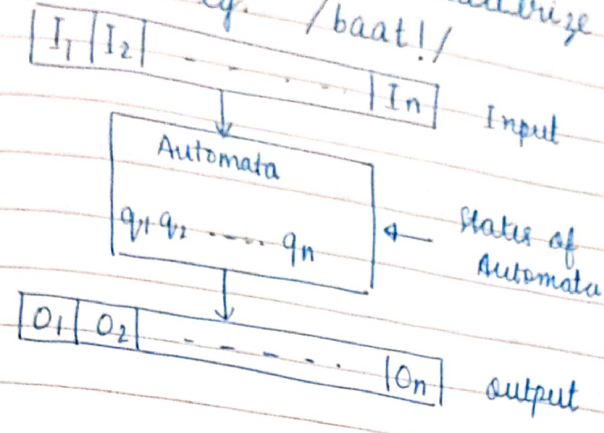
UNIT-I

Introduction to Finite Automata

Finite Automata is the simplest machine to recognize patterns. It is used to characterize regular language. For eg: /baat!/



Symbols

0, 1, a, b, A, ...

Alphabet

Σ → finite set of symbols

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

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

String

Finite sequence of symbols

eg: 1010

apple

lppl

Language

Set of strings

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

L = set of string of length 2

$= \{00, 01, 10, 11\}$

Power of an alphabet (Σ^k)

k is the length of string

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

$\Sigma^1 = \{a, b, c\}$

$\Sigma^2 = \{ab, bc, ac, ba, cb, ca\}$

It is a set of strings where k is the length of strings.

Deterministic Finite Automata (DFA)

DFA consists of 5 tuples $(Q, \Sigma, \delta, q_0, F)$

i) $Q \rightarrow$ finite set of states

ii) $\Sigma \rightarrow$ finite set of input symbols

iii) $\delta \rightarrow$ Transition function that takes a state and input symbol as arguments and returns another state.

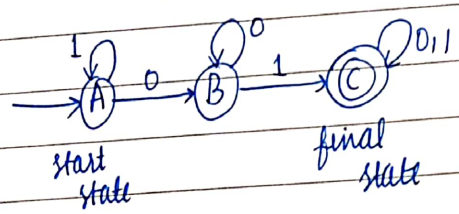
$\delta(p, a) = q$ where p is a state
a is a symbol

iv) $q_0 \rightarrow$ start state

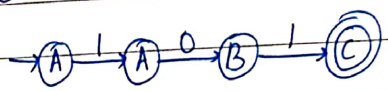
v) $F \rightarrow$ set of final or accepting state

① Design a DFA which accepts all strings with a substring $\{0, 1\}$

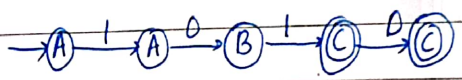
$\Sigma = \{0, 1\}$



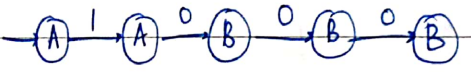
* 101 → valid



* 1010 → valid



* 1000 → invalid



→ not a final state so invalid

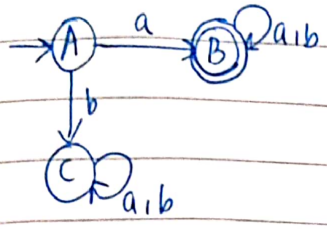
$Q = \{A, B, C\}$
$\Sigma = \{0, 1\}$
$q_0 = A$
$F = \{C\}$

δ	0	1
A	B	A
B	B	C
C	C	C

Transition function

② Construct a DFA over $\{a, b\}$ which accepts languages for all string starting with a.

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

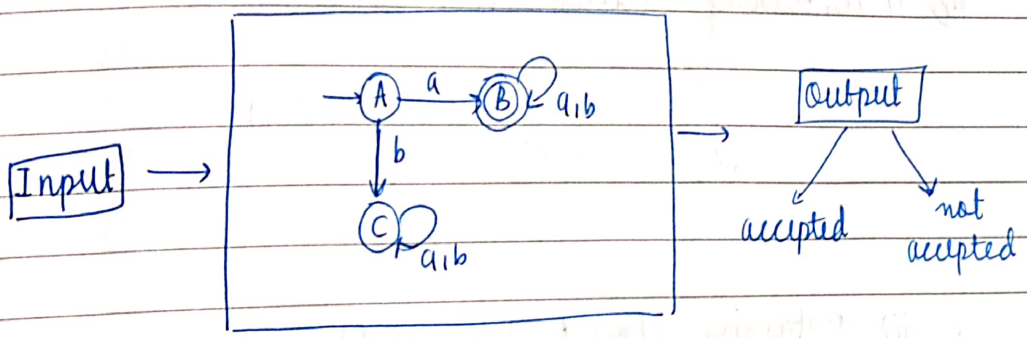


δ	a	b
A	B	C
B	B	B
C	C	C

Trap state

16/04/24

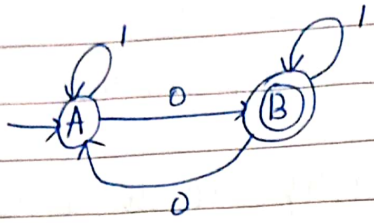
Graphical representation of DFA



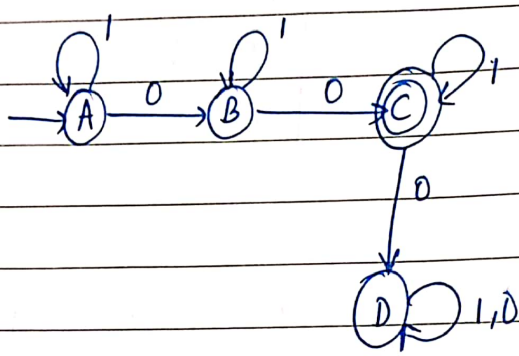
FSM (Finite State Machine)

- DFA - Deterministic Finite Automata
  - 5 tuples
  - graphically
  - computation is done mechanically
- NFA - Nondeterministic Finite Automata
  - 5 tuples

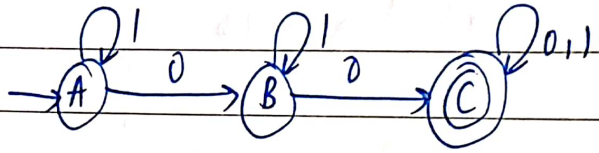
(3) Design DFA with $\Sigma = \{0,1\}$ accepting strings containing odd no. of zeros



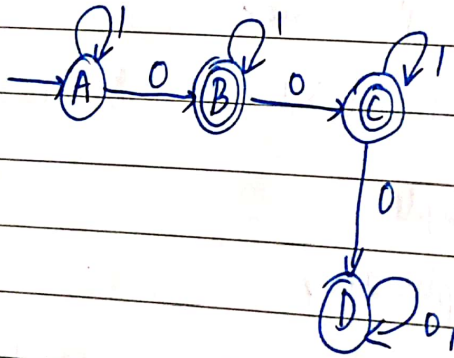
(4) i) Design a DFA over (0,1) accepting strings containing exactly two zeros.



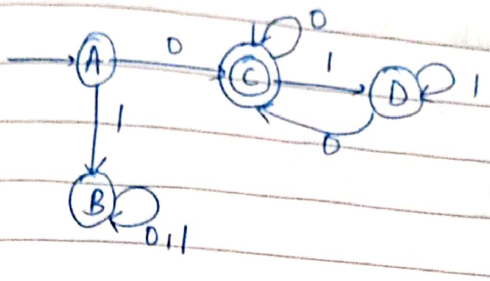
(5) ii) containing atleast two zeros.



6 iii) containing atmost two zeros

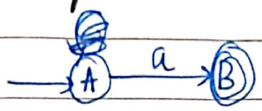


iv) Strings starting and ending with 0

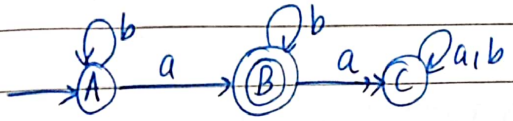


note:
1 I/P ⟶ 2 states
2 I/P ⟶ 3 states
n I/P ⟶ n+1 states

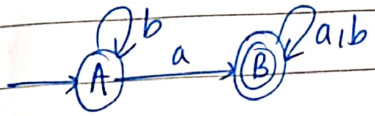
v) exactly one a



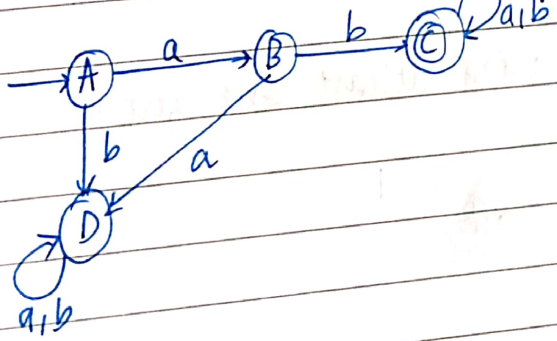
vi) exactly one a on $\Sigma = \{a, b\}$



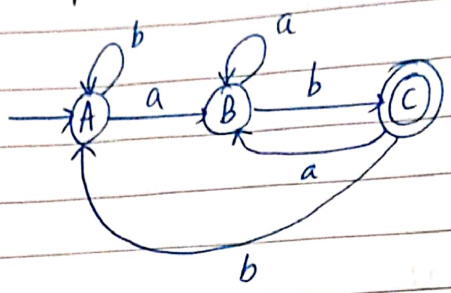
vii) atleast one a



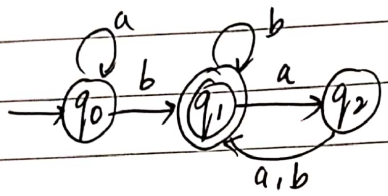
⑤ i) Obtain DFA to accept strings starting with ab. Accept strings of a's and b's.



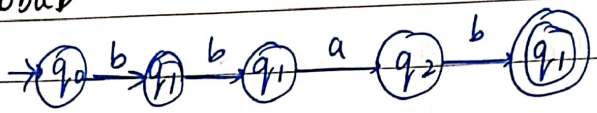
ii) ending with ab



Q) Does the DFA accept string bbab?

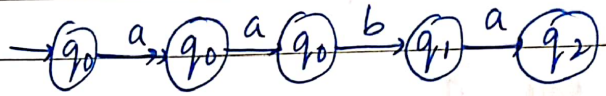


i) bbab



Yes

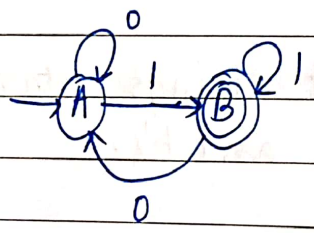
ii) aaba



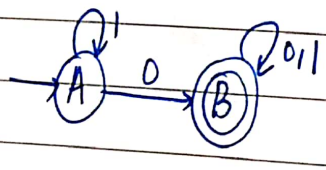
NO

Q) DFA ending with 1.

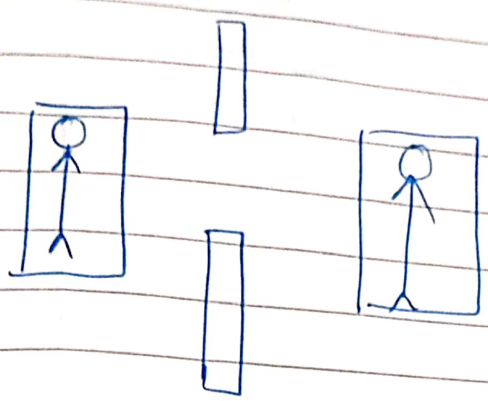
$\Sigma = \{0, 1\}$



Q) DFA that contains atleast one zero.

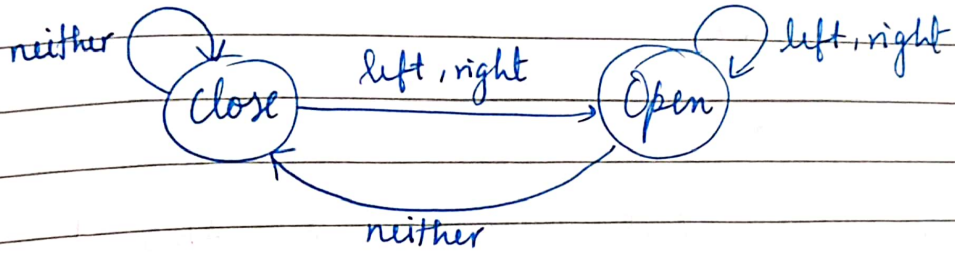


Q) Design the logic behind automatic doors in Walmart.

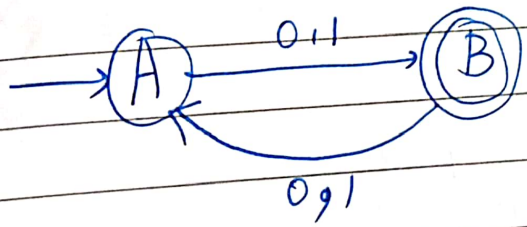


Walmart

States = { open, close }
Input = { left, right, neither }



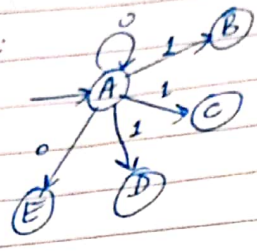
Q) DFA for binary strings of odd length.



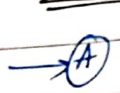
Non-Deterministic Finite Automata (NFA)

DFA	NFA
• Only one next state	• Multiple next states
• doesn't accept null strings (ϵ)	• accepts null strings (ϵ)

Eg:

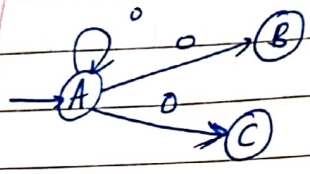


2 states



I/P : 0 — A, B, AB, ϕ — 4 next states

3 states



I/P: 0 — A, B, C, ~~AB~~, BC, AC, ~~ABC~~, ϕ — 8 next states

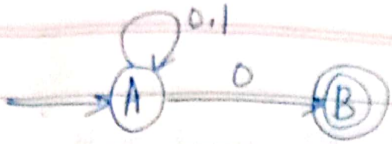
NFA

It consists of 5 tuples $(Q, \Sigma, q_0, F, \delta)$

i) Q - set of states
ii) Σ - set of ~~for~~ inputs
iii) q_0 - start state
iv) F - set of final states
v) δ - transition function

$$\delta = Q \times \Sigma \rightarrow 2^Q$$

sign NFA with inputs $\Sigma = \{0, 1\}$ that epts all strings that end with zero.



$Q = \{A, B\}$
$\Sigma = \{0, 1\}$
$q_0 = A$
$F = \{B\}$

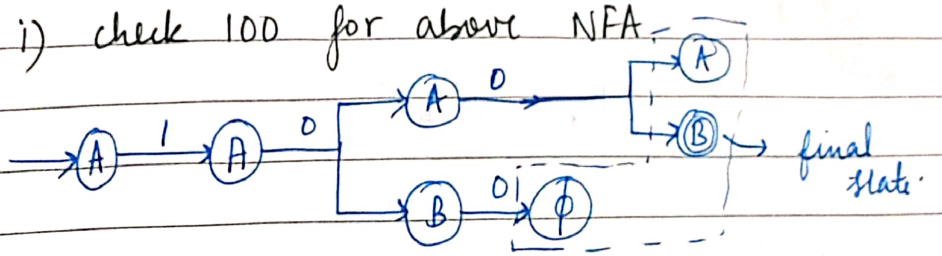
$\delta = Q \times \Sigma \longrightarrow 2^2 = 4$

$A \times 0 \longrightarrow A$
$A \times 0 \longrightarrow B$
$A \times 1 \longrightarrow A$
$B \times 0 \longrightarrow \phi$
$B \times 1 \longrightarrow \phi$

Transition table

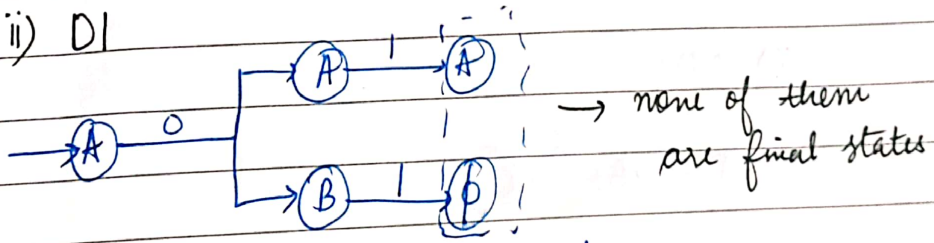
δ	0	1
A	{A, B}	A
B	ϕ	ϕ

Eg: i) check 100 for above NFA:-



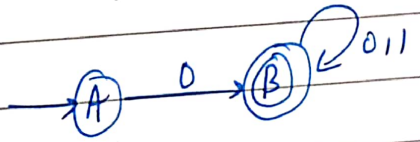
O/P : Input is accepted

ii) 01

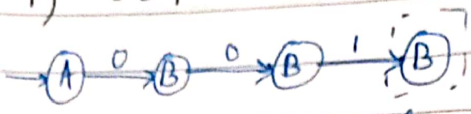


O/P : Input is not accepted

(3) L = { set of all strings that start with zero }
$\Sigma = \{0, 1\}$



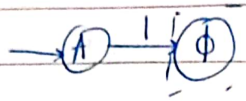
eg: i) 001



final state

o/p: Input is accepted

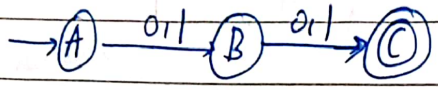
ii) 101



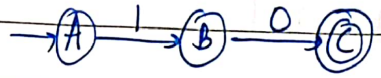
not final state

o/p: Input is not accepted

(A) L = {set of all strings over (0,1) of length 2}

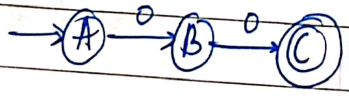


eg: i) 10



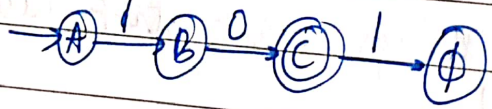
accepted

ii) 00



accepted

iii) 1011



not accepted

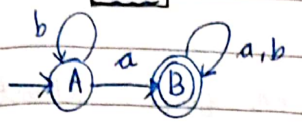
22/9/24 **Convert NFA to DFA**

DFA = $(Q, \Sigma, q_0, F, \delta) \Rightarrow \delta = Q \times \Sigma \to Q$.

NFA = $(Q, \Sigma, q_0, F, \delta) \Rightarrow \delta = Q \times \Sigma \to 2^Q$

↳ differ in transition fn. (δ)

DFA



$Q = \{A, B\}$

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

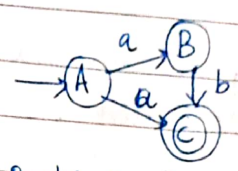
$q_0 = A$

$F = \{B\}$

Transition function (δ)

states	I/P a	b
A	B	A
B	B	B

NFA



need not tel for all I/P in each state

$Q = \{A, B, C\}$

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

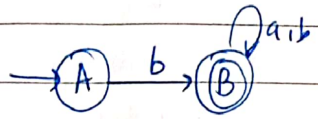
$q_0 = A$

$F = \{C\}$

Transition func^n (δ)

~~A~~ a	a	b
A	B,C	ϕ
B	ϕ	C
C	ϕ	ϕ

eg:

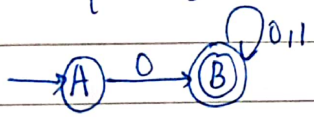


NFA

strings that start with b.

① $L = \{$ set of all strings over (0,1) that starts with 0 $\}$

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



NFA

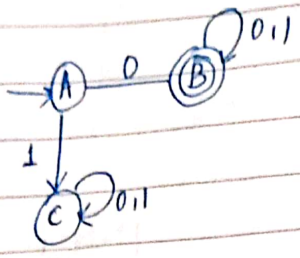
Transition state of NFA

	0	1
A	B	ϕ
B	B	B

Step 1. Transition state for DFA → If no where (ϕ), in DFA, create trap state

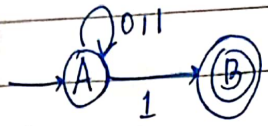
	0	1
A	B	C
B	B	B
C	C	C

Step2 Draw the DFA from δ

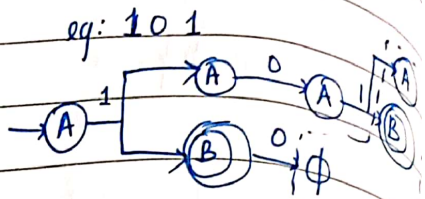


(2) L = { set of all strings over (0,1) that ends with 1 }

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



eg: 1 0 1



Transition table of NFA.

δ	0	1
A	A	{A, B}
B	ϕ	ϕ

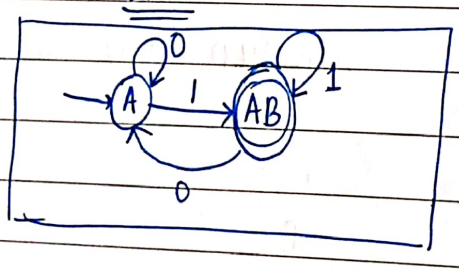
Transition table of DFA → if multiple states, combine to one.

	0	1
A	A	AB
AB	A	AB
~~dead~~	~~dead~~	~~dead~~

AB → new state → union of A & B

i.e A X 0 → A ∪ B for 0. → A.

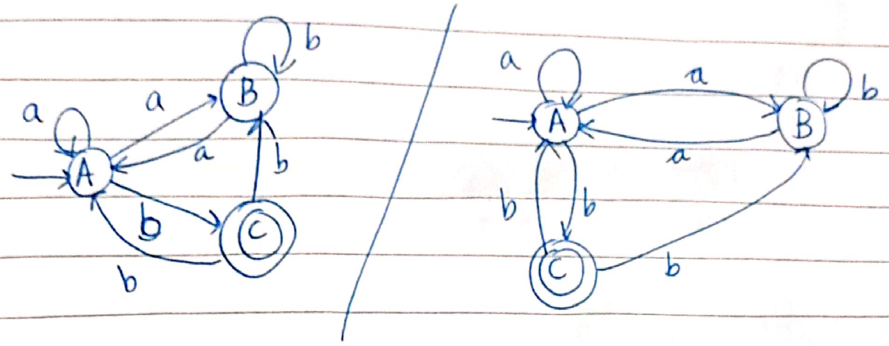
DFA



Find equivalent DFA for NFA given by $L = \{\{A,B,C\}, (a,b), \delta, A, \{C\}\}$ where δ is given by

	a	b
→ A	{A,B}	C
B	A	B
(C)	ϕ	{A,B}

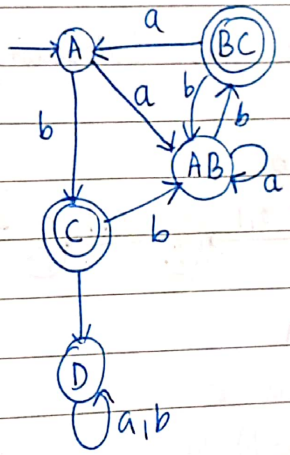
$Q = \{A,B,C\}$ $\Sigma = \{a,b\}$ $q_0 = A$ $F = \{C\}$



Transition table for DFA

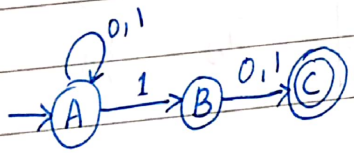
	a	b
→ A	AB	C
AB	AB	BC
(BC)	A	AB
(C)	D	AB
D	D	D

DFA



Design an NFA for language that accepts all strings over (0,1) in which the second last symbol is always 1. Then convert it into equivalent DFA.

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



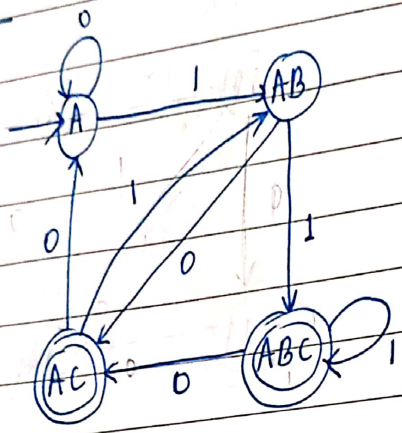
Transition table

	0	1
A	A	{A,B}
B	C	C
C	ϕ	ϕ

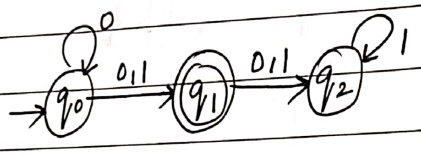
Transition table for DFA

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

DFA



Convert the following NFA into its equivalent DFA.



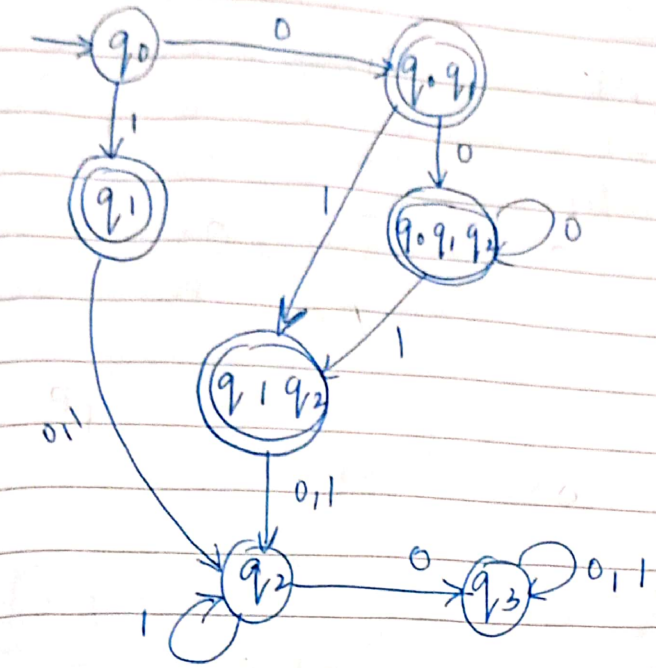
TS of NFA:

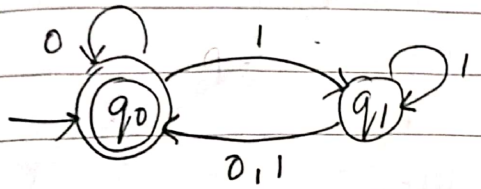
	0	1
q_0	$\{q_0, q_1\}$	q_1
q_1	q_2	q_2
q_2	ϕ	q_2

TS of DFA.

	0	1
$\rightarrow q_0$	$q_0 q_1$	q_1
$q_0 q_1$	$q_0 q_1 q_2$	$q_1 q_2$
$q_0 q_1 q_2$	$q_0 q_1 q_2$	$q_1 q_2$
$q_1 q_2$	q_2	q_2
q_1	q_2	q_2
q_2	q_3	q_2
q_3	q_3	q_3

DFA





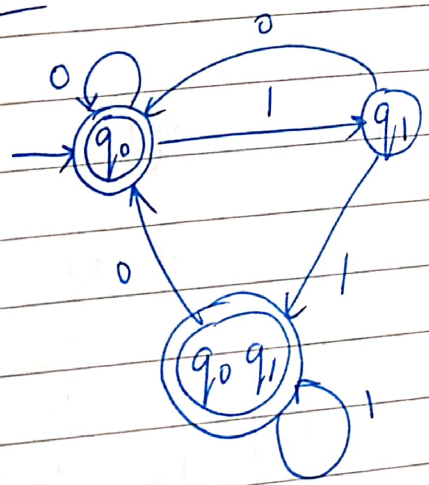
TS of NFA

	0	1
q_0	q_0	q_1
q_1	$\{q_0$ ~~q_1~~	$\{q_0, q_1\}$

TS of DFA

	0	1
q_0	q_0	q_1
q_1	q_0	q_0q_1
q_0q_1	q_0	q_0q_1

DFA



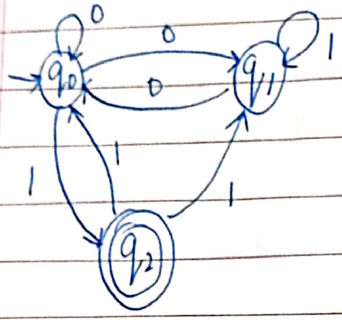
iii) TS of NFA

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

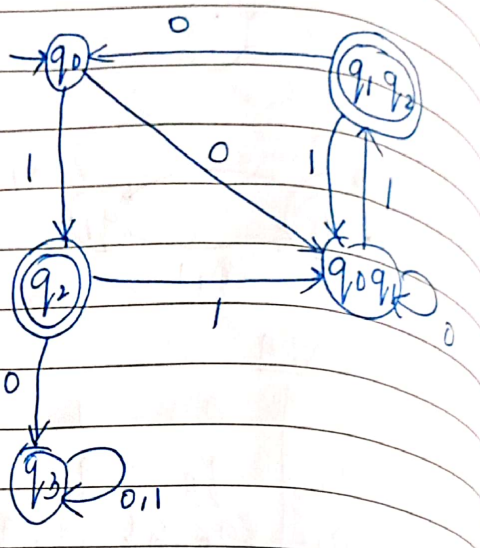
TS of DFA

δ	0	1
→ q_0	q_0q_1	q_2
q_0q_1	q_0q_1	q_1q_2
q_1q_2	q_0	q_0q_1
q_2	q_3	q_0q_1
q_3	q_3	q_3

NFA

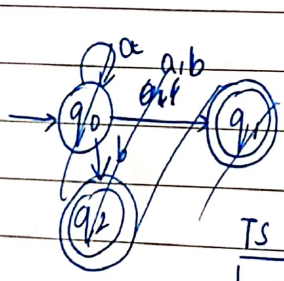


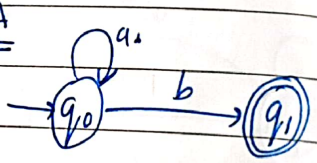
DFA



⑥ Construct NFA to accept all strings with arbitrary no. of a's followed by a single b.

NFA





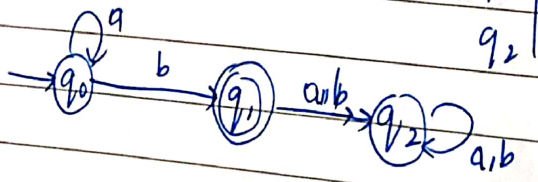
TS of NFA

	a	b
q_0	q_0	ϕ q_1
q_1	ϕ	ϕ

TS of DFA

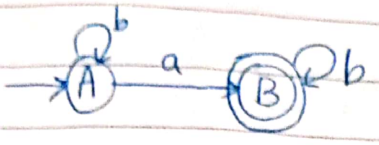
	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2

DFA:



Design NFA that accept string with exactly one a over $\Sigma = \{a, b\}$

NFA



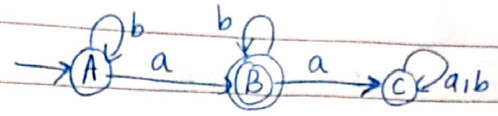
TS of NFA

δ	a	b
A	B	A
B	ϕ	B

TS of DFA

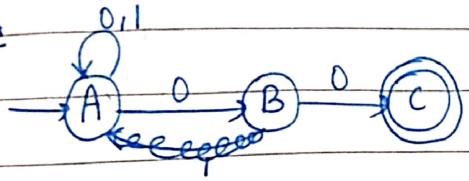
δ	a	b
A	B	A
B	C	B
C	C	C

DFA



Design an NFA that accepts all strings ending with 00 over $\Sigma = \{0, 1\}$.

NFA



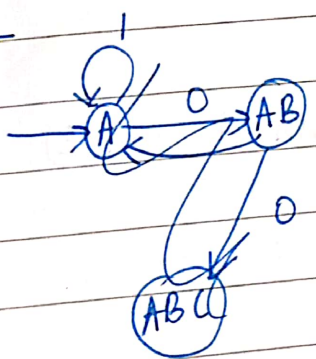
TS of NFA

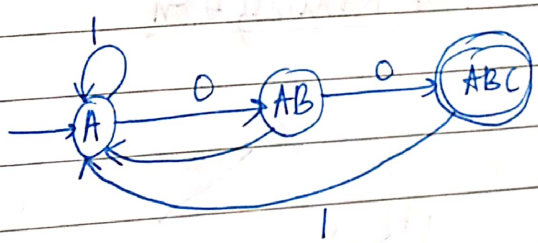
δ	0	1
A	{A,B}	A
B	C	ϕ
C	ϕ	ϕ

TS of DFA

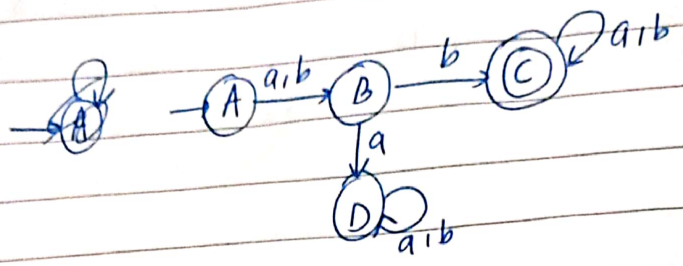
δ	0	1
A	AB	A
AB	ABC	A
ABC	ABC	A

DFA

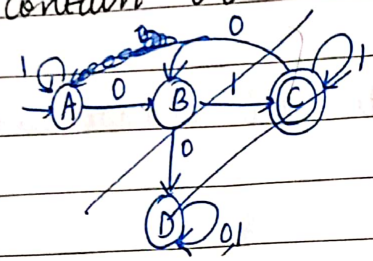


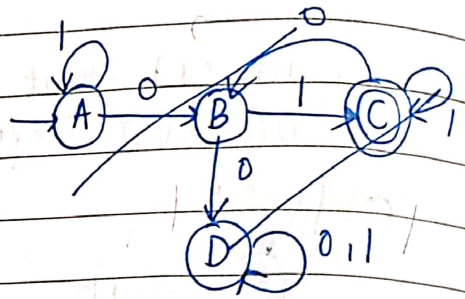


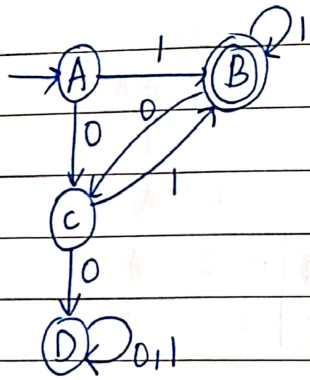
(9) Design DFA that accepts set of all strings with b as second letter over (a|b)



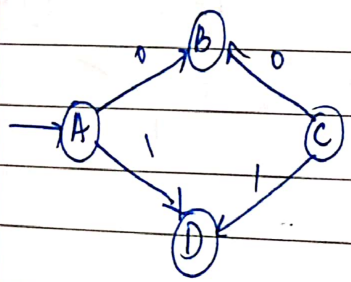
(10) Obtain DFA that accepts all strings on $\Sigma = \{0,1\}$ that ends with 1 and do not contain 00





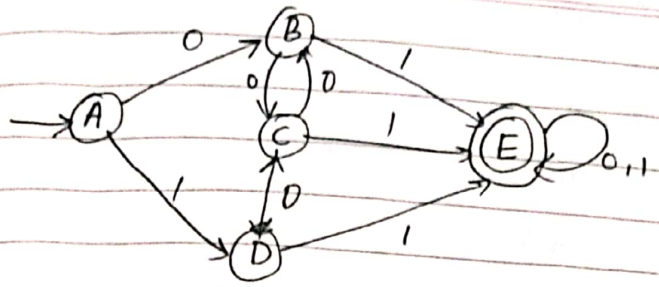


5/4/24 **DFA Minimization**



Here both (A) and (C) are going to (B) when 0. So we can combine them.

(1)



δ	0	1
A	B	D
B	C	E
C	B	E
D	C	E
E	E	E

State Transition Table

Step 1: Remove the states that are not reachable from start state
[In this example, all states are reachable]

Step 2: Divide the states into
i) Set of non final states
ii) Set of final states

Step 3: Split the set of states if possible based on if they behave the same or not

zero equivalence → Π_0 = {A, B, C, D} ① {E} ②

A → 0 → B → ① A → 1 → D → ①
B → 0 → C B → 1 → E → ②
} They do not belong in same set as they don't behave same

A → 0 → B → ① A → 1 → D → ①
C → 0 → B C → 1 → E → ②
} not same behaviour

B → 0 → C → ① B → 1 → E → ②
C → 0 → B C → 1 → E
} same behaviour — same set

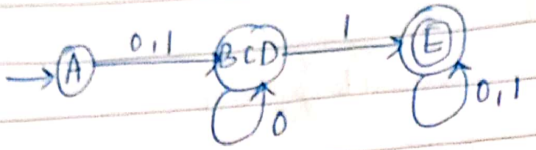
A → 0 → B → ① A → 1 → D → ①
D → 0 → C D → 1 → E → ②
} not same

∴ Π_1 = {A} ③ {B, C, D} ④ {E} ⑤

Π_2 = {A} {B, C, D} {E}

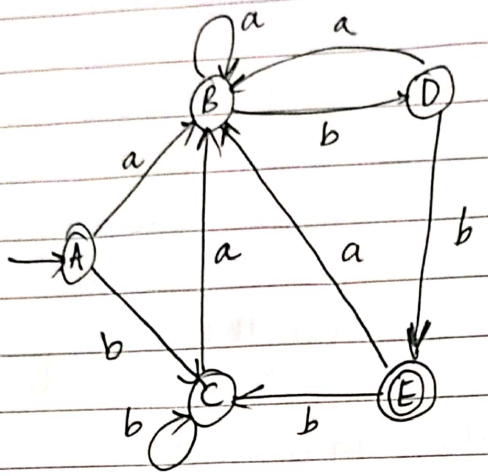
Stop the splitting when last two equivalance are same.

Step 4: Draw the DFA by considering each set as a single state and refer to δ.



Minimized DFA

(2)



Transition state

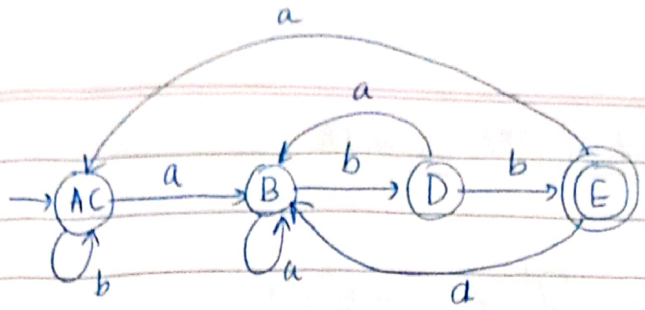
δ	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

$\Pi_0 = \{A, B, C, D\}\ \{E\}$

$\Pi_1 = \{A, B, C\}\ \{D\}\ \{E\}$

$\Pi_2 = \{A, C\}\ \{B\}\ \{D\}\ \{E\}$

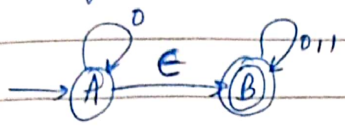
$\Pi_3 = \{A, C\}\ \{B\}\ \{D\}\ \{E\}$



3/6/24

∈ - NFA

∈ - null symbol.

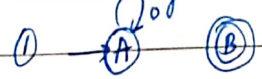


∈-NFA defined using 5 tuples:

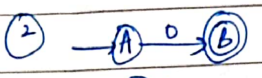
i) Q - set of states
ii) Σ - set of inputs
iii) q_0 - start state
iv F - set of final state
v) δ - transition function : ~~$\delta: Q \times \Sigma \to 2^Q$~~

~~$\delta: A \times 0 \to A$~~,

Note: In NFA, if two states A & B.

①  (go to A)

$\delta: A \times 0 \to A, B, AB, \phi$

②  (go to B)

③  (go to both)

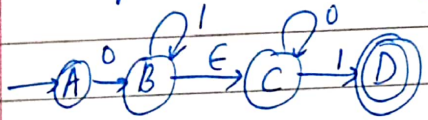
④ No where

In ∈-NFA, $\delta = Q \times \Sigma \cup \epsilon \to 2^Q$

$= A \times 0 \cup \epsilon \to A, B, AB, \phi$

- By seeing nothing, you can switch from one state to another.
- Every state on ∈ goes to itself.

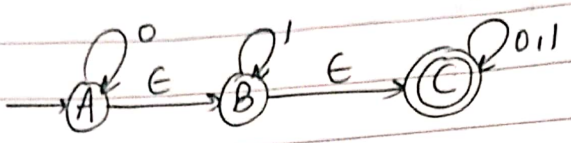
eg:



start with zero and end with 1

Convertion of ϵ-NFA to NFA

(1)



note: it accepts empty string

[1] Create the below table

State	ϵ^* Input	ϵ^*

Epsilon closure (ϵ^*) → it is a set of states that can be reached from a particular state only by seeing the 'ϵ' symbol.

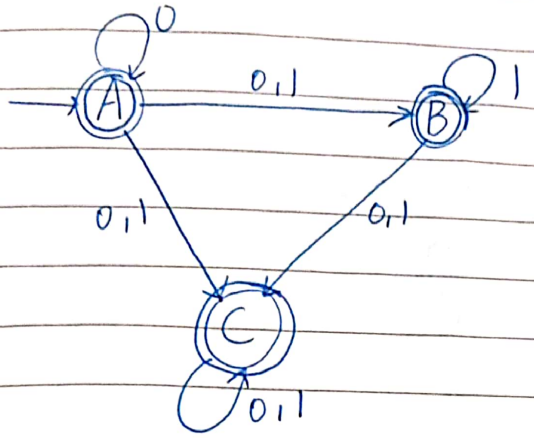
State	ϵ^*	0	ϵ^*
A	A	A	A, B, C
	B	ϕ	
	C	C	C
A	ϵ^*	1	ϵ^*
	A	ϕ	
	B	B	B, C
	C	C	C
B	ϵ^*	0	ϵ^*
	B	ϕ	
	C	C	C
B	ϵ^*	1	ϵ^*
	B	B	B, C
	C	C	C
C	ϵ^*	0	ϵ^*
	C	C	C
C	ϵ^*	1	ϵ^*
	C	C	C

[2] Transition table for NFA

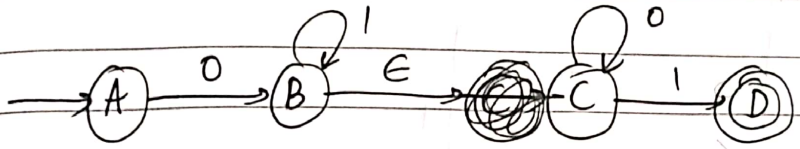
δ	0	1
A	{A, B, C}	{B, C}
B	C	{B, C}
C	C	C

* Final state any state that can reach the final state only by seeing ϵ

[3] Draw the NFA



(2)

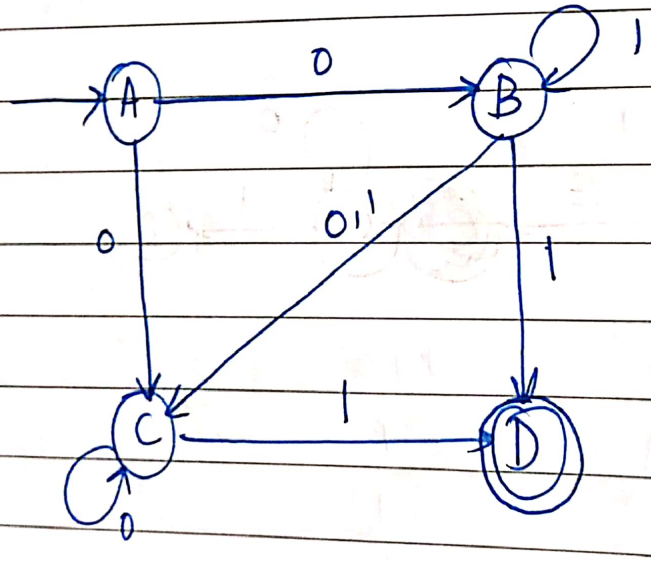


State	ϵ*	0	ϵ*
A	A	B	B, C
	ϵ*	1	ϵ*
	A	ϕ	
B	ϵ*	0	ϵ*
	B	ϕ	
	C	C	C
	ϵ*	1	ϵ*
	B	B	B, C
	C	D	D
C	ϵ*	0	ϵ*
	C	C	C
	ϵ*	1	ϵ*
	C	D	D

D	ϵ^*	0	ϵ^*
	D	—	ϕ
	ϵ^*	1	ϵ^*
	D	—	ϕ

Transition table

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



UNIT - II

Regular expression

The language accepted by finite automata can be easily described by simple expressions called regular expression.
The language accepted by some regular exprenion are referred as regular languages.
eg:
a^* - zero or more combination of a
a^+ - one or more combination of a

Operations on regular languages

1) Union
Let L1 and L2 be two regular languages, then
L1 ∪ L2 = L3 will also be a regular language

2) Concatenation
L1 · L2 = L3 → regular ~~or~~ language

3) Kleene closure
L^* → closure of L1.
→ zero or more combination of L1

Write R.E for language accepting
i) all combinations of a over $\Sigma = \{a\}$
R.E = a^*

ii) all combinations of a except null string over $\Sigma = \{a\}$
R.E = a^+

iii) string containing any no. of a's and b's. over $\Sigma = \{a, b\}$

R.E = $(a+b)^*$

L = { ε, a, b, ab, ba, aa, ... }

iv) Strings starting with 1 and ending with 0 over $\Sigma = \{0,1\}$

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

v) Strings starting and ending with a 'a' & having any combination of b's in b/w

R.E = ab^*a

vi) strings starting with a but not have consecutive b's over $\Sigma = \{a,b\}$

R.E = $a(a+ab)^*$

vii) strings ~~starting with~~ containing any no. of a's followed by any ~~any~~ no. of b's and followed by any no. of c's over $\Sigma = \{a,b,c\}$

R.E = $a^*b^*c^*$

viii) Strings having even length of over $\Sigma = \{0\}$

R.E = $(00)^*$

ix) strings having at least one zero and atleast one 1 over $\Sigma = \{0,1\}$

~~R.E = $(0^+1^+ + 1^+0^+)^+$~~

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

x) strings that do not contain the substring 01 over $\Sigma = \{0,1\}$

R.E = 1^*0^*

xi) strings ending with 011

R.E = $(0+1)^*011$

7/5/24

xii) ~~strings~~ $\{(11)^n \mid n \geq 0\}$

R.E = $(11)^*$

xiii) strings with even no of zeros:

~~R.E = $(1+00)^*$~~ R.E = $(1^*01^*01^*)^* + 1^*$

xiv) string with odd number of b's over $\Sigma = \{a, b\}$

~~R.E = $(a^*ba^*)^*$ + a^*~~

R.E = $(a^*ba^*ba^*)^*ba^*$ or ~~$a^*(ba^*)^*$~~ $a^+(ba^*ba^*)^*ba^*$

xv) strings that contain aa or bb

R.E = $(a+b)^*(aa+bb)^*(a+b)^*$

xvi) strings having atleast two b's

~~R.E = $(a+b)^*bb(a+b)^*$~~

R.E = $(a+b)^*b(a+b)^*b(a+b)^*$

xvii) string's that contain exactly two b's

R.E = $a^*ba^*ba^*$

xviii) Strings that don't contain aa and ending with b

~~R.E = $(ab+b)^*$~~

R.E = $(b+ab)^*$

Note: $a^+ = aa^*$

xix) strings with alternate a's and b's

R.E = ~~$(ab)^*$~~ $(\epsilon+b)(ab)^*(\epsilon+a)$

(OR) $(ab)^* + (ba)^* + a(ba)^* + b(ab)^*$

xx) strings with one pair of consecutive zeros

R.E = $(1+01)^*00(10+1)^*$

xxi) strings of a's and b's of even length.

$R.E = (ab + ba + aa + bb)^*$

(or)

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

xxii) strings of a's and b's of odd length

$R.E = (a+b)(ab+aa+ba+bb)^*$

(or)

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

xxiii) strings of 0's and 1's having at least ~~of one pair~~ three consecutive zeros

$R.E = (0+1)^* 000 (0+1)^*$

xxiv) strings of 0's and 1's having no two consecutive zeros.

$R.E = (1+01)^*(0+\epsilon)$

xxv) strings of a's and b's whose length is less than or equal to 10

$R.E = (\epsilon + a + b)^{10}$

xxvi) strings starting with a and ending with b

$R.E = a(a+b)^* b$

xxvii) strings whose 10th symbol from right is a

$R.E = (a+b)^* a (a+b)^9$

xxviii) words with two or more letters but begin and end with same letter $\Sigma = \{a, b\}$

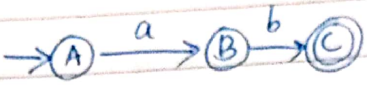
$R.E = a(a+b)^* a + b(a+b)^* b$

xxix) strings whose length is even or multiple of 3 or both

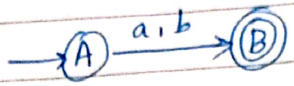
$R.E = ((a+b)(a+b))^* + ((a+b)(a+b)(a+b))^*$

Regular Expression to Finite Automata

Concatination

ab ⟹ 

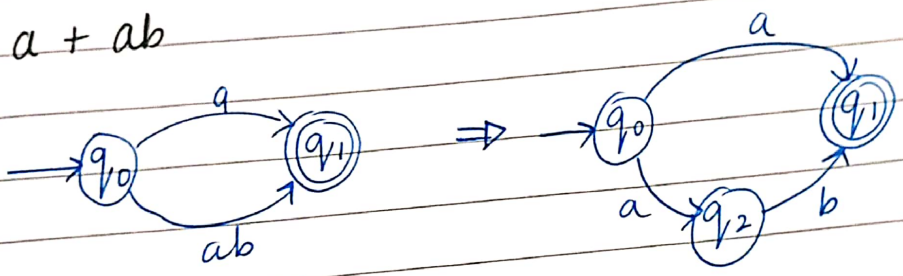
Union

a+b ⟹ 

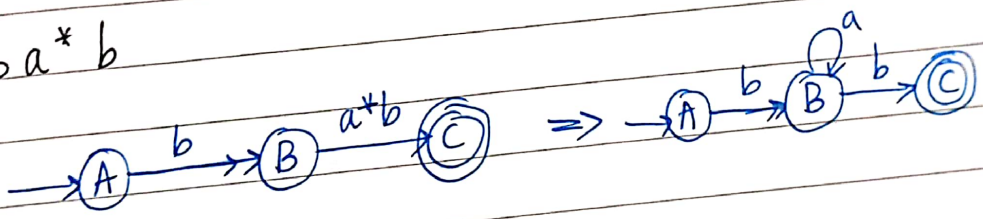
Kleen closure

a^* ⟹ 

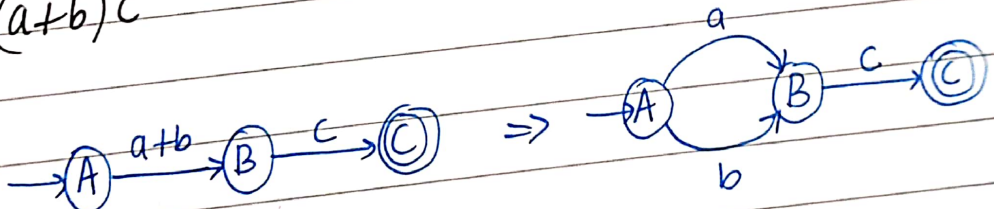
a + ab



ba^*b

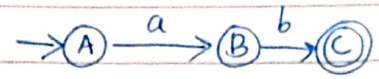


(a+b)c

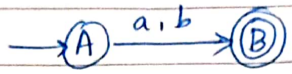


Regular Expression to Finite Automata

Concatenation

ab ⟹ 

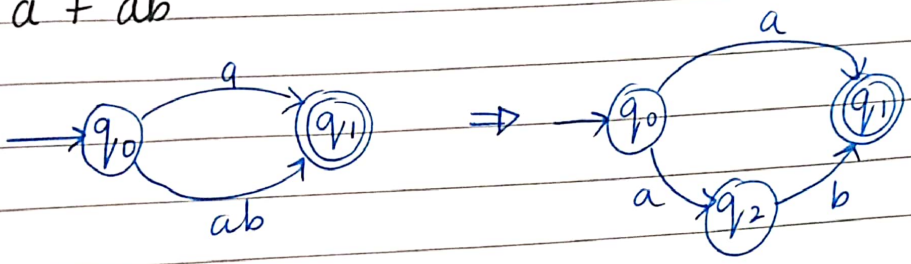
Union

a+b ⟹ 

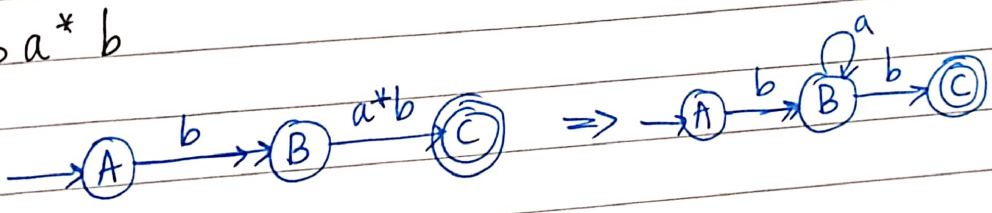
Kleen closure

a^* ⟹ 

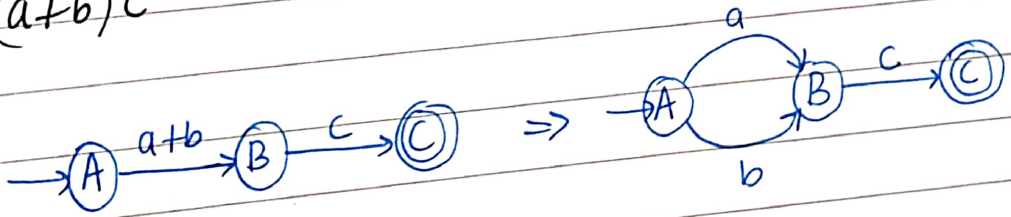
a + ab



ba* b



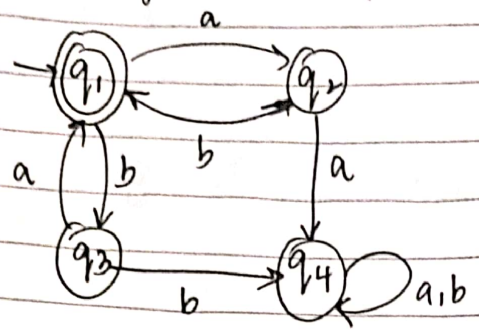
(a+b)c



13/5/24

How to find R.E. for the following DFA:

using Arden Theorem (1)



note: Arden's Theorem

$R = Q + RP$

$\Downarrow$

$R = QP^*$

Identities:

i) $\epsilon R = R$

ii) $\epsilon + RR^* = R^*$

Incoming transitions

$q_1 = \epsilon + q_2 b + q_3 a$

$q_2 = q_1 a$

$q_3 = q_1 b$

$q_4 = q_2 a + q_3 b + (a+b) q_4$

Final state:

$q_1 = \epsilon + q_2 b + q_3 ba$

$= \epsilon + q_1 ab + q_1 ba$

$q_1 = \epsilon + q_1 (ab + ba)$

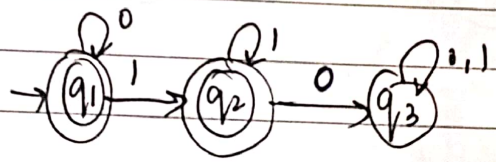
By Arden's theorem

$q_1 = \epsilon (ab+ba)^*$

By identity (i)

Ans: $\boxed{q_1 = (ab+ba)^*}$

(2)



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

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

$q_0 = q_1$

$q_F = \{q_1, q_2\}$

Incoming transition

$q_1 = \epsilon + 0 q_1 0$

$q_2 = \epsilon + q_1 1 + q_2 1$

$q_3 = q_2 0 + q_3 0 + q_3 1$

ii) Prove $L = \{ww \mid w \in \{0,1\}\}$ is not a Regular Language

Final state:

$q_1 = \epsilon + q_1 0$

By Arden's theorem

$q_1 = \epsilon 0^*$

$\boxed{q_1 = 0^*}$

$q_2 = q_1 1 + q_2 1$

$= 0^* 1 + q_2 1$

By Arden's Theorem,

$\boxed{q_2 = 0^* 1 1^*}$

R.E = Union of both final states

$= q_1 \cup q_2$

$= q_1 + q_2$

$R.E = 0^* + 0^* 1 1^*$

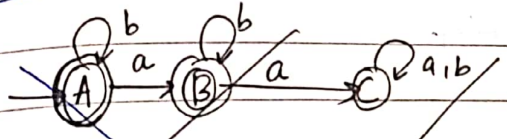
$= 0^* + 0^* 1 1^*$

$= 0^* (\epsilon + 1 1^*)$

By applying identity (ii), $\epsilon + RR^* = R^*$

$\boxed{R.E = 0^* 1^*}$

(3)



$Q = \{A, B, C\}$

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

$q_0 = A$

$F = \{B, A\}$

$A = \epsilon + Ab + Ca$

$B = Aa + Bb$

$C = Ba + Ca + Cb$

Final state:

$B = Aa + Bb$

By Arden's Theorem,

$B = Aab^*$

$B = \epsilon b^* ab^*$

$\therefore R.E = A \cup B$

$= b^* \cup b^* ab^* = b^* + b^* ab^* = b^* (\epsilon + ab^*)$

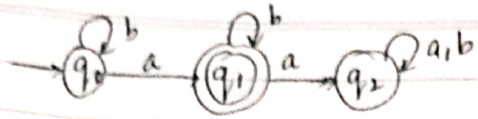
$A = \epsilon + Ab$

By Arden's theorem,

$A = \epsilon b^*$

$\boxed{A = b^*}$

③



$q_0 = \epsilon + q_0 b$

$q_1 = q_0 a + q_1 b$

$q_2 = q_1 a + q_2 a + q_2 b$

Final state:

$q_1 = q_0 a + q_1 b$

$= (\epsilon + q_0 b) a + q_1 b$

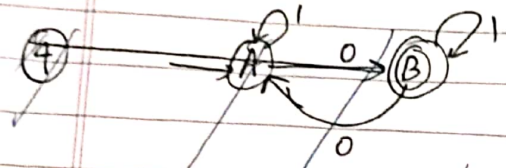
$= b^* a + q_1 b$

$\boxed{q_1 = b^* a b^*}$

$q_0 = \epsilon + q_0 b$

$= \epsilon b^*$

$q_0 = b^*$



~~$A = A1 + B1 + \epsilon$~~

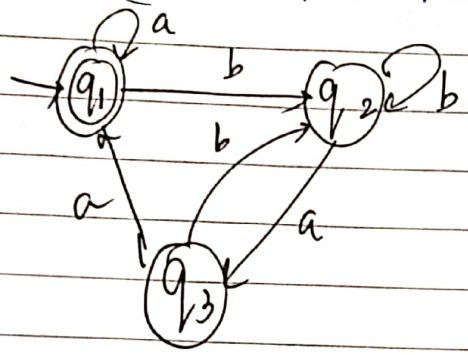
~~$B = A0 + B1$~~

~~$A = A\epsilon + A1 + (A+B)$~~

~~$B = A0 + B1$~~

~~$= (A1 + B1 + \epsilon)0 + B1$~~

④



$q_1 = \epsilon + q_1 a + q_3 a$

$q_2 = q_1 b + q_2 b + q_3 b$

$q_3 = q_2 a$

$q_2 = q_1 b + q_2 ab + q_2 b$

$= q_1 b + q_2 (ab + b)$

$q_2 = q_1 b (ab + b)^*$

~~$q_1 = \epsilon + q_1 a + q_3 a$~~

~~$= \epsilon + q_2 aa + q_1 a$~~

$q_1 = \epsilon + q_1 a + q_3 a$

$= \epsilon + q_1 a + q_2 aa$

$= \epsilon + q_1 a + q_1 b(ab+b)^* aa$

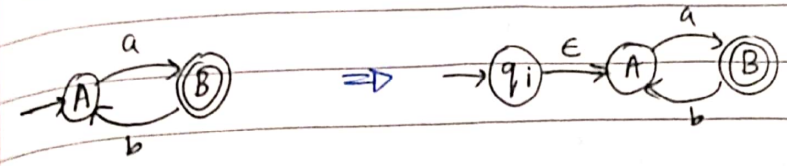
$q_1 = \epsilon + q_1 (a + b(ab+b)^* aa)$

$\therefore q_1 = \epsilon (a+b(ab+b)^* aa)^*$

$\boxed{q_1 = (a + b(ab+b)^* aa)^*}$

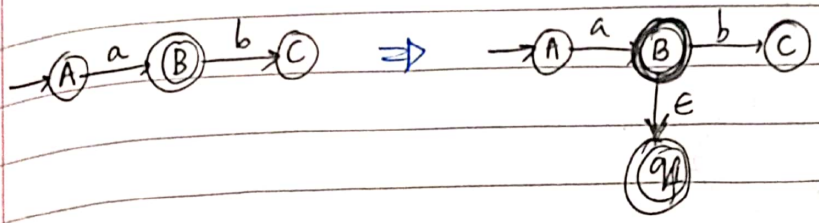
14/5/24 **To find RE for the following FA using State elimination method**

Step i)



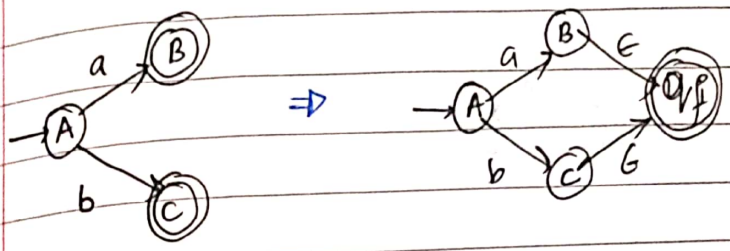
Initial state
- incoming edge

Step ii)



~~Last~~ Final state
- outgoing edge

Step iii)

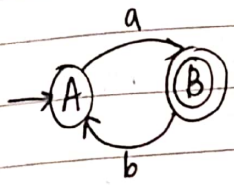


Two final states
- make it to ~~two~~ one

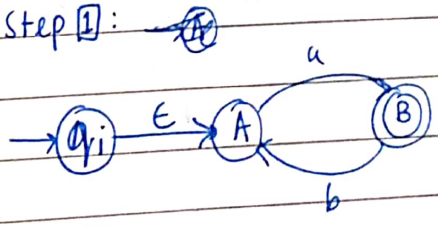
Step iv) Eliminate all states one by one except the initial and final state.

Problems:

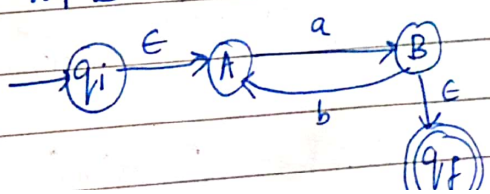
(1)



Step 1:

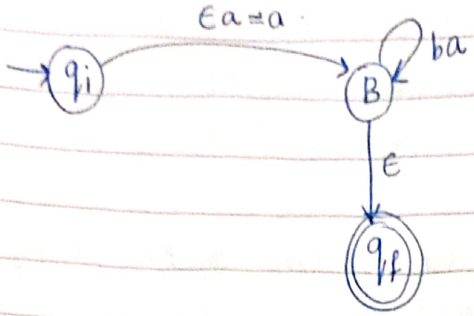


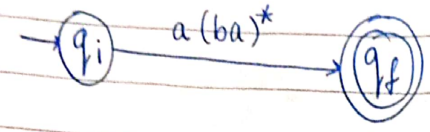
Step 2:



Step [3] : ~~No~~ Only one final state

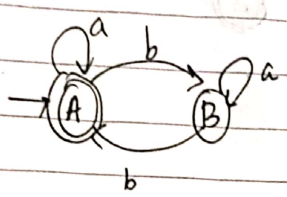
Step [4] :



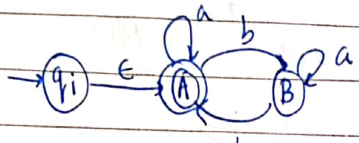


∴ R.E : $a(ba)^*$

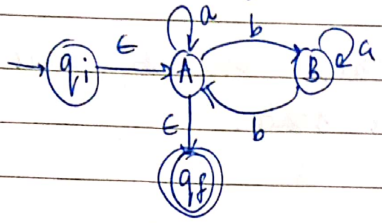
(2)



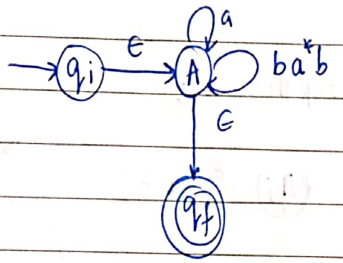
i)

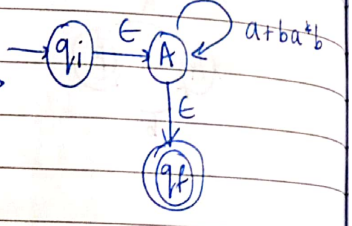


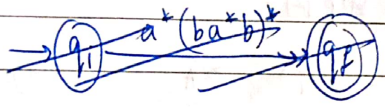
ii)

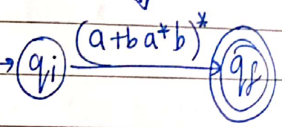


iv)



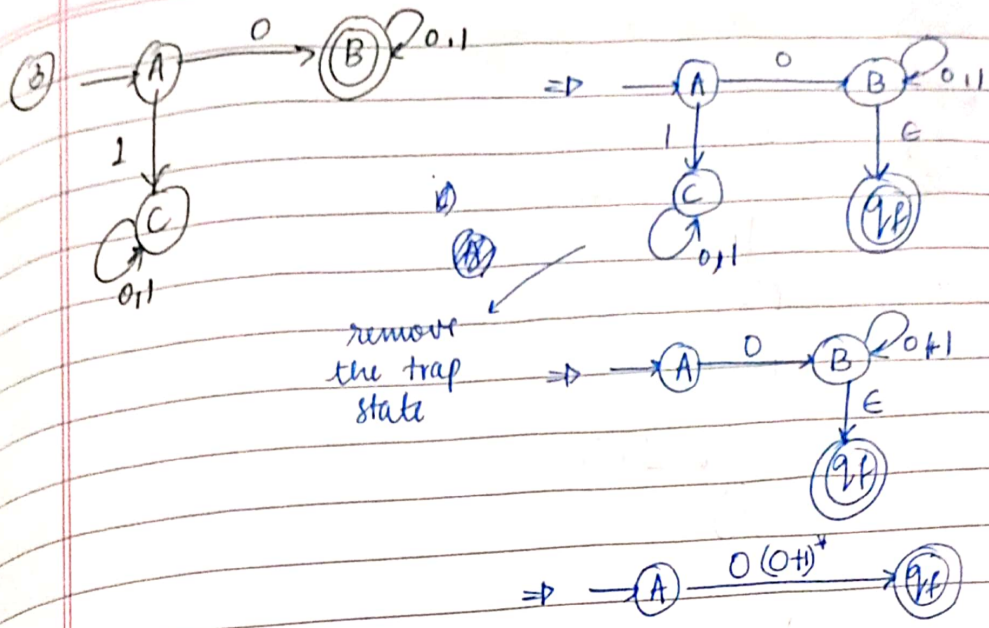




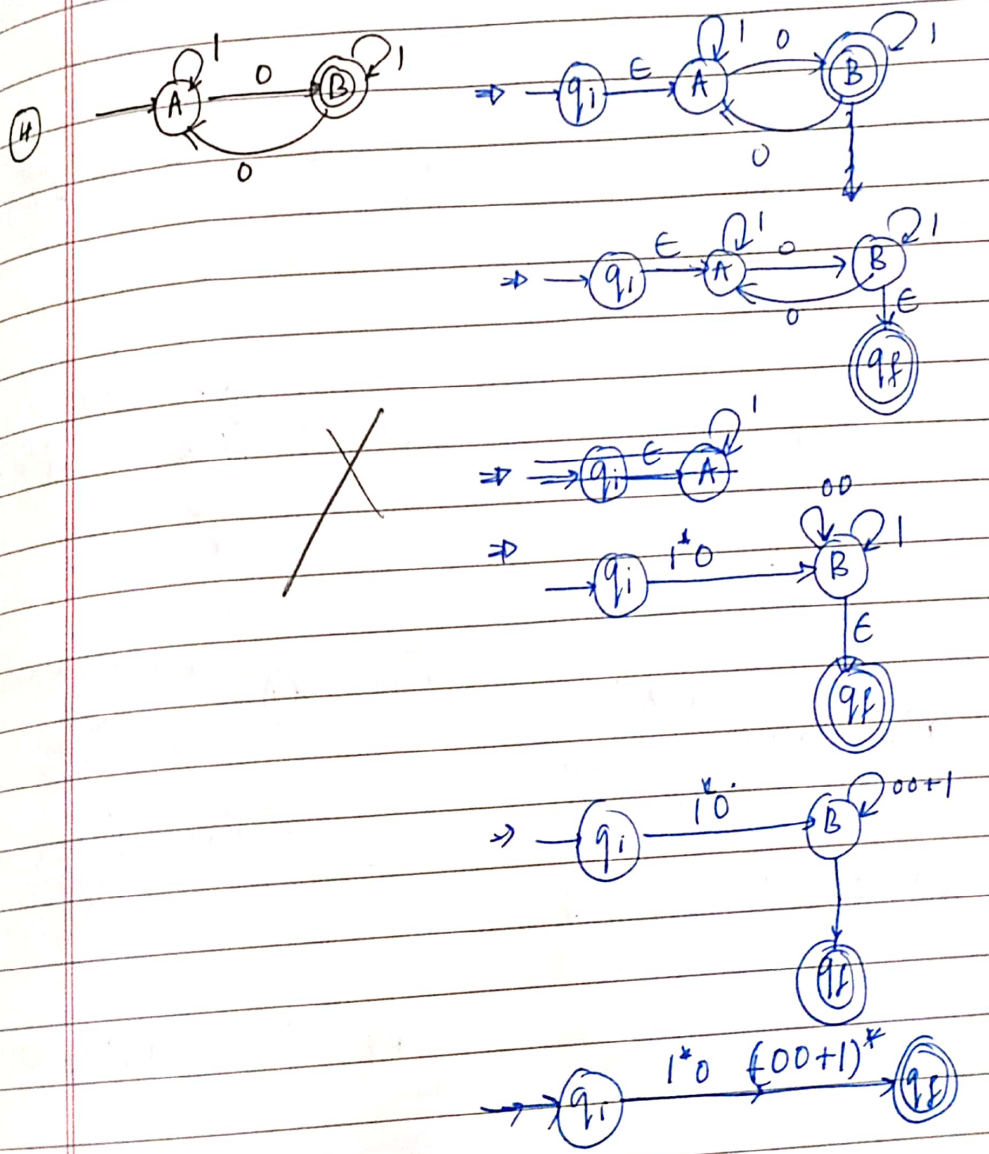


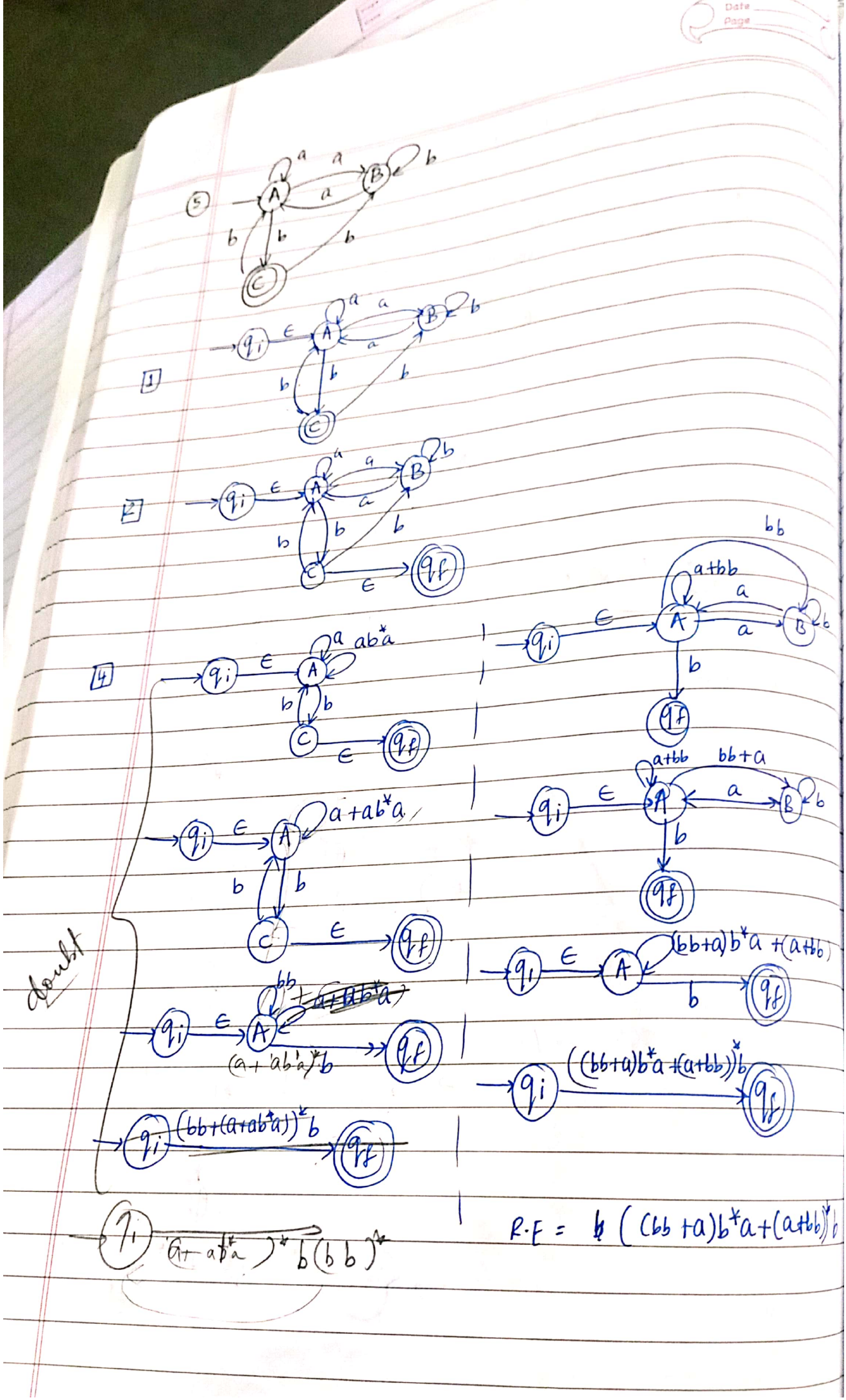
~~∴ R.E = a*(ba*b)*~~

R.E = $(a+ba^*b)^*$



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





10/5/24

Identity Rules of Regular expression

1) $\phi + R = R$
   where R is a regular expression
   where ϕ is an empty set
2) $\phi \cdot R = R \cdot \phi = \phi$
3) $\epsilon \cdot R = R \cdot \epsilon = R$ where ϵ is a null string
4) $\epsilon^* = \epsilon$ & $\phi^* = \epsilon$
5) $R + R = R$ eg: $a + b$ $a + a = a$
6) $R^* \cdot R^* = R^*$ eg: $(ab)^* (ab)^* = (ab)^*$
7) $(R^*)^* = R^*$ eg: $(ab)^{**} = (ab)^*$
8) $RR^* = R^*R = R^+$
9) $\epsilon + RR^* = \epsilon + R^*R = \epsilon + R^+ = R^*$
10) $(P+Q)R = PR + QR$ eg: $(a+b)c = ac + bc$.
11) $(PQ)^* P = P(QP)^*$
12) $(P+Q)^* = (P^* \cdot Q^*)^* = (P^* + Q^*)^*$

Arden's Theorem

If P and Q are two regular expressions over Σ, and if P doesn't contain ϵ, then the following equation in R given by

$R = Q + RP$

has a unique solution i.e.

$R = QP^*$

proof:

$R = Q + RP$

$= Q + QP^*P$ $\because R = QP^*$

$= Q(\epsilon + P^*P)$ $\because \epsilon + R^*R = R^*$

$R = QP^*$

Prove that $R = QP^*$ is the unique solution.

$R = Q + RP$

$= Q + [Q + RP]P$

$= Q + QP + RP^2$

$= Q + QP + (Q+RP)P^2$

$= Q + QP + QP^2 + RP^3$

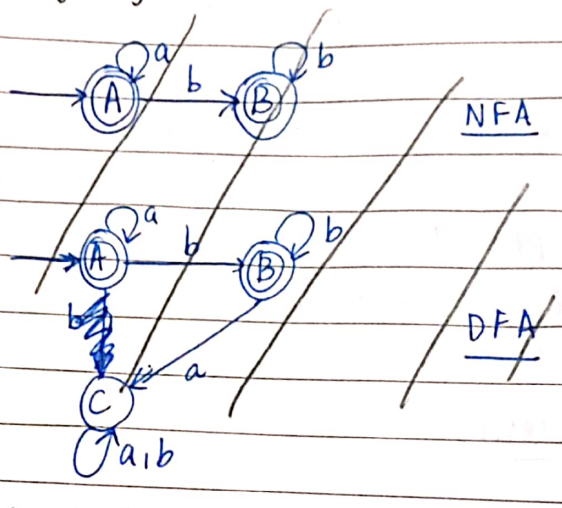
$= Q + QP + QP^2 + \cdots + QP^n + RP^{n+1}$

$= Q + QP + QP^2 + \cdots + QP^n + QP^*P^{n+1}$

$= Q(\epsilon + P + P^2 + \cdots + P^n + P^*P^{n+1})$

$\boxed{R = QP^*}$

Q] Design a FA for the language $L = \{a^n b^n \mid n \geq 0\}$ over $\Sigma = \{a, b\}$



We can not design a FA for above language, as there is no enough memory to remember the count value. So it is not a regular language.

Regular language

It is regular language iff some finite state machine recognizes it.

eg: $L = \{a^n b^n \mid n \geq 0\}$ is not a regular language.

$L = \{yy \mid y \in (0,1)^*\}$ is not a regular language.

Note: FA has very limited memory

It cannot store or count strings.

1/5/24

Pumping Lemma

It is used to prove that language is not regular.

If substring of a string is repeated many times and if the resultant string is also available in language 'L' then we can say it is regular.

Steps:

1. Consider a language as a regular language.
2. Assume a constant 'C' and select the string 'w' such that $|w| \geq C$ where $|w|$ is length of w
3. Divide 'w' as xyz such that
   i) $|y| > 0$
   ii) $|xy| \leq C$
4. Check string xy^iz belongs to L for $i \geq 0$

(1) Prove that $L = \{a^n b^n \mid n \geq 0\}$ is not a regular language.

Step 1: Assume L is a regular language

Step 2: Let $C = 4$, $w = aabb$

Here $|w| \geq C$ i.e $4 \geq 4$.

Step 3: $w = aabb$

Divide w as x y z where

$x = a$ $\quad |x| = 1$

$y = ab$ $\quad |y| = 2$

$z = b$ $\quad |z| = 1$

$\therefore |y| > 0$ and $|xy| = 3 \leq C$

Step 4: When $i = 0$, $xy^iz = xz$

$= ab \in L$

$i = 1$ $\quad xyz = aabb \in L$

$i = 2$ $\quad xy^2z = a\,xyyz = aababb \notin L$

Hence the given language is not a regular language.

② Prove that $L = \{a^i b^j \mid i \leq j\}$ is not a regular language.

$L = \{ab^*, abB, aabb, aabbb, abbb, \ldots\}$

Step 1: Assume L is a regular language

Step 2: Let $c = 2$ and $w = aabbb$

$\therefore |w| \geq c$

$5 \geq 2$

Step 3: Divide w as xyz such that

$x = a \Rightarrow |x| = 1$

$y = a \Rightarrow |y| = 1$

$z = bbb \Rightarrow |z| = 3$

$\therefore |y| \geq 0 \Rightarrow 1 \geq 0$

$|xy| \leq c \Rightarrow 2 \leq 2$

Step 4: when $i = 0$, $xy^i z = xz = abbb \in L$

$i = 1$, $xyz = aabbb \in L$

$i = 2$, $xy^2z = aaabbb \in L$

$i = 3$, $xy^3z = aaaabbb \notin L$

$\therefore$ The above language is not regular.

③ Prove that $L = \{a^{2n} \mid n \geq 1\}$ is not a regular language.

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

Step 1: Assume L is a regular language

Step 2: Let $c = 3$, and $w = aaaa$

$\therefore |w| \geq c$ i.e $4 \geq 3$

Step 3: Divide w as xyz

$x = a \Rightarrow |x| = 1$

$y = a \Rightarrow |y| = 1$

$z = aa \Rightarrow |z| = 1$

$\therefore |y| > 0 \Rightarrow 1 \geq 0$

$\therefore |xy| \leq c \Rightarrow 2 \leq 3$

Step 4: When $i = 0$, $x^0y^iz = xz = aaa \notin L$

$\therefore$ The above language is not regular.

Prove that $L = \{yy \mid y \in \{0,1\}\}$ is not a regular language.

$L = \{00, 11, 0101, 110110, \ldots\}$

Step 1: Assume L is a regular language

Step 2: Let $C = 3$ and $w = 0101$

$\therefore |w| \geq C \Rightarrow 4 \geq 3$

Step 3: Divide w as XYZ

$X = 0$

$Y = 1$

$Z = 01$

$\therefore |X| \geq 0 \Rightarrow 1 > 0$

$|XY| \leq C \Rightarrow 2 \leq 3$

Step 4: When $i = 0$, $XY^iZ = XZ = 001 \notin L$

$\therefore$ The above language is not regular

Applications of Regular Expressions

Regular expression in UNIX

Notations / meta characters:

1) . → single character

2) [A-Z] → character class for uppercase letters

(A+B+C+ ---- +Z)

· [aeiou] → vowel

(a+e+i+o+u)

3) ? → zero or one

\+ → one or more

* → zero or more

Lexical Analysis

It is the first stage in compiler design.

eg: To identify if it is identify or keyword or value, etc.

if: [a-z A-Z][A-Z a-z 0-9]* return identifier;

[0-9]+ return NUM;

'if' | 'float' | 'char' return KEYWORD;

Text Search

a) like in UNIX text editor

eg: RIT[a-z A-Z 0-9]+

eg: RIT5 , RIT cs , RIT cs03 , etc