8|9|23

Theory of computation :

- It is a branch of computer science that deals with how efficiently problem can be solved on a model of computation using an algorithm i.e. it is ~~connected~~ concerned with how problems can be efficiently solved by using algorithms.

- The main purpose of theory of computation is to develop a formal mathematical model of computation that reflects the real world computer.

This course covers : 1. complexity theory
2. computability theory
3. Automata theory

Automata theory :

derived from the word automaton → related to automation

- Automata theory is the study of abstract computational devices.
- It mainly deals with the logic of computation w r t simple message referred to as automata.
- Automata enables scientists to understand how machines compute the function & solve problems.

→ Used in compiler design, hardware, AI

Terminologies :

- Tuple → NO. of elements in a sequence
- Set vs Sequence

Set	Sequence
↓	↓
order doesn't matter	order matters

9/9/23

- Alphabets - Finite set of symbols
  Denoted by capital greek letter Σ

  Eg → $\Sigma = \{a, b\}$
  $\Sigma = \{a, b, c, d, \cdots z\}$
  $\Sigma = \{\#, 2, \Delta\}$

- Symbols (set) - Entities / individual objects which can be any character, special character, number, or picture.
  (Γ)
  ↑ Typewriter
  Eg: $\Gamma = \{0, 1, a, b\}$

- Strings (w) - A string over an alphabet is a finite collection of symbols from the alphabet

  Eg: Let $\Sigma = \{0, 1\}$
  A string over Σ can be 001110
  $w = 001110$

  → length of string $w = |w| = 6$

  → w^R = reverse string

  $w^R = 011100$

- Language - A language "L" over alphabet Σ is any set of strings made up of symbols from Σ.
  - A language which is formed over Σ is finite or infinite

  Eg: Let $\Sigma = \{a, b\}$

  L_1 = {set of all strings with length 2}
  = {aa, ab, ba, bb} → finite

  L_2 = {set of all strings starts from a}
  = {a, aa, ab, aab, aba, ...} → infinite

- ϵ → empty string
  ϕ → empty language

- The set of all strings over Σ is denoted by Σ^*
  If you define any language, it will be a subset of Σ^*
  $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$

- union
- Intersection
- concatenation
- Closure / Kleene star (L^*) Of language : The closure / Kleene star of a language L is the language L^* obtained by ~~it sets~~ uniting L^K for multiple times. Where L^K is the language obtained by concatenating L to itself K times.

{It is the set of all possible string of all possible lengths including the empty string ϵ }

↑ Kleen star for string Σ^*

$L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cdots$

where L^K = concatenation of L & L^{K-1}

$= LLL \cdots L$ ← K times →

empty string = ω
" language = ϕ

→ $\overline{L} = L^* - L$

14/9/23 • $\Sigma^* = \underbrace{\Sigma^0}_{\text{set of all strings of length zero}} \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cdots$

Eg: $\Sigma = \{0, 1\}$
Then $\Sigma^* = \{\epsilon, 0, 1, 00, 01, \cdots\}$
$\Sigma^+ = \{0, 1, \cdots\}$
Positive closure / Kleen star : (Σ^+)

$\Sigma^+ = \Sigma^* - \{\epsilon\}$

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$

Boolean Logic

$x \to y$
$= \neg x \vee y$

AND	$\wedge$
OR	$\vee$
NOT	$\neg$
XOR	$\oplus$
EQUALITY	$\longleftrightarrow$
IMPLICATION	$\longrightarrow$

Types of Proof

(1) Proof by Induction

(2) " " Contradiction

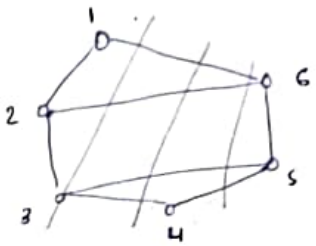
(3) " " Construction : ~~Our goal is to create / find a formula for creati~~

Theorem : For each even no. $n > 2$ there exists a 3 regular graph with n nodes

(Proof by Construction)

** Our goal is to create / find a formula for creating a 3-regular graph for any $n \geq 4$ where n is even

Proof : $n \geq 4$, n is even



Eg 1, let $n = 6$, $\frac{n}{2} = 3$

$V = \{1, 2, 3, 4, 5, 6\}$

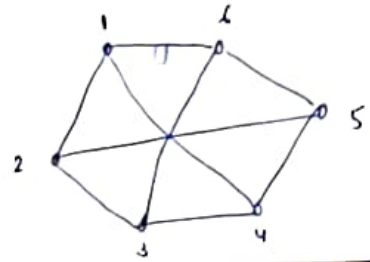
$G(V, E)$

$E = \{\{i, i+1\} ; 1 \leq i \leq n-1\} \cup \{(n, 1)\}$ → connecting adjacent vertices

connecting last edge for cycle

$\cup \{(i, i + \frac{n}{2}) : 1 \leq i \leq \frac{n}{2}\}$

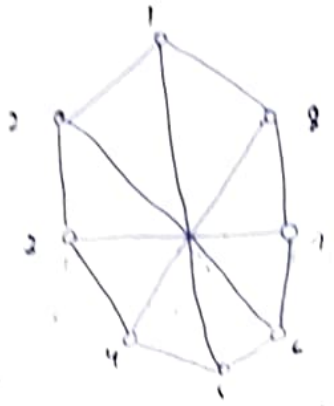
↓ connection opp vertices



15/9/28

Eg 2 $n = 8$, $V = \{1, 2, \cdots 8\}$

$E = \{(1,2), (2,3) \cdots (8,1)(1,5)(2,6)(3,7)(4,8)\}$

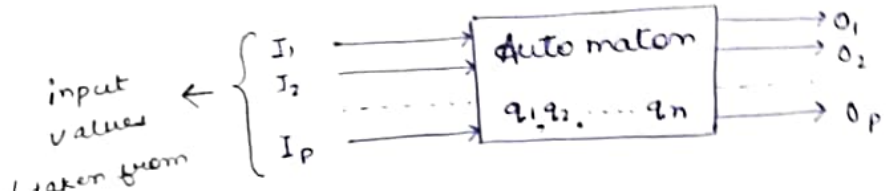


Introduction to Finite Automata (FA)

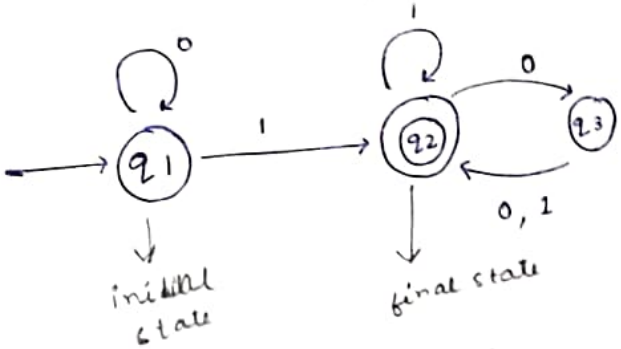
Finite Automaton / Finite State Machine.

- An automaton is an abstract computing device / machine.
  An automaton is defined as mathematical model of a system with discrete inputs & outputs. ~~The system~~
- The system can be in a finite no. of internal states.

A model of discrete Automaton



Def^n of FA : Finite Automaton consists of a finite set of states & set of transitions from state to state that occur on input symbols chosen from an alphabet Σ. The following fig depicts a finite automaton ~~is~~ M_1



- initial state : an arrow from nowhere
- final state : double circle

Eg. of automaton machine
(State diagram / State Transition diagram)

Eg-
suppose for $\Sigma = \{0, 1\}$
the string $w = 1101$

→ reads left to right

FOR $w = 1101$

$q_1 \to q_2 \to q_2 \to q_3 \to q_2$

starting from q_1 & ending at q_2 so this this string is accepted

$w_1 = 0101$, $w_2 = 01$, $w_3 = 1$, $w_4 = 1100$, $w_5 = 0100$

$w_6 = 011$, $w_7 = 01100$

All these are accepted strings

Pattern : All strings that end with 1

+

" " having even no. of 0(s) after last 1.

Q① Design a finite automaton which accepts all the binary strings ending at 1.

② Design a finite automation that accepts all binary strings having even no. of 0's after the last 1.

A file automation M is defined as a 5 tuple

$M = (Q, \Sigma, \delta, q_0, F)$

Where Q = finite set of states.

Σ = (symbols) finite set of inputs called alphabet.

δ = delta is a mapping fn / transition fn which maps $\delta : Q \times \Sigma$ into Q

$q_0 \in Q$ is the initial state

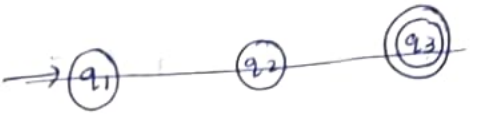
$F \subseteq Q$: set of final states

$M_1 = (\{q_1, q_2, q_3\}, \delta, q_1, \{q_2\})$

(2) $\delta : Q \times \Sigma$ into Q , $Q = \{q_1, q_2, q_3\}$

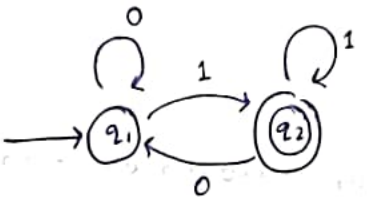
$\Sigma = \{0, 1\}$

Input / Output	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2



16|9|23

Eg. Design a ~~finite~~ FA that accepts all strings that ends at a 1 only.



Input strings : 1
0 1
1 1
0 0 1
0 1 1
1 0 1
1 1 1
⋮

L = set of all possible strings of all possible lengths ending at 1.

The Automaton recognizes the language L & accepts the string.

Regular language

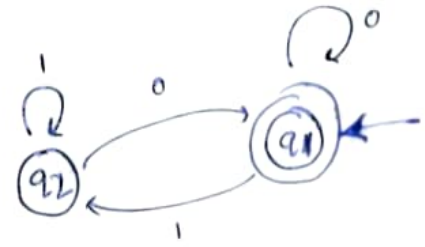
- A language is called a regular language if some finite automaton recognizes it.
- Let for an automaton M_1, which accepts all strings of language $A = \{w \mid w$ ends at a $1\}$ then $L(M_1) = A$.

For the above eg :

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

Q (2) Design an automaton that accepts an empty string ε & any string that ends at 0 only

Sol^n



Lang = { 0
0 0
1 0
0 0 0
⋮ }

$M = (\{q1, q2\}, \{0,1\}, \delta, q1, \{q1\})$

	0	1
q1	q1	q2
q2	q1	q2

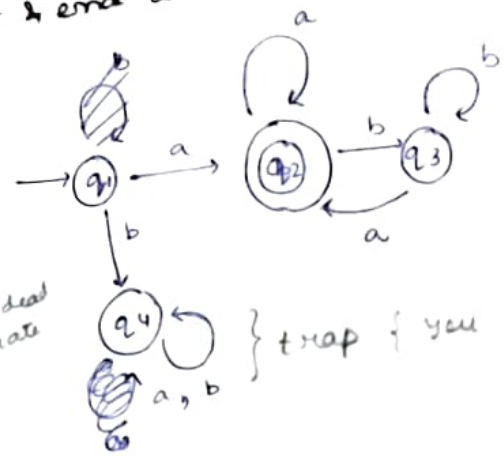
$\delta(q1, 1) = q2$

$\delta(q2, 1) = q2$

$\delta(q2, 0) = q1$

(3) Design an automaton that accepts all strings that start & end with a, where the alphabet is $\Sigma = \{a, b\}$

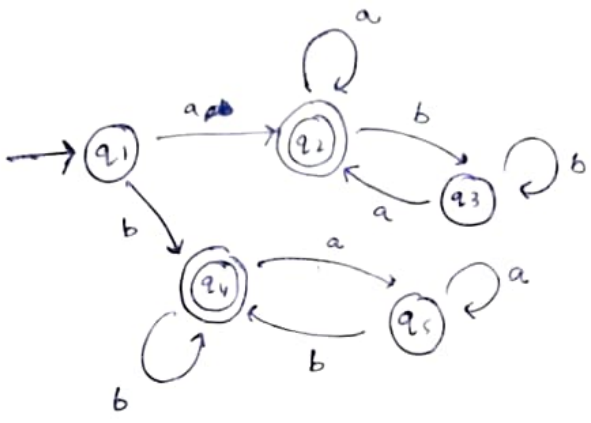
Sol^n



L = { a
a a
a b a
a a a
⋮ }

} trap (you can't go to final state)

(b) All string start & end at a OR start & end at b



~~M = { (~~

M = ({ q1, q2, q3, q4, q5 }, { a, b }, δ, q1, { q2, q4 })

Mapping Technique :

	a	b
q_1	q_2	q_4
q_2	q_2	q_3
q_3	q_2	q_3
q_4	q_5	q_4
q_5	q_5	q_4

21/9/23

$\delta(q_1, abbaba)$

$= \delta(q_2, bbaba)$

$= \delta(q_3, baba) = \delta(q_3, aba) = \delta(q_2, ba) = \delta(q_3, a)$

$= q_2$

$w_2 = abbab$; this string will be trapped / rejected

Types of FA

- FA without output
  - Deterministic FA (DFA)
  - Non-deterministic FA (NFA)
  - E-NFA
- FA with output
  - Mealy Machine
  - Moore machine

- DFA

→ DFA is nothing but a finite automata whose operation is completely determined by their input.

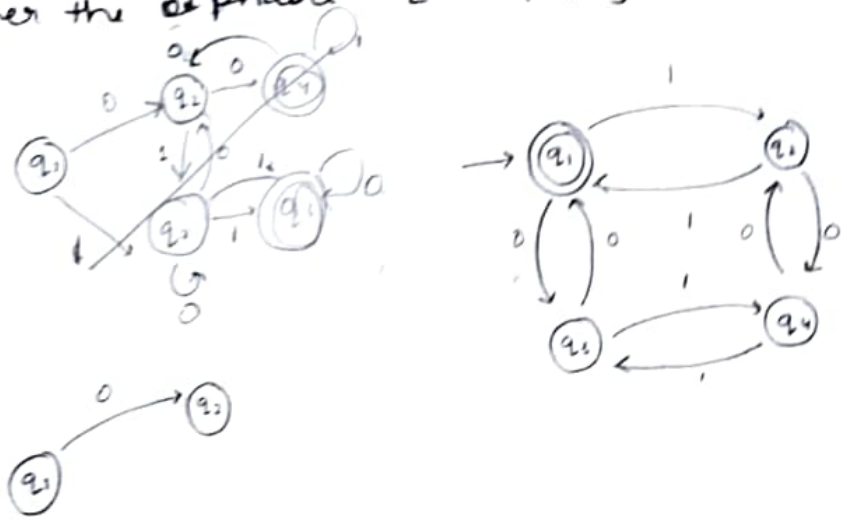
→ It is simply a language recognition devices.

→ In DFA, even a current state, we know what the next step will be. It has only one unique next state

I has no choice / randomness

→ It is simple & easy to design

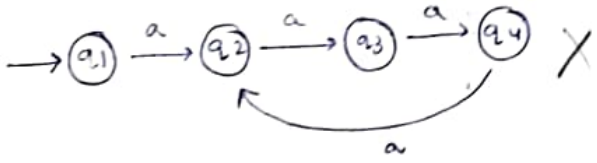
(1) Construct a DFA that will accept all strings with even no.s of zero's and even no. of 1's over the alphabet $\Sigma = \{0, 1\}$

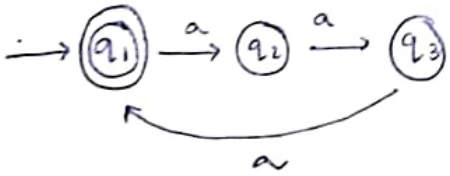


22| 9|23

(2) Design a DFA that will accept all strings with no of a's divisible by 3 over the alphabet $\Sigma = \{a\}$.

Ans

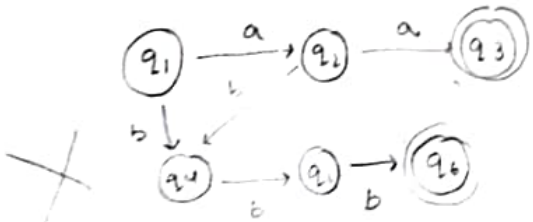


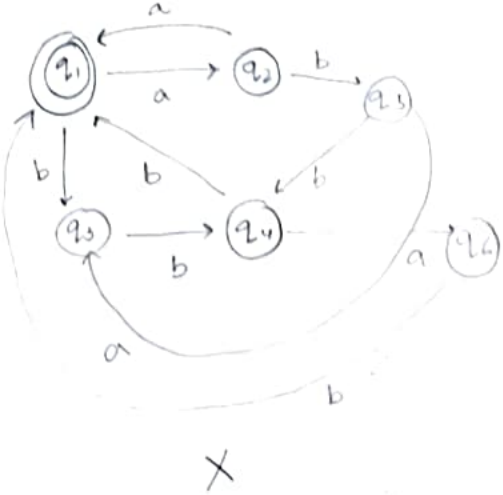


(3) Design a DFA that will accept all strings with no. of a's divisible by 2 & no. of b's div by 3 over $\Sigma = \{a, b\}$

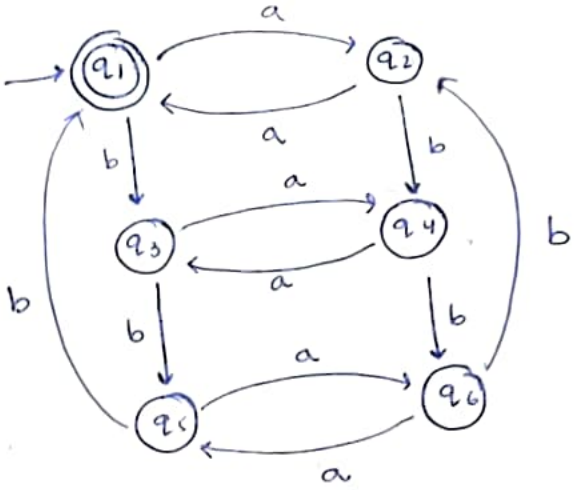
i/p strings

a a
b b b
a b a b b
a b b a b
a b b b a
a a b b b

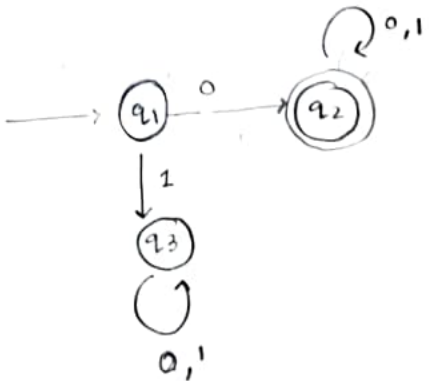




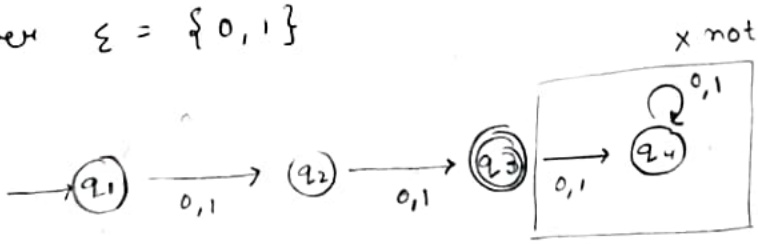
X



(4) " " " " " all strings that start with a "0" over $\Sigma = \{0, 1\}$



(5) Construct a DFA that will accept all strings of length 2 over $\Sigma = \{0, 1\}$



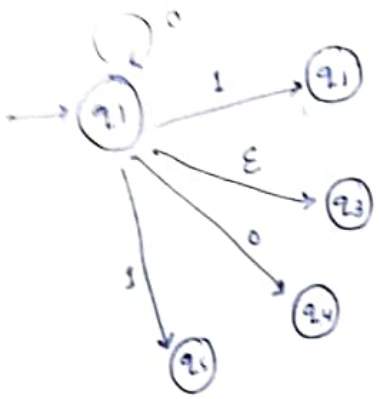
only 3 states needed

23/9/23

Non-deterministic FA (NFA)

→ In NFA, for some given state, & input symbol, the transition function "δ" may lead to a set of states that is a subset of Q.

→ In NFA, several choices may exist for the next state at any point.



• In NFA, empty string can also change state

★★ → An NFA is defined as a 5 tuple $(Q, \Sigma, \delta, q_0, F)$

where 1. Q is the finite set of states

2. Σ is the " " " input symbols

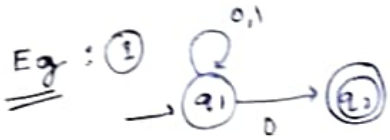
3. $q_0 \in Q$ is the initial state

4. $F \subseteq Q$ is " final "

5. δ is the mapping f^n or Transition function that maps $\delta : Q \times \Sigma \rightarrow 2^Q \approx P(Q)$ i.e. all possible subsets of Q

$P(Q)$ → all possible subsets of Q (power set of Q)

Eg :

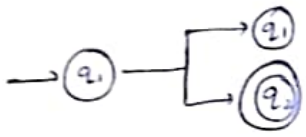


NFA N_1

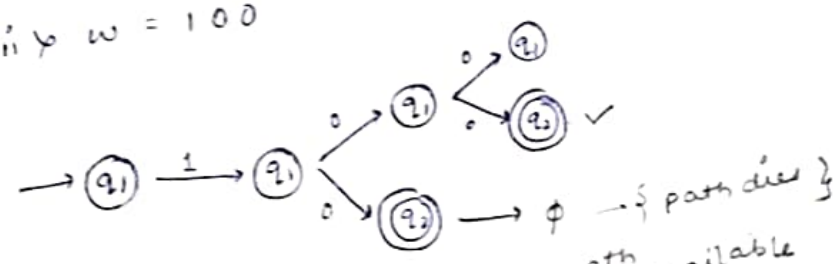
Here N_1 will accept all strings that ends at '0' ~~other~~ over $\Sigma = \{0,1\}$

∴ if at least 1 cond^n reaches final state, then string is accepted

∴ if Input string = 0



∴ if $w = 100$



∴ path not available

string accepted

★ $N_1 = (\{q_1, q_2\}, \{0,1\}, \delta, q_1, \{q_2\})$

② δ mapping :

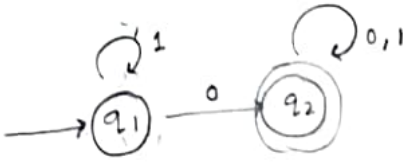
	0	1
q_1	$\{q_1, q_2\}$	q_1
q_2	ϕ	ϕ

** If Q has n elements then, no. of states possible from 1 set to another will be 2^n

Eg : $Q = \{q_1, q_2\}$

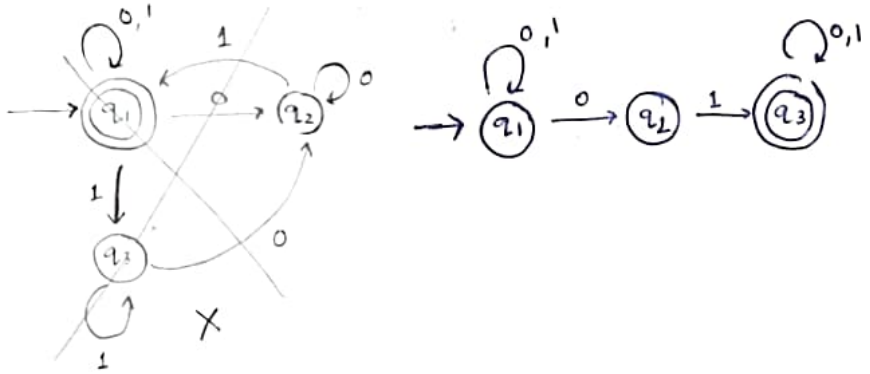
$q_1 \longrightarrow q_1, q_2, \{q_1, q_2\}, \phi$

Q① construct an NFA that accepts every string that has 0 over $\Sigma = \{0, 1\}$



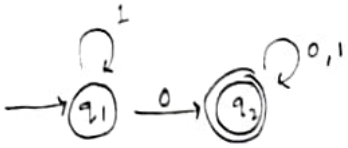
28/9/23 ~~Eg 1~~ Ex ① construct an NFA that accepts all strings with 01 over $\Sigma = \{0, 1\}$

:01"



i/p string
01
001
010
101
011

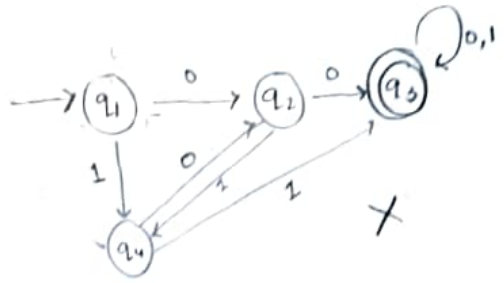
② " " NFA " " " " that consisting '0' over $\Sigma = \{0, 1\}$



i/p
0
00
01
10
000
001
110

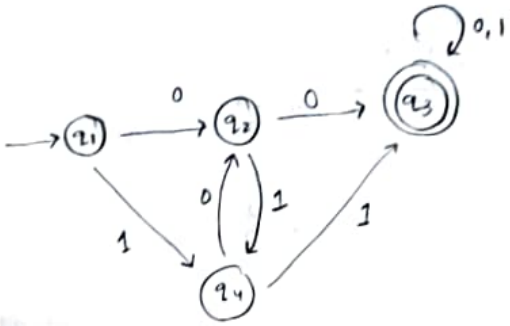
③ Construct the NFA will accept all strings with 2 consequtive 0's or 2 consecutive 1's over $\Sigma = \{0, 1\}$

Sol^n



i/p

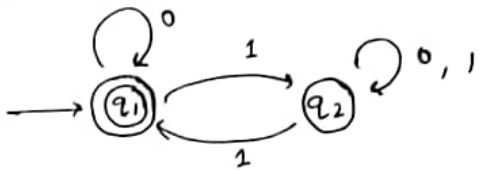
0 0 0 1
0 0
1 1
0 0 1
1 1 0
0 1 1
1 0 0



30/9/23 **Equivalence of NFAs & DFAs**
(i.e. conversion from NFA to DFA)

Non-determinism is a generalization of determinism, so every deterministic finite automaton (DFA) is automatically a non deterministic finite automaton (NFA).
That is, every DFA is an NFA but not vice versa.

Eg: (1) convert the given NFA to the equivalent DFA



$\delta : Q \times \Sigma \longrightarrow 2^Q$

Transition mapping table :

	0	1
→ q_1	q_1	q_2
q_2	q_2	$\{q_1, q_2\}$

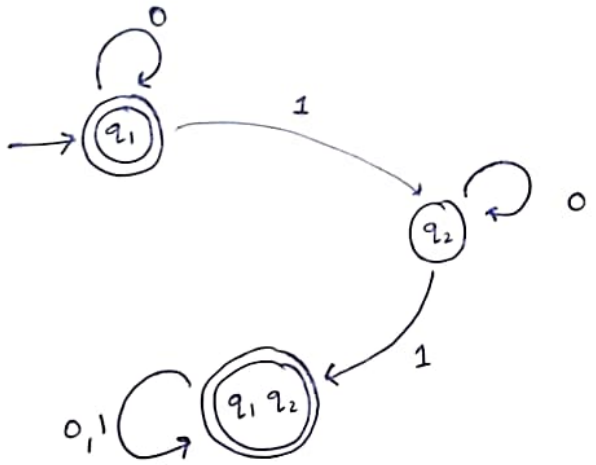
Transition mapping table of DFA :

	0	1
→ q_1	q_1	q_2
q_2	q_2	q_1q_2
q_1q_2	q_1q_2	q_1q_2

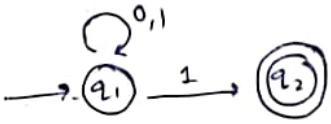
q_1q_2 → new state

union from δ of NFA table

DFA :



Q② Convert the given NFA to equivalent DFA

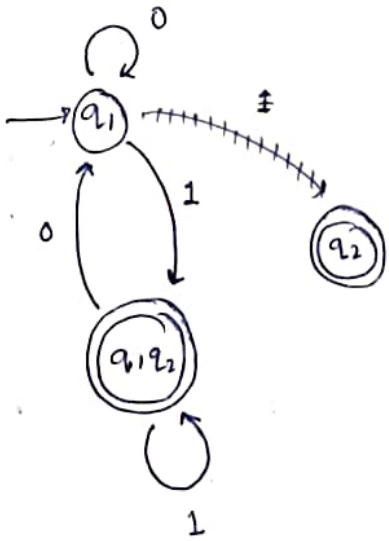


δ : NFA

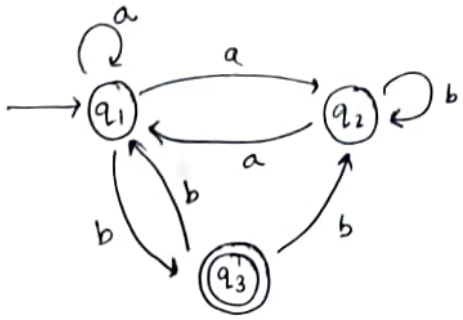
	0	1
→ q_1	q_1	$\{q_1, q_2\}$
q_2	ϕ	ϕ

δ_{DFA} :

	0	1
q_1	q_1	q_1q_2
q_2	ϕ	ϕ
q_1q_2	q_1	q_1q_2



(3) Convert the given NFA to DFA



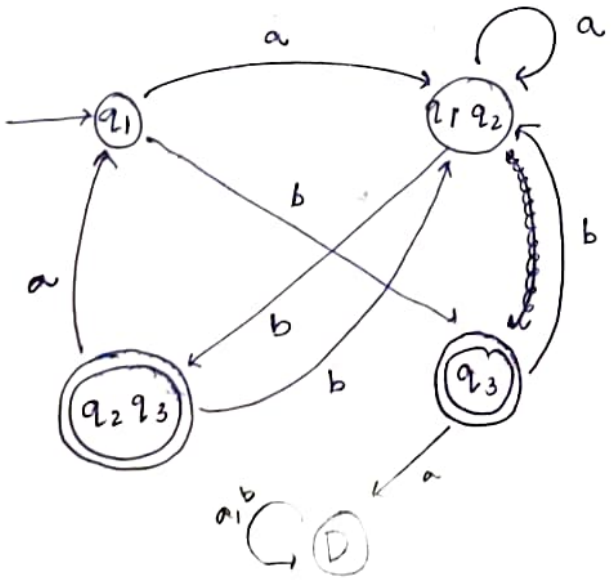
δ_{NFA} :

	a	b
q_1	$\{q_1, q_2\}$	q_3
q_2	q_1	q_2
q_3	ϕ	$\{q_1, q_2\}$

δ_{DFA} :

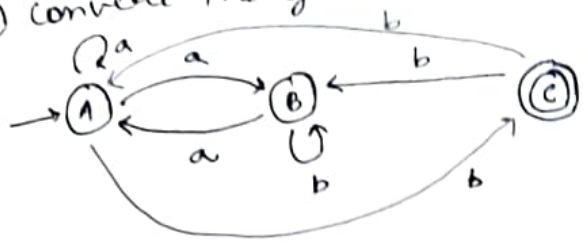
	a	b
q_1	$q_1 q_2$	q_3
$q_1 q_2$	$q_1 q_2$	$q_2 q_3$
q_3	D	$q_1 q_2$
$q_2 q_3$	q_1	$q_1 q_2$
D	D	D

dead / trap state as ϕ isn't available in DFA



5|10|23

(4) Convert the given NFA to DFA.

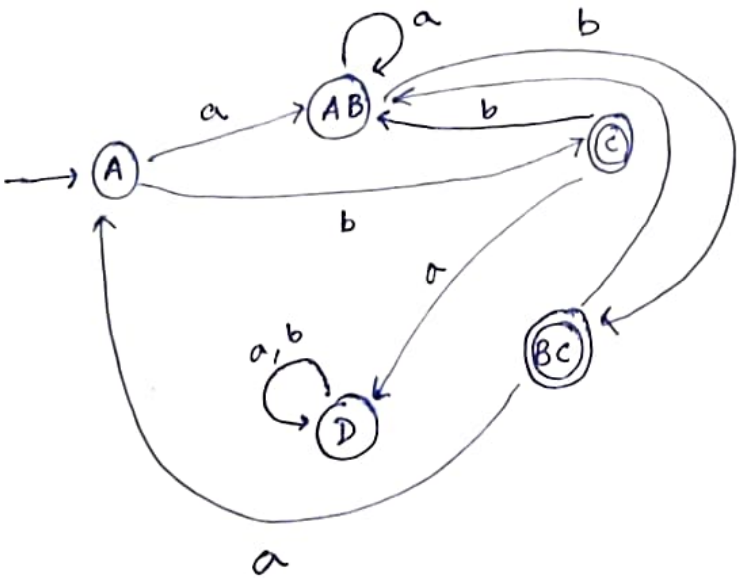


Sol^n δ_{NFA} :

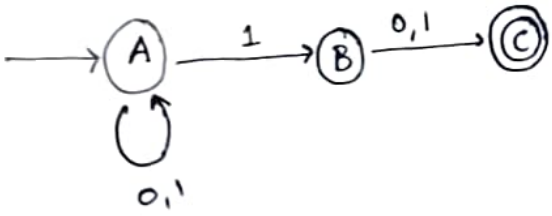
	a	b
→ A	{A, B}	C
B	A	B
C	ϕ	{A, B}

δ_{DFA} :

	a	b
→ A	AB	C
AB	AB	BC
C	D	AB
Bc	A	AB
D	D	D



5)



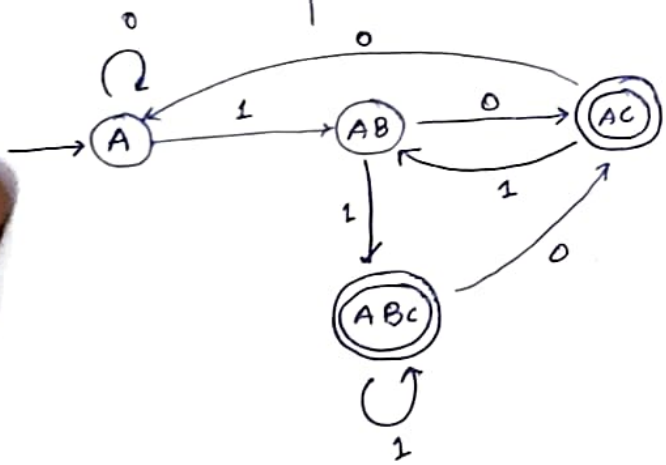
Sol^n

δ_{NFA} :

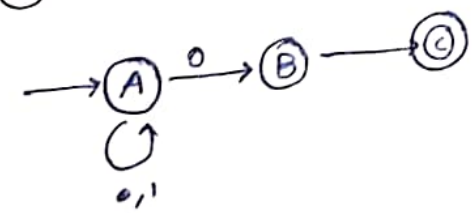
	0	1
→A	A	{A, B}
B	C	C
C	ϕ	ϕ

δ_{DFA} :

	0	1
→A	A	AB
AB	AC	ABC
AC	A	AB
ABC	AC	ABC



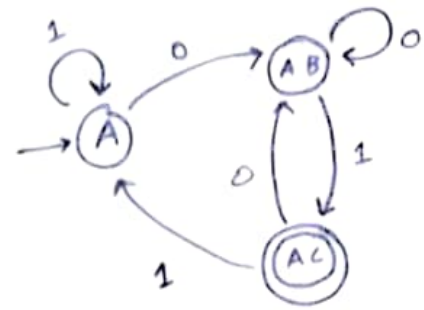
6) Convert NFA to DFA



δ_{NFA} :

	0	1
→ A	{A, B}	A
B	ϕ	C
C	ϕ	ϕ

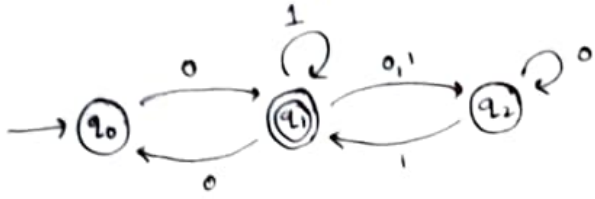
State diagram :-



δ_{DFA} :

	0	1
→ A	AB	A
AB	AB	AC
AC	AB	A

(7)



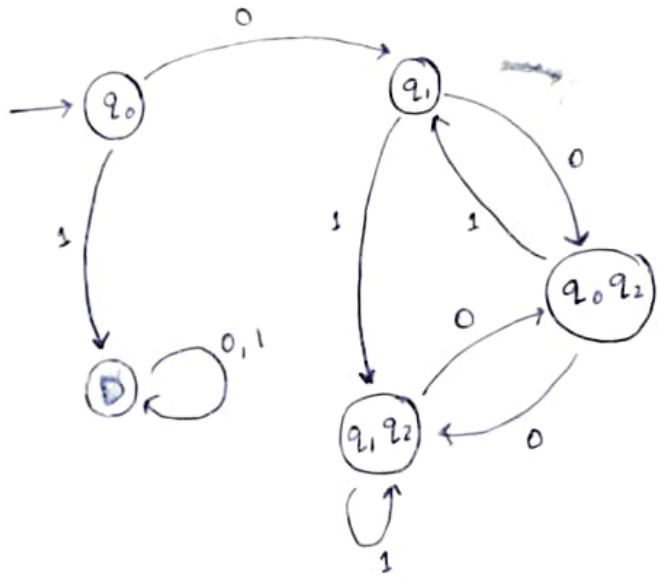
δ_{NFA} :

	0	1
→ q_0	q_1	ϕ
q_1	$\{q_0, q_2\}$	$\{q_1, q_2\}$
q_2	q_2	q_1

δ_{DFA} :

	0	1
→ q_0	q_1	D
q_1	$q_0 q_2$	$q_1 q_2$
$q_0 q_2$	$q_1 q_2$	q_1
$q_1 q_2$	$q_0 q_2$	$q_1 q_2$
D	D	D

State diagram :



Steps for converting NFA To DFA

1. Convert the given NFA to its equivalent transition mapping table
2. Create the DFA's initial state
3. Create the DFA's transition mapping table
4. Create the DFA's final states
5. Simplify the DFA ; i.e. -
   - i) remove unreachable states
   - ii) merge equivalent state (states that have the same transition rules for all input symbols can be merged into single state)
   - iii) Remove dead states.

Repeat steps 3 To 5 until no further simplified is possible.

6/10/23

Epsilon NFA (ε-NFA)

* ε represents empty symbol

Eg:

δ :

	ε
A	A
B	{B, C}
C	C

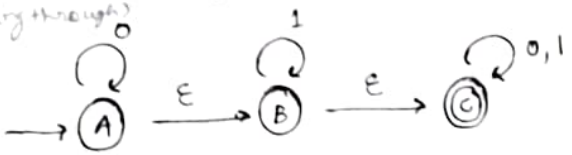
* $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \longrightarrow 2^Q$

Conversion from ε-NFA To NFA

- The procedure for converting any ε-NFA to its equivalent NFA is that -
  - → For each state that we have, check where does this state go on ε^* (epsilon closure).
  - → Then the set of state that we get here have to be checked on which state does they go on getting a particular input & the set of states that will get here have to be again checked on to which state do they go on ε^* again.

ε^* (epsilon closure) : It means all the states that can be reached from a particular state only by seeing the ε symbol.
(Or by travelling through)

Eg ①



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

states	ε^*
A	{A, B, C}
B	{B, C}
C	{C}

	ε^*	0	ε^*
A	A	A	A, B, C
	B	ϕ	ϕ
	C	C	C
B	B	ϕ	ϕ
	C	C	C
C	C	C	C

	ε*	1	ε*
A	A	ϕ	ϕ
	B	B	B, C
	C	C	C
B	B	B	B, C
	C	C	C
C	C	C	C

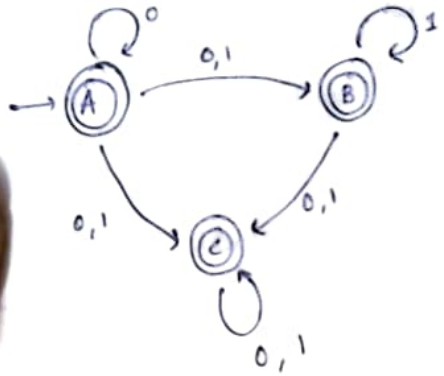
Transition mapping of the NFA

	0	1
A	{A, B, C}	{B, C}
B	C	{B, C}
C	C	C

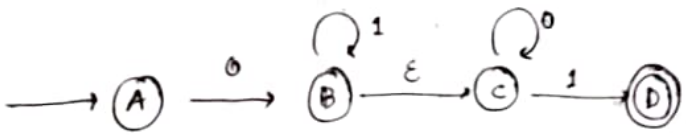
of ε-NFA

- A state which can reach to the final state on seeing an ε symbol will be the final state of ~~ε-NFA~~ of the regular NFA.

So, A, B, C all are final states



Eg (1) Convert the following ε-NFA to its equivalent NFA.



Sol^n Transition mapping table of NFA

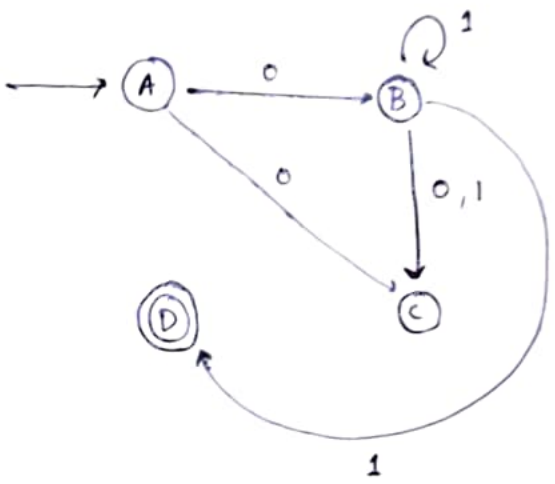
	0	1
A	{B, C}	ϕ
B	C	{B, C, D}
C	C	D
D	ϕ	ϕ

ϵ^* table

States	ϵ^*
A	A
B	{B, C}
C	C
D	D

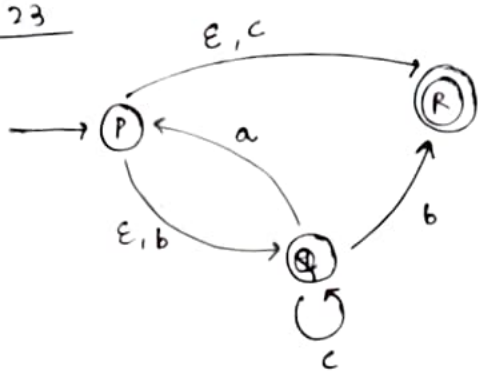
	ϵ^*	0	ϵ^*
A	A	B	{B, C}
B	B	ϕ	ϕ
	C	C	C
C	C	C	C
D	D	ϕ	ϕ

	ϵ^*	1	ϵ^*
A	A	ϕ	ϕ
B	B	B	{B, C}
	C	D	D
C	C	D	D
D	D	ϕ	ϕ



7 | 10 | 23

(3)



Convert the ϵ-NFA to its equivalent NFA.

Soln

ϵ^* table

State	ϵ^*
P	{P, Q, R}
Q	Q
R	R

Transition mapping of NFA :

	a	b	c
P	{P, Q, R}	{Q, R}	{Q, R}
Q	{P, Q, R}	R	Q
R	ϕ	ϕ	ϕ

	ε*	a	ε*
P	P	ϕ	ϕ
	Q	P	{P, Q, R}
	R	ϕ	ϕ
Q	Q	P	{P, Q, R}
R	R	ϕ	ϕ

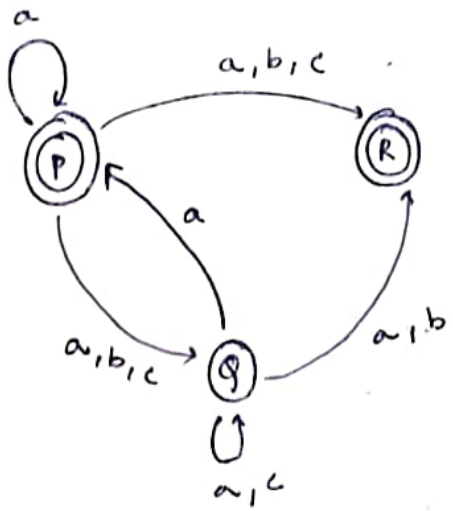
	ε*	b	ε*
(P)	P	Q	Q
	Q	R	R
	R	ϕ	ϕ
(Q)	Q	R	R
(R)	R	ϕ	ϕ

	ε*	c	ε*
P	P	R	R
	Q	Q	Q
	R	ϕ	ϕ
Q	Q	Q	Q
R	R	ϕ	ϕ

State diagram

ε-NFA To NFA

Equivalent NFA



Regular Expression

Regular Language - Any language that can be defined by a finite state machine.

- Regular expressions are the expression is built of by using regular operations eg - U, ., *
- The regular expressions are used to describe the regular languages. The value of a regular expression is a regular language i.e. Regular expressions generate the regular languages.

Eg: $(0 \cup 1)\,0^* \approx (0+1)\cdot 0^*$

- Regular expressions are used for representing certain sets of strings in an algebraic fashion.

Certain Rules about Regular Expression

1. Any terminal symbol i.e. symbols $\in \Sigma$ including ϵ and ϕ are regular expressions.
2. The union of 2 regular expressions is also a regular expression i.e. if R_1 & R_2 are 2 regular expressions then $R_1 \cup R_2$ is ~~also~~ a regular expression.
3. ~~to~~ The concatenation of 2 regular expression is also a regular expression i.e., If R_1 & R_2 are two regular expressions then $R_1 R_2$ or $R_1 \cdot R_2$ is also a regular expression.
4. The iteration/closure of a regular expression is also a regular Expression i.e. If R is a regular expression then R^* is also a regular expression.

Formal definition of Regular Expression

Say that R is a regular expression if R is :

(1) 'a', for some symbol 'a' in the alphabet Σ

(2) ε

(3) ϕ

(4) $(R_1 \cup R_2)$, $\approx^{(R_1+R_2)}$ where R_1 & R_2 are regular expressions

(5) $(R_1 \circ R_2)$ (concatenation), $\approx^{R_1 R_2}$ where " " " " " " "

(6) R_1^*, where R_1 is a regular expression

12|10|23

Eg : $R = (0 \cup 1)\,1^*$; $\Sigma = \{0, 1\}$

R is regular expression

Examples of Regular Expression

(1) $\{0, 1, 2\}$ ← Language

$R = 0 \cup 1 \cup 2 = 0 + 1 + 2$

(2) $\{\varepsilon, a\}$

$R = \varepsilon a$

(3) $\{a, b, ab, aaa, bb\}$

$R = a + b + ab + aaa + bb$

(4) $\{\}$

$R = \phi$

(5) $\{0, 00, 000, \cdots\}$

$R = 0^+$

(6) $\{\varepsilon, 0, 00, \cdots\}$

$= 0^*$

① $L = \{ \varepsilon, 0, 00, 000, \cdots, 10 \}$

$R = 0^* + 10$

Identities of Regular Expression

1. $\phi \cup R = R \cup \phi = R$
2. $\phi R = R\phi = \phi$ [concatenating the empty language ϕ to any set yields the empty set ϕ]
3. $\phi R + R\phi = \phi$
4. $\varepsilon R = R\varepsilon = R$
5. $\varepsilon^* = \varepsilon$
6. $\phi^* = \varepsilon$
7. $R + R = R$
8. $R^* R^* = R^*$
9. $R R^* = R^* R$
10. $(R^*)^* = R^*$
11. $\varepsilon + RR^* = \varepsilon + R^* R = R^*$
12. $(PQ)^* P = P(QP)^*$
13. $(P+Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$
14. $(P+Q)R = PR + QR$

OR

$R(P+Q) = RP + RQ$

Eg ① $L = \{ w \mid$ ~~length etc~~ of w is at least 2 $\}$

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

$L = \{ aa, ab, ba, bb, \cdots \}$

$R = aa, ab, ba, bb, aaa$

$= a(a+b) + b(a+b) \cdots$

$= (a+b)(a+b)(a+b)^*$

$R = aaa + aab + abb + bbb + bba + bab$

$= aa(a+b) + bb(a+b) + ba(b+a)$

$= (a+b)\{aa + bb + ba\}$

$= (a+b)\{aa + b(b+a)\}$

$=$

Eg (2) $L = \{w \mid \text{length of } w \text{ is atmost } 2\}$

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

$L = \{\varepsilon, a, b, aa, ab, ba, bb\}$

$R = \varepsilon + a + b + aa + ab + ba + bb$

$= \varepsilon + (a+b) + (a+b)^2$

$= (\varepsilon + a + b) + (\varepsilon + a + b)^2$

$= (\varepsilon + a + b)(\varepsilon + \varepsilon + a + b)$

$= (\varepsilon + a + b)(\varepsilon + a + b)$

A regular expression R describes the language L

13/10/23 Examples on Regular language

① $R = (0+1)\,1^*$; $\Sigma = \{0,1\}$

R describes the language with strings we

L = starts with 0 or 1, followed by any no. of ones

(OR)

L: It denotes all strings starting with 0 or 1, followed by any no. of ones.

② $R = (0+1)^*\ 00\ (0+1)^*$

L = denotes all strings of 0's or 1's, with at least 2 consequetive zeros.

③ $R = (1+10)^*$

L = Denotes all strings of 0's & 1's begins with 1 & not having 2 consecutive 0's.

④ $R = (0+1)^*\ 011$

L = Denotes all strings of 0's & 1's ending with 011

⑤ $R = 0^*\,1\,0^*\,1\,0^*$

L = Denotes all strings ~~of 0's & 1's starting with 0 & having~~ of 0's & 1's ~~& ending~~ exactly 2 ones

⑥ $R = 01 + 01$

L = {01} only

⑦ $\Sigma = \{0,1\}$; $R = \Sigma^*\,1\,\Sigma^*$

L = at least one 1

⑧ $0^*\,1\,0^*$

L = exactly one 1

⑨ $\Sigma^*\,001\,\Sigma^*$

L = substring 001

Example ① P.T. the regular expression

$\varepsilon + 1^*(011)^* (1^*(011)^*)^*$ is equal to $(1+011)^*$

Soln We have to prove

$$\varepsilon + 1^*(011)^* (1^*(011)^*)^* = (1+011)^*$$

Proof :

$$LHS = \underbrace{\varepsilon} + \underbrace{1^*(011)^*}_{R} \underbrace{(1^*(011)^*)^*}_{R^*}$$

$$= (1^*(011)^*)^* \quad [\because \text{AS } \varepsilon + PR^* = R^*]$$

$$= (1+011)^* \quad [\because \text{AS } (P^*Q^*)^* = (P+Q)^*]$$

$$= RHS$$

② P.T. the regular expression

$(1+00^*1) + (1+00^*1)(0+10^*1)^* (0+10^*1)$

is equal to $0^*1(0+10^*1)^*$

Proof : $LHS = (1+00^*1) + (1+00^*1)(0+10^*1)^*$

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

$$= (1+00^*1) + (1+00^*1)\{(0+10^*1)^* - \varepsilon\}$$

$$= (1+00^*1) + (1+00^*1)(0+10^*1)^* -$$

$$(1+00^*1)$$

$$= (1+00^*1)(1+$$

$$= 1(\varepsilon + 00^*)1(0+10^*1)^*$$

$$= 0^*1(0+10^*1)^* \quad [\because \varepsilon + RR^* = R^*]$$

$$= RHS$$