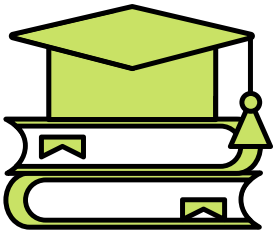


Gatenotes.in

GATE
Computer Science

Ravindrababu Ravula GATE CSE

Hand Written Notes

-: SUBJECT :-

Theory of Computation

Division: Roll:

Subject: T.O.C

UNIQUE

Serial Number	Date	Title	Page Number	Teacher's Sign/Remarks
1.		• Introduction of T.O.C. (1)		
2.		• Regular expression and finite Automata. (2)		
3.		• Regular Language and context free language. (2) (3)		
4.		• Context free grammars and push down automata. (3)		
5.		• Turing Machine. (4)		
6.		• Closure properties of language. (1,2,3,4)		
7.		• Uncountability. (4)		
8.		• undecidability. (4)		
9.		• pumping lemma. (2)		

INTRODUCTION OF TOC

- **Automata** :- Study of abstruct computing devices or machines.
- **Symbol** :- A symbol is an abstruct entity.
  Example : a, b, 0, 1, ...
- **Alphabet (Σ)** :- An Alphabet is a finite collection of Symbols.
  Example : $S = \{a, b, c\}$
  $\Sigma = \{0, 1\}$
- **String** :- It is a sequence of Symbols.
  Example :- $\Sigma = \{a, b\}$
  strings :- $\{\underline{a}, \underline{b}, \underline{aa}, \underline{ab}, \underline{ba}, \underline{bb}\}$ here have '6' string.
  ↳ length-2
  → Empty (Null) string can be denoted by ($\in$ or Λ or λ).
  [If the length of string is zero, such string is called empty string]

- **Language** :- collection of string.
  [where strings is restricted over given alphabet]
  → $\Sigma = \{a, b\}$
  L_1 = set of all string of length 2.
  $= \{aa, ab, ba, bb\}$
  Where language L_1 is finit language.

  → $\Sigma = \{a, b\}$
  L_2 = set of all string where each string starts with 'a'.
  $= \{a, aa, ab, aaa, aab, aba, ...\}$
  where, L_2 is infinit language.

- **Length of string** :- No of symbols contained in a string.
  $|\{ab\}| \Rightarrow$ length '2'. $|\in| \Rightarrow$ length '0'.

- **Prefix of a string :-** If $w = xy$, for some string y, then x is a prefix of w.

  example - $w = \{0011\}$

  $\Rightarrow \{\underset{x}{001}\ \underset{y}{1}\} \Rightarrow \{\underset{x}{00}\ \underset{y}{11}\} \Rightarrow \{0, 00, 001\}$

- **Suffix of a string :-** If $w = xy$, for some string x, then y is a suffix of w.

  example - $w = \{0011\}$

  $= \{\underset{x}{001}\ \underset{y}{(1)}\} \Rightarrow \{1, 11, 011\}$

  $\rightarrow$ suffix.

- **Kleen closure :-**

$\rightarrow$ It is denoted by * (asterisk) after the name of the alphabet is Σ^*. This notation also known as the Kleene Star.

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

Σ^0 = set of all strings of length '0'. $= \{\epsilon\}$.

Σ^1 = set of all strings of length '1' $= \{a, b\}$

$\Sigma^2 = \Sigma . \Sigma = \{a, b\}\ \{a, b\}$

$= \{aa, ab, ba, bb\}$ = set of all strings of length '2'.

$\Sigma^3 = \Sigma\ \Sigma\ \Sigma =$ set of all strings of length '3'.

$|\Sigma|^3 = 8$ (no of string)

$\vdots$

$|\Sigma|^n = 2^n$ = set of all strings of length 'n'.

$\therefore \boxed{\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots\ldots\ldots}$

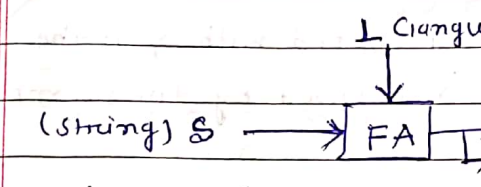
$= \{\epsilon\} \cup \{a, b\} \cup \{aa, ab, ba, bb\} \cup \ldots\ldots\ldots$

Σ^* = set of all string possible over $\{a, b\}$ [Universal set]

$\boxed{\Sigma^* = \Sigma^+ \cup \{\epsilon\}}$.

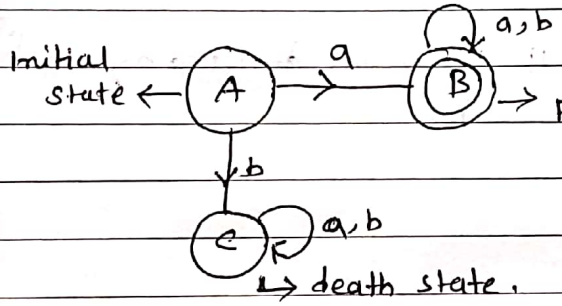
$\Sigma^* = \boxed{L_1 \quad L_2 \quad L_3}$ $\quad \boxed{L_1 \subseteq \Sigma^* \\ L_2 \subseteq \Sigma^* \\ L_3 \subseteq \Sigma^*}$

→ no of string possible over Σ^* is infinite.
→ no of Language possible over Σ^* is infinite.



→ A program is a string.
→ C-programming language = set of all valid program.

L = set of all strings which starts with 'a'.
L = { a, aa, ab, aaa, ... }



string = aab (present in Language or not)
this string are accepted by FA. $A \xrightarrow{a} B \xrightarrow{a} B \xrightarrow{b} (B)$

→ the string will be accepted by FA, if after scaning entire string we reach in final state from initial state.

string = bba
this string are not accepted by FA. $A \xrightarrow{b} C \xrightarrow{b} C \xrightarrow{a} C$

- **positive closure:**

→ The '+' (plus operation) is some times called positive closure.
→ If $\Sigma = \{a, b\}$, then,

$\Sigma^+ = \{ a, aa, ab, ba, bb, \ldots \}$

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

- **Substring of a string :-**

→ A string 'w' = abc

here, ab, bc are substring of w.

But ac are not substring of w.

- **Concatenation of two string :-**

→ If $x, y \in \Sigma^*$, then x concatenated with y is the word formed by the symbol of 'x' followed by the symbols of 'y'.

→ This is denoted by $x \cdot y$, it is same as xy.

- **Reversal of a string :**

→ Given a string w, it reversal denoted by w^R is the string spelled back wards.

$w = ab$

$w^R = ba$.

- **Grammar :**

→ It enumerates string of the language.

→ It is a finite set of rules defining a language.

→ A grammar is defined as 4-tuples (V, T, P, S)

Where,
- V → Set of non terminals.
- T (Σ) → Set of input terminals.
- P → finite set of production rule.
- S → start of symbol.

example :- → start (S).

(production) P = $\begin{cases} S \rightarrow aSB \\ S \rightarrow aB \\ B \rightarrow b \end{cases}$

here, $V = \{S, B\}$

$T = \{a, b\}$

→ Getting a string from a grammar is called Derivation.

Derivation on aabb

$S \rightarrow aSB$
$\rightarrow aaBB \quad [S \rightarrow aB]$
$\rightarrow aabB \quad [B \rightarrow b]$
$\rightarrow \boxed{aabb} \quad [B \rightarrow b]$

→ entire process called derivation.
→ sequential forms.

Q-1 Construct a grammar given the following language,
$L = \{$ Set of all string length '2' $\}$
$\Sigma = \{a, b\}$

→ $L = \{aa, ab, ba, bb\}$ $\quad (a+b)\ (a+b)$; A A

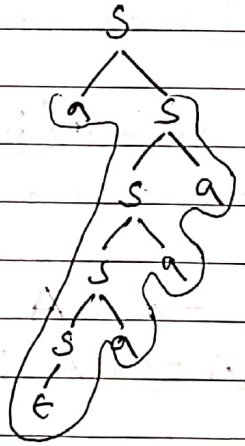
$S \rightarrow aa/ab/ba/bb.$
$S \rightarrow AA$
$A \rightarrow a/b$

→ production rules.

Q-2 construct grammar, given the following language,
$L = \{a^n / n \geq 0\}$

→ $L = \{\epsilon, a, aa, aaa, \cdots\}$

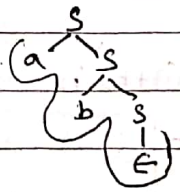
P.R→
$S \rightarrow aS/\epsilon$
$S \rightarrow Sa/\epsilon$



(a^4).

Q-3 Construct grammar, to generate $(a+b)^*$.

→ $\boxed{S \rightarrow aS/bS/\epsilon}$



(ab)

Q-4 Construct a grammar, given the following language,
$L = \{$ set of all string of length atleast '2' $\}$

→ $L = \{aa, ab, ba, bb, aaa, aab \cdots\cdots\}$
$(a+b)_A\ (a+b)_A\ (a+b)^*$

production rules,

$S \rightarrow AAB$
$A \rightarrow a/b.$
$B \rightarrow aB/bB/\epsilon.$

Q-5 C.A.G, given the following language,
$L = \{$ string of length at most '2' $\}$.

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

$(a+b+\epsilon)_A\ (a+b\epsilon)$

production rules,

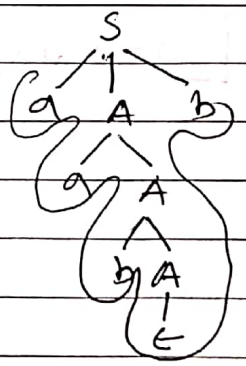
$S \rightarrow AA$
$A \rightarrow a/b/\epsilon$

Q-6 $L = \{$ start with 'a' and end with 'b' $\}$.

→ $a\ (a+b)^*\ b$

production rules,

$S \rightarrow aAb$
$A \rightarrow aA/bA/\epsilon$



aabb

Q-7 $L = \{$ set of all string starting and ending with different symbol $\}$.

→ $a(a+b)^*b + b(a+b)^*a$

PR:- $S \rightarrow aAb/bAa.$
$A \rightarrow aA/bA/\epsilon$

Q-8 $L = \{$ Set of all string starting and ending with same symbol $\}$

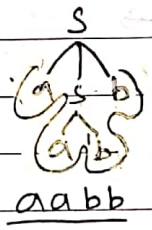
→ $a(a+b)^*a + b(a+b)^*b$.

$S \to aAa / bAb / b / a / \epsilon$
$A \to aA / bA / \epsilon$

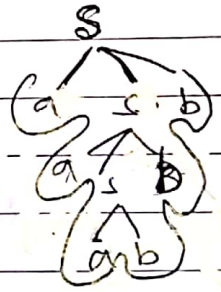
Q-9 construct a grammar, given the following language.
$L = \{a^n b^n / n \geq 1\}$

→ $L = \{a^n b^n / n \geq 1\}$
$= \{ab, aabb, \cdots\}$

$S \to aSb / ab / \epsilon$



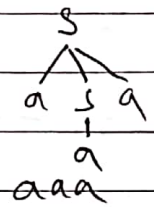
aabb



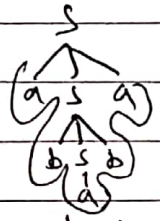
aaabbb

Q-10 construct grammars that generate set of all palindrom - WW^R, WaW^R, WbW^R

→ $S \to aSa / bSb / a / b / \epsilon$



aaa



abaaba

Q-11 Construct grammar that generate "even length string".

→ $L = \{aa, ab, ba, bb, aaaa, \cdots\}$
$((a+b)(a+b))^*$

$S \to BS / \epsilon$
$B \to AA$
$A \to a / b$

Q-12 $L = \{a^n b^m / n, m \geq 1\}$

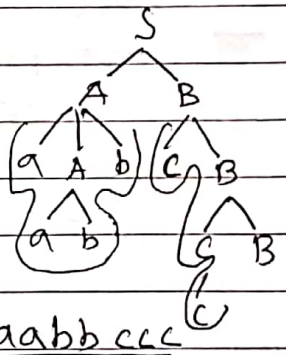
→ $L = \{ab, aab, aabb, abb, \cdots\}$

$S \rightarrow AB$	
$A \rightarrow aA / a$	$\rightarrow a^n$
$B \rightarrow bB / b$	$\rightarrow a^m$

Q-13 $L = \{a^n b^n c^m / n, m \geq 1\}$

→

$S \rightarrow AB$
$A \rightarrow aAb / ab$
$B \rightarrow cB / c$

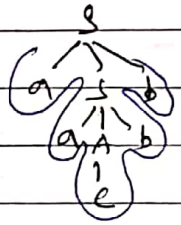


aabb ccc

Q-14 $L = \{a^n c^m b^n / n, m \geq 1\}$

→

$S \rightarrow aSb / aAb$
$A \rightarrow cA / c$



aacbb

Q-15 $L = \{a^n b^n c^m d^m / n, m \geq 1\}$

→

$S \rightarrow AB$	
$A \rightarrow aAb / ab$	$\rightarrow a^n b^n$
$B \rightarrow cBd / cd$	$\rightarrow c^m d^m$

Q-16 $L = \{a^n b^{2n} / n \geq 1\}$

→ $S \rightarrow aSbb / abb$

Q-17 $L=\{a^{m+n}\ b^m\ c^n \mid m,n \geq 1\}$ $\{a^n b^m c^m d^m \mid n,m \geq 1\}$ (X)

$\rightarrow L=\{a^m a^m b^m c^n \mid m,n \geq 1\}$

$S \rightarrow aSc \mid aAc$.
$A \rightarrow aAb \mid ab$.

Q-18 $L=\{a^n b^{n+m} c^m \mid n,m \geq 1\}$

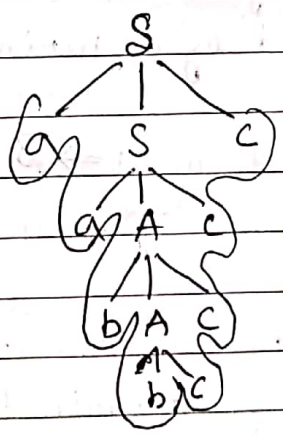
$\rightarrow L=\{a^n b^m b^m c^m \mid n,m \geq 1\}$

$S \rightarrow AB$
$A \rightarrow aAb \mid ab$
$B \rightarrow bBc \mid bc$

Q-19 $L=\{a^n b^m c^{n+m} \mid n,m \geq 1\}$

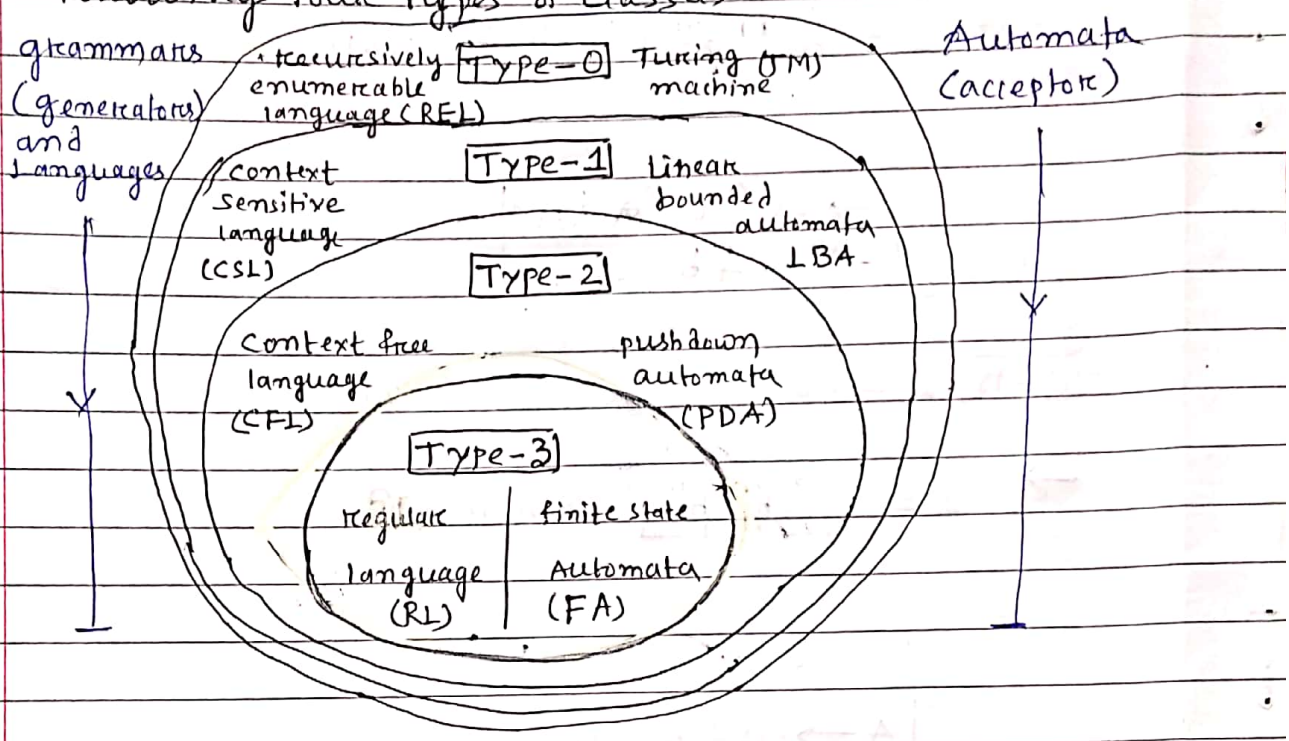
$\rightarrow L=\{a^n b^m c^m c^n \mid n,m \geq 1\}$

$S \rightarrow aSc \mid aAc$
$A \rightarrow bAc \mid bc$



$aabbcccc \Rightarrow a^{(n)}\ b^{(m)}\ c^{(2+2)}$
$\Rightarrow a^2 b^2 c^4$.

- Comsky Hierarchy: Comsky hierarchy consists of following four types of classes-



(Chomsky Hierarchy)

- Types of Grammers :-

① TYPE-0 Grammar (Unrestricted Grammar):

→ These are unrestricted grammar which include all formal language.

→ This grammar generate exactly all language that can be recognized by a turing machine.

→ Rules are of the form $\alpha \rightarrow \beta$

(α must have at least 1 Non-terminal.)

→ Where, α and β are are arbitrary sequence of terminals and non-terminals and $\alpha \neq \epsilon$.

ex- $A \rightarrow aA \mid bA$ | $Xa \rightarrow Ba$

② TYPE-1 Grammar (Context Sensitive Grammar):

→ language defined by type-1 grammars are accepted by the linear bounded automata.

→ Rules are of the form, $|\alpha| \leq |\beta|$
length of 'α' is less than or equal to length of 'β'.

$A \to \epsilon$ is not allowed unless A is a start symbol.
✗ $AB \to a$, ✓ $A \to aB$

③ TYPE-2 Grammar (Context free Grammar)

→ language defined by type-2 grammar are accepted by push down automata.
→ Rules are of the form $\alpha \to \beta$.
Where, α = single nonterminal.

$A \to a$ ✓ $Aa \to bB$ ✗.

④ TYPE-3 Grammar (Regular Grammar)

→ language define by type-3 grammars are accepted by finite state automata.
→ Regular grammar can follow either right linear or left linear.

→ (right linear grammar)

$A \to \alpha B / \beta$
$A, B \in V$ (nonterminal)
$\alpha, \beta \in T^*$ (terminal)

example,
$A \Rightarrow aB$
$B \to aB/bB/a/b$

→ Left linear grammar

$A \to B\alpha / \beta$
$A, B \in V$
$\alpha, \beta \in T^*$

example,
$A \to Ba$
$B \to Ba/Bb/a/b$

If, $A \to Ba/a$, $B \to aB/a$ → not Type 3 gramma.

- Chomsky hierarchy Examples-

• Identify grammar :-

1) Example :

$S \rightarrow aSb \mid aA \mid B$

$B \rightarrow aA \mid a$

$C \rightarrow cD \mid D$

→ ans is (Type-0)

It will be accepted by Turing machine.

2) Example

$S \rightarrow a \mid aA \mid Bb$

→ (Type-2).

It will be accepted by Push Down automata.

3) Example-

$S \rightarrow aSb \mid ab$ → (Type-2) grammar.

- Type-0 class is also called as :

→ Unrestricted Grammar.
→ Recursively Enumerable languages.
→ Turing Machine.

- Type-1 class is also called as :

→ Context Sensitive grammar.
→ Context Sensitive language.
→ Linear Bounded Automata.

- Type-2 class is also called as :

→ Context Free Grammar.
→ Context Free language.
→ Push down automata.

- Type-3 class is also called as :

→ Regular Grammar.
→ Regular language.
→ finit automata.

- Finite Automata (FA):
  → Machines with fixed amount of unstructured memory, accepts regular language.
  → Application of FA: useful for modeling chips, communication protocols, adventure games, some control system, etc.

- push down Automata (PDA):
  → Finite Automata with unbounded structured memory in the form of a push down stack, accepts CFL.
  → Application of PDA: ~~Compilers~~ useful for modeling parsing, compilers, programming language design.

- Turing Machine (TM):
  → Finite automata with unbounded tape accepts or enumerates recursively enumerable language.

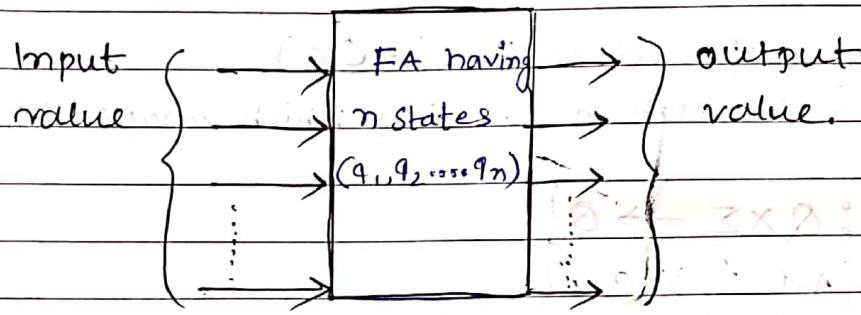
  → equivalent to RAM's and various programming language models.
  → Application of TM: Model for general sequential computation (real computer).

classmate
Date
Page 15

Regular Expression and Finite Automata.

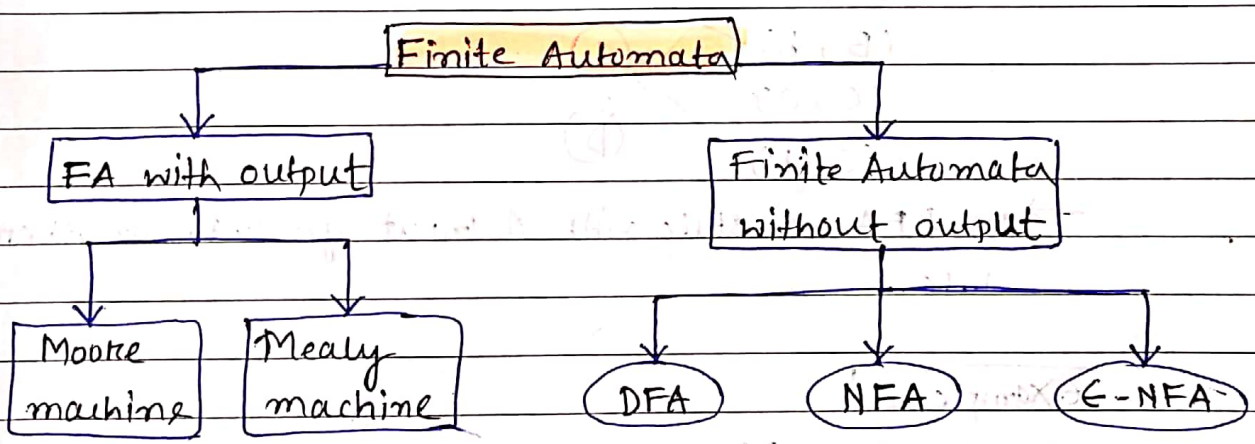
- **Finite Automata :-**

→ A Finite state Machine (FSM) or finite state automata is an abstract machine used in the study of computation and language that has only a finite, constant amount of memory.



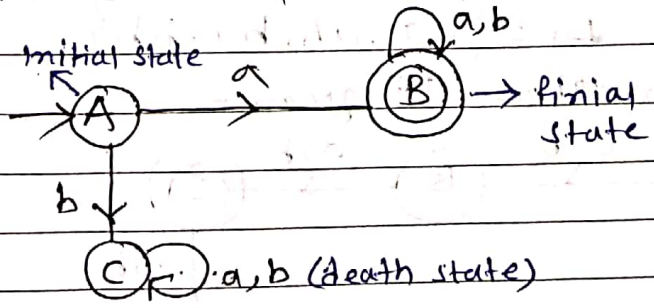
Model of finite automata

- **Types of Finite Automata :**



- **DFA (Deterministic Finite Automata) :-**

→ A finite Automata is defined as 5-tuples $(Q, \Sigma, \delta, q_0, F)$.



Where,

Q = set of all states ⇒ {A, B, C}

Σ = Input ⇒ {a, b}

q_0 = Initial state ⇒ 'A'.

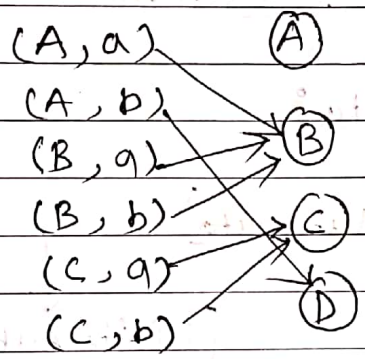
F = finial state ⇒ {B}
↳ set of.

→ Q is the superset of F. (F ⊂ Q)

δ = transation function which maps Q × Σ into Q.

$$\delta : Q \times \Sigma \rightarrow Q$$

{A, B, C} X {a, b}



→ in DFA, a state with 1 Input go into another one state.

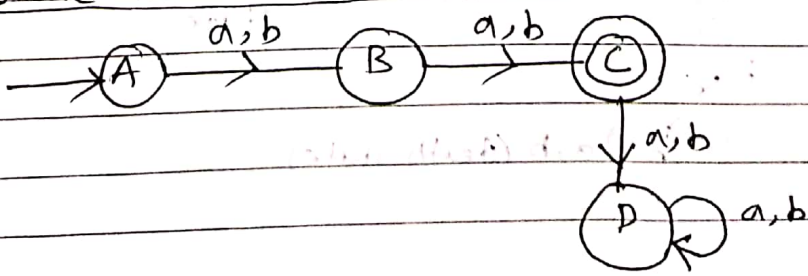
Example - 1

Construct A DFA that accept set of all string over {a, b} of length '2'.

→ Σ = {a, b}.

language, L = {aa, ab, ba, bb}.

State transation Diagram -



- String accept → Scan the entire string, if we reach in a finial state from initial state. Then string will be accepted by FA.

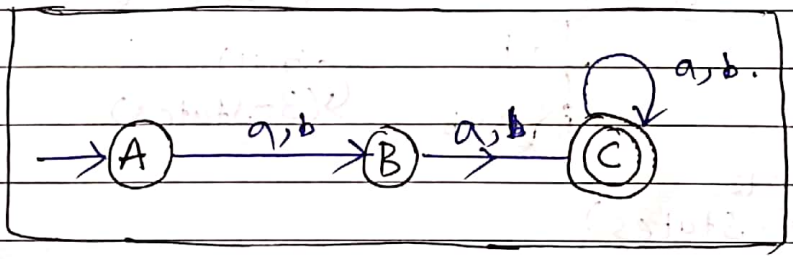
- language accept → A Finite Automata is said to accept a language if all the string in the language are accepted and the string not in the Language are rejected.

Ex-2

Construct a DFA which accepts all the String $\{a,b\}$ where the string length is ~~accept~~ atleast '2'. $|w| \geq 2$

→ $\Sigma = \{a,b\}$

$L = \{aa, ab, ba, bb, aaa, aba, baa, bba \ldots\ldots\ldots\}$



State transition Diagram,

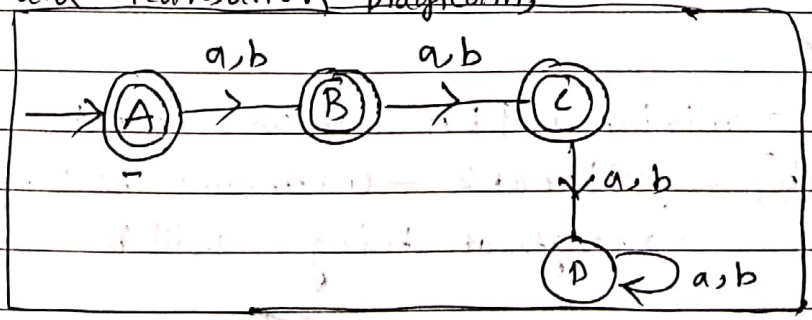
Ex-3

Construct a DFA, $\Sigma = \{a,b\}$, $w \in \{a,b\}$, $|w| \leq 2$.

→ $\Sigma = \{a,b\}$

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

State transition Diagram,



Ex-4 $w \rightarrow$ string

(*) $|w|=2$, $|w| \geq 2$, $|w| \leq 2$ .. (gate)

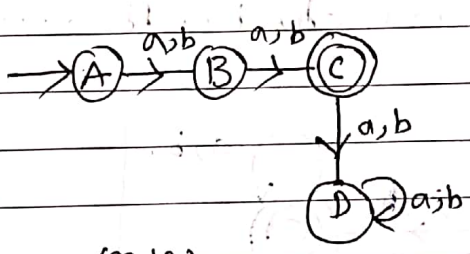
→ When, string length $|w|=n$, then
no of state required $(n+2)$.

When, $|w| \geq n$, then
no of state required $(n+1)$.

When, $|w| \leq n$, then
no of state required $(n+2)$

• $|w|=2$

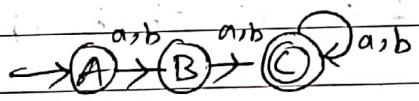
$L=\{aa, ab, ba, bb\}$



$(n+2)$
(4-states)

• $w \geq 2$

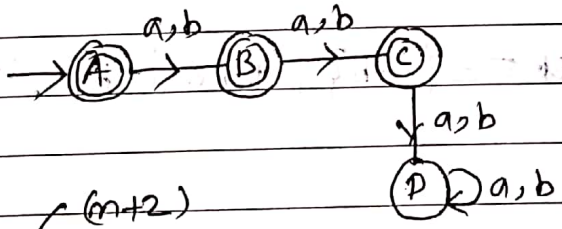
$L=\{aa, ab, aaa..\}$



$(n+1)$
(3-states)

• $|w| \leq 2$

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



$(n+2)$
(4-states)

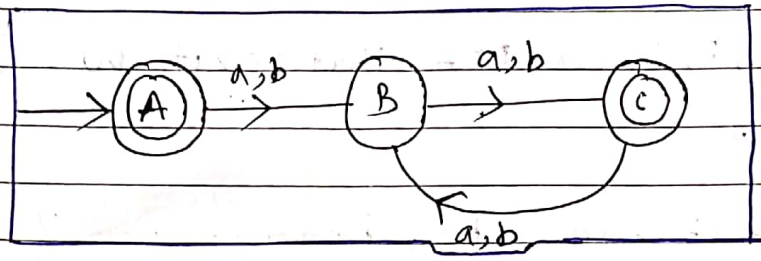
Example-4

Construct a minimal DFA which accept all string over $\{a,b\}$, $|w| \bmod 2 = 0$ (means the length of the string ↳ length of string will be even)

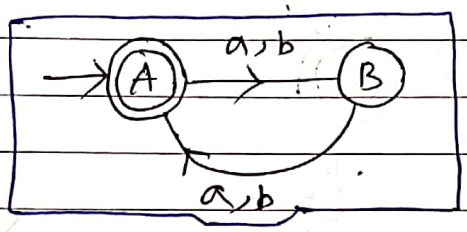
→

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

$L = \{\epsilon, aa, ab, ba, bb, aaaa, \dots, bbbb, \dots\}$



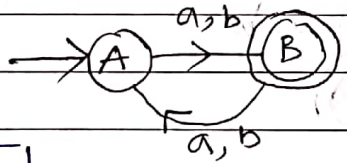
OR



(State transition Diagram)

Ex-5, $|w| \bmod 2 = 1$

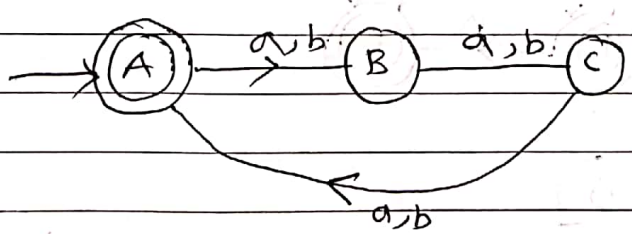
→ $L = \{a, b, aaa, bbb, \dots aaaaa, \dots\}$



Ex-6,

$|w| \bmod 3 = 0$

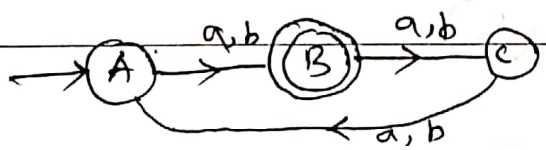
→ $L = aaa\{\epsilon, aaa, aba, bbb \dots\}$



finial state 'A'.

Ex-7, $|w| \bmod 3 = 0\,1$

→ $L = \{a, b, aaaa, bbbb, \dots\}$



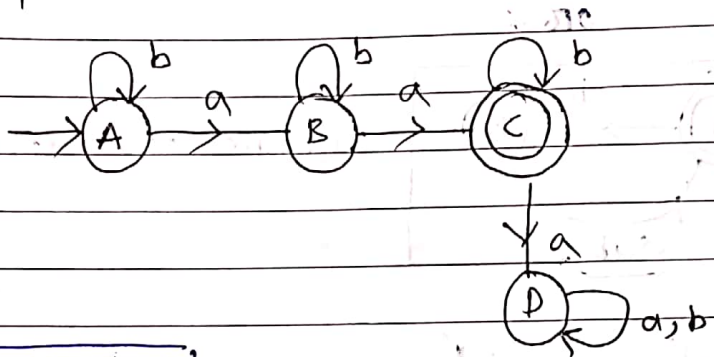
** If $|w| \bmod n = 0$, then no of states are 'n'.

Example - 8

Construct a minimal DFA, that accept $w \in \{a,b\}^*$

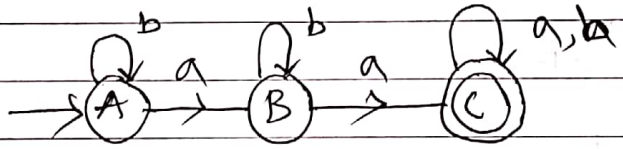
1. $n_a(w) = 2$

→ $L = \{aa, aab, baa, aba, bbaa \ldots\ldots\}$



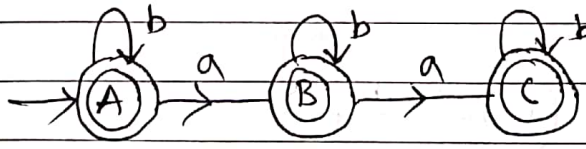
2. $n_a(w) \geq 2$

→ $L = \{aa, aaab, aaaaba \ldots\ldots\}$



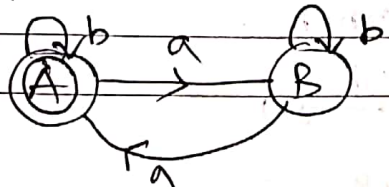
3. $n_a(w) \leq 2$

→ $L = \{\epsilon, a, b, aa, ab, ba, bb\}, babb, \ldots\ldots\}$



4. $n_a(w) \bmod 2 = 0$

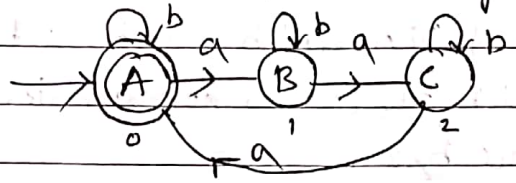
→ $L = \{aa, aab, aabb, aaaa, aaaaaa \ldots\ldots\}$



5. $n_a(w) \bmod 3 = 0$

→ $L = \{aaa, \dots aaaaaa \dots\}$

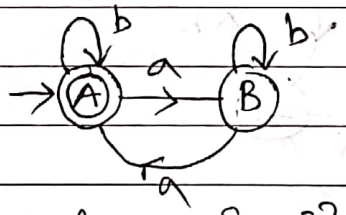
State transition diagram –



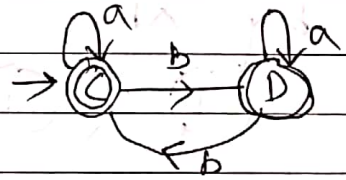
6. Construct a minimal DFA, where $w \in \{a,b\}^*$

$$n_a(w) \cong 0 \bmod 2$$
&&
$$n_b(w) \cong 0 \bmod 2$$

→ $L = \{\epsilon, aa, bb, aabb, abab, \sout{bb}aaaa, \dots bbbb \dots\}$



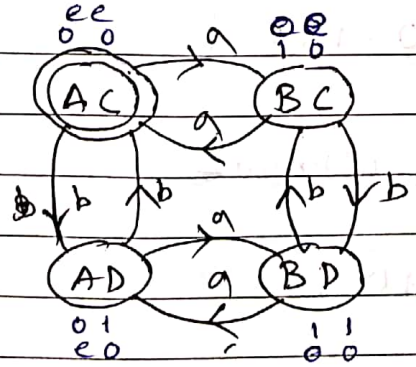
# of state {A, B}



# of state {C, D}

Cross product method –

$\{A,B\} \times \{C \times D\} = \{AC, AD, BC, BD\}$



$A \xrightarrow{a} B$	$A \xrightarrow{b} A$
$C \to C$	$C \to D$

e or e = {ee, eo, oe} ~~state~~ (finial state).

**

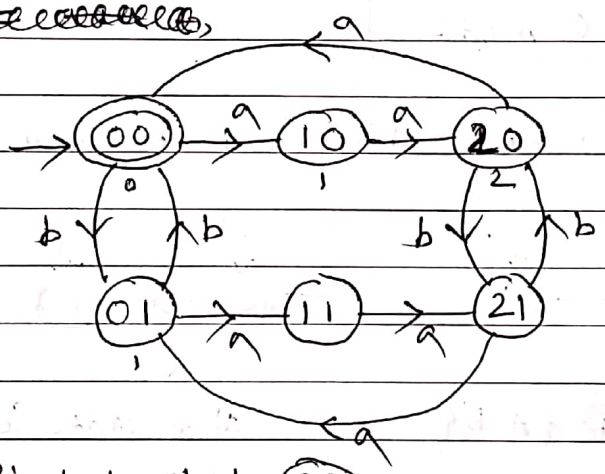
→ If one Automata contain 'n' state and the other automata contain 'm' number of states then there cross product going to contain, (m×n) states.

⑦ Construct a minimal DFA which accepts set of all strings over {a, b} in which no of a's are divisible by 3 and no of b's are divisible by 2.

→ $w \in \{a,b\}^*$

$n_a(w) \equiv 0 \bmod 3$

$n_b(w) \equiv 0 \bmod 2$



finial state (00)

When, $n_a(w) \bmod 3 > n_b(w) \bmod 2$

finial state = {10, 20, 11, 21}

When, $n_a(w) \bmod 3 = n_b(w) \bmod 2$

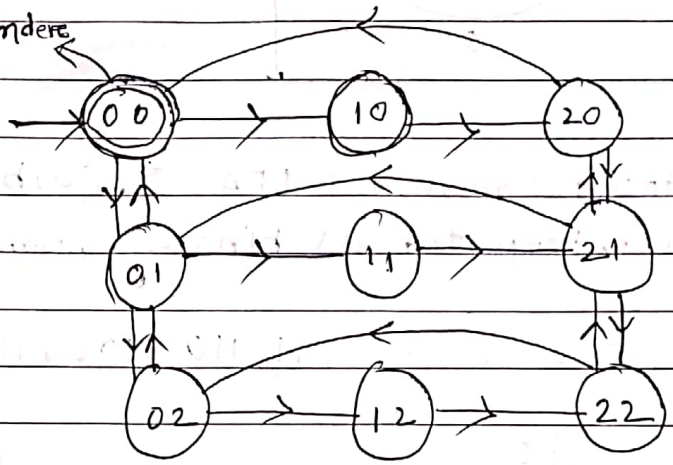
finial state = {00, 11}

⑧ Construct a minimal DFA, $w \in \{a,b\}^*$,

$n_a(w) \cong 0 \bmod 3$

$n_b(w) \cong 0 \bmod 3$

→ Remainders



When, $n_a(w) \bmod 3 = 1$

&&

$n_b(w) \bmod 3 = 2$, then

final state = $\{1, 2\}$

When, $n_a(w) \bmod 3 > n_b(w) \bmod 3$, then

final state = $\{10, 20, 21\}$

→ $n_a(w) \cong 0 \bmod n$

$n_b(w) \cong 0 \bmod m$

Then the minimum no of states in automata is '$m \times n$'.

Example-3] — ①

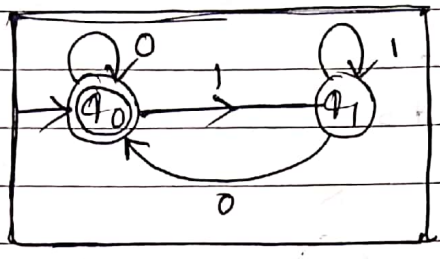
Construct a minimal DFA, which accepts set of all string over $\{0,1\}$, which when interpreted as binary number is divisible by '2'.

→ $\Sigma = \{0,1\}$

$w \in \{0,1\}^*$

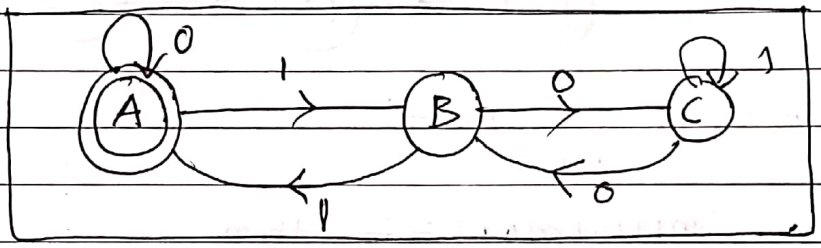
If we divided any no by 2 remainder can be '0' or '1'.

$L = \{0, 00, 000, 0000, \ldots, 10, 100, 110, \ldots\}$



(2) construct a minimal DFA, $\Sigma = \{a, b\}$, $w \in \{a, b\}^*$ when interpreted as binary number is divisible by 3.

→ $L = \{0, 00, 000 \ldots, 11, 110, 1100, 1111 \ldots\}$



→ Finite Automata can represent in two ways
- state ~~Diagram~~ transation Diagram.
- State transation table.

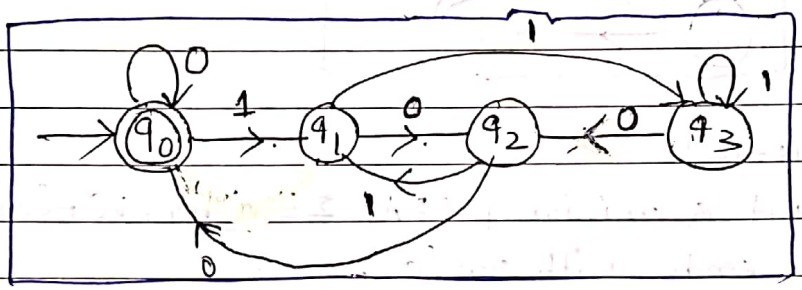
State transtion table,

	0	1
→*A	A	B
B	C	A
C	B	C

(3) Construct a minimal DFA, which accept all set of all string over {0,1} which when interpreted as a binary number is divisible by '4'.

→ $\Sigma = \{0, 1\}$, $\forall w \in \{0,1\}^*$

$L = \{0, 100, 000, \cdots, 100, 1000, 1100, 10000, 10100, \cdots\}$



State transaction table -

	0	1
* → q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

→ If the reminder is = 0
then FS → q_0
reminder is = 1
FS → q_1
reminder is = 2
FS → q_2
reminder is = 3
FS → q_3

Example - 10

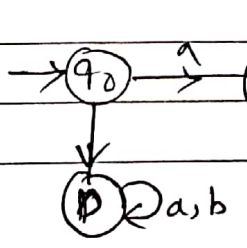
Construct a minimal DFA, which accepts set of all strings, over {a,b} where each string starts with an 'a'.

→ $\Sigma = \{a, b\}$

$w \in \{a, b\}^*$

language, $L_1 = \{a, aa, ab, aaa \ldots\}$

State transaction Diagram,

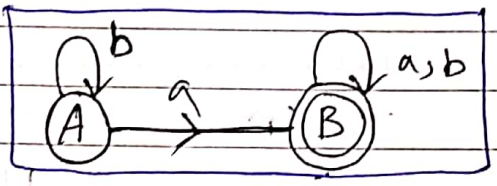


→ If the language is infinite then take the smallest string and make its skeleton.

Example-11

Construct a minimal DFA, $\Sigma = \{a, b\}$, $w \in \{a, b\}^*$ "where each string contains 'a'."

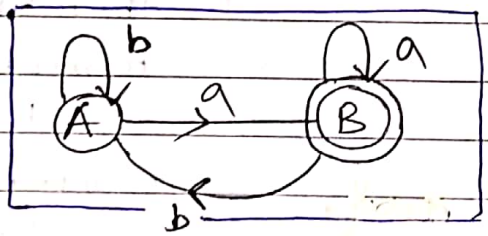
→ $L = \{a, aa, ab, ba, aaa, \ldots\}$



Example-12

Construct a minimal DFA, $\Sigma = \{a, b\}$, $w \in \{a, b\}^*$ String ends with an 'a'.

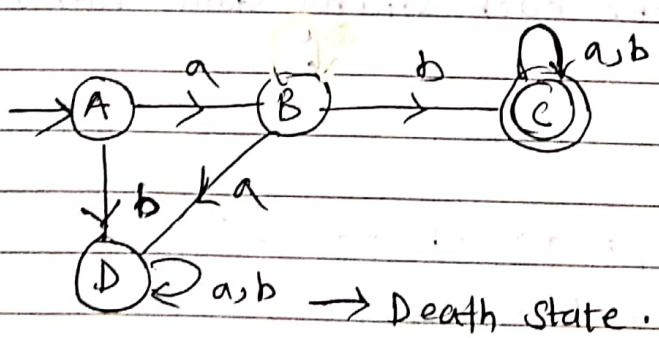
→ $L = \{a, aa, ba, aaa, bba, baa, bbba \ldots\}$



Example-13

Construct a minimal DFA, $\Sigma = \{a, b\}$, $w \in \{a, b\}^*$ "each string start with 'ab'".

→ $L = \{ab, aab, abb, \ldots\}$



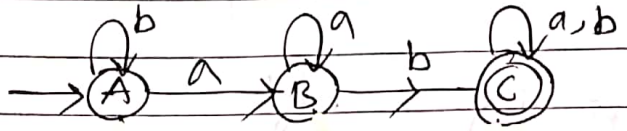
→ Death State.

Example – 14

DFA, containing ab.

→ Let $\Sigma = \{a, b\}$, $w \in \{a, b\}^*$

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

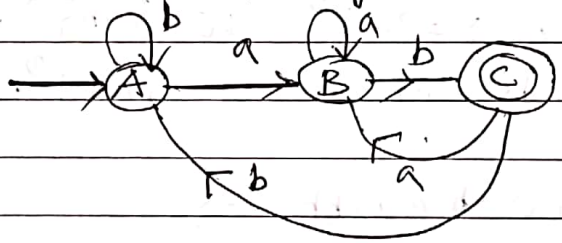


Example – 15

Construct a minimal DFA, ends with "ab".

→ $L = \{ab, bbab, aaab \ldots\}$

State transition diagram–

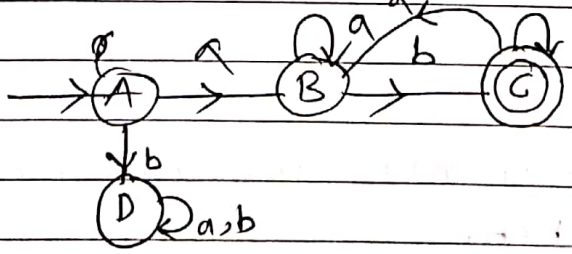


Ex – 16

Construct a minimum DFA, starts with 'a' and ends with 'b'.

→ $\Sigma = \{a, b\}$, $w \in \{a, b\}^*$

$L = \{ab, aabb, abab, abbab \ldots\}$

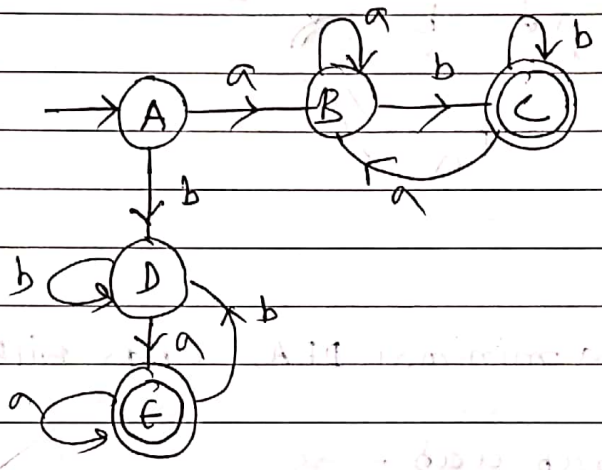


Ex-17

Construct a DFA, $\Sigma=\{a,b\}$, $w\in\{a,b\}^*$
" start and ends with different symbol".

→ $\Sigma=\{a,b\}$, $w\in\{a,b\}^*$.

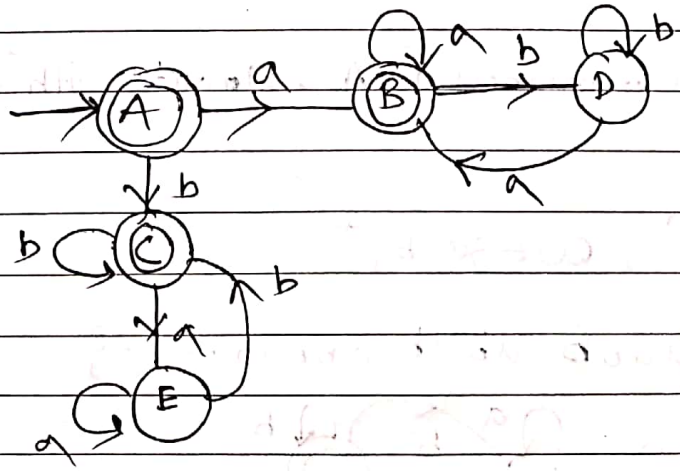
$L=\{ab, ba, abb, bba \ldots\}$



Ex-18

Construct a DFA, $\Sigma=\{a,b\}$, $w\in\{a,b\}^*$.
" start and ends with same symbol".

→ $L=\{\epsilon, a, b, aa, bb \ldots\}$



→ here, Example – 17 and 18 are compliment of each other.

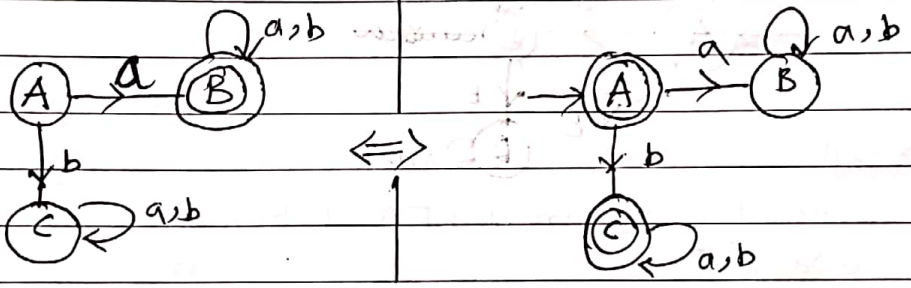
Ex-18

→ L_1 and L_2 are 2-language, then L_1 is said complement of L_2,

$$L_1 = \Sigma^* - L_2$$

- Complementation of DFA :

L_1 = {Starting with 'a'} | L_2 = {not starting with 'a'}



$$\overline{L_1} = L_2$$

→ Complimentation methode apply for only DFA not for NFA.

→ Every thing has to be same only ~~giving~~ finial state will be changed ^.

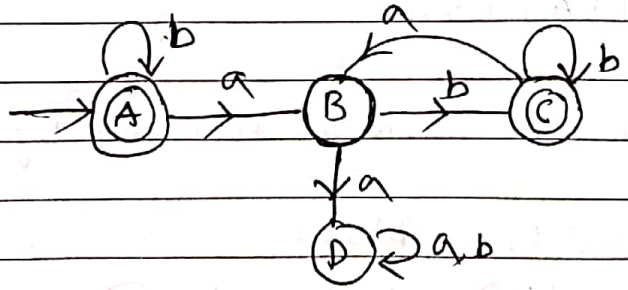
Ex-19

Construct a DFA, $w \in \{a,b\}^*$, P

Every 'a' should be followed by a 'b'.

→ $L = \{\epsilon, ab, abb, abab, babb, \cdots, b, bbb, \cdots\}$

State transcation Diagram -

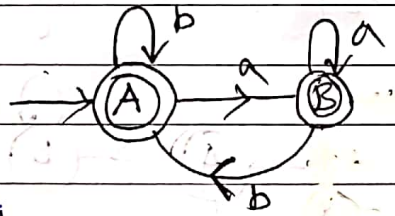


Ex-20

Construct a DFA, where, $w \in \{a,b\}^*$
Every 'a' should not never followed by a 'b'.

→ $L = \{\epsilon, a, b, aa, ba, bb, aaa, bba, \ldots\}$

State transation Diagram-

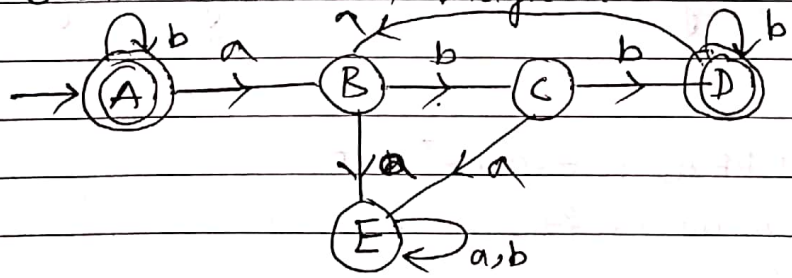


Ex-21

construct a minimal DFA, where
$w \in \{a,b\}^*$
every 'a' should be followed by 'bb'.

→ $L = \{\epsilon, abb, \ldots\}$

State transation Diagram-

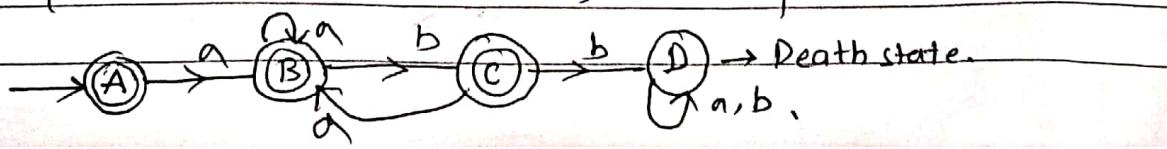


final state = {A, D}.

Ex-22

Construct a DFA, where.
$w \in \{a,b\}$
every 'a' should never be followed by a 'bb'.

→ $L = \{\epsilon, a, aa, ab, ba, bb, aaa, \ldots\}$

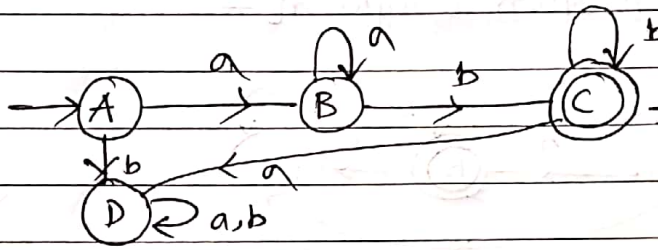


Ex-23

Construct a minimal DFA which accepts, $L = \{a^n b^m / n, m \geq 1\}$

→ $L = \{ab, aab, aabb, aabb \cdots\}$

State transcation diagram-

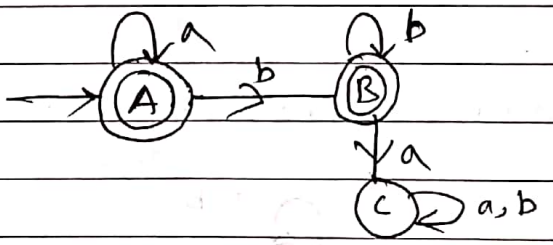


Ex-24

Construct a minimal DFA, which accepts $L = \{a^n, b^m / n, m \geq 0\}$

→ $L = \{\epsilon, a, aa, aaa, \cdots, b, bb, bbb \cdots, ab, aabb \cdots\}$

State transcation Diagram-

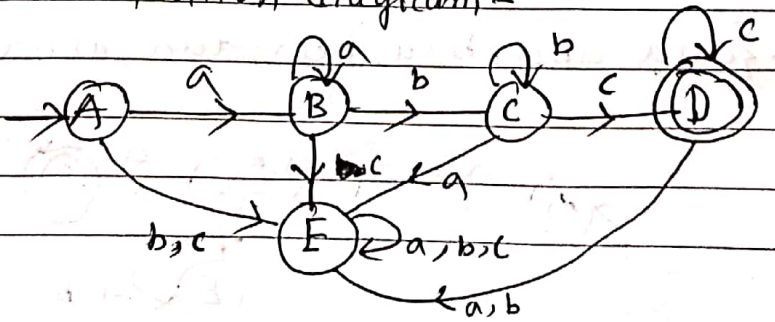


Ex-25

Constructs a minimal DFA, which accepts $L = \{a^n b^m c^l / n, m, l \geq 1\}$

→ $L = \{abc, aabbcc \cdots\}$

State transcation diagram-

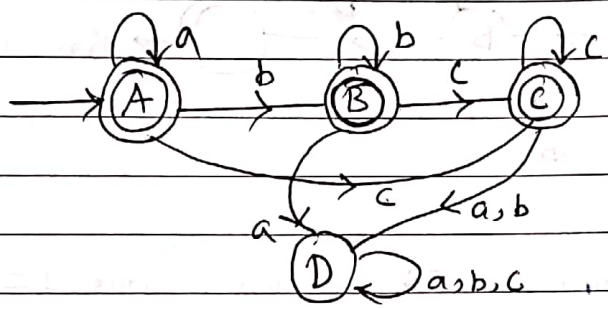


Ex-26

Construct a minimal DFA, which accepts,
$L = \{a^n b^m c^l / n, m, l \geq 0\}$.

→ $L = \{\epsilon, a, aa \cdots, b, bb \cdots, c, cc, \cdots, abc, aabbcc\}$

State transition Diagram –

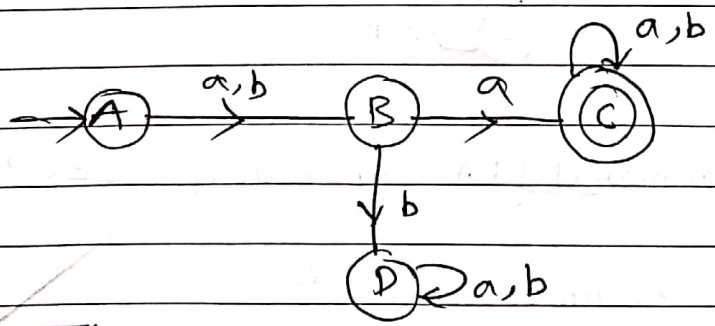


Ex-27

Construct a minimal DFA which accepts set of all strings over $\{a,b\}$ such that second symbol from L.H.S is 'a'.

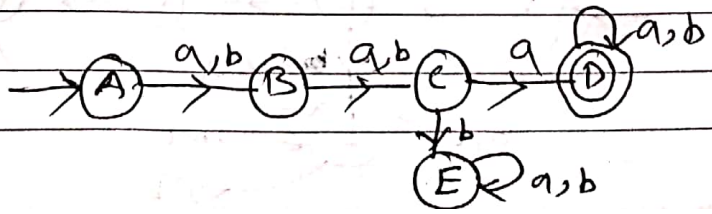
→ $\Sigma = \{a,b\}$

$L = \{aa, ba, aaa, baa, bba, \cdots\cdots\}$



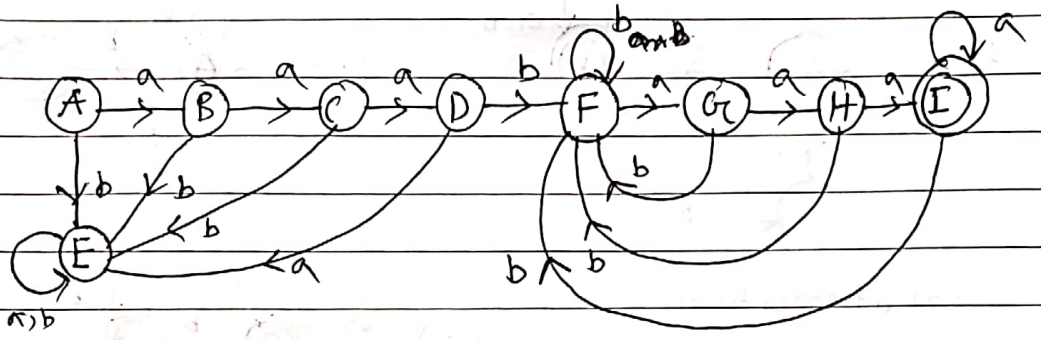
Ex-28 3rd symbol from L.H.S is a.

→ $L = \{aaa, aba, baa, bba, aaa, abaa \cdots\}$



Ex-29 construct a DFA which accepts set of all string over $\{a,b\}$ where strings are of the form 'a^3bwa^3' where 'w' is any string over $\{a,b\}$.

→ $L = \{a^3b\in a^3, a^3b\,a\,a^3, a^3b\,b\,a^3, \ldots\}$



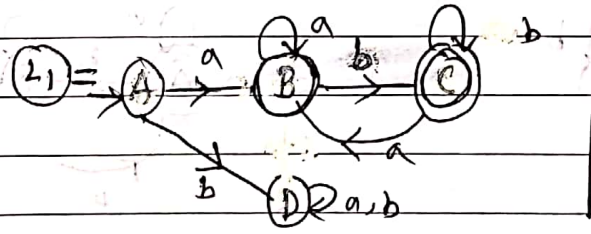
- OPERATION ON DFA like –
  (I) Union (II) Concatenation (III) cross product (IV) Complementation. (V) Reversal.

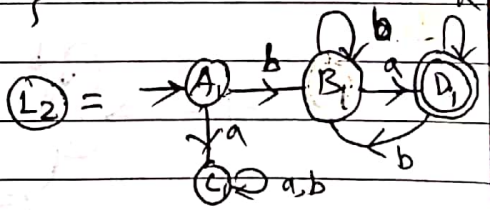
① Union :-

Ex – starts and ends with different symbol.

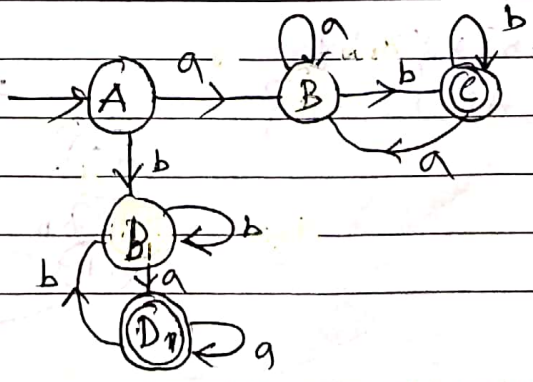
→ $L_1 = \{ab, aab, aba, abb, \ldots\}$

$L_2 = \{ba, baa, bba, \ldots\}$





$L_1 \cup L_2$ =

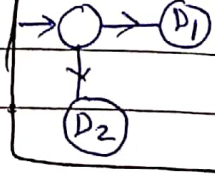


**

$L_1 = D_1$

$L_2 = D_2$

$L_1 \cup L_2$



② Concatenation — $L_1 = D_1$, $L_2 = D_2$ / $L_1 \cdot L_2 = D_1 \cdot D_2$.

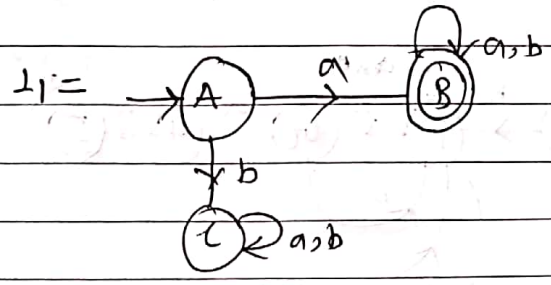
→ starting with 'a' and ending with 'b'.

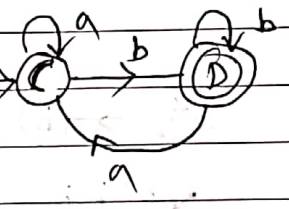
(starting with a)

$L_1 = \{a, aa, ab, aaa \cdots\}$

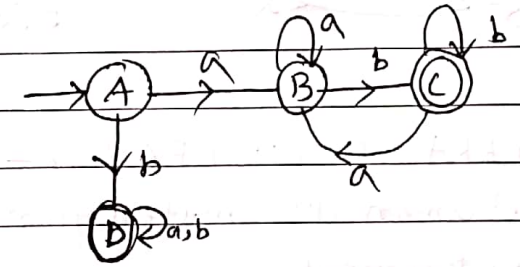
Ending with b)

$L_2 = \{b, ab, bb, aab, bab, bbb, \cdots\}$

$L_1 =$ 

$L_2 =$ 

Concatenation,

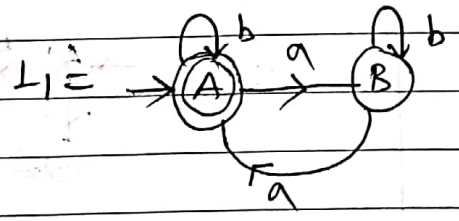
$L_1 \cdot L_2 =$ 

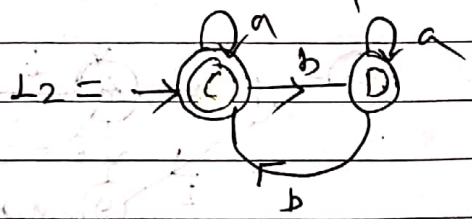
③ Cross product methode —

→ even no of 'a's and even num of b's.

$L_1 = \{\epsilon, aa, aab, aaaa, aaaaaa \cdots\}$

$L_2 = \{\epsilon, bb, bba, bbbb, bbbbbb \cdots\}$

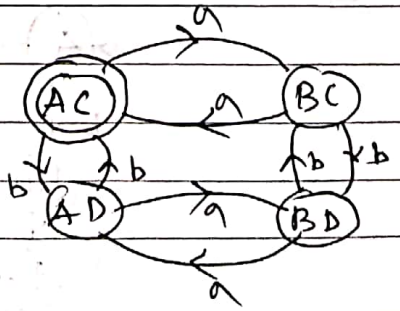
$L_1 =$ 

$L_2 =$ 

$L_1 \times L_2$

$= \{A, B\} \times \{C, D\}$

$= \{AC, AD, BC, BD\}$



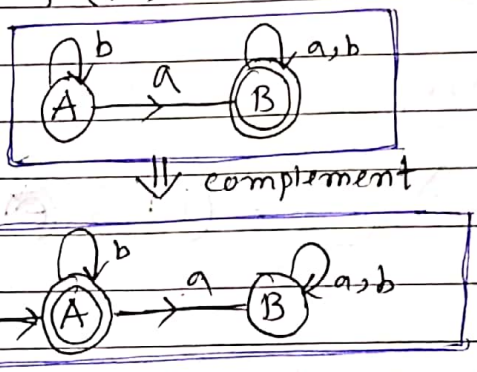
④ Complement :-

→ Does not contain 'a'.

$L_1 = \{\text{containing 'a'}\}$

$= \{a, aa, ab, ba, aaa \cdots\}$

$\bar{L}_1 = \{\epsilon, b, bb, bbb, bbbb \cdots\}$



⑤ Reversal :

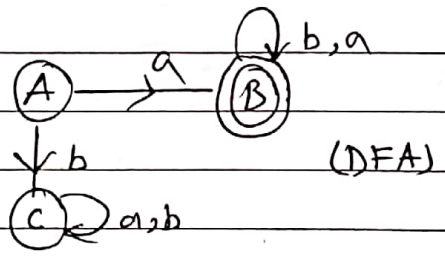
→ $aWb \Leftrightarrow bWa$.

$L_1 = \{\text{start with a}\}$

$= \{a, aa, ab, aaa, aba, aaaa \cdots\}$

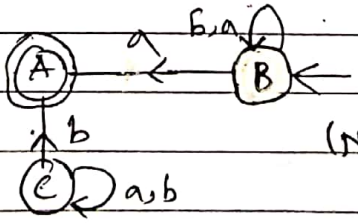
$L_1^R = \{a, aa, ba, aaa, aba, aaaa \cdots\}$

$L_1 =$



(DFA)

$L_1^R =$



(NFA)

**

$L_1 \rightarrow$ DFA $\rightarrow (DFA)^R \rightarrow L_1^R \rightarrow$ NFA / DFA

(30) Ex-30

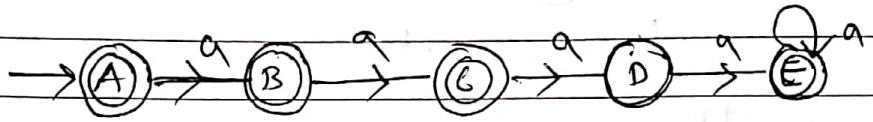
Construct a minimal DFA over $\{a\}$.

1. For $\{a^n / n \geq 0, n! = 3\}$
2. For $\{a^n / n \geq 0, n! = 2, n! = 4\}$.

→ $\Sigma = \{a\}$

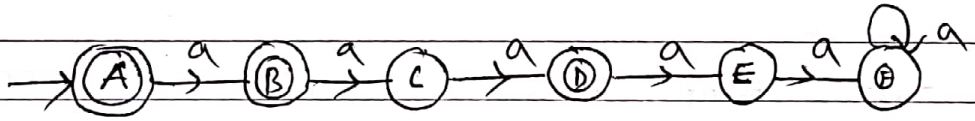
(1) $L = \{a^n / n \geq 0, n! = 3\}$

$L = \{\epsilon, a, aa, aaaa, \cdots\}$



(2) For $L = \{a^n / n \geq 0, n! = 2, n! = 4\}$

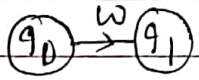
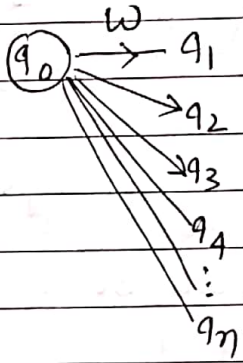
$= \{\epsilon, a, aaa, aaaaa, \cdots\}$



→ In NFA, not need to show death state.

NFA (Non-Deterministic Finite Automata):

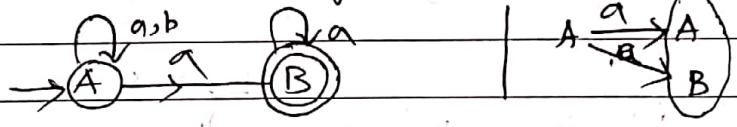
→ Every DFA is NFA.

DFA	NFA
$\delta: Q \times \Sigma \to Q$	$\delta: Q \times \Sigma \to 2^Q$
	

Ex-1 Construct a NFA, ~~where~~ which accepts set of all string over $\{a,b\}$ such that $L = \{$ ends with 'a' $\}$

→ $L = \{ a, aa, ba, aaa, baa \ldots \}$

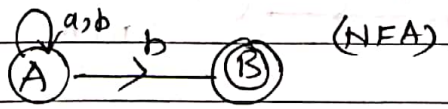
State transation Diagram,



→ [The string 'a' is accepted by NFA, if start with initial state and ~~end~~ if reach at least 1 state is final]

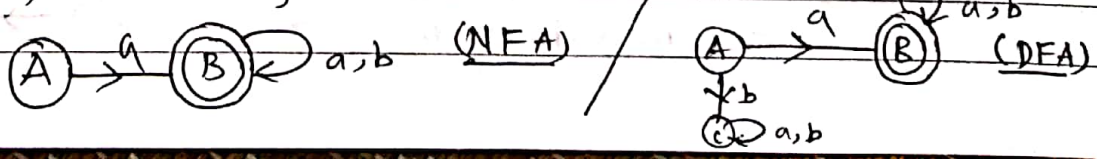
Ex-2 $L = \{$ ending with b $\}$

→ $L = \{b, ab, abb \ldots\}$



Ex-3 L = ~~Starting~~ {String starts with 'a'}

→ $L = \{a, aa, ab \ldots\}$

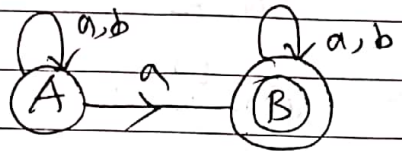


bababa

Ex-3

L = { set of all strings containing 'a' }.

→ $L = \{a, aa, ab, ba, aaa, aab ----\}$

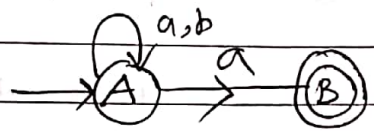


(NFA)

Ex-4

L = { set of all strings ends with 'a' }.

→ $L = \{a, aa, ba, \ldots\}$

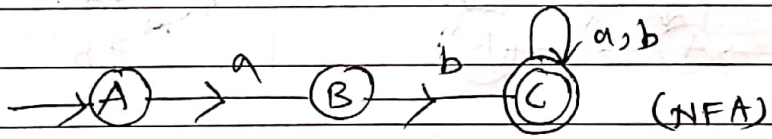


(NFA)

Ex-5

L = { all string ~~Start~~ with 'ab' }

→ $L = \{ab, abb, abab, \ldots\}$

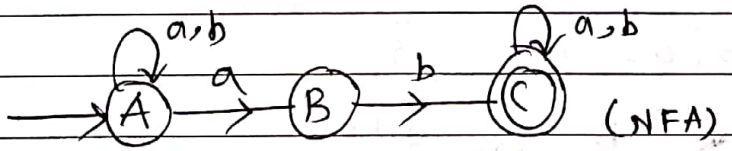


(NFA)

Ex-6

L = { set of all strings contains 'ab' }

→ $L = \{ab, abb, aabb ----\}$

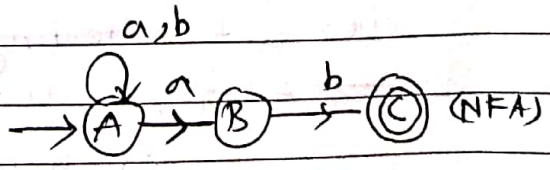


(NFA)

ex-7

L = { ending with 'ab' }

→ $L = \{ab, aab, bab, \ldots\}$

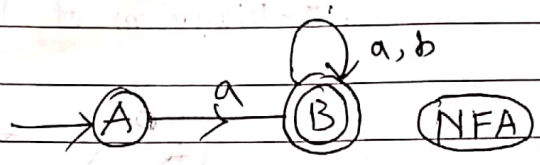


(NFA)

- CONVERSION of NFA to DFA for the Example (1) "all strings start with a".

→ L = { a, aa, ab, aab, aba, ... }

State transition Diagram of NFA.



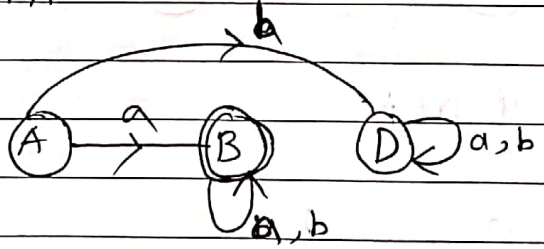
STT of NFA –

	a	b
→ A	B	∅
* B	B	B

STT of DFA –

	a	b
→ A	B	D
* B	B	B
D	D	D

STD of DFA –



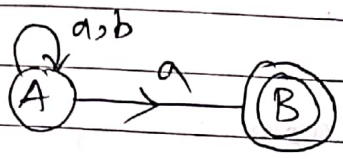
Example-2

Conversion of NFA to DFA :

L = "all strings ends with a".

→ { a, aa, abba, aaa, aba, ... }

ST-Diagram of NFA-



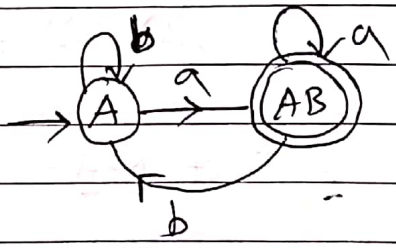
ST-table of NFA

	a	b
A	{A,B}	{A}
B	{∅}	{∅}

ST-Table of DFA

	a	b
→ A	[AB]	[A]
*[AB]	[AB]	[A]

ST-Diagram of DFA



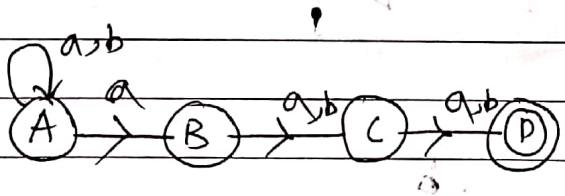
EX-2)

Conversion of NFA to DFA,

$\perp$ = { all strings in which third symbol from R.H.S is 'a' }

→ $\perp$ = {aaa, abb, aba, baaa, ----}

ST-D of NFA



ST-Table of NFA

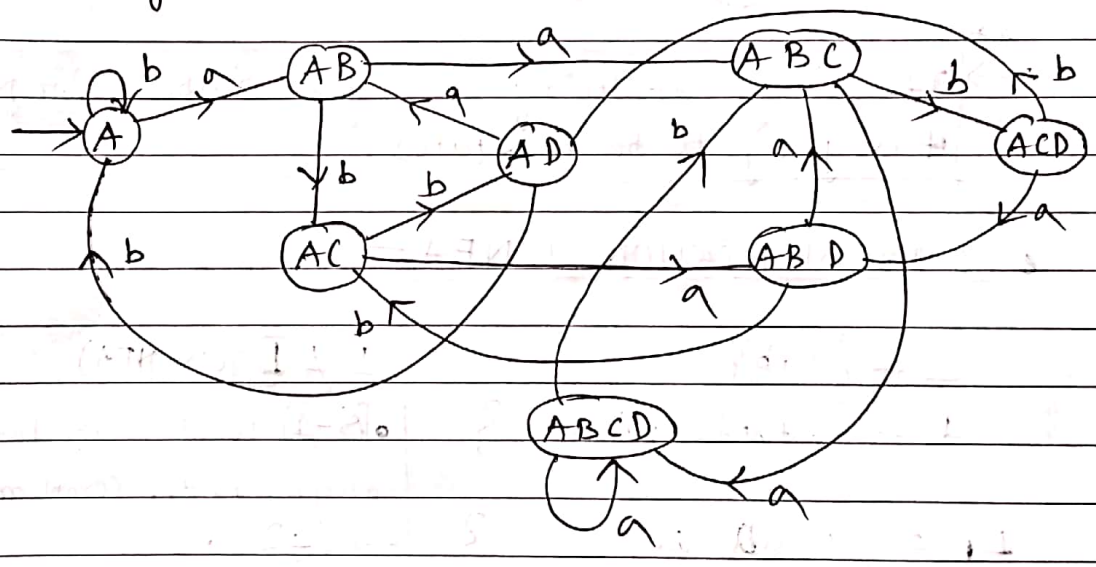
	a	b
→ A	{A,B}	{A}
B	{C}	{C}
C	{D}	{D}
* D	{∅}	{∅}

✓T-Table of DFA from ST of NFA -

	a	b
→ [A]	[A, B]'	[A]'
[A B]	[ABC]'	[AC]'
[AC]	[ABD]'	[AD]'
* [AD]	[AB]'	[A]'
[ABC]	[ABCD]	[ACD]'
*[ABD]	[ABC]'	[AC]'
*[ACD]	[ABD]'	[AD]'
*[ABCD]	[ABCD]'	[ACD]'

any state contain D

ST-Diagram of DFA from above TT of DFA -

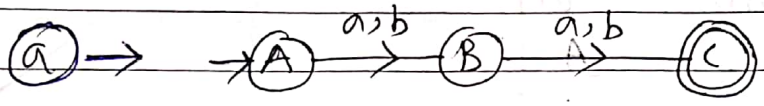


*** → If an minimal NFA Contain 'n' states then an DFA Contain 2^n states in worst case.

$n \longrightarrow 2^n$

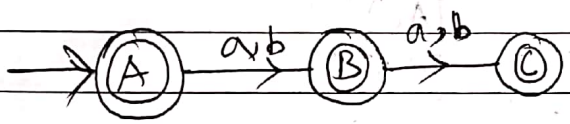
• Ex- NFA for string of length –

a) exactly 2. b) at most 2 c) at least 2

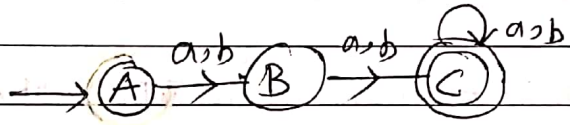


$L = \{aa, ab, ba, bb\}$

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



c → $L = \{aa, ab, ba, bb, aaa, \ldots\}$



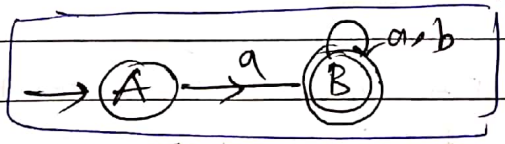
** → If we have ~~a~~ string of length 'n' then for NFA it is going to be 'n' states.

• COMPLEMENTATION of NFA –

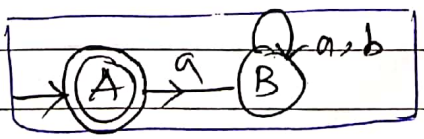
$\Sigma = \{a, b\}$

$L = \{$ starts with a $\}$

$L_1 = \{a, aa, ab, \ldots\}$



Then $\overline{L} =$

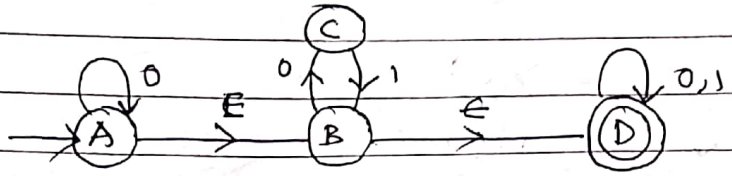


$L \neq \overline{L}$ (in NFA)

• Q-1 What is the language accepted by the complement of NFA.
→ $\{\epsilon\}$.

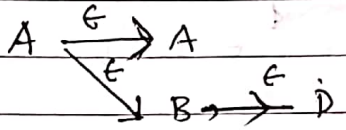
• Q-2 What is the complement of language accepted by NFA.
→ $\{\epsilon, b, bb, bbb, \ldots, ba\}$.

- **Epsilon NFA :-** (ϵ-NFA)



→ For ϵ-NFA : $Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$

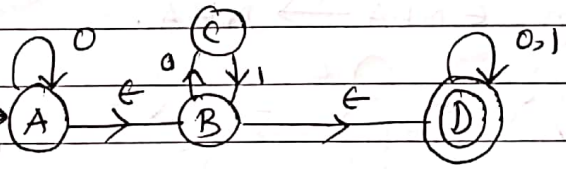
→ ϵ-closure (A) = {A, B, D}



Ex-1

- **Conversion ϵ-NFA to NFA :**

State transition Diagram of ϵ-NFA

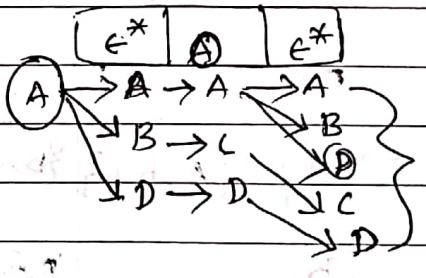


⇓ conversion NFA

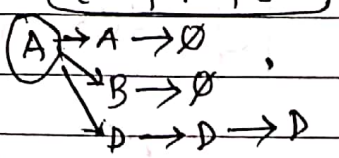
State transition table of NFA :

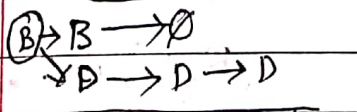
	0	1
A	{A, B, C, D}	{D}
B	{C, D}	{D}
C	{∅}	{B, D}
D	{D}	{D}

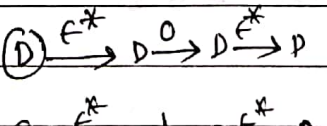
ϵ^*	1	ϵ^*



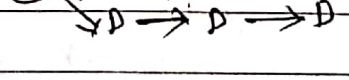
ϵ^*	1	ϵ^*

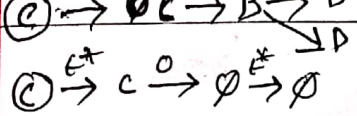




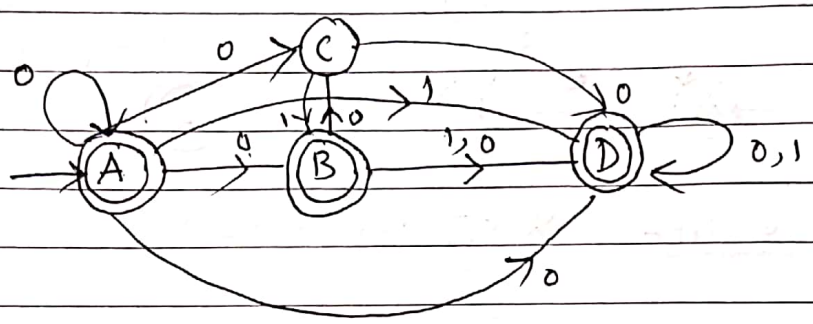


B	ϵ^*	0	ϵ^*





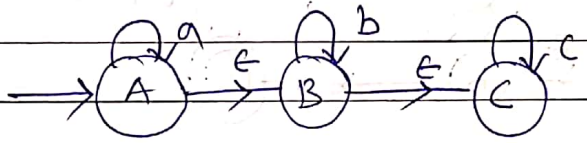
from the state ~~table~~ transation Diagram - ~~no~~



→ 'A' is going to 'D' by seeing 'ϵ' so 'A' will be the finial state.
→ 'B' is going to 'D' " " 'ϵ' so 'B' " " " .

EX-2

conversion ϵ NFA ⟶ NFA.

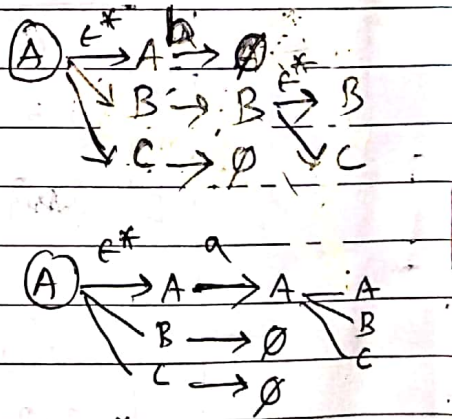


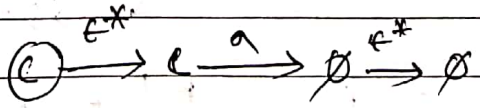
(ϵ NFA)

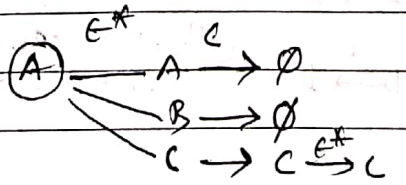
↓ conversion.

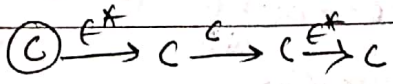
transation table :-

		a	b	c
→	A	{A, B, C}	{B, C}	{C}
	B	{∅}	{B, C}	{C}
	C	{∅}	{∅}	{C}

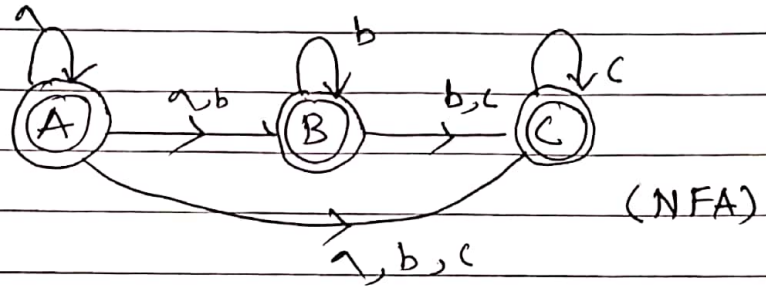








State transation Diagram ~~for~~ from state transation table:



(NFA)

→ all the DFA, NFA and ϵ-NFA are equal in power.

Minimisation of DFA:

** Two state called equivalent,

(P,q) equivalent,

$\delta(P,w) \in F$

$\Rightarrow \delta(q,w) \in F$

OR

$\delta(P,w) \notin F$

$\Rightarrow \delta(q,w) \notin F$

When,

Length of string $|w|=0$, 0 equivalent.

$|w|=1$, 1 equi.

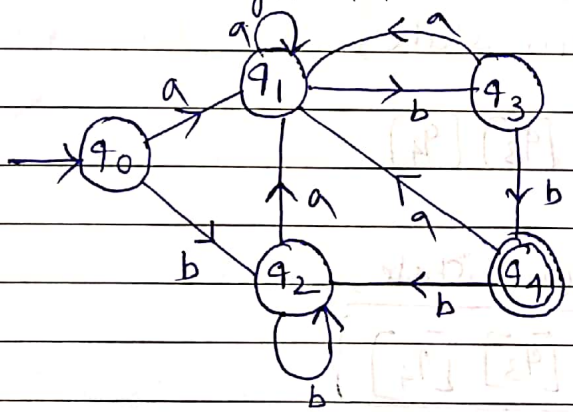
$|w|=2$, 2 equi.

⋮

$|w|=n$, n equivalent.

Example-1

Minimise the given DFA-



→ Step-1: Identify the Initial state and final state.

Step-2: Delete all the state that not reachable from initial state. (in final stat)

Step-3: Draw state transaction table.