

THOC

Congruence mod n

$a \equiv_n b$ ie. $(a-b)$ is an integer multiple of n

For eg. $a = 5$
 $b = 1$
 $n = 4$

$$a \equiv_4 b$$

$$\therefore a-b = n*k$$

$$\text{Just eg. } 4 = 4*k \quad k = 1$$

Eg. Find pairs of a,b if $n=3$

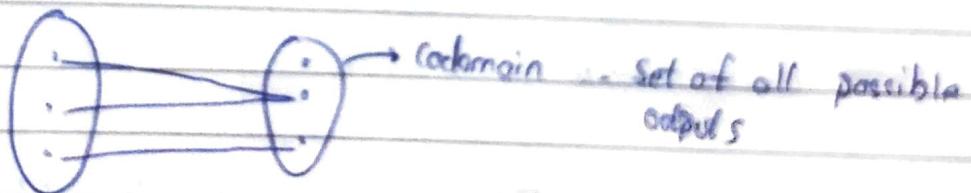
Sol:

$$a \equiv_3 b$$

$$R = \{(0,3), (0,0), (1,0), (1,4), (2,1), \dots\}$$

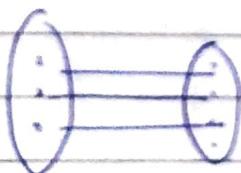
3 equivalence classes we will get as we have $(a-b) \mod 3$

⇒ Onto



Not onto because codomain range ≠ codomain
 i.e. some elements of range are missing

⇒ Into



For one value of x only one value of y should be in codomain

"Into fun"



Q.

$$f: \mathbb{R} \rightarrow \mathbb{R} \quad f(x) = x^2$$

Into Onto

$$f: \mathbb{R} \rightarrow \mathbb{R}^+ \quad \text{No} \quad \text{Yes}$$

$$f: \mathbb{R}^+ \rightarrow \mathbb{R} \quad \text{Yes} \quad \text{No}$$

$$f: \mathbb{R}^+ \rightarrow \mathbb{R}^+ \quad \text{Yes} \quad \text{Yes}$$

INTO : Only one value of x matches with only one value of y in codomain

ONTO : Range = Codomain

$$a^k = aaaa\dots a$$

$$a \in \Sigma$$

$$L \subseteq \Sigma^*$$

$$x^k = xx\ddots \dots x$$

$$x \in \Sigma^*$$

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

$$L^k = LLL\dots L$$

$$a^0 = \lambda$$

$$\Sigma^0 = \{\lambda\}$$

$$x^0 = \lambda$$

$$L^0 = \{\lambda\}$$

\Rightarrow Concatenation

Apply in strings as well as languages set of strings

x
y

xy

$$L_1 \cdot L_2 \subseteq \Sigma^*$$

$$L_1 \cdot L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$$

e.g. L_1

$$\{a, b\}$$

$$L_2 = \{c, d\} = \{ac, ad, bc, bd\}$$

e.g.

$$L_1 = \{ab, bab\}^* \xrightarrow{o \rightarrow \infty} \cup \{bab, bbb\}^*$$

$$L_2 = \{x \in \{a,b\}^* \mid \begin{matrix} \nearrow \text{number of } a \\ n_a(x) \geq n_b(x) \end{matrix}\} \xrightarrow{\text{number of } a}$$

Soln

$$L_2 = \{ab, aab, aabb, aaab, aba, ba, \dots\}$$

\hookrightarrow represents
arbitrary time
concatenate
 $ab \& bab$

$$L_1 = \{ab, bab, abbab, babab, ababab, \dots \} \quad (\text{neglecting } bab, bbbb)$$

\hookrightarrow neglecting ~~ab, bab~~ \hookrightarrow missing ab & bbbb

e.g.

$$L = \{byb \mid y \in \{a,b\}^*\}$$

Soln

$$L = \{bb, babb, baaab, bbbb, bababb, \dots\}$$

e.g. Either start with b or end with b

$$L = \{byb \mid y \in \{a,b\}^*\}$$

\Rightarrow Recursive Defⁿ

$$\text{For eg. } n! = \begin{cases} 1 & n=0 \\ n(n-1)! & n>0 \end{cases}$$

Recursive defⁿ of L^*

$$1) \wedge \in L^*$$

$$2) \text{ For any } x \in L^* \& y \in L^*, xy \in L^*$$

$$3) \text{ No string other than (1) \& (2) will come in } L^*$$

Q. Recursive defⁿ of Palindrome strings

- Ans.
- 1) $\lambda \in L^*$
 - 2) For any $x \in L^* \& y \in L^*, xy = yx$
 - 3) No

- Q1
- 1) $\lambda \in Pal$ { Null strings }
 - 2) For any $a \in \Sigma, a \in Pal$ { Single character }
 - 3) For any $x \in Pal \& a \in Pal, axa \in Pal$
 - 4) No string is in pal unless it can be obtained by using 1, 2 & 3.

Q1 The set ω of all strings of the form 0_+^{ij} where $i \geq 2j$ and $i, j \in \mathbb{N}$

Q2 The set V of all strings of the form 0_+^{ij} where $j \leq i \leq 2j$

Q.

→ Structural Induction

* Regular Language

→ A regular language over an alphabet Σ is one that can be obtained from given basic languages using the operations Union, Concatenation & Kleene.

$$\begin{aligned} L_1 &= \{\alpha\}, \quad L_2 = \{\beta\} \\ L_1 L_2 &= \{\alpha\beta\} \quad L_1 \cup L_2 = \{\alpha, \beta\} \end{aligned} \quad \left. \begin{array}{l} \text{These are regular languages} \\ \hline \end{array} \right\}$$

→ A regular expression is an algebraic repres' of regular language.

Lang R.E.

$\{\}\}$

λ

$\{\}\} \rightarrow \text{remove}$

$\{0\}$

0

$\{\}\} \rightarrow C$

$\{001\}$

001

$\cup \rightarrow +$

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

$0 + 1$

$$\{0, 10\}$$

$0 + 10$

$$\{0, 1\} \{001\} \quad (0 + 1) 001$$

$$\hookrightarrow \{0\} \cup \{1\} \{001\}$$

$$\{110\}^* \{0, 1\} \quad (110)^* (0 + 1)$$

$$\{10, 11, 11010\}^* \quad (10 + 11 + 11010)^*$$

$$\{0, 10\}^* (\{11\}^* \cup \{001, 1\}) \quad (0 + 10)^* ((11)^* + 001 + 1)$$

$$\{0\}$$

\emptyset

$$L_1 \cup L_2$$

$$\gamma_1 + \gamma_2$$

$$L_1 \cdot L_2$$

$$\gamma_1 \gamma_2$$

$$L_1^*$$

$$\gamma_1^*$$

1	2	3	4	5	6	7	8	9	10	11	12
01*	(01)*	1*	(0+1)*	(0+1)*	1*	0*	01*	10101	01*	10101	01*

$$\text{Q. } (0+1)^* 01 (0+1)^* + 1^* 0^* = (0+1)^*$$

Are they equal or not.. yes equal

$$SOL^n (0+1)^* = \{1, 0, 1, 00, 11, \dots\}$$

$$\text{Q. Strings of even length } \Sigma = \{0, 1\}$$

$$L = \{\cancel{00}, 01, 10, 11, 0000, \dots\} = \{00, 01, 10, 11\}$$

$$RE = (00 + 01 + 10 + 11 + 0000 + \dots) = (00 + 01 + 10 + 11)$$

Q

Strings with an odd number of 1's

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

$$(0^* 1 0^*) (0^* 1 0^*)^* 0$$

$$(0^* 1 0^* 1) (0^* 1 0^*)^*$$

$$(0^* 1 0^*) (0^* 1 0^* 1 0^*)^*$$

$$SOL^n$$

Also check that
null string must
also be accepted
in the answer.

$$L = \{0\}^* \{1, 11, 111, 1111, \dots\}$$

$$\text{Ans. } 0^* 1 0^* (10^*)$$

$$0^* 1 0^* (10^* 1 0^*)$$

$1 (00+11)^*$ → It does not contain all the languages

Q

Strings ending in 1 & not containing 00, $\Sigma = \{0, 1\}$

$$(1 + 0)^+$$

Q

R.E. for identifiers of C language. $\Sigma = \{a, \dots, z, A, \dots, Z, 0, \dots, 9, -, _\}$

int temp ✓

int 2tempX

int _temp ✓

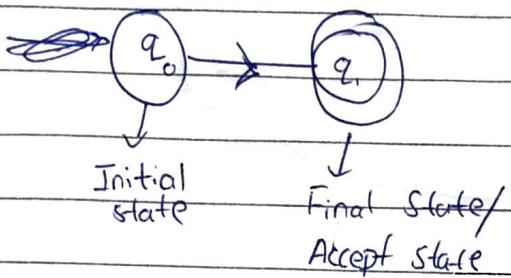
int temp ✓

$$L = \{a, \dots, z, A, \dots, Z\}$$

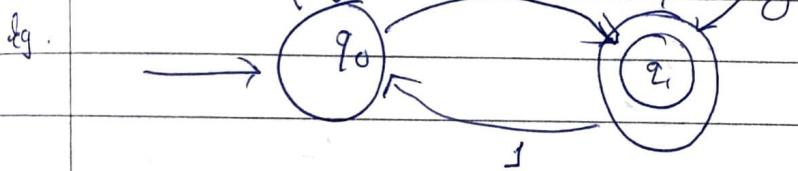
$$d = \{0, \dots, 9\}$$

$$RE = (d + -)(d + d + -)^*$$

| FINITE AUTOMATA |

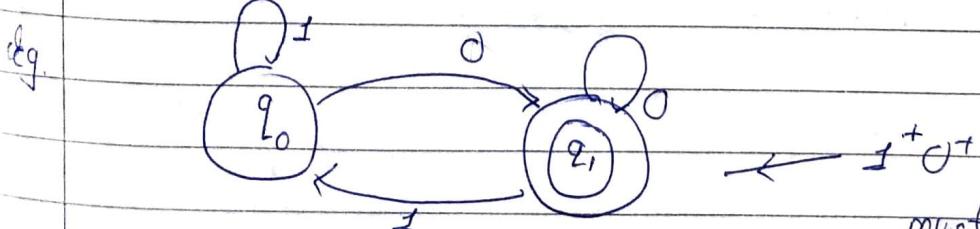
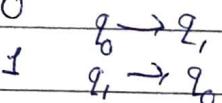


input State



initial state $\rightarrow q_0$

If 01 is input the 0 for 0



0, 10, 010, 1000

It must end with 0
to achieve q_2

Ans. $(0+1)^* 0$

OR

$(0^* 1^*)^* 0$

exactly 2 0's $\rightarrow \underline{1^*} 0 \underline{1^*} 0 \underline{1^*}$

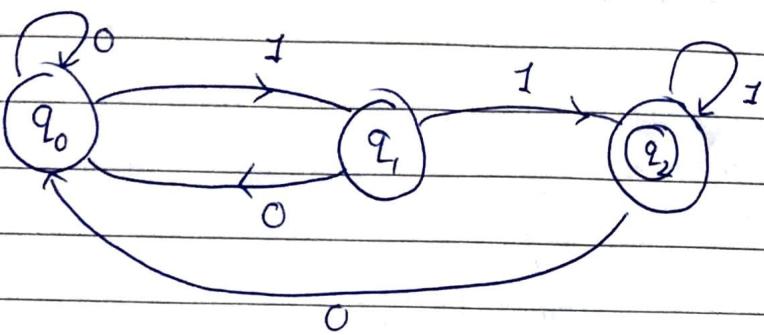
At least 2 0's $\rightarrow (0+1)^* 0 (0+1)^* 0 (0+1)^*$

d.g.

②

This is final automata with final state = initial state & it will accept null string

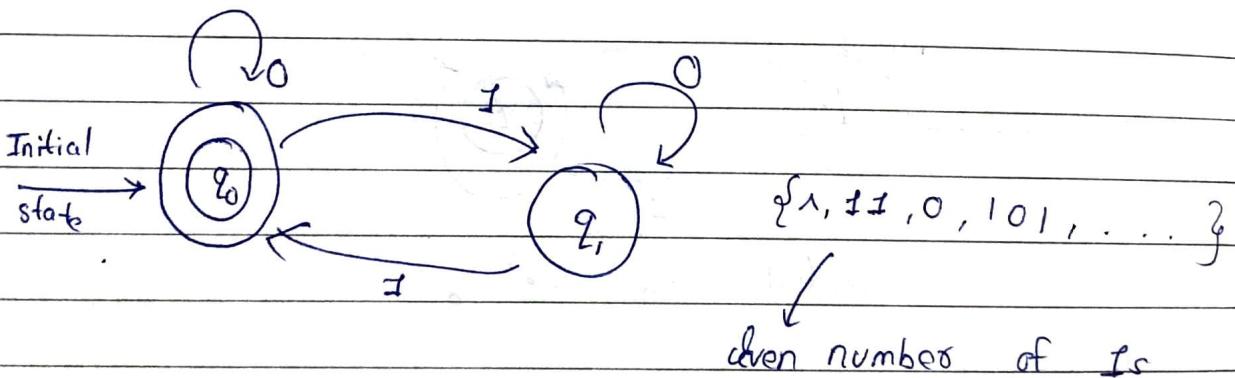
d.g.



Language Strings should end ~~not~~ with 11

$$R.E. = (0+1)^* \cdot 11 \cdot \cancel{(0+1)^*}$$

d.g.



$$\begin{aligned}
 R.E. = & \cancel{(0+11)^*} \overbrace{\left[0^* (10^* 1)^* 0^* \right]}^{\text{OR}} \\
 & \left[\left(0^* 10^* 10^* \right)^* \right]
 \end{aligned}$$

\Rightarrow Defⁿ of finite Automata or FSM (Finite State machine)

It is a 5-tuple

1) Θ is a finite set $\Theta = \{q_0, q_1, \dots\}$

2) Σ is an alphabet of input symbols

3) $q_0 \in Q$

4) $A/F/q_f \subseteq Q$ \rightarrow More state than one final

5) Transition funⁿ $\delta : \Theta \times \Sigma \rightarrow Q$

Ex. $L_1 = \{\lambda, a, aa, aaa, \dots\} \xrightarrow{\text{Ans}} \text{AFA}$

Ex. $L_2 = \{\lambda, a, b, ab, bb, aa, \dots\}$

solⁿ AFA

Extended transition funⁿ

$\delta^* : \Theta \times \Sigma^* \rightarrow \Theta$

Ex. $q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{c} q_3$

$$\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$$

$$= \delta(\delta(\delta^*(q_0, a), b), c)$$

$$= \delta(\delta(\delta(\delta^*(q_0, \lambda), a), b), c)$$

$$= \delta(\delta(\delta(q_1, a), b), c)$$

$$= \delta(\delta(q_2, ab), c)$$

$$= \delta(q_3, abc)$$

Q Let $M = (\Theta, \Sigma, q_0, A, \delta)$ be an FA.
we define "the fun"

$$\delta^*: \Theta \times \Sigma^* \rightarrow \Theta$$

as follows

$$1) \text{ For any } q \in \Theta, \delta^*(q, 1) = q$$

$$2) \text{ For any } q \in \Theta, y \in \Sigma^* \text{ & } a \in \Sigma$$

$$\delta^*(q, ya) = \delta(\delta^*(q, y), a)$$

\Rightarrow Acceptance of an FA

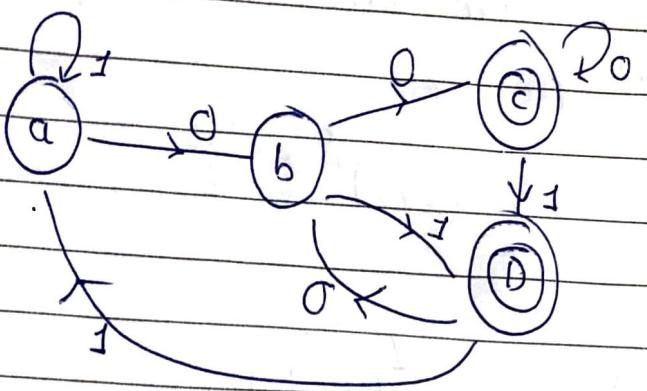
Let $M = (\Theta, \Sigma, q_0, A, \delta)$ be an F.A. A string $x \in \Sigma^*$ is accepted by M if $\delta^*(q_0, x) \in A$

If a string is not accepted, we say it is rejected by M . The language accepted by M or the language recognized by M , is the set

$$L(M) = \{x \in \Sigma^* \mid x \text{ is accepted by } M\}$$

If L is any language over Σ , L is accepted as recognized by M iff $L = L(M)$

Ex.

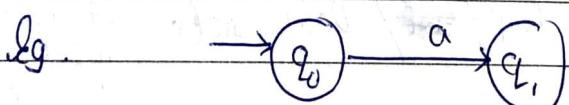


Ex. FA, binary strings divisible by 3

⇒ Types of automata

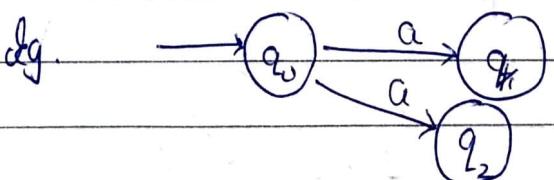
- 1) DFA (Deterministic)
- 2) NFA (Non-deterministic)

→ Det DFA mean that for transition fun" we have clearly possible state to reach then it is called DFA



For every input we need to have state transition to one state.

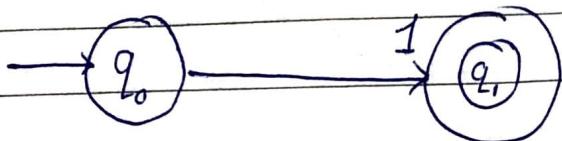
→ NFA mean that for transition fun" we have no clearly possible state to reach then its NFA.



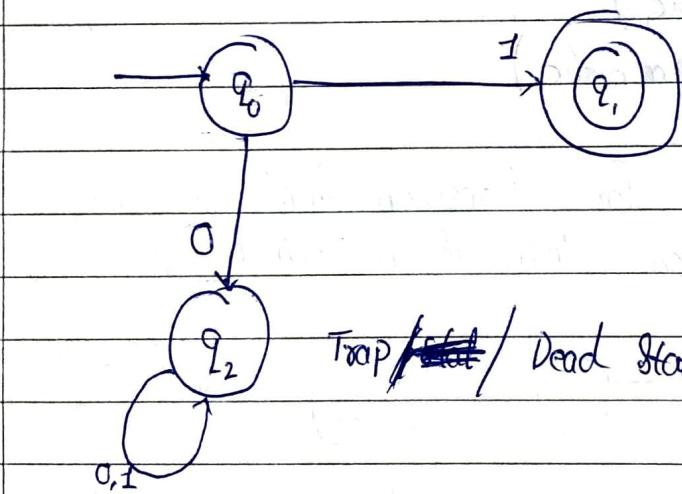
Q Construct a DFA which accept all strings starts with 1

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

Soln R.E. = $1 \cdot (0+1)^*$, $L = \{1, 11, 10, 101, \dots\}$



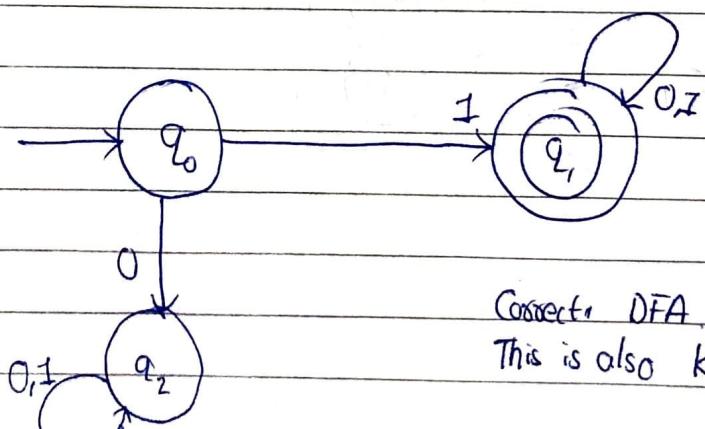
- If we make a self loop for input 0 in q_0 then it will accept string 01 too. So NP.
- If we make $q_0 \xrightarrow{0} q_2$ then also for input 0 it will reach final state. So NP.
- So make another state



Trap ~~/~~ Dead State (It indicates that if input is 0 then there is no chance to reach final state)

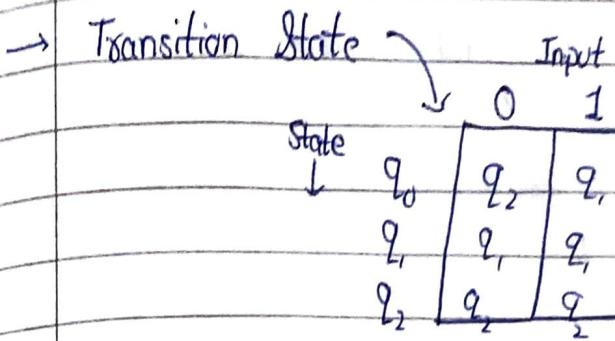
Did because in DFA it is necessary.

- Self loop for input 0, 1 & self loop



Correct DFA.

This is also known as transition diagram.

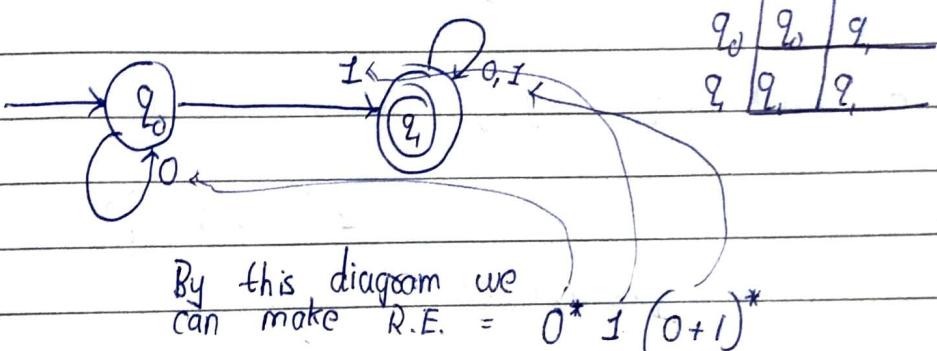


Q $\Sigma = \{0, 1\}$, strings contain 1

$$L = \{1, 01, 101, \dots\}$$

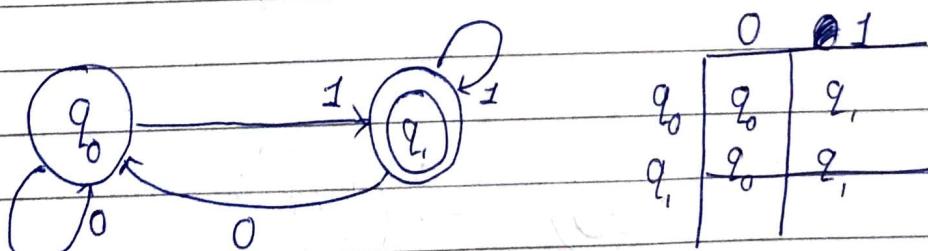
$$\text{R.E.} = \cancel{(0+1)^*} \cdot 1^+ \quad \text{or} \quad \underline{(0+1)^*} \ 1 \ \underline{(0+1)^*}$$

DFA



Q Strings end with 1

$$\text{R.E.} = (0+1)^* \cdot 1, L = \{0, 001, 101, \dots\}$$

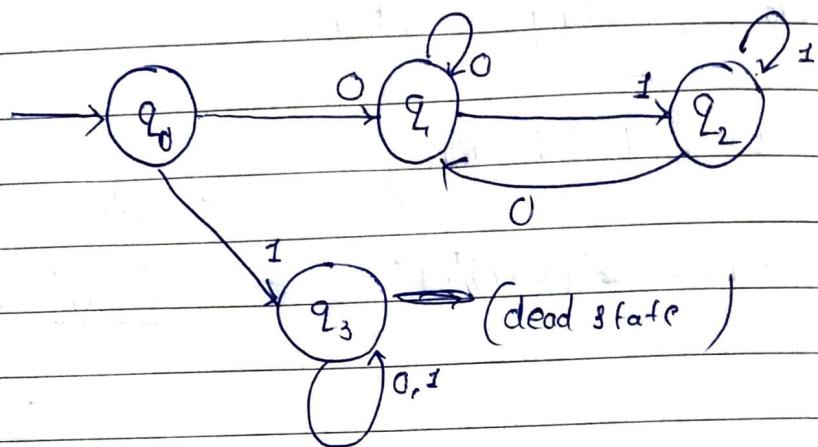


→ From q_1 for input 0 ~~then 1~~ we do self loop.
then 10 string will also be accepted.

Q) DFA strings start with 0 end with 1

01, 001, 0111

$$\text{DFA}^{\text{RE}} = 0(0+1)^*1$$

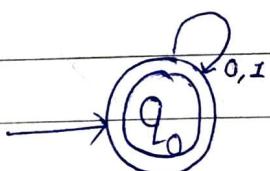


Q) DFA which accept language consist of all string 0 & 1

Soln

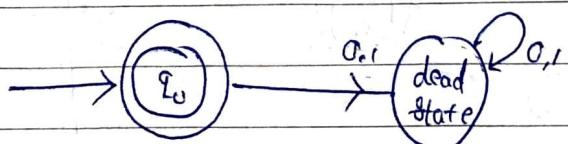
$$\Sigma^* = \{0, 1\}^*$$

$$L = \{ \lambda, 0, 1, 00, 01, \dots \}$$

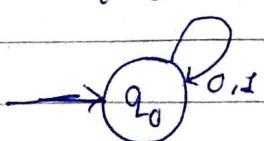


Q) DFA $\Sigma = \{0, 1\}$, $L = \{\lambda\}$

Soln



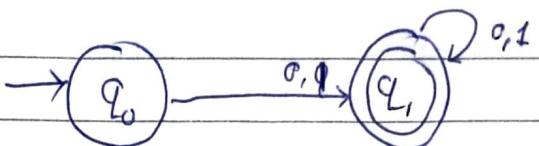
Q) DFA, $L = \{\lambda\}$, $\Sigma = \{0, 1\}$



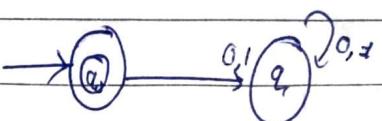
No final state as we don't want to accept anything.

Q DFA

$\Sigma = \{0, 1\}$, $\Sigma^* - \{1\}^*$ (The DFA which accepts all strings except null)



Another approach



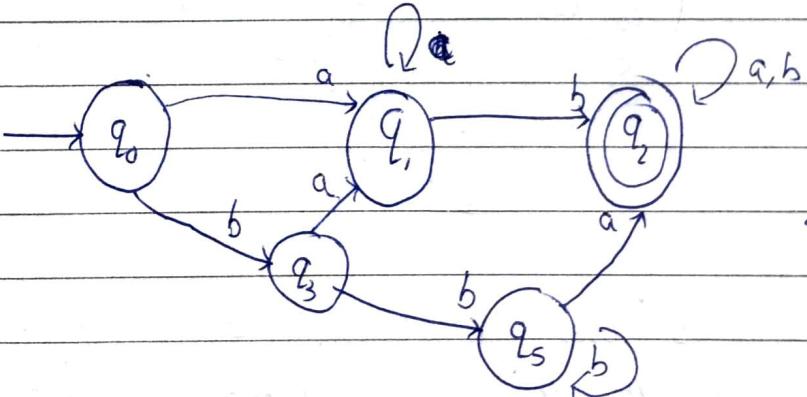
do for null $L = \{1\}^*$
then do conversion
~~interchange~~ ~~final & no~~

Final state \rightarrow Non-final
Non-final \rightarrow final.

Q DFA contains either strings ab or bba

$$L = \{ab, bba, aaab, bbaab, \dots\} \quad abba$$

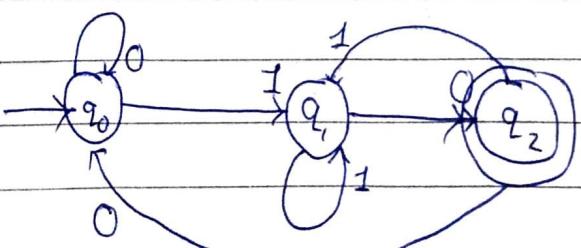
~~$(ab)^*$~~ $(a+b)^*(ab+bba)^*(a+b)^*$



Q strings end with 10, $\Sigma = \{0, 1\}$

$$L = \{ \text{0xx} 10, 010, 110, \dots \}$$

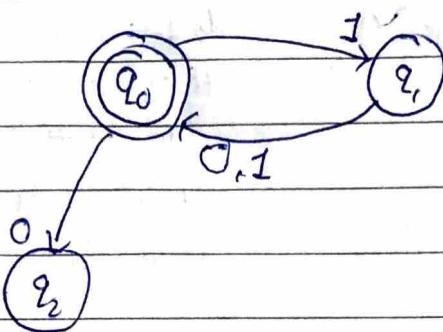
Test case:
01010101110



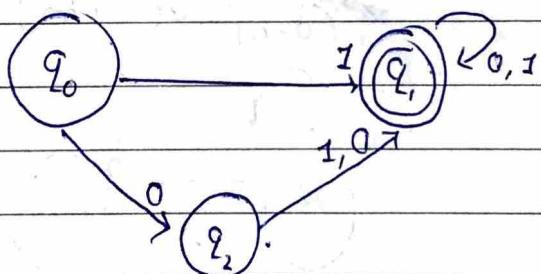
Q DFA

$$\left. \begin{array}{l} RE = (11 + 10)^* \\ RE = (0+1)^*(1+00)(0+1)^* \end{array} \right\}$$

Soln 1) $L = \{1, 11, 10, 1110, 1010, 11111, \dots\}$



2) $L = \{1, 00, 110, 000001, \dots\}$



R.L = A language L over an alphabet Σ is regular iff there is an FA with input alphabet Σ that accepts L .

Eg. $L_1 = \{1\}$, this is R.L.

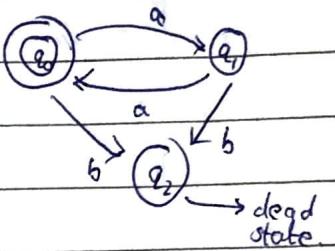
→ If DFA/NFA or RE can be constructed for then the language can be known as R.L.

→ Finite Automata does not have any memory

Q. a^{2n} is this a R.L.

$$L = \{aa, aaga, aaaca, \dots\}$$

→ Here we can create DFA so it is R.L.



e.g. $a^n b^n$, Regular language or not.

\Rightarrow Distinguishing one string from another.

Let L be a language in Σ^* & x any string in Σ^*
 The set L/x is defined as

$$L/x = \{ z \in \Sigma^* \mid xz \in L \}$$

→ Two strings x & y are said to be distinguishable w.r.t. L if $L/x \neq L/y$. Any string Z that is in one of the two sets but not the other is distinguish x & y w.r.t. L , if $L/x = L/y$, x & y are indistinguishable w.r.t. L .

Eg. DFA whose number is divisible by 3

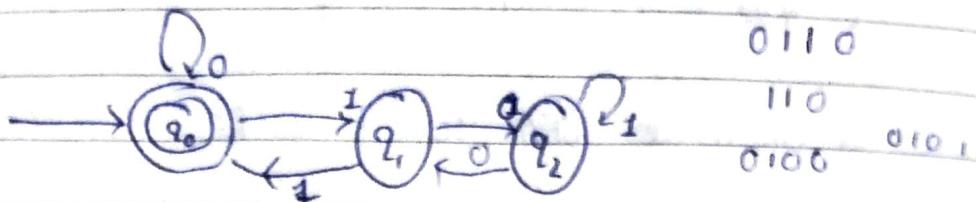
0000 → accepted

0011 → accepted (\because Decimal value 3)

0110 → "

Solⁿ 3 values of remainders = 0, 1, 2 , L =

→ Construct states = Total number of values of remainder

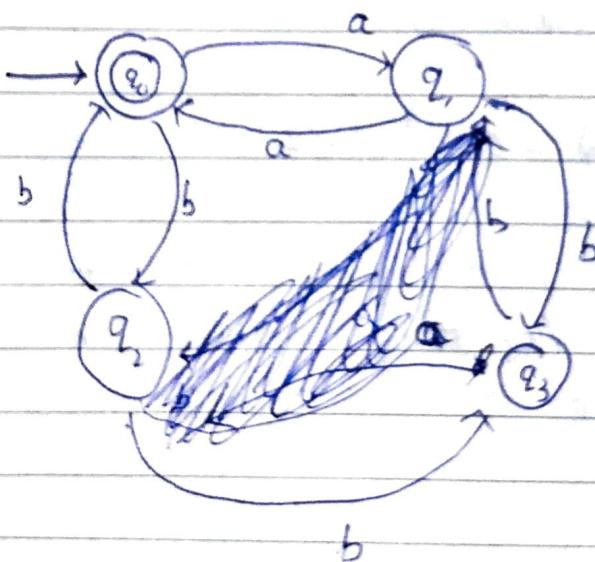


Just see values & put it in
state as per values .

Q DEF DFA , no. of a's & no. of b's are even
 $\Sigma = \{a, b\}$

$$L = \{ \lambda, aa, bb, abab, babababa, \dots \}$$

Solⁿ



Just changing the
final state we can
get all the combinations

- 1) A odd, B odd
- 2) A even, B odd
- 3) A odd, B even

⇒ Closure properties of RL

1) If we have 2 languages L_1, L_2 , then on doing union of both language, if the resultant language is R.L. then we can say that R.L. is closed under union.

$$\text{e.g. } L_1 = \{0, 00, 000, \dots\} = 0^+ \\ L_2 = \{1, 11, 111, \dots\} = 1^+$$

$$L_1 \cup L_2 = \{0, 1, 00, 11, 001, 0111, \dots\} = (0^+ + 1^+)$$

2) RL are closed under intersection

3) RL are closed under concatenation

4) RL are closed under Kleen's closure, complement & difference too.

$$L^* = L_1 \cdot L_2 \cdot L_3 \dots \xrightarrow{\text{As RL is closed under concatenation it is automatically closed under Kleen's closure}}$$

e.g. $L_1 = \{x / 00 \text{ is not a substring of } x\}$ (First we will create DFA of 00 as a substring & then will interchange final & non-final states)

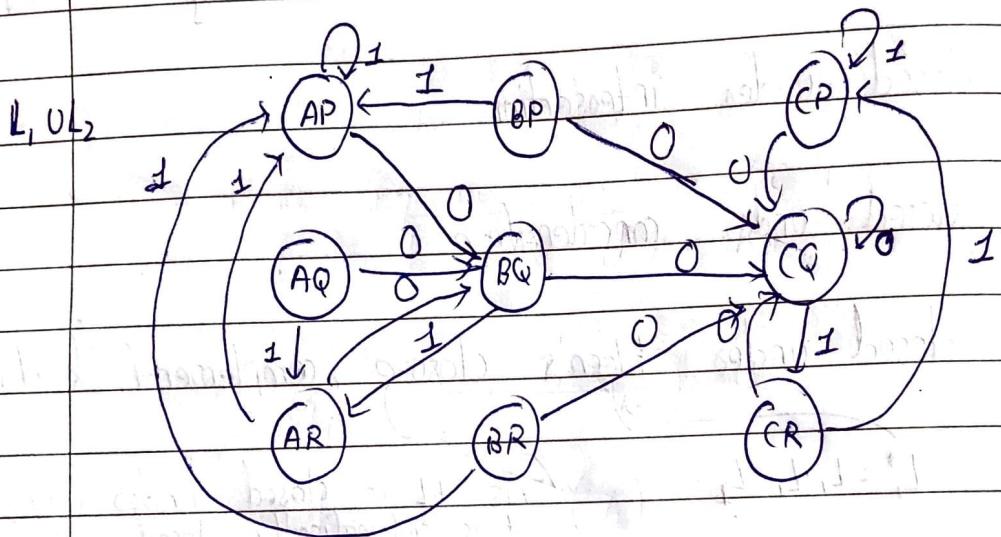
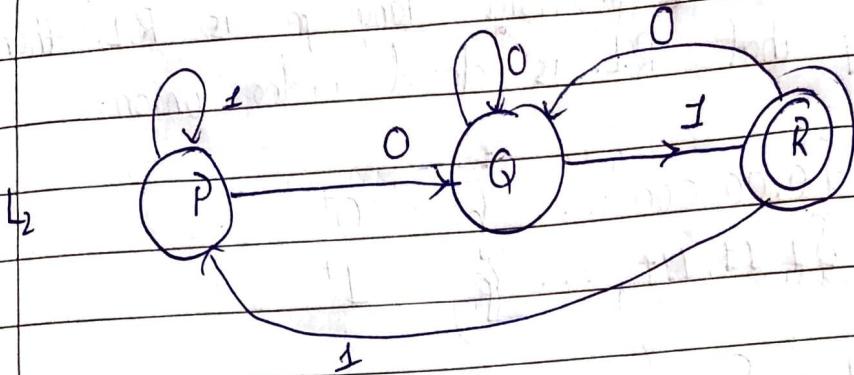
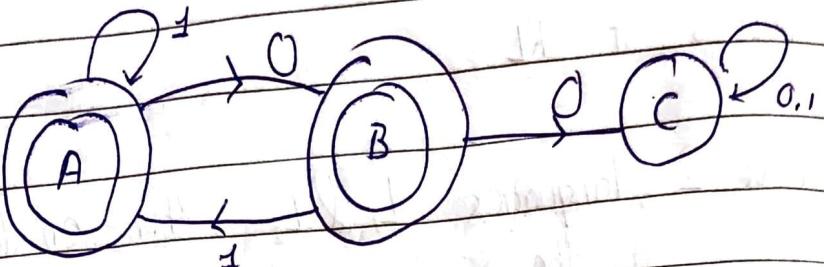
$$L_2 = \{x / x \text{ ends with } 01\}$$

We will take union of these 2 languages

$$L_1 = \{01, 10, 11, 0, 1, \dots\}$$

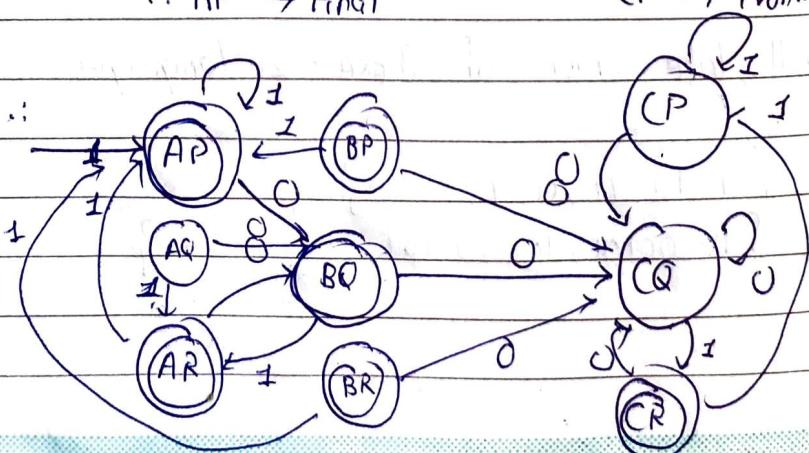
$$L_2 = \{10, 00101, 101, 11101, 001, \dots\}$$

Inter changing
initial & final stages
in substituting



$$L_1 \cup L_2 = \{0, 1, 01, 10, \dots\}$$

(For union) If ^{at least} any one state from both languages is final then we will make that state as final state
 For eg. $A \rightarrow \text{Final}$ $A \cup C \rightarrow \text{Non-final}$
 $P \rightarrow \text{Non-final}$ $P \rightarrow \text{Non-final}$
 $\therefore AP \rightarrow \text{Final}$ $CP \rightarrow \text{Non-final}$



→ For intersection . if $p \in F_1$ & $q \in F_2$ then
only (p, q) as final state

From book

→ Automata

Th^m3.4 Suppose $M_1 = (\Theta, \Sigma, q_1, A_1, \delta_1)$ &
 $M_2 = (\Theta_2, \Sigma, q_2, A_2, \delta_2)$ accept
 L_1 & L_2 respectively . Let M be an FA defined
by $M = (\Theta, \Sigma, q_0, A, \delta)$ where

$$\Theta = \Theta_1 \times \Theta_2$$

$q_0 = (q_1, q_2)$ & transition fun' δ is
defined by the formula

$$\delta((p, q), a) = \{(\delta_1(p, a), \delta_2(q, a))\}$$

$$\text{e.g. } \delta((A, P), O) = \{(\delta_1(A, O), \delta_2(P, O))\}$$

For union $A = \{ (p, q) \mid p \in A_1 \text{ or } q \in A_2 \} , L = L_1 \cup L_2$
Ex

Intersection $A = \{ (p, q) \mid p \in A_1 \text{ and } q \in A_2 \} , L = L_1 \cap L_2$

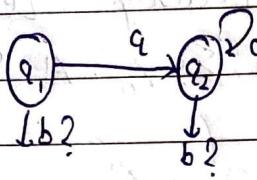
Complement

Difference $A = \{ (p, q) \mid p \in A_1 \text{ and } q \notin A_2 \} , L_1 - L_2$

* NFA

→ Unlike DFA, if any transition is not covered then also it will work for NFA

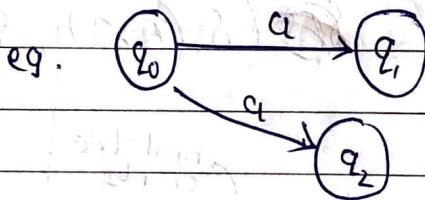
For eg. $\Sigma = \{a, b\}$



This is NFA
as it states q_1 & q_2
does not have transition
for b

~~Also~~

→ Also if for any one transition we have reached to more than 1 states then (also) its NFA.



→ A non deterministic finite Automata is a 5-tuple $M = (\Theta, \Sigma, q_0, A, \delta)$, Θ & Σ are non-empty finite sets

$$q_0 \in \Theta$$

$$A \subseteq \Theta$$

Transition fun" of NFA , $\delta: \Theta \times \Sigma \rightarrow 2^Q$
 " " " " DFA , $\delta: \Theta \times \Sigma \rightarrow Q$

As it might combine with all elements
of power set

⇒ Non-Recursive defⁿ of δ^* for an NFA

→ For an NFA, $M = (\Theta, \Sigma, q_0, A, \delta)$ & $p \in Q, \delta^*(p, \lambda) = \{q\}$

For NFA $\delta^*(q_0, x)$ → string
 DFA $\delta(q_0, a)$ → single character as input

→ For any $p \in Q$, & any $x = a_1 a_2 a_3 \dots a_n \in \Sigma^*$

$\delta^*(p, x)$ is the set of all states q for which there is a sequence of states $P = p_0 p_1 \dots p_{n-1}, p_n = q$ satisfying

$p_i \in \delta(p_{i-1}, a_i)$ for each i with $1 \leq i \leq n$

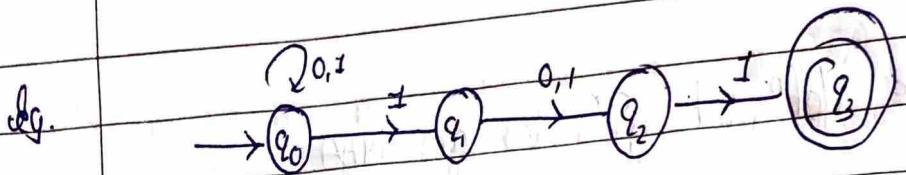
⇒ Recursive defⁿ of δ^* for an NFA

Let $M = (\Theta, \Sigma, q_0, A, \delta)$ be an NFA. The funⁿ $\delta^* : \Theta \times \Sigma^* \rightarrow 2^Q$ is defined as follows :

1) For any $q \in Q, \delta^*(q, \lambda) = \{q\}$

2) For any $q \in Q, y \in \Sigma^* \& a \in \Sigma$

Representation $\delta^*(q, ya) = \bigcup_{x \in \delta^*(q, y)} \delta(x, a)$



$$\delta^*(q_0, \text{aa}) = \bigcup_{\gamma \in \delta^*(q_0, 1)} \delta(\gamma, a)$$

$\frac{1}{x} \frac{1}{a}$
 ↓ last character a
 Remaining y

$$= \bigcup_{\{q_0, q_1\}} \delta(\gamma, 1)$$

$$= \delta(q_0, 1) \cup \delta(q_1, 1)$$

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

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

~~δ^*~~

$$\delta^*(q_0, 01) = \bigcup_{\gamma \in \delta^*(q_0, 0)} \delta(\gamma, 1)$$

$$= \bigcup_{\{q_0\}} \delta(\gamma, 1)$$

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

$$= \{q_0\}$$

$$\delta^*(q_0, 111) = \bigcup_{\gamma \in \delta^*(q_0, 11)} \delta(\gamma, 1)$$

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

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

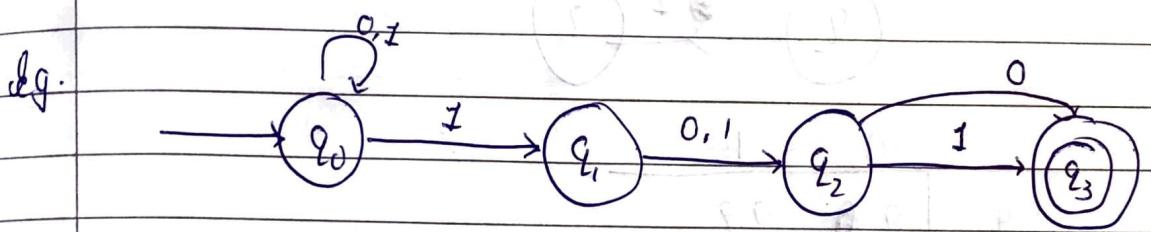
$$\delta^*(q_0, 011) = \bigcup_{\alpha \in \delta^+(q, 01)} \delta(\alpha, 1)$$

$$= \bigcup_{\{q_0, q_1\}} \delta(q, 1)$$

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

⇒ Conversion of NFA to DFA

By subset construction method



Initial state → q_0
final state → q_3

transition state

	0	1
q_0	q_0	$q_0 q_1$
q_1	q_2	q_2
q_2	q_3	q_3
q_3	-	-

	0	1
q_0	q_0	$q_0 q_1$
$q_0 q_1$	$q_0 q_1$	$q_0 q_1, q_2$
q_2	-	-

How $q_0 q_1$ came?

→ Check previous row, on input 0 we reached q_0 & on input 1 we reached $q_0 q_1$
→ So for input 1 we have q_1 as additional part which is not included in input 0.

→ So check for every input which is not covered previously

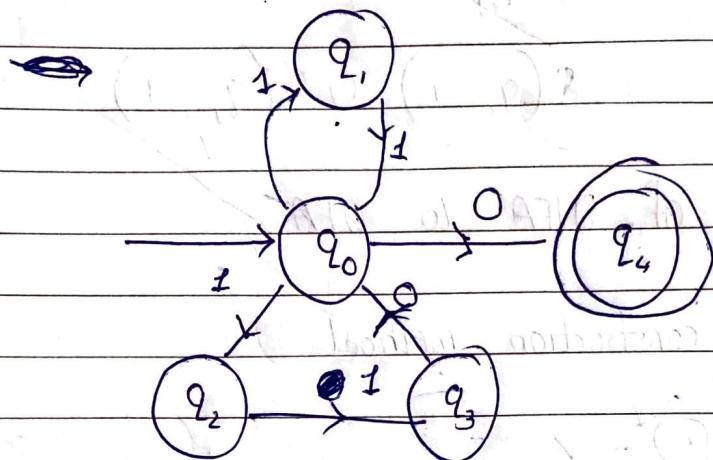
	0	1
q_0	q_0	$q_0 q_1$
$q_0 q_1$	$q_0 q_1$	$q_0 q_1, q_2$
$q_0 q_2$	$q_0 q_2$	$q_0 q_2, q_3$
$q_0 q_1 q_2$	$q_0 q_1 q_2$	$q_0 q_1 q_2, q_3$
$q_0 q_3$	$q_0 q_3$	$q_0 q_3$
$q_0 q_2 q_3$	$q_0 q_2 q_3$	$q_0 q_2 q_3$
$q_0 q_1 q_3$	$q_0 q_1 q_3$	$q_0 q_1 q_3$
$q_0 q_1 q_2 q_3$	$q_0 q_1 q_2 q_3$	$q_0 q_1 q_2 q_3$

Now all states are covered.

→ Now we can create ~~an~~ a DFA with 8 states & the final state can be determined by the state which contains q_3 .

Q

NFA



	0	1
q_0	q_4, q_3	q_1, q_2
q_1	-	q_0
q_2	q_3	-
q_3	q_0	-
q_4	-	-

NFA table

	0	1
q_0	q_4	q_1, q_2
q_4	-	-
q_1, q_2	q_3	q_0, q_3
q_0, q_3	q_0, q_4	q_1, q_2
q_0, q_4	q_4	q_1, q_2

DFA with 5 states

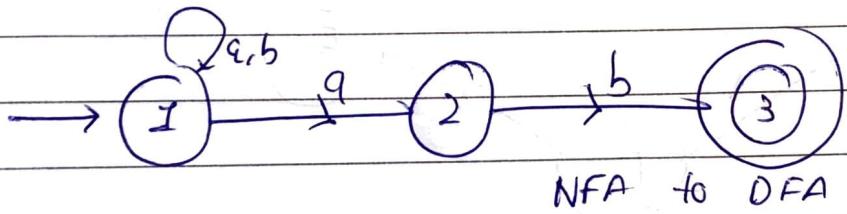
For '-' we can convert it into \emptyset & consider it as a new state

	0	1
q_0	q_4	q_1, q_2
q_4	\emptyset	\emptyset
q_1, q_2	\emptyset	q_0, q_3
q_0, q_3	q_0, q_4	q_1, q_2
q_0, q_4	q_4	q_1, q_2

∴ Total 6 states & as it contains q_4 , q_5 & q_6
so 2 states are final states

HW
Q

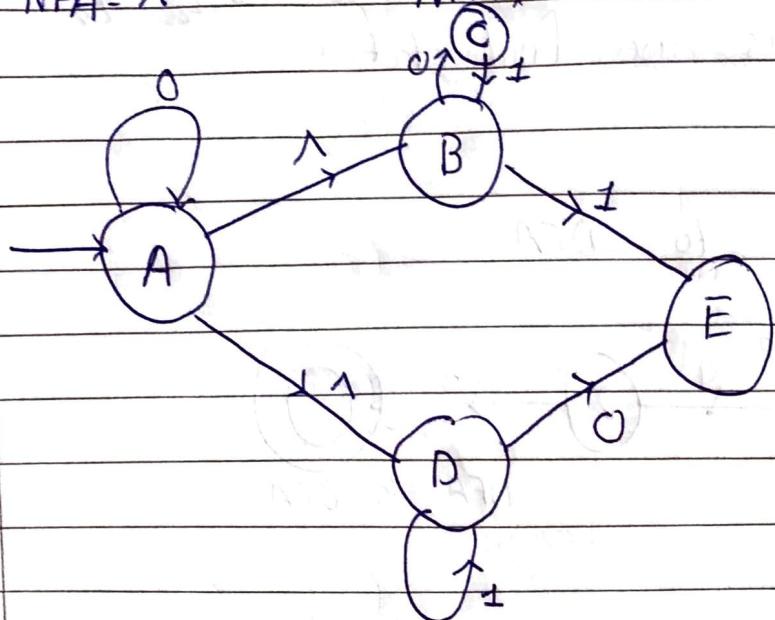
Convert NFA to DFA



NFA to DFA

$\Rightarrow \text{NFA-}\lambda \rightarrow \text{NFA}$, conversion

Sq.



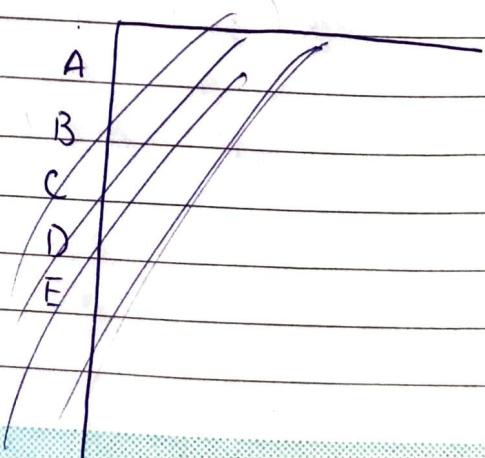
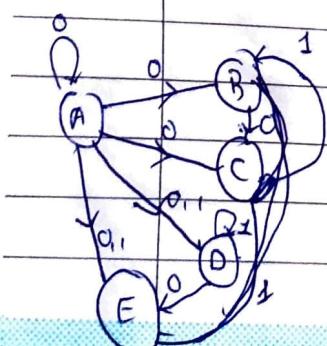
- Steps to do
- 1) Convert $\text{NFA-}\lambda \rightarrow \text{NFA}$
 - 2) $\text{NFA} \rightarrow \text{DFA}$.

$(\delta_0)^n$

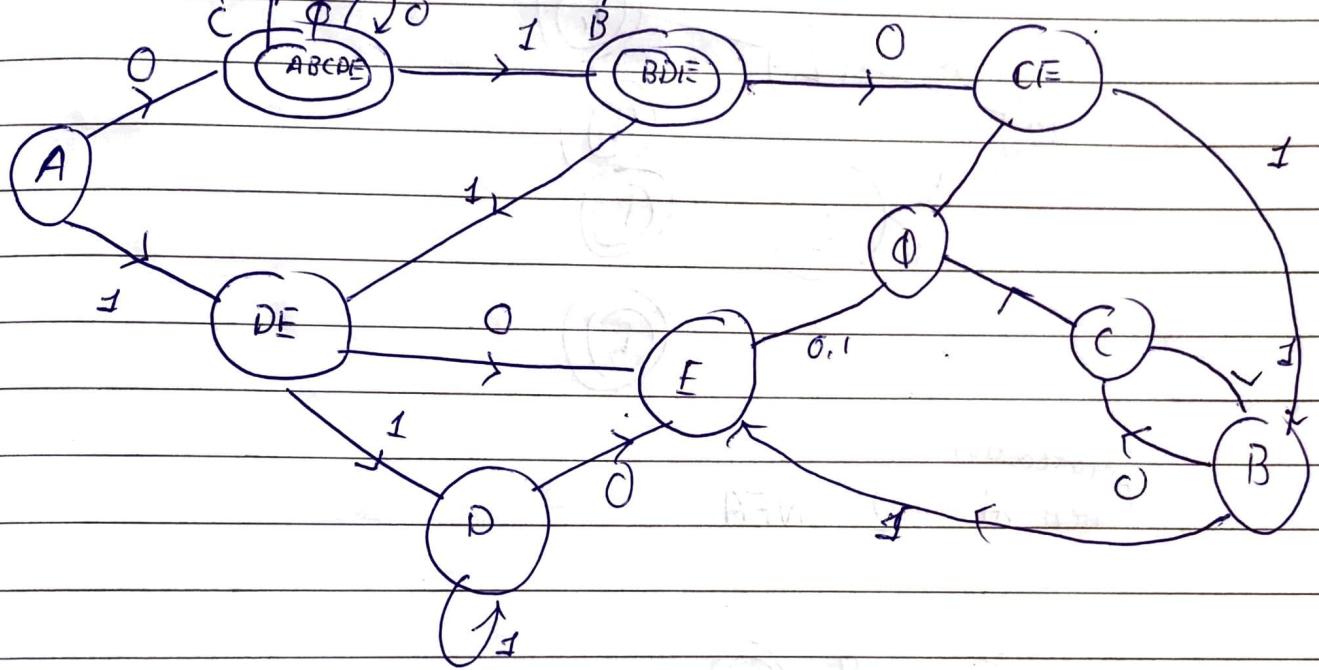
For calculation just
hide the null part.

		$\delta(q, 1)$	$\delta(q, 0)$	$\delta^*(q, 1)$	$\delta^*(q, 0)$	$\delta^*(q, 1)$
A	B	A	-	$\{A, B, C\}$	$\{E\}$	$\{D, E\}$
B	-	C	E	$\{C\}$	$\{E\}$	\emptyset
C	-	-	B	\emptyset	$\{B\}$	$\{B\}$
D	-	E	D	$\{E\}$	$\{D\}$	\emptyset
E	-	-	-	\emptyset	\emptyset	\emptyset

$\text{NFA} \rightarrow \text{DFA}$



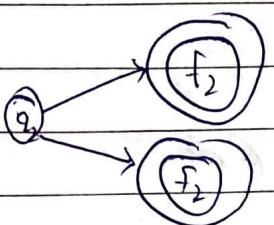
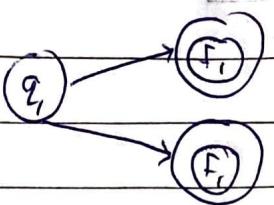
A	0	1
ABCDE	ABCFDE	DE
ABCDE	ABCDFE	BDFE
DE	E	D
BDFE	C E	DE
E	Φ	Φ
D	E	D
CE	Φ	B
B	C	E
Φ	Φ	Φ
C	Φ	Φ
ABCDE	Φ	Φ



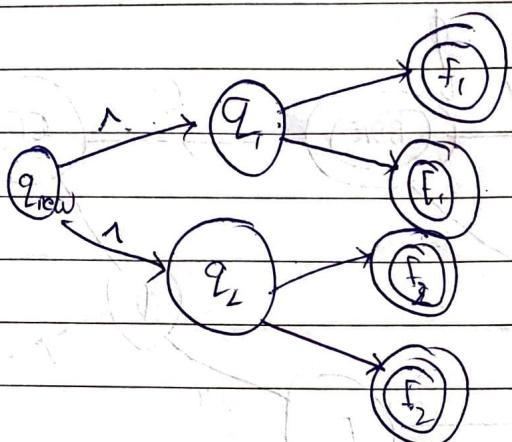
⇒ Kleen's theorem

→ Part 1 :- Any regular language can be accepted by a finite automata

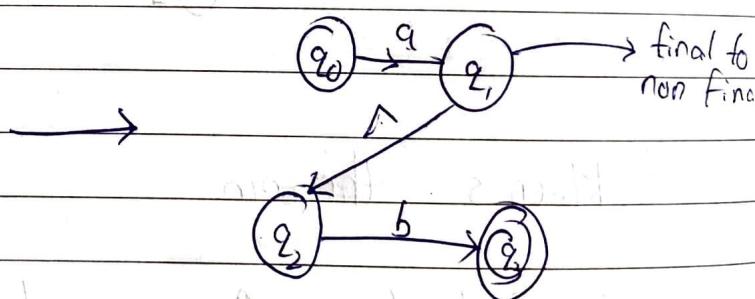
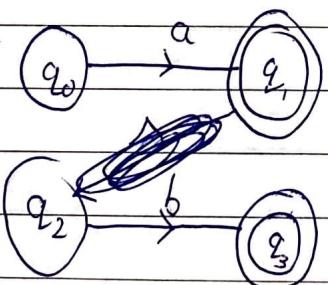
Part 2 :- The language accepted by any finite automata is regular.



To do union of 2 NFA
we will create new state
& will do null transition



Concatenation
Union of 2 NFA



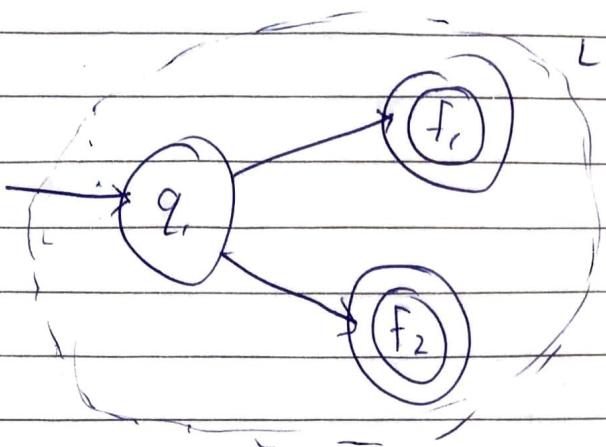
→ Kleen's closure (self loop)

$$L = \{a, aca, acaa, \dots, a^n\} = a^*$$

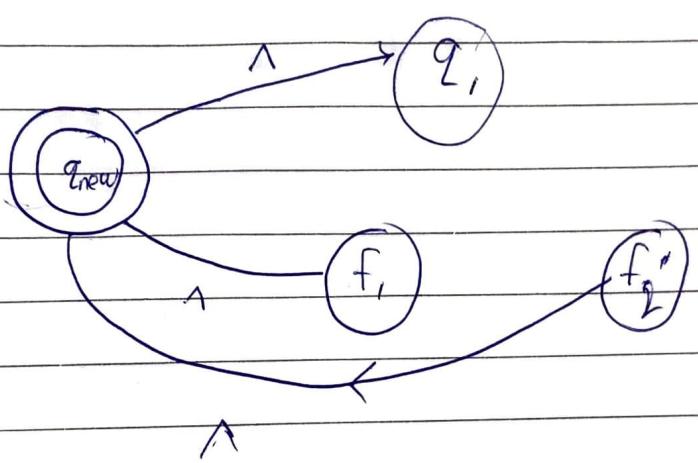
We will just need one NFA



Q) To Find Kleen's closure : L^*



Soln'



Q) $(010 + 1)^*$ Construct NFA

Concatenation Union

Soln'

First do ~~determine~~ FA of 010 & FA of 1
then do union & then do Kleen's closure