THOC

Congruence mod n

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

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

$a \equiv_4 b$

$\therefore a - b = n * k$

Just eg $4 = 4 * k$ $\quad k = 1$

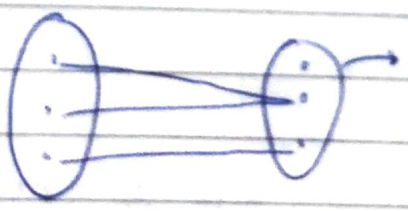
Eg. Find pairs of a,b if $n = 3$

Sol^n $a \equiv_3 b$

$$R = \{(a,3), (a,c), (1,4), (4,1), \ldots\}$$

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

$\Rightarrow$ Onto

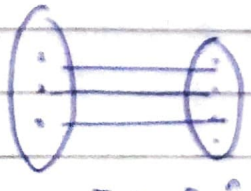


codomain ... set of all possible outputs

Not onto because ~~codom~~ range ≠ codomain

i.e. some elements of range are missing

$\Rightarrow$ Into



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

Into fun^

Date:

eg. $f: R \rightarrow R \qquad f(x) = x^2$

	Into	Onto
$f: R \rightarrow R^+$	No	yes
$f: R^+ \rightarrow R$	Yes	No
$f: R^+ \rightarrow R^+$	Yes	Yes

$a^k = aaa \ldots a \qquad a \in \Sigma \qquad L \subseteq \Sigma^*$
$x^k = xxx \ldots x \qquad x \in \Sigma^*$
$\Sigma^k = \Sigma\Sigma\Sigma \ldots = \{x \in \Sigma^+ \mid |x| = k\}$

$L^k = LLL \ldots L$

$a^0 = \Lambda \qquad \Sigma^0 = \{\Lambda\}$
$x^0 = \Lambda \qquad L^0 = \{\Lambda\}$

⇒ Concatenation

Apply in strings as well as languages (set of strings)

$x \qquad xy$
y

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

$L_1 L_2 = \{xy \mid x \in L_1 \ \& \ y \in L_2\}$

e.g. $L_1 \quad L_2$
$\{a, b\} \ \{c, d\} = \{ac, ad, bc, bd\}$

e.g. $L_1 = \{ab, bab\}^{*\ (0 \text{ to } \infty)} \cup \{b\}\{bb\}^{*\ (0 \text{ to } \infty)}$

$L_2 = \{ x \in \{a,b\}^* \mid n_a(x) \geq n_b(x) \}$ (number of a, number of b)

Soln

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

$L_1 = \{ab, bab, abbab, babab, ababab, \ldots\}$ (neglecting $\{b\}\{bb\}$)

$b, bbb, abbbb, \ldots$

neglecting $\{ab, bab\}^*$

mixing ab & bbb

* represents arbitrary time concatenate ab & bab

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

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

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

$L = \{ by \cup yb \mid y \in \{a,b\}^* \}$

⇒ Recursive Def^n

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

Recursive def^n of L^*

1) $\wedge \in L^*$
2) For any $x \in L^*$ & $y \in L^*$, $xy \in L^*$
3) No string other than (1) & (2) will come in L^*

Eg. Recursive def^n of Palindrome strings

Data
1) $\wedge \in L^*$
2) For any $x \in L^*$ & $y \in L^*$, $xy = yx$
3) No

Soln
1) $\wedge \in Pal$ {Null string}
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_1^{ij} where $i \geq 2j$

Q2 The set V of all strings of the form 0_1^{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.

$L_1 = \{a\}$, $L_2 = \{b\}$
$L_1L_2 = \{ab\}$ $L_1 \cup L_2 = \{a, b\}$ } These are regular languages

→ A regular expression is an algebraic repre^n of regular language.

Lang	R.E.	
$\{\wedge\}$	$\wedge$	$\{\} \rightarrow$ remove.
$\{0\}$	0	$\{\} \rightarrow ()$
$\{001\}$	001	$\cup \rightarrow +$
$\{0\} \cup \{1\} = \{0,1\}$	0+1	
$\{0,10\}$	0+10	
$\{1,\wedge\}\{001\}$	$(1+\wedge)001$	
$\rightarrow \{1\} \cup \{\wedge\}\{001\}$		
$\{110\}^*\{0,1\}$	$(110)^*(0+1)$	
$\{10, 111, 11010\}^*$	$(10+111+11010)^*$	
$\{0,10\}^*(\{11\}^* \cup \{001, \wedge\})$	$(0+10)^*((11)^* + 001 + \wedge)$	
$\{\emptyset\}$	$\emptyset$	
$L_1 \cup L_2$	$\gamma_1 + \gamma_2$	
$L_1 \cdot L_2$	$\gamma_1\gamma_2$	
L_1^*	γ_1^*	

Eg. $(0+1)^* 01 (0+1)^* + 1^*0^* = (0+1)^*$

Are they equal or not.. yes equal

Soln $(0+1)^* = \{\wedge, 0, 1, 00, 11, \ldots\}$

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

$L = \{00, 01, 10, 11, 0000, \ldots\} = \{00, 01, 10, 11\}^*$

$RE = (00 + 01 + 10 + 11 + 0000 + \ldots) = (00+01+10+11)^*$

Q Strings with an odd number of 1's

1. $(0^*10^*)(0^*10^*1)^*0^*$
2. $(0^*10^*1)^*(0^*10^*)$
3. $(0^*1\,0^*)(0^*10^*10^*)^*$

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

Soln

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

$L = \{0\}^*\{1, 111, 11111, \ldots\}$

Ans: $0^*10^*(10^*10^*)^*$

$1(00+11)^*$ ⟶ It does not contain all the language

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

$(1+01)^+$

Q R.E. for identifier of C language. $\Sigma = \{a, \ldots z, A, \ldots Z, 0, \ldots 9, \_\}$ (_ : underscore)

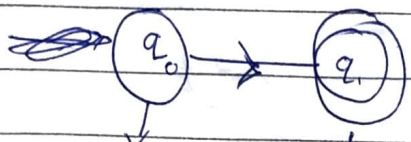
int temp ✓
int 2tempX
int _temp ✓
int temp_ ✓

$l = \{a, \ldots z, A, \ldots Z\}$

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

RE. = $(d + \_)(d + d + \_)^*$

FINITE AUTOMATA

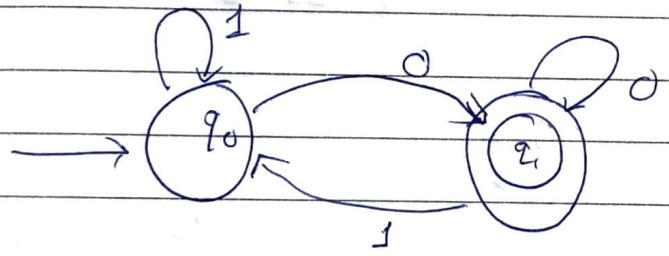


Initial state

Final State/ Accept state

input State

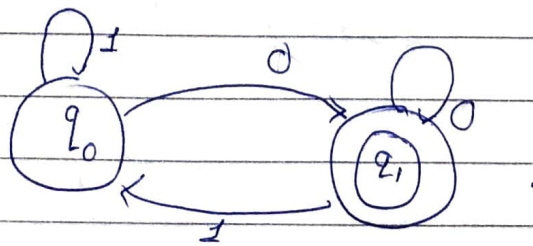
Eg.



initial State → q_0

If 01 is input the ● for 0 $q_0 \rightarrow q_1$
1 $q_1 \rightarrow q_0$

Eg.



1^+0^+

It must end with 0 to achieve q_1

0, 10, 010, 1000

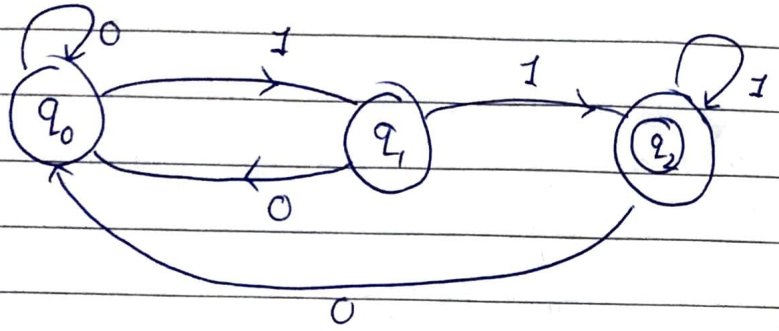
Ans. $(0+1)^*0$
OR
$(0^*1^*)^*0$

Exactly 2 0's ⟶ $1^*01^*01^*$

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

Eg. (q) This is final automata with final state = initial state & it will accept null string

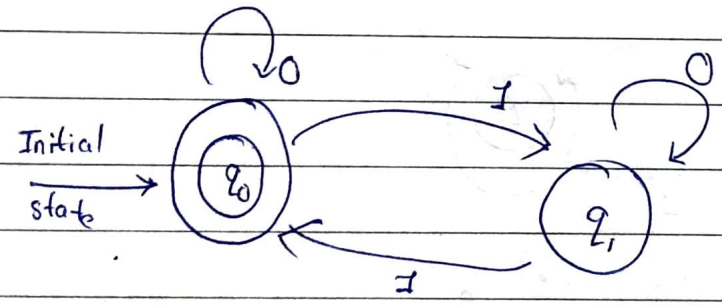
Eg.



~~Language~~ Strings should end ~~11~~ with 11

R.E. = $(0+1)^*.11.$

Eg.



{ ∧, 11, 0, 101, ... }

↓

even number of 1s

∴ R.E. = ~~$0^*10^*10^*$~~ ~~$(00+11)^*$~~ $(0^*(10^*1)^*0^*)^*$

OR

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

⇒ Def^n of finite Automata or FSM (Finite State machine)

It is a 5-tuple

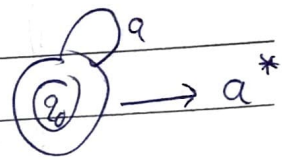
1) Q is a finite set $Q = \{q_0, q_1\}$

2) Σ is an alphabet of input symbols

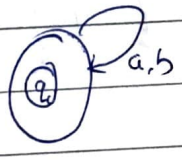
3) $q_0 \in Q$

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

5) Transition fun^n $\delta : Q \times \Sigma \longrightarrow Q$

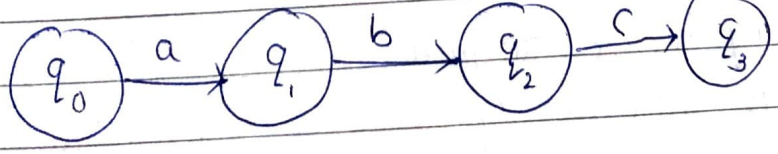
Eg. $L_1 = \{\wedge, a, aa, aaa, \ldots\}$ → Ans → 

Eg. $L_2 = \{\wedge, a, b, ab, bb, aa, \ldots\}$ 

sol^n 

Extended transition fun^n

$\delta^* : Q \times \Sigma^* \longrightarrow Q$

Eg. 

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

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

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

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

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

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

Q Let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA. We define the fun^n

$$\delta^* : Q \times \Sigma^* \longrightarrow Q$$

as follows

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

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

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

⟹ Acceptance of an FA

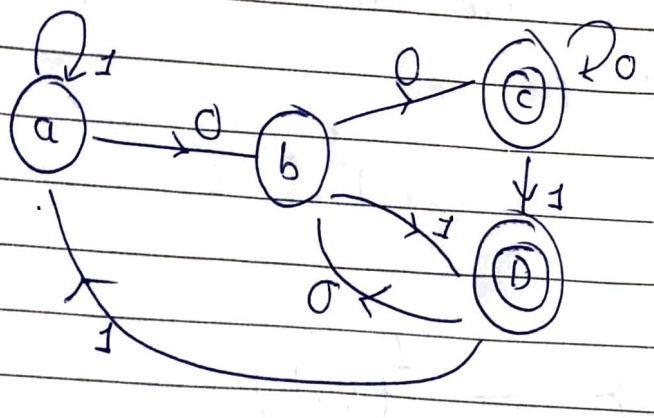
Let $M = (Q, \Sigma, q_0, A, \delta)$ be an F.A. A string $x \in \Sigma^*$ is accepted by M if $\delta^*(q_0, x) \in A$

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

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

if L is any language over Σ, L is accepted or recognized by M ~~iff~~ iff $L = L(M)$

Eg

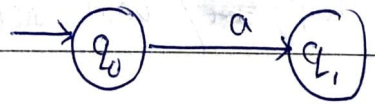


Eg FA, binary strings divisible by 3

⇒ Types of automata

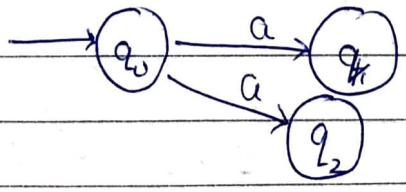
1) DFA (Deterministic)
2) NFA (Non-deterministic)

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

Eg. 

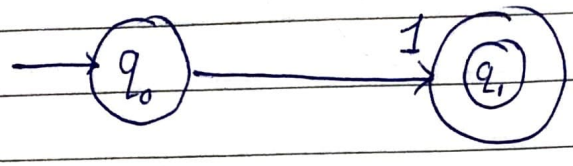
For every input we need to have ~~state~~ transition to one ~~state~~.

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

Eg. 

Q Construct a DFA which accept all strings starts with 1 $\Sigma = \{0, 1\}$

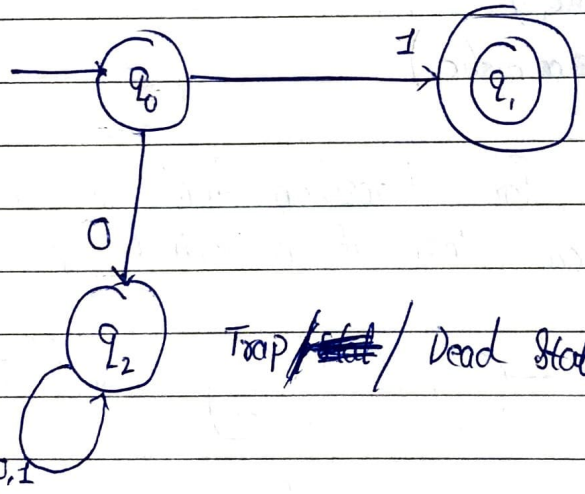
Soln $R.E = 1 \cdot (0+1)^*$, $L = \{1, 11, 10, 1011, \ldots\}$



→ If we make a self loop for input 0 in q_0 then it will accept string 01 too. So NP.

→ If we make $q_0 \xrightarrow{0} q_1$ then also for input 0 it will reach final state. So NP.

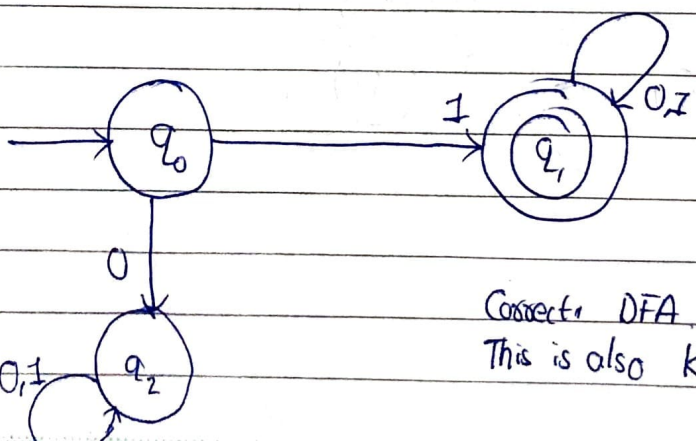
→ So make another state



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 to q_1 self loop



Correct DFA

This is also known as transition diagram.

→ Transition State

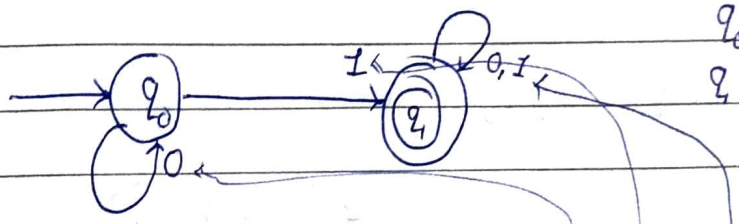
State	Input 0	Input 1
q_0	q_2	q_1
q_1	q_1	q_1
q_2	q_2	q_2

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

$L = \{1, 01, 100, \ldots\}$

R.E. = $(0+1)^* \cdot 1^+$ or $(0+1)^* \; 1 \; (0+1)^*$

DFA

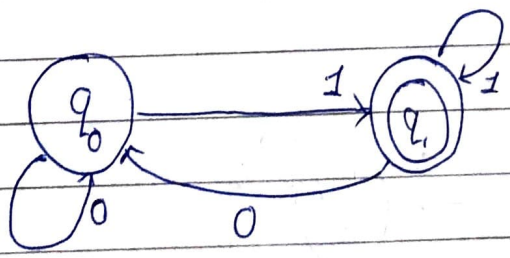


	0	1
q_0	q_0	q_1
q_1	q_1	q_1

By this diagram we can make R.E. = $0^* \; 1 \; (0+1)^*$

Q Strings end with 1

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



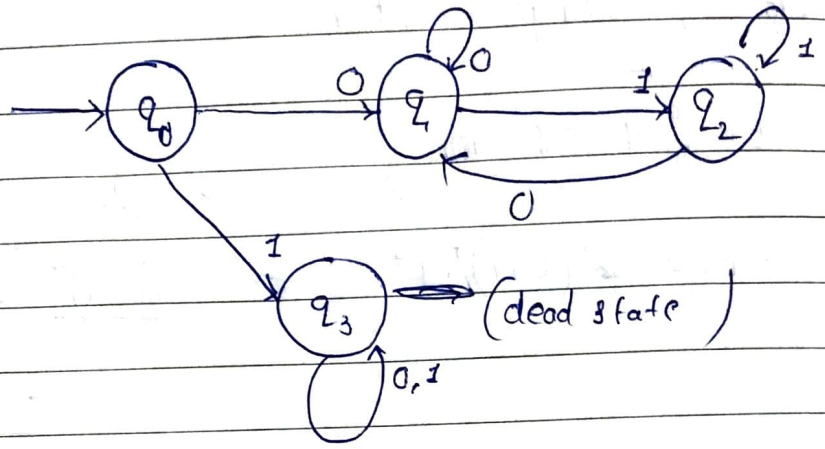
	0	1
q_0	q_0	q_1
q_1	q_0	q_1

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

Q DFA strings start with 0 end with 1

01, 001, 0111

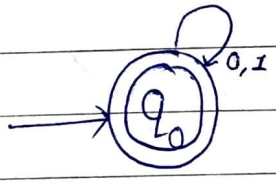
~~DFA~~ RE = $0(0+1)^* 1$



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

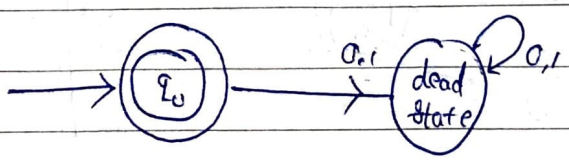
Soln $\Sigma^* = \{0, 1\}^*$

$L = \{\wedge, 0, 1, 00, 01, \ldots\}$

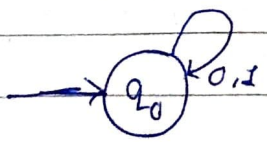


Q DFA $\Sigma = \{0, 1\}$, $L = \{\wedge\}$

Soln



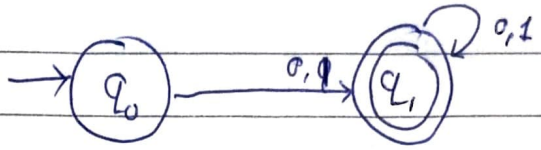
Q DFA, $L = \{\}$, $\Sigma = \{0, 1\}$



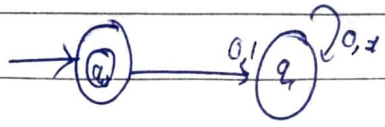
No final state as we don't want to accept anything.

Page No.: YOUVA
Date:

Q DFA $\Sigma = \{0, 1\}$, $\Sigma^* - \{\wedge\}$ (The ~~DFA~~ RE which accepts all string except null)



Another approach

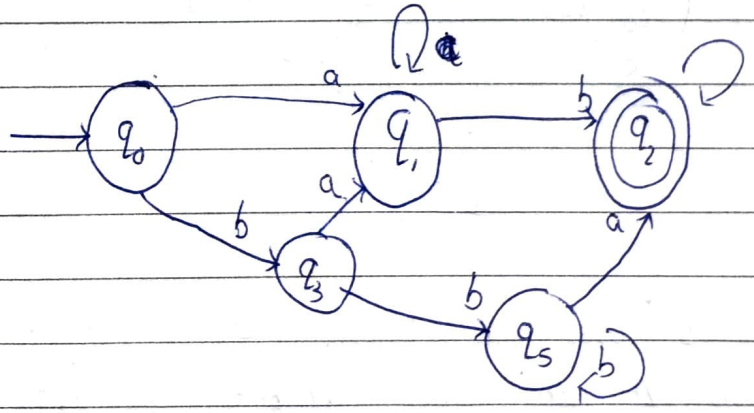


do for null $L = \{\wedge\}$
then do conversion
~~interchange final & no~~
Final state → Non-final
Non final → final.

Q DFA contains either strings ab or bba

$L = \{ab, bba, aaab, bbaab, \overset{abbba}{\ldots}\}$

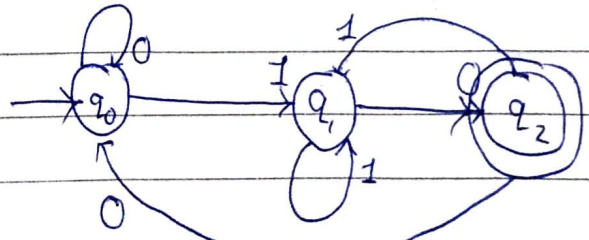
~~RE~~ $(a+b)^* (ab + bba)^+ (a+b)^*$



Made initially with ab & bba & then did connect all the states.

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

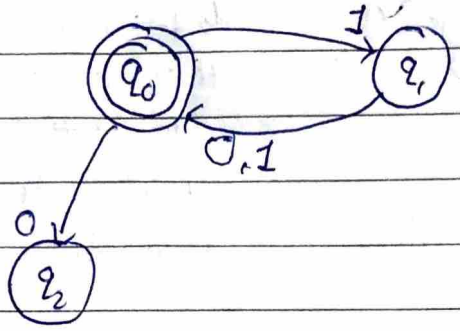
$L = \{10, 010, 1110, \ldots\}$ Test case: 01010101110



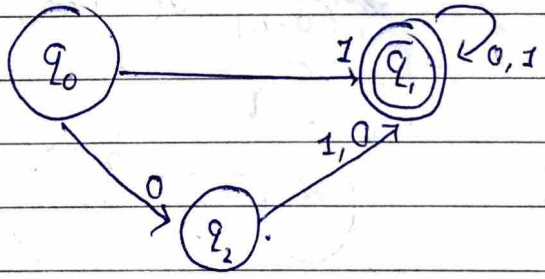
Q DFA

1) $RE = (11+10)^*$
2) $RE = (0+1)^*(1+00)(0+1)^*$

Soln 1) $L = \{\wedge, 11, 10, 1110, 1010, 111111, \ldots\}$



2) $L = \{1, 00, 110, 0000001, \ldots\}$



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

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

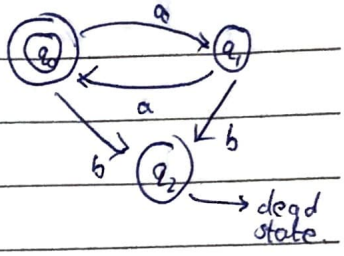
→ If DFA/NFA or RE can be constructed then the language can be known as R.L.

→ Finite Automata does not have any memory

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

$L = \{aa, aaaa, aaaaaa, \ldots\}$

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



Eg. $a^n b^n$, Regular language or not.

⇒ Distinguishing one string from another.

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

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

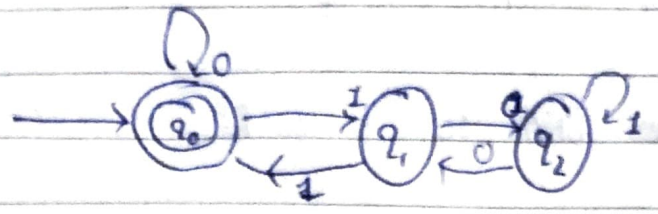
→ Two strings x & y are said to be distinguishable w.r.t. L if $L/x \neq L/y$. Any string Z that is in one of the two sets but not the other is distinguish x & y wrt. L. if $L/x = L/y$, x & y are indistinguishable w.r.t. L.

Eg. DFA whose number is divisible by 3

0000 ⟶ accepted
0011 ⟶ accepted (∵ Decimal value 3)
0110 ⟶ "

Sol^n 3 values of remainder = 0, 1, 2 , L =

→ Construct states = Total number of values of remainders



0 1 1 0
1 1 0
0 1 0 0

0 1 0 1

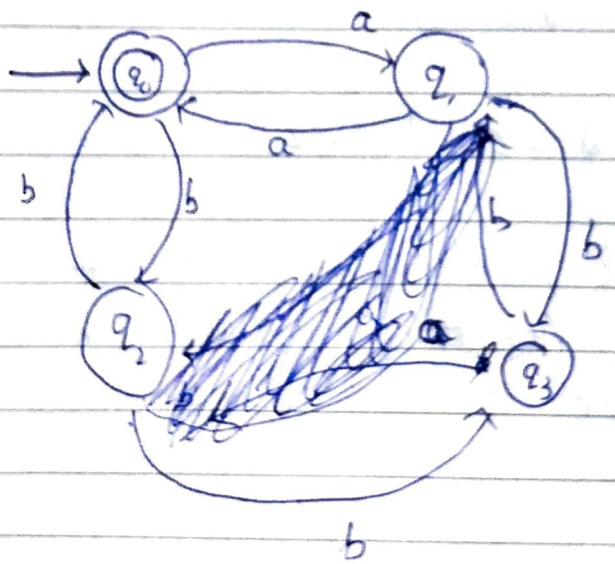
Just see values & put it in state as per values.

Q ~~DF~~ DFA , no. of a's & no. of b's are even

$\Sigma = (a, b)$

$L = \{ \wedge, aa, bb, abab, bababa, \ldots \}$

Sol^n



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

Date: YOUVA

⇒ 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.

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

$L_1 \cup L_2 = \{0, 1, 00, 11, 001, 01111, \dots\} = (0^+ + 1^+)$

2) RL are closed under intersection

3) RL are closed under concatenation.

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

$L_1^* = L_1 \cdot L_1 \cdot L_2 \cdot \dots$ (As RL is closed under concatenation it is automatically closed under kleen's closure)

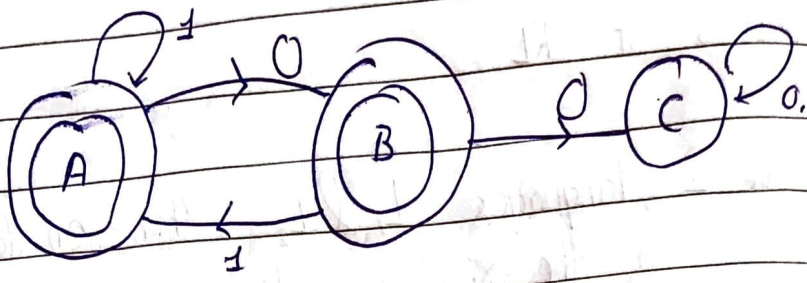
Eg. $L_1 = \{x \mid 00$ is not a substring of $x\}$ (First we will create DFA of 00 as a substring & then will interchange final & non-final states)

$L_2 = \{x \mid x$ ends with $01\}$

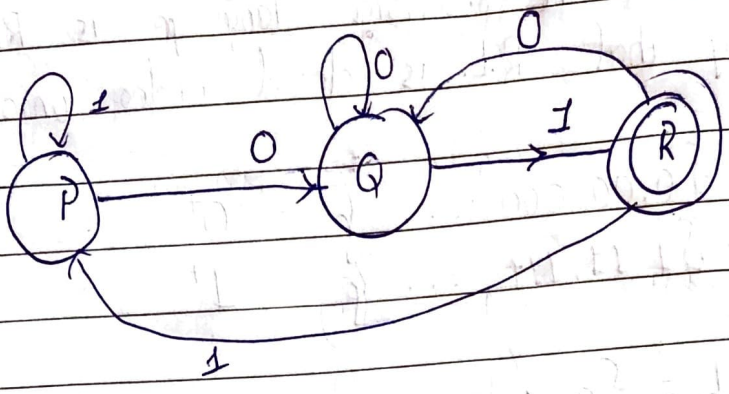
We will take union of these 2 languages

Soln $L_1 = \{01, 10, 11, 0, 1, \dots\}$
$L_2 = \{10, 00101, 101, 11101, 001, \dots\}$

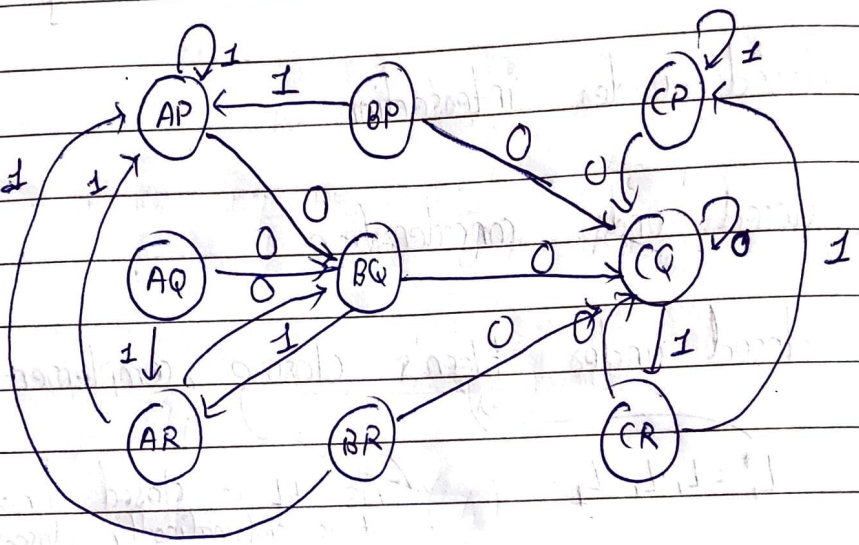
Interchanging stages of initial & final in ∞ substring



L_2



$L_1 \cup L_2$



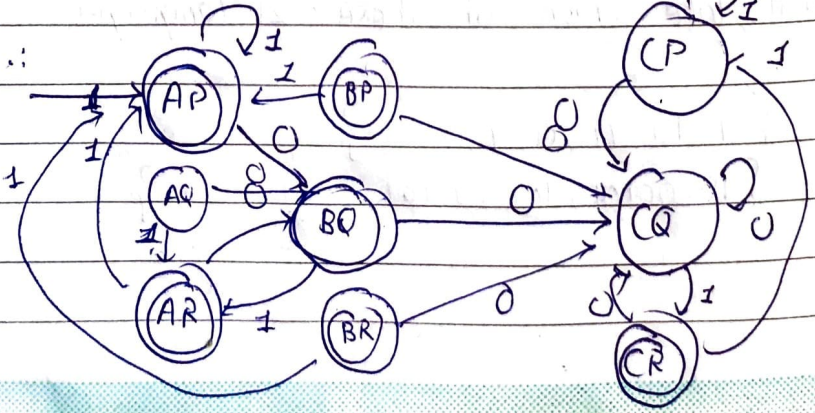
$L_1 \cup L_2 = \{0, 1, 01, 10, \ldots\}$

(For union) If any one state from both languages is final then we will make that state as final state

at least

For eg. A → Final
P → Non-final
∴ AP → Final

A C → Non-final
P → Non-final
CP → Non-final



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

Th^m 3.4 (→ from book) Suppose (→ Automata) $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ & $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ accept L_1 & L_2 respectively. Let M be an FA defined by $M = (Q, \Sigma, q_0, A, \delta)$ where

$$Q = Q_1 \times Q_2$$

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

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

Eg. $\delta((A, P), 0) = (\delta_1(A, 0), \delta_2(P, 0))$

For union $A = \{(p, q) \mid p \in A_1$ (→ Final state of RL_1) or $q \in A_2$ (→ Final state of RL_2)$\}$, $L_1 \cup L_2$

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

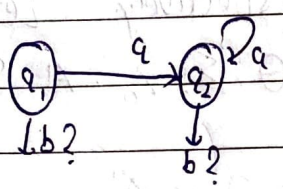
Complement

Difference $A = \{(p, q) \mid p \in A_1$ 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. $\Sigma = \{a, b\}$

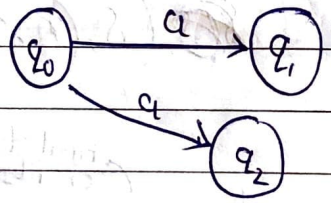


This is NFA as states q_1 & q_2 does not have transition for b

~~Also~~

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

eg.



→ 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^n of NFA, $\delta : \Theta \times \Sigma \longrightarrow 2^Q$

" " " DFA, $\delta : \Theta \times \Sigma \longrightarrow Q$

As it might combine with all elem of power set

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

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

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

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

$\delta^*(P, x)$ is the set of all states q for which there is a sequence of states $P = P_0 P_1 \ldots P_{n-1}$, $P_n = q$ 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

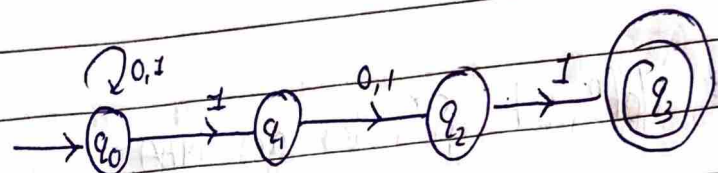
Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. The funⁿ $\delta^* : Q \times \Sigma^* \longrightarrow 2^Q$ is defined as follows:

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

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

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

Eg.



$$\delta^*(q_0, 11) = \bigcup_{r \in \delta^*(q_0, 1)} \delta(r, 1) \quad \begin{matrix} \to a \\ \to y \end{matrix}$$

1 1
y a
→ last character a
→ Remaining y

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

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

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

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

~~δ^*~~

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

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

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

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

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

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

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

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

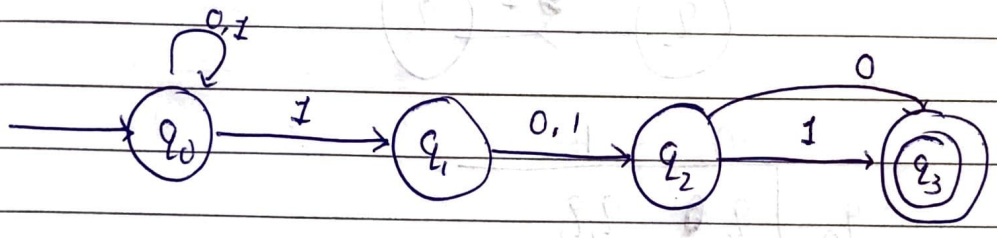
$$= \bigcup_{\{q_0, q_2\}} \delta(\sigma, 1)$$

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

⇒ Conversion of NFA to DFA

By subset construction method

Eg.



Initial state → q_0
final state → q_3

transition table

	0	1
q_0	q_0	q_0q_1
q_1	q_2	q_2
q_2	q_3	q_3
q_3	-	-

	0	1
q_0	q_0	q_0q_1
q_0q_1	q_0q_2	$q_0q_1q_2$

How q_0q_1 came ?
→ Check previous row, on input 0 we reached q_0 & q_1 on input 1 we reached q_0q_1
→ So for input 1 we have q_1 as addition part which is not included in input 0.

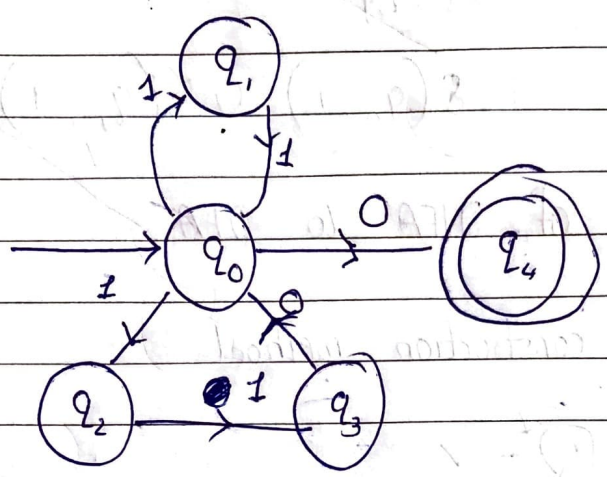
→ So check for every input which is not covered previously

	0	1
q_0	q_0	q_0q_1
q_0q_1	q_0q_2	$q_0q_1q_2$
q_0q_2	q_0q_3	$q_0q_1q_3$
$q_0q_1q_2$	$q_0q_2q_3$	$q_0q_1q_2q_3$
q_0q_3	q_0	q_0q_1
$q_0q_1q_3$	q_0q_2	$q_0q_1q_2q_3$
$q_0q_2q_3$	q_0q_3	$q_0q_1q_3$
$q_0q_1q_2q_3$	$q_0q_2q_3$	$q_0q_1q_2q_3$

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	$q_1 q_2$
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	–	–
$q_1 q_2$	–	$q_0 q_3$
$q_0 q_3$	$q_0 q_4$	$q_1 q_2$
$q_0 q_4$	q_4	$q_1 q_2$

~~DFA with 5 states~~

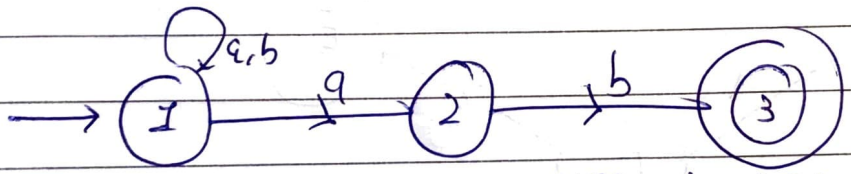
For '–' we can convert it into ϕ & consider it as a new state

	0	1
q_0	q_4	$q_1 q_2$
q_4	ϕ	ϕ
$q_1 q_2$	ϕ	$q_0 q_3$
$q_0 q_3$	$q_0 q_4$	$q_1 q_2$
$q_0 q_4$	q_4	$q_1 q_2$
ϕ	ϕ	ϕ

∴ Total 6 states ℓ as it contains q_4, q_4 ℓ $q_0 q_4$ so 2 states are final states

HW Q

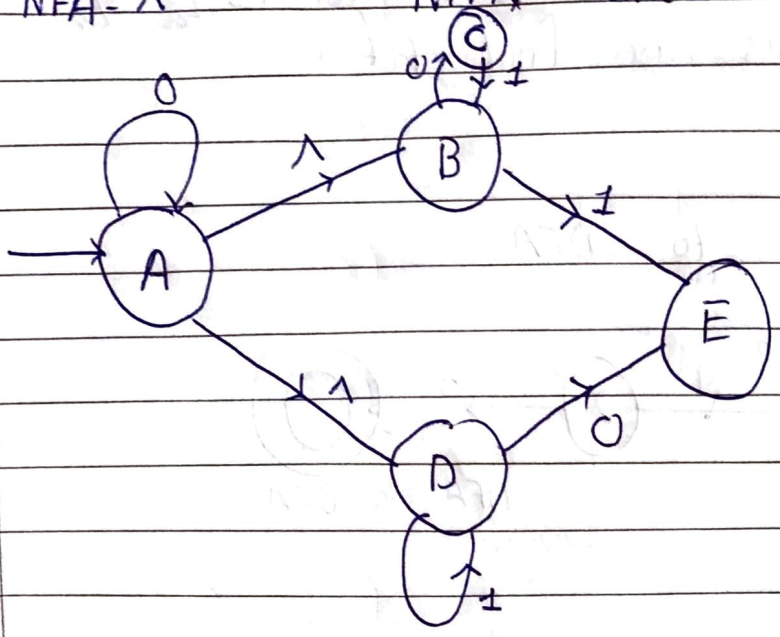
Convert NFA to DFA



NFA to DFA

⇒ NFA-Λ ⟶ NFA, conversion

Eg.



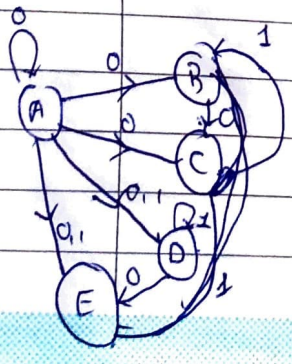
Steps to do 1) Convert NFA-Λ ⟶ NFA
2) NFA ⟶ DFA.

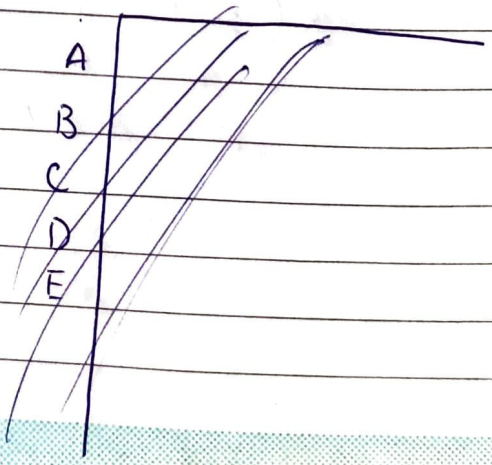
Soln

↗ For calculation just hide the null past.

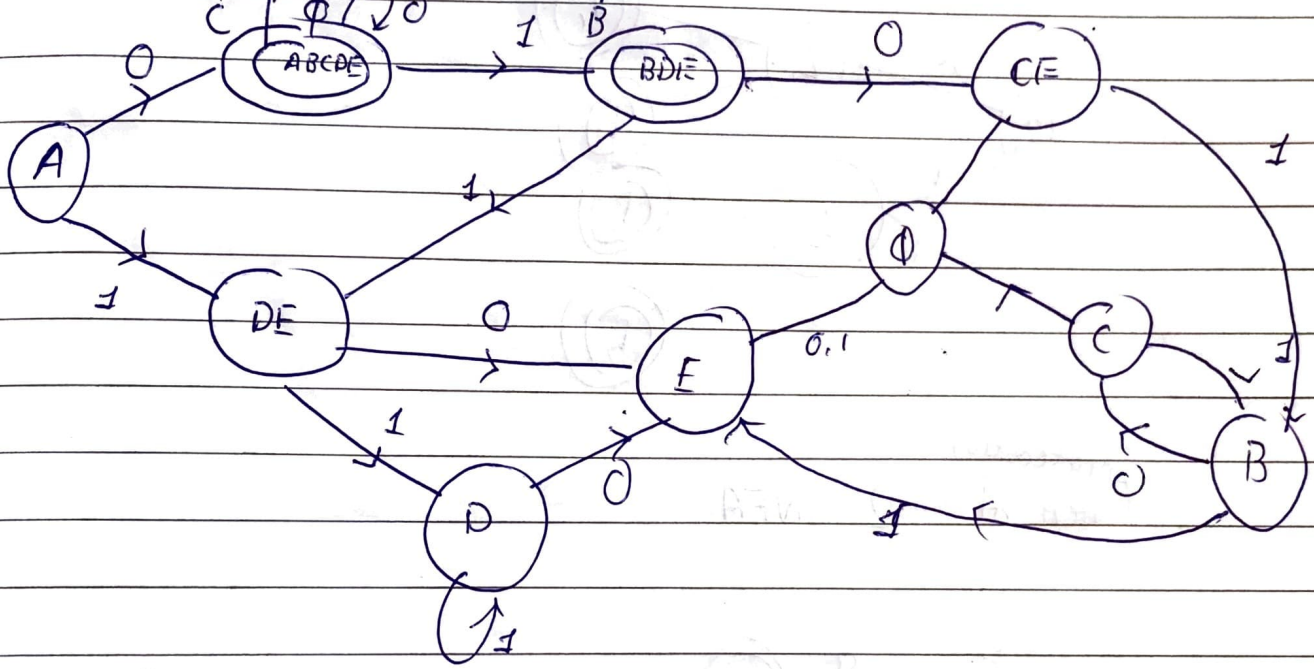
	$\delta(q,1)$	$\delta(q,0)$	$\delta(q,1)$	$\delta^*(q,0)$	$\delta^*(q,1)$
A	B	A	-	{A, B, C, D, E}	{D, E}
B	-	C	E	{C}	{E}
C	-	-	B	ϕ	{B}
D	-	E	D	{E}	{D}
E	-	-	-	ϕ	ϕ

NFA ⟶ DFA





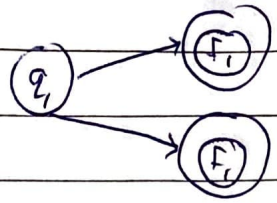
	0	1
A	ABCDE	DE
ABCDE	ABCDE	BDE
DE	E	D
BDE	CE	DE
E	ϕ	ϕ
D	E	D
CE	ϕ	B
B	C	E
ϕ	ϕ	ϕ
C	ϕ	B

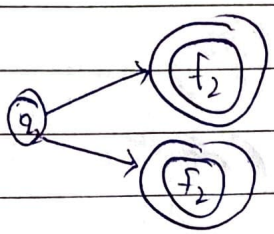


⇒ Kleen's theorem

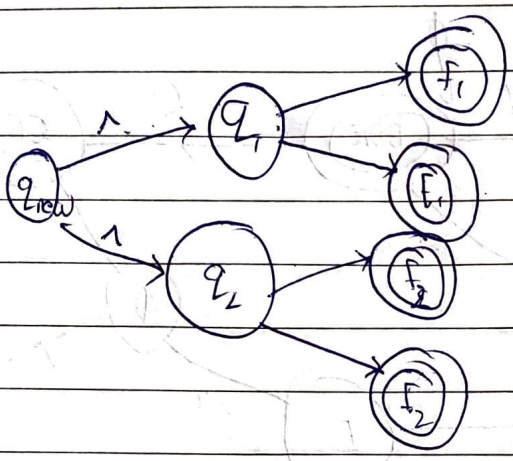
→ ~~An~~ Part 1 :- Any regular language can be accepted by a finite automata

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

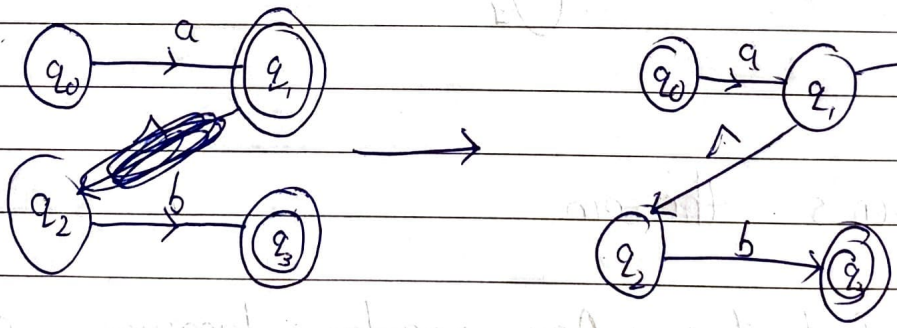




To do union of 2 NFA we will create new state & will do null transition



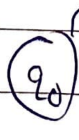
Concatenation ~~Union~~ of 2 NFA



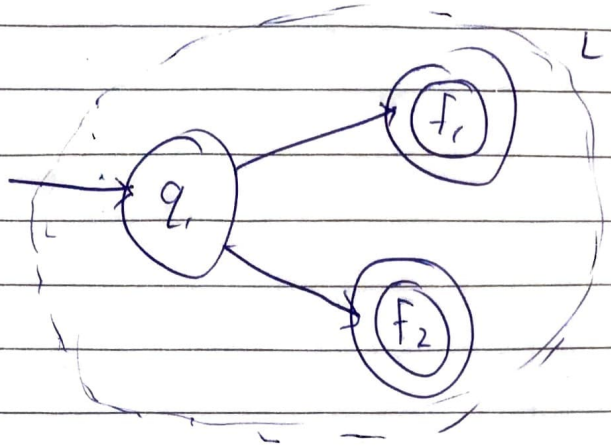
⟹ Kleen's closure (self loop)

$L = \{a, aa, aaa, aaaa, \ldots\} = a^*$

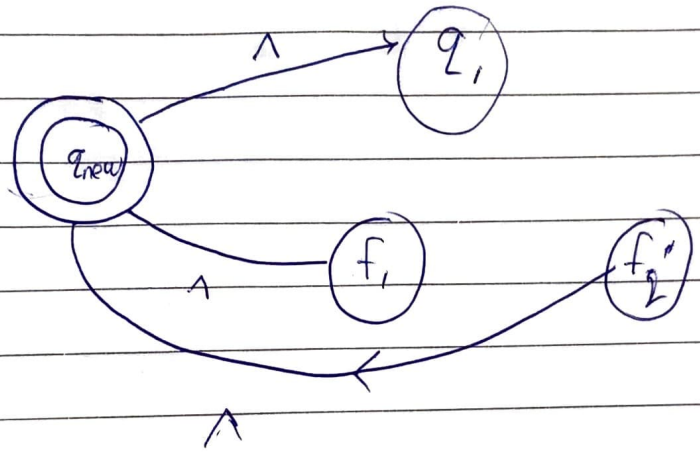
We will just need one NFA



Q ~~He~~ Find Kleen's closure : L^*



Soln



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

Concatenation Union

Soln First do ~~automata~~ FA of 010 & FA of 1 then do union & then do kleen's closure