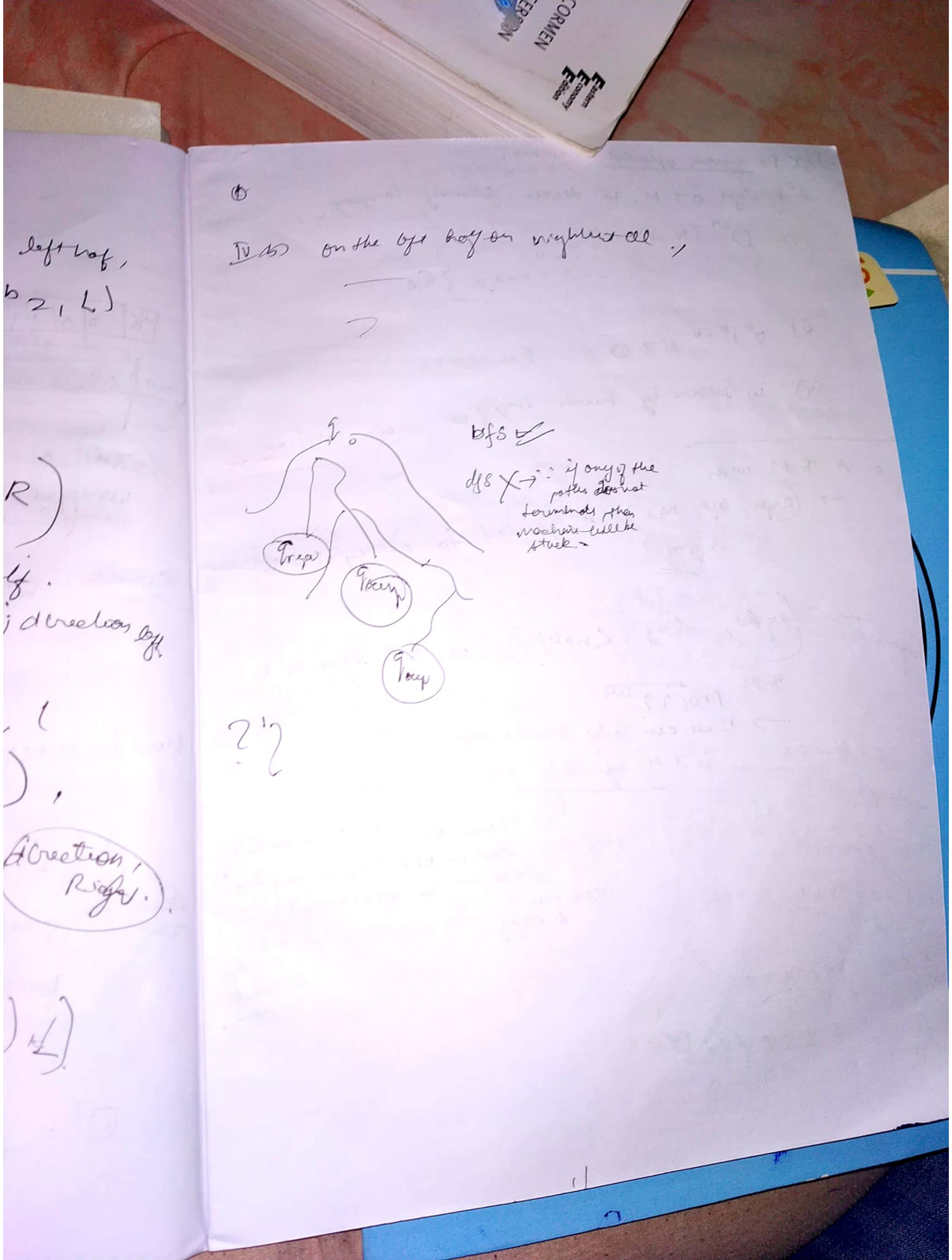
Case III: a) Original mode is was in right half. Not on the leftmost cell; direction left.

$\delta'\left(\langle q_1, 0\rangle, \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}\right) = \left(\langle r, 0\rangle \begin{pmatrix} b_1 \\ a_2 \end{pmatrix}, L\right)$

Case IV: a) On left half (bottom track in new machine), Not on the right most cell of half direction Right.

$\delta'\left(q_1, 1\right), \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}) = \left(\langle r, 1\rangle, \begin{pmatrix} a_1 \\ b_2 \end{pmatrix}, L\right)$

Case IV b): on the right half in leftmost cell



Do questions uploaded on edusrer..

- Design a T.M. to decide following Language..

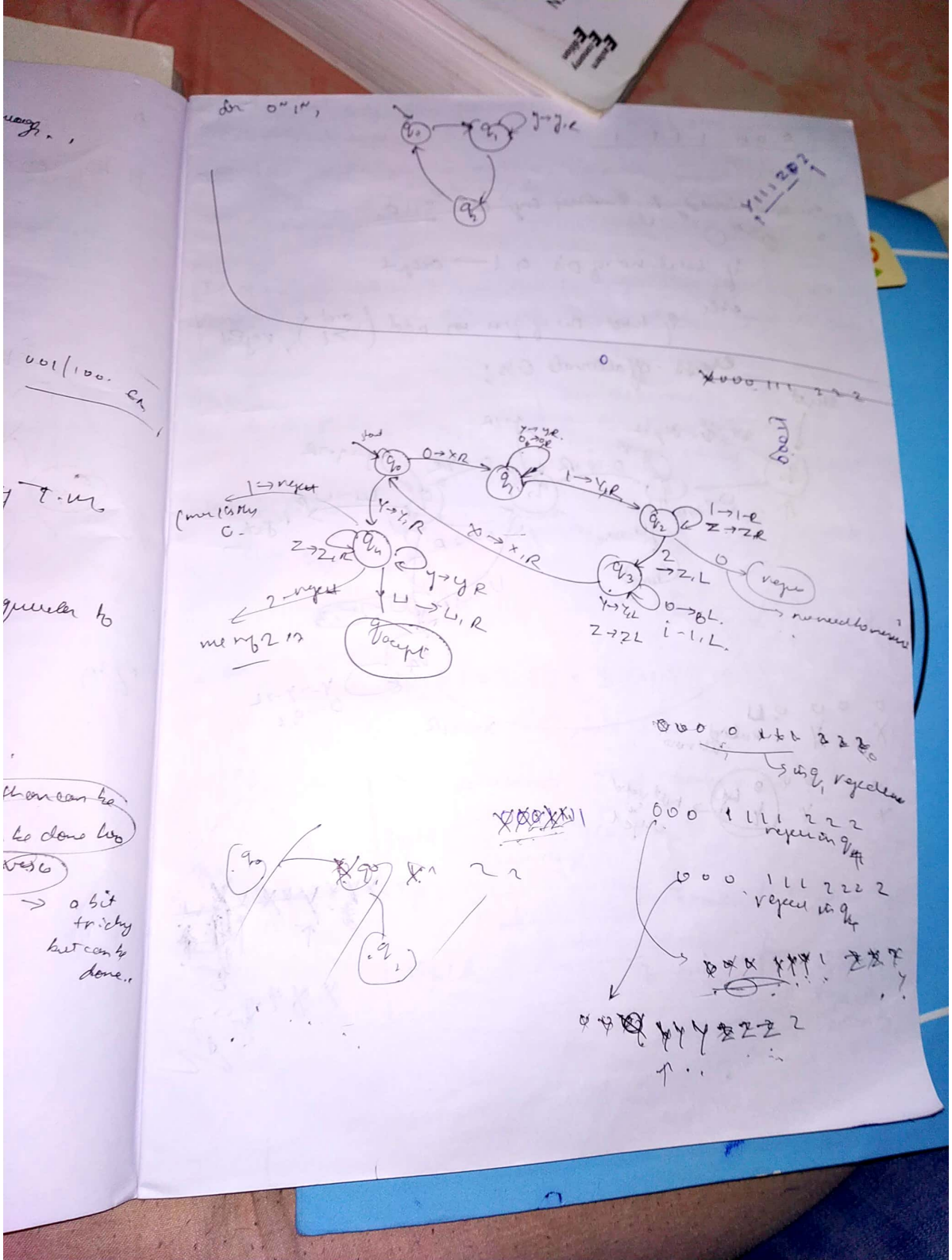
i) 0^{2N} $(N \geq 0)$ — 0, 0000, 00000000 ... } EA.

ii) $0^N 1^N 2^N$, $N \geq 0$ — Ex. 001122

iii) w followed by mirror image of w — Ex. 001|100.

- A FM with
  - → (Right, left, rest) is equivalent to ordinary T.M. proof?
  - → (Right, Stay, <no left>) is not equivalent to T.M. proof??
    - → ∵ we can only traverse once.
    - So it is equivalent to DFA. not TM.
      - ↳ show anything that can be done on DFA can be done here also & vice-versa
      - DFA to TM is easy
      - → a bit tricky but can be done..

Scanned by CamScanner



0 0 0 1 1 1 1

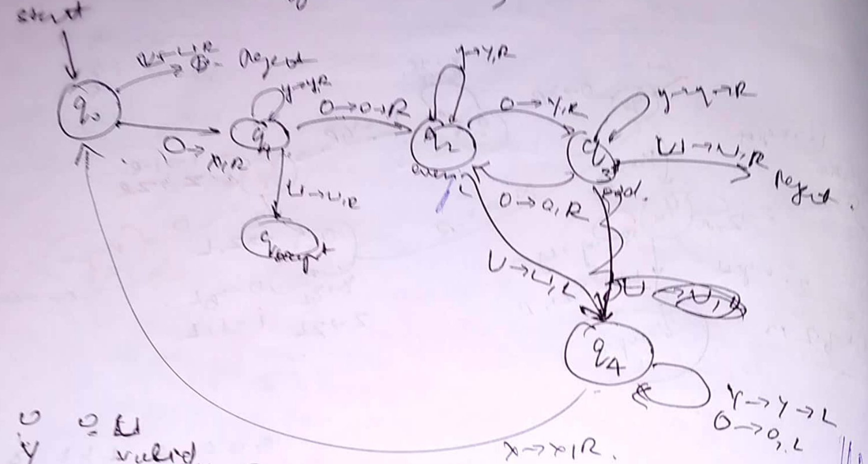
- ~~w followed by ... w' 0110.~~
- 0^{2^n}

  if total no. of 0's is 1 → accept

  else

  if total no. of 0's is odd (and > 1), reject

  Cross off alternate 0's;



0 0 0 0 ⊔

X Y valid till now

0 0 0 0 ⊔

X Y Y ⊔ → not valid so reject

T.M. not only give Yes/NO ans. can also do some calculation & compute.

(use multi tape in addition, multiplication, compare etc, if not explicitly stated to be single tape), : it will give simpler steps than tape 2..

If N is a non-deterministic TM that recognizes a language A., then there is an equivalent deterministic TM, M that recognizes the same language A.

(Deterministic) $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

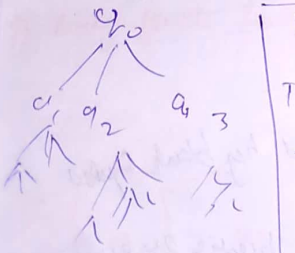
(Non Deterministic) $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$

• Set of all subsets of $Q \times \Gamma \times \{L,R\}$.

let b be the max no of choices for any member in δ. (of w),

max size of $\delta(q, a)$ or $\max |\delta(q, a)|$

$b = |Q| \times |\Gamma| \times \frac{2}{2}$

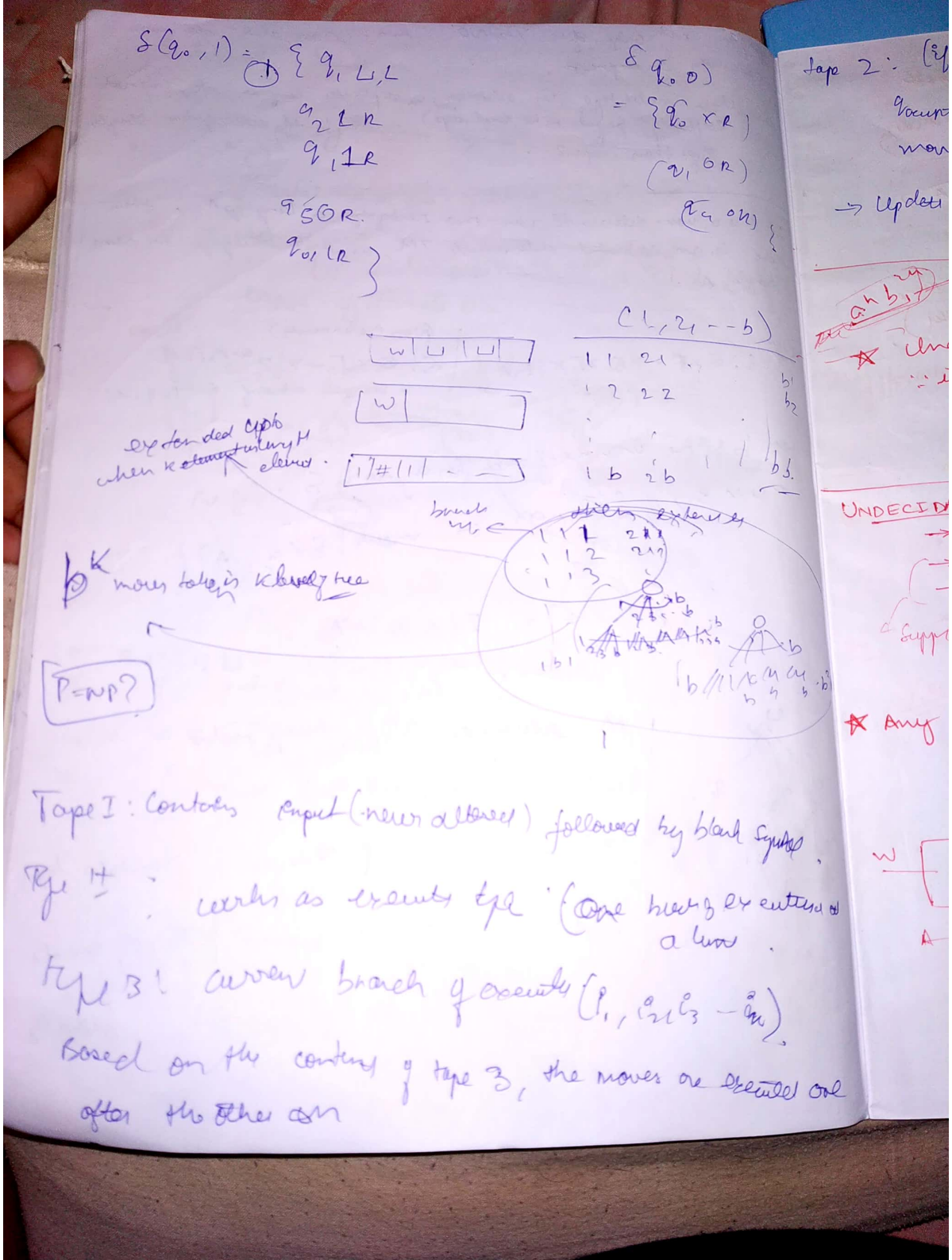


M, deterministic 3 tape turing machine.

Tape 1 (input tape) (never changes)

Tape 2 (execute / simulation tape)

Tape 3 current branch of execution tape or address tape



Tape 2 : (if it goes to q_{accept}) most of also goes to q_{acc}

q_{accept} (halt). If any moves is halt then accept,

move to next step.

→ Update current branch on tape 3. ← Continue.

$a^n b_1^{2n}$

★ Unary notation can be used in T.M's for operations ∵ it is easy (if not mentioned in question not to use unary).

UNDECIDABILITY

→ what languages are turing decidable?

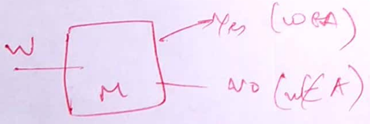
→ Are there languages that are not decidable? — Yes

→ Are there any languages that are not recognizable? — Yes.

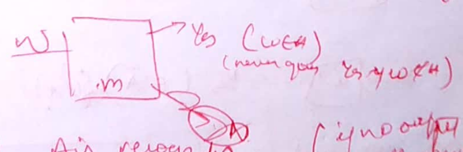
Suppose $\Sigma = \{0, 1\}$, then there are ∞ languages.

★ Any finite language is turing decidable, (J.P. sir said definitely decidable) (recursive)

(and hence turing recognizable also) (recursively enumerable).



A is decidable



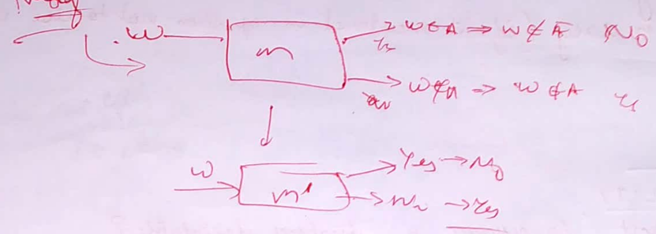
A is recognizable (if no output then both possibility w∈A & w∉A)

If there is a turing machine M that recognizes A & halts on all input, then A is turing decidable (M decides)

Q. A is turing decidable. What can we say about $\bar{A}$?
($\bar{A}$ is complement of A, if $w \in A \rightarrow w \notin \bar{A}$
& if $w \notin A \rightarrow w \in \bar{A}$, over same alphabet).

A. Then $\bar{A}$ is also turing decidable.

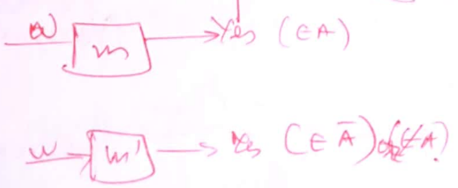
Proof



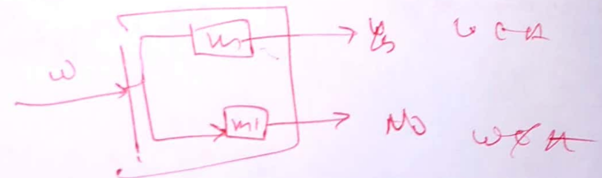
★ The set of "recursive" (or turing decidable) languages is closed under complementation

Q. A is turing recognizable } Can we say
$\bar{A}$ is turing recognizable } something more about A?

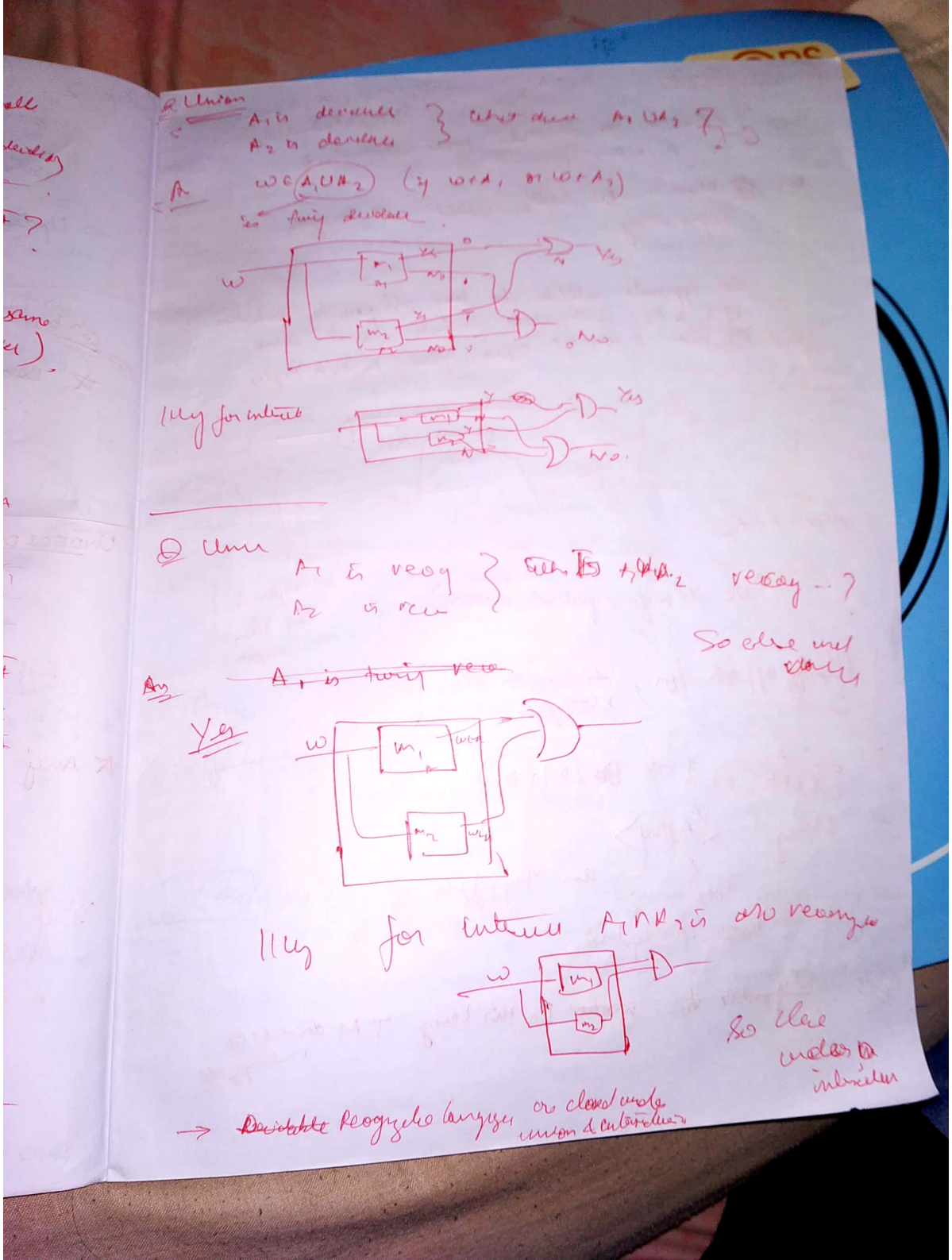
Ans A will be turing decidable



So we construct



So decidable

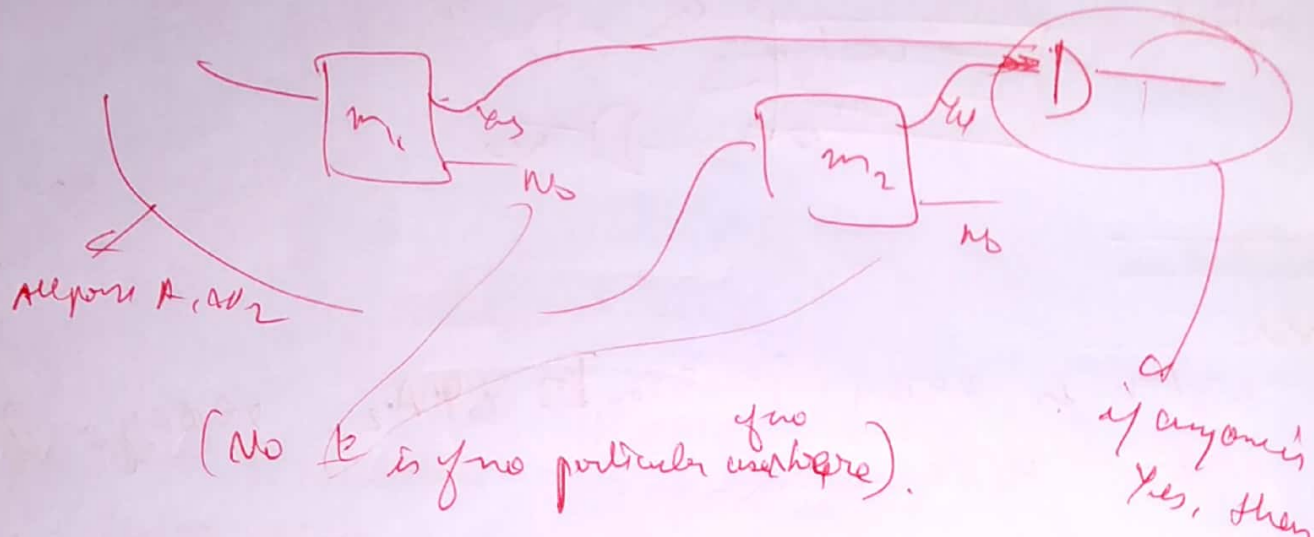


Concatenate $A_1 \cdot A_2$ → $w_1 \cdot w_2 \in A_1 \cdot A_2$ if $w_1 \in A_1$ & $w_2 \in A_2$

Q → If $A_1 \cap A_2$ are turn decidable then what about $A_1 \cdot A_2$?

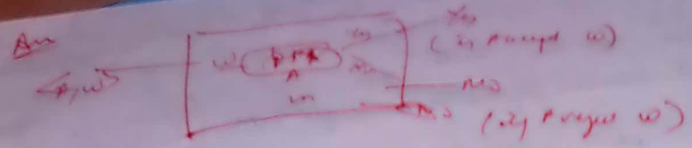
Ans · Note → w_1, w_2 could not be given separately.

So approach will be to try all possible combination of A_1 & A_2. & then check for each i, j at some point both A_1, A_2 get accept. Then $A_1 \cdot A_2$ is accepted.



Turing approach for, decidable also recognizer

Ans



Algorithm
→ Run DFA A with input w
→ If A accept w, say Yes. (go to q_{accept})
else say No.

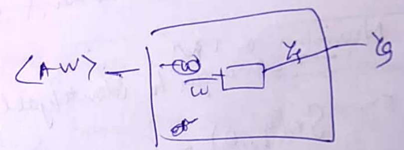
Details
→ Simulation w with DFA with Same set of state rules in T.M.
→ whatever alphabet of the language.

If ⟨T.M, A, w⟩ the problem is,

DFA →Yes →No always but TM →Yes →No →∞ problem.

$L_{TM} = \{\langle M, w\rangle : M \text{ accept } w\}$

Theorem: L_{TM} is undecidable



∴ No default output for No, So it is not turing decidable.

First step: We "construct" or define a language in such a way that there is no turing machine that recognize that language. (using 'Diagonalization' technique)

$0 + 1 + 2 \ldots$

$2 \quad , \; (2^k - 1)$

Idea

① "Number" all the T.M., each number represents a single T.M. (given a no. we must be able to identify machine M_i)

② "Number" each input string (over {0,1}). (given a number, we should be able to identify the word w_n.)

Ex for $L = \Sigma^*(0,1)$,

w_1	0	→	1 (numbering string)
w_2	1	—	2
w_3	00	—	3
w_4	01	—	4
w_5	10	—	5
w_6	11	—	6
w_7	000	—	7
w_8	001	—	8
w_9	010	—	9

* Note, decimal equivalent will cause problem, ∵ 0, 000, 00 etc will have same no. then.

length ($k = 4$), value in dec ($n = 5$)

0101 ↓

$2^1 + 2^2 + 2^3 + \ldots + 2^{k-1} + (n+1)$

$= 2^k - 1 + n$

number $= 2^k + (n-1)$ → how?

other way round,

($i = 18$) = ? (given index as i)

$k = \lfloor \log(i+1) \rfloor$, $n = ?$

put k in

$i = 2^{\lfloor \log(i+1) \rfloor} + (n-1)$

$i = 2^k + n - 1$

$n = (i+1) - 2^k$

15 → 0000
16 → 0001
17 → 0010
18 → 0011

Exercise

① given a no. 'i', what is string w_i?

② given a string w, [we know length k & number n] how to find i st. $w = w_i$?

Deliberately left.

Number a TM.

~~$2 + 2^3 + 2^?$~~ enough to identify all moves ("encode" all move in δ).

Ex. 0000

$\delta(q_0, 0) = (q_1, \sqcup, R)$

$\delta(q_1, 1) = (q_3, 1, L)$

~~$\delta(q_1, 0) = (q_{accept}, 0, R)$~~

$\delta(q_3, 0) = (q_{accept}, 0, R)$

$\delta(R_0, 1) = (q_0, 1, R)$

- first order states in such a way that q_1 is start state, q_2 is accept state.

M. (given a no. we must be able to identify machine M_i)

to identify the word w_n)

Exercise

given no. 'i', what is string w_i?

given a string w, [we … & number n how to find p s.t. $w = w_p$?

[i of i]

000
001 delimiter to get
010
011

… in δ).

…, R)

…, R)

… is start state,

old names: q_0 → new names: q_1; q_{accept} → q_2; q_1 → q_3; q_3 → q_4.

writing moves after renaming,

$\delta(q_1, 0) = (q_3, \sqcup, R)$
$\delta(q_3, 1) = (q_4, 1, L)$
$\delta(q_4, 0) = (q_2, 0, R)$
$\delta(q_1, 1) = (q_1, 1, R)$
$\delta(q$

$\{0, 1, \sqcup\}$ → x_1, x_2, x_3 $\{L, R\}$ → d_1, d_2

00
01 now these (5 nos) will identify the turing machine.

00
01
10
000
001
011
011

$\delta(q_i, x_j) = (q_k, x_l, d_m)$

for $\delta(q_1, 0) = (q_3, \sqcup, R)$,

one way of encoding is,

$0\,|\,0\,|\,0\,|\,000\,|\,000\,|\,00$
(0^i, 0^j, 0^k, 0^l, 0^m)
↳ one used as delimiter to separate 0.

Ex, L → 0, R → 00, q_1 → 0, q_2 → 00, …, q_n → 0000, 0 → 0, 1 → 00, ⊔ → 000

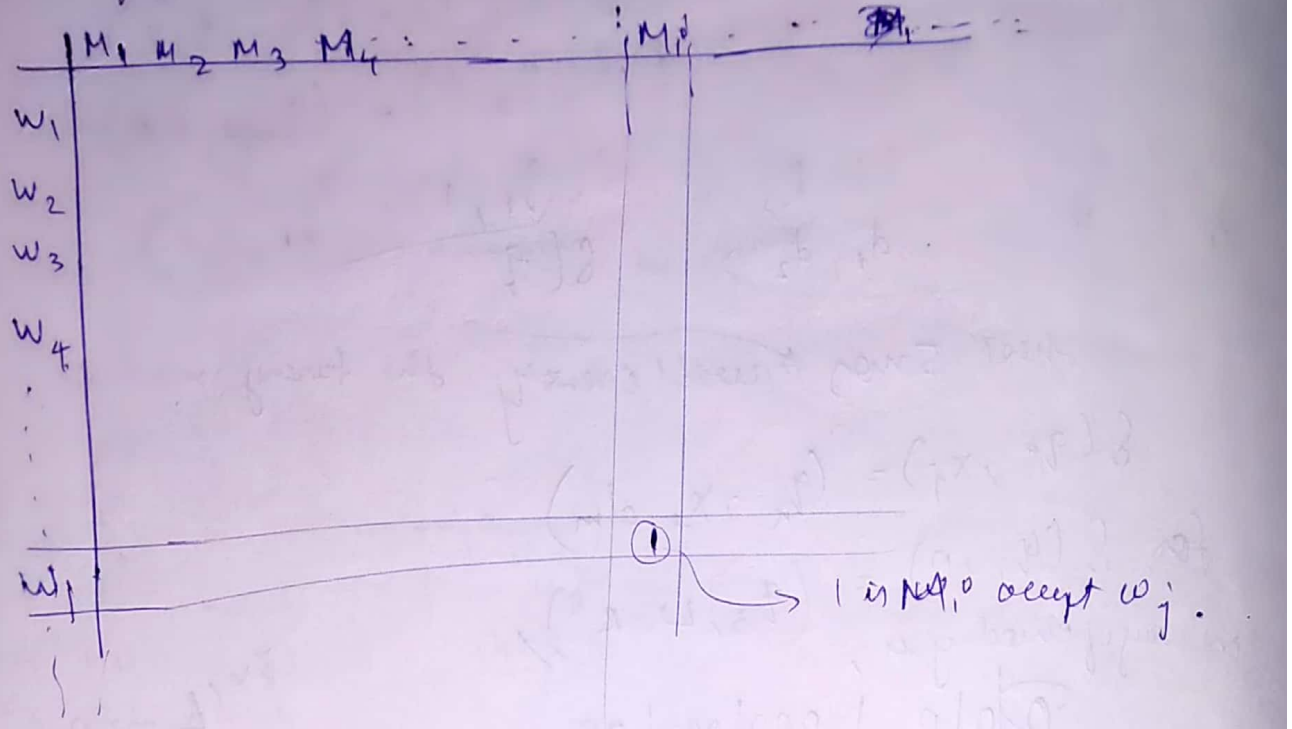
So for numbering turing machine,

111 <move 1>11 <move 2>11 <move 3>11 <move 4>11 — 11 <move m>
↳ 11 used as delimiter in b/w. (2 ones)

↳ given a "number" for the turing machine.

But same TM can have different nos., based on order of moves,
but a TM will have a number & given a no., it will represent one TM, (surjective/onto) (Q (queue …)) so it's surjective from us:- TM → ano. ano → TM.
eventually diff. nos. can represent same TM.

Defining a language s.t., there is no TM that recognizes the language.



L_d (diagonal) = $\{w_i : \text{entry } \langle i, i \rangle \text{ is } 0 \text{ in this matrix}\}$

from

$L_d = \{w_i : M_i \text{ does not accept } w_i\}$.

→ Show that L_d is not recognized by any machine M_j

Assume → Assume, M_j identifies L_d,

what about entry $\langle j, j \rangle$?, does M_j accept w_j ?

{If w_j in L_d}

Case I

From the defn of L_d, if $w_j \in L_d$, $\Rightarrow M_j$ does not accept w_j

But our assumption is that M_j recognizes L_d.

M_j accept w_j → contradiction

Case II If $w_j \notin L_d$, M_j recognizes L_d only if it accept any of L_d;

it doesn't accept w_j (because $w_j \notin L_d$) $\Rightarrow w_j \in L_d$ (by defn of L_d) → contradiction

what about entry $\langle j, j \rangle$?

case I $\langle j, j \rangle = 0$ if $w_j \in L_d$ & M_j recognizes

case 2 $\langle j, j \rangle = 1$

w_j ∈ L_d

$\rightarrow w_j$ ∉ .

w .

①

12/09/19

$L_{TM} = \{ \langle M, w \rangle , M \text{ accepts } w \}$

given any $\langle M, w \rangle$! Does M accept w?

ATM : show that this is undecidable .

o Construct a language that is not even turing recognizable

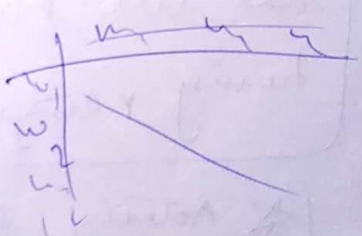
- Number all T.M.'s ,

[Each number uniquely represent a single T.M. & it it doesn't represent any T.M.) .

$M_1 \; M_2 \; M_3 \; M_4 \; \ldots$

L_d (diagonal) $= \{ w_i \text{'s entry } (i, i) \text{ is 0 in the matrix} \}$

$L_d = \{ w_i \cdot M_i \text{ doesn't accept } w_i \}$



Claim : L_d is not recognizable by any T.M.

[L_d is not turing recognizable] .

i) what could be entry $\langle j, i \rangle$ in matrix

ii) Does M_j accept w_j .

Scanned by CamScanner

entry $\langle i,j \rangle$: Case 1 Let M_j accepts L_d

then $w_j \notin L_d$.

But : M_j accept w_j

→ $w_j \notin L_d$ & M_j accept w_j (so w_j accepts things outside L_d also), So contradiction than M_j accept L_d.

→ But by our assumption that M_j accepts L_d & $w_j \notin L_d$

$\Rightarrow \langle i,j \rangle = 0$

Contradiction

Case 2 $\langle j,j \rangle = 0$

then $w_j \in L_d$.

By assumption that M_j recognizes L_d,

M_j must accept L_d, or entry$\langle i,j \rangle = 1$

(contradiction)

So we have a Langg L_d which is not even turing recognizable

(* Actually there are ∞ such Languages which are not turing recognized.)

If L_{TM} is decidable (if we have a TM M_{TM} that decides L_{TM}), then L_d is also decidable.

Th: The Language $TM = \{\langle M, w \rangle = M \text{ accepts } w\}$ is undecidable.