17-12-2024

Notations:

Alphabet:

- It is a finite non empty set of symbols

Notation: We use symbol Σ (sigma) to denote an alphabet.

Language:

- A collection of sentences of finite length and all from alphabets and symbols. • $L \subseteq \Sigma^*$

String: A string or word is a finite sequence of symbols chosen from Σ.

- Null string is denoted by ϵ (or) e (or) λ

$\Sigma^0 = \epsilon$

$\Sigma^1 = \{0, 1\}$ $\Sigma^2 = \{00, 01, 10, 11\}$ $\Sigma^3 = \{000, 001, 010, 100\}$

Σ^k = set of all k length strings.

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots\ldots$

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

$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \ldots\}$

$\Sigma^+ = \{0, 1, 00, 01, 10 \ldots\ldots\}$

concatenation of $x = 000$

$y = 101$

$x.y = 000101$

* If the st
count non
Ex: 01 ∈
• Concater
• Concate
L = {ϵ,
L = {ϵ
[all s
one's]
• Emp
Ex:
*L =
*L =
Th
G
ε

* If the string contains ε(epsillon) we should count non ε charecters.

Ex: 01 ε 10ε 00ε 11ε $|w| = 6$.

- Concatenation of ε with U(string) is U.
- Concatenation of U with ε is U.

$L = \{\epsilon, 01, 0011, \ldots\} = \{0^n 1^n \mid n \geq 0\}$ [0's followed by 1's]

$L = \{\epsilon, 01, 10, 0011, 1100, 0101, 1010, 1001 \ldots\}$

[all strings of with equal no of zero's and one's.]

- Empty language is denoted by ϕ.

Ex:

$*L = \{\} = \phi \rightarrow$ It is a null string.

$*L = \{\epsilon\} \rightarrow$ zero length string it can't be ϕ

The Membership Problem:

Given a string $w \in \Sigma^*$ and a language L over Σ, decide whether or not $w \in L$.

Example:

Let w = 10011

Q) Is the w ∈ the language of strings with equal no. of 0's and 1's?

Sol: $w \in L$

Binary relations on strings:

Prefix: u is a prefix of u if there is a w such that $u = uw$

ε is a prefix of o since $o = o\varepsilon$

Suffix: u is a suffix of u if there is a w such that $v = wu$

ε is a suffix of o since $o = o\varepsilon$

Substring: u is a substring of v if there are x and y such that $v = xuy$

Ex: o is a substring of o since $\varepsilon \cdot o \cdot \varepsilon$

- Prefix and suffix are special cases of substring.

What is Grammar: A grammar can be regarded as a device that enumerates the sentence of a language.

- A finite set of rules defining a language.

Sentences:

- These are the strings of symbols.

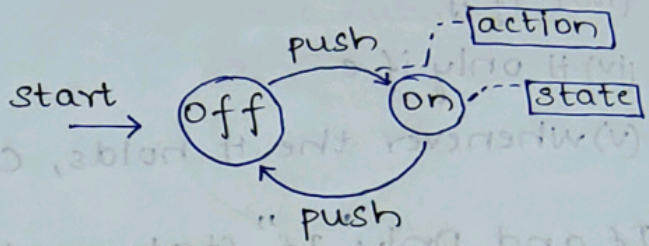
Ex: $S \to 0A$ $\quad B \to 1B$

$A \to 1A$ $\quad B \to 0F$

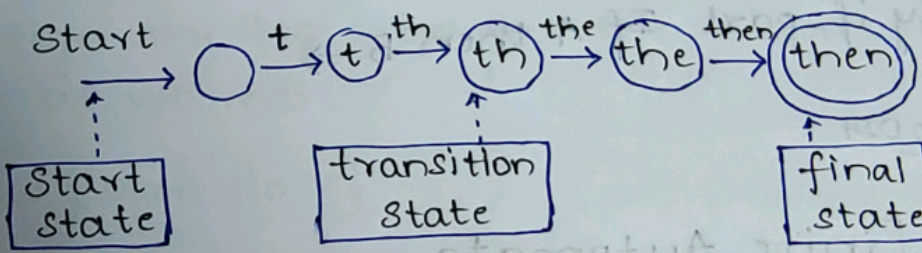
$A \to 0B$ $\quad F \to \epsilon$

Finite Automata : Examples

- On/off switch



- Modelling recognition of the word



Quantifiers:

"$\forall$" for all or for every [Universal proofs]

"$\exists$" There exists [Existential proofs]

Proving Techniques:

- By contrapositive
- By contridiction
- By induction.

Different ways of saying the same thing

"If H then C"

(i) H implies C

(ii) H $\Rightarrow$ C

(iii) C if H

(iv) H only if C

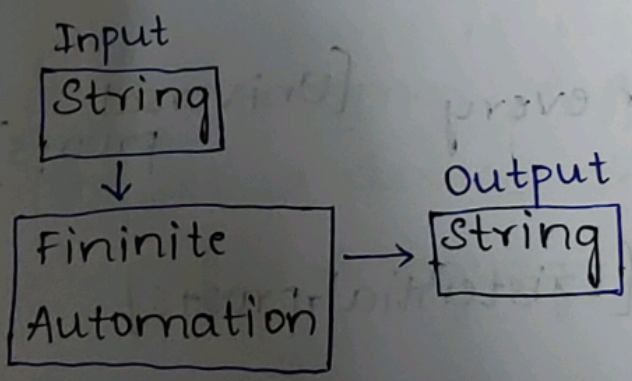
(v) whenever the H holds, C follows

"If and only If" statements :

- if part If B then A $\Leftarrow$
- only if part If A then B $\Rightarrow$

19-12-2024

Finite Automata

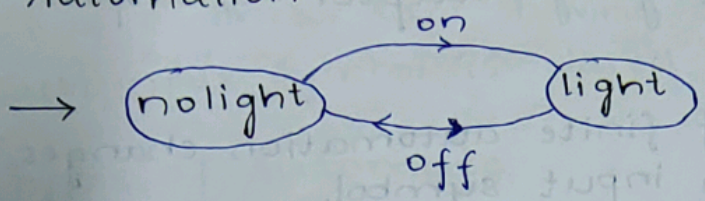


Ex: Design logic behind an electric bulb

Analysis:

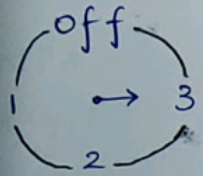
States={nolight, light}, Input={off, on} Finite

Automation

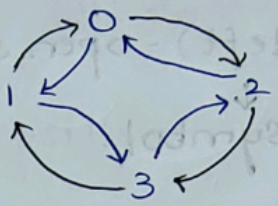


Ex: Multispeed fan

Diagram:



Finite Automation.



Analysis:

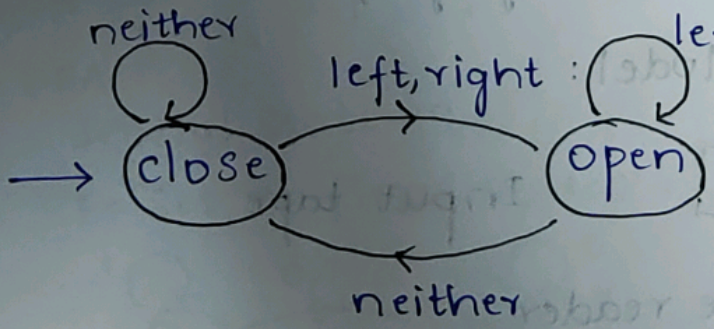
state = {0, 1, 2, 3}

Input = {CW, A, CW}

Ex: Design the logic behind the automatic doors at walmart.

states = {open, close}

Input = {left, right, neither}



← Transition state diagram

Basic feautres of finite Automata:

* Extremly limited memory
* finite set of states

* It acts as a language acceptor i.e, outputs "yes" (or) "no"

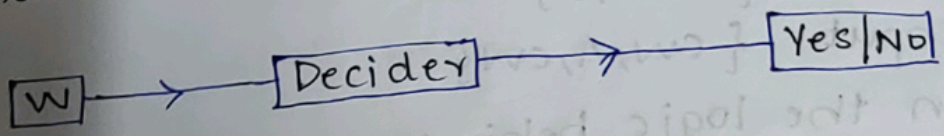
* current state of finite automation changes when it reads an input symbol.

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

Ex: δ(close, left) = open state

(close: state, left: symbol)

What is decision symbol?

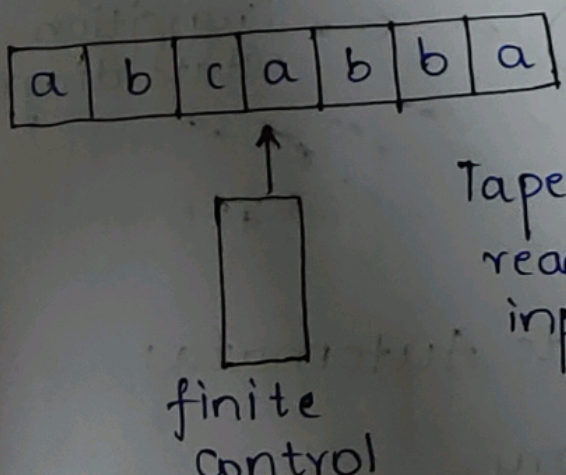


* A computer that solves a decision problem is a decider.

Input to a decider : A string w.

* The output will be in only yes/no.

Finite Automata Model:



Input tape:
It is a linear tape having some no of cells
Each input symbol is placed in each cell.

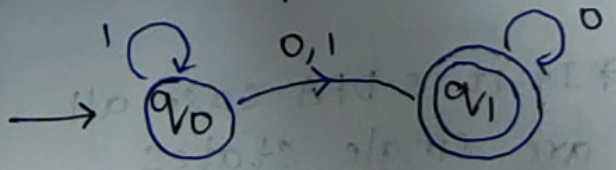
Types of Automata:

1. DFA (deterministic finite automata)
2. NFA (non-deterministic finite automata)
   ↓
   ϵ-NFA
   $\delta(q, \epsilon) = q$ → def of epsillon

1. DFA: It refers to the uniqueness of the computation. the machine goes to one state only one for a particular input charecter.
DFA doesnot accept the null move.

2. NFA: It is used to transmit any number of states for a particular input.
It can accept the null move.

DFA:
Initial State = q_0
F = Set of final states



$\delta(q_0, 0) = q_1$

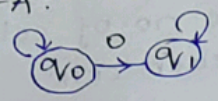
$\delta(q_0, 1) = q_0$

$\delta(q_1, 0) = q_1$ $\delta(q_1, 1) = q_1$

$Q = \{q_0, q_1\}$ → finite set of states

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

NFA:



$\delta(q_0, 0) = q_1$

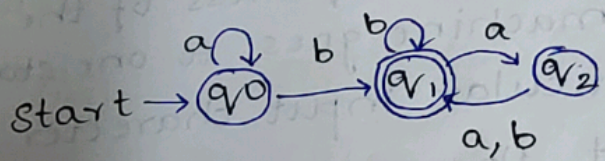
$\delta(q_0, 1) = \{q_0, q_1\}$

$\delta(q_1, 0) = q_1$

* Every DFA is NFA but NFA is not DFA.
* There can be multiple final states in both DFA and NFA

- DFA - lexical Analysis in compiler
- NFA -

Example: Does the DFA accept the string bbab?



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

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

Initial state = q_0

$F = \{q_1\}$

Here F is set of final states

$\delta(q_0, a) = q_0$

$\delta(q_0, b) = q_1$

$\delta(q_0, b) = q_1$

$\delta(q_1, a) = q_2$

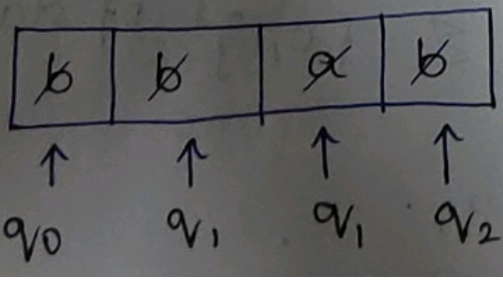
$\delta(q_1, b) = q_1$

$\delta(q_2, a) = q_1$

$\delta(q_2, b) = q_1$

* If it is DFA case all are single states

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



q_1 which is a final state

$\therefore$ bbab is accepted by DFA

NFA but
FA.
multiple
in both

bbab?

δ	a	b
→ q_0	q_0	q_1
* q_1	q_2	q_1
q_2	q_1	q_1

→ Transition table

Formalities:
Deterministic Finite Acceptor DFA

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

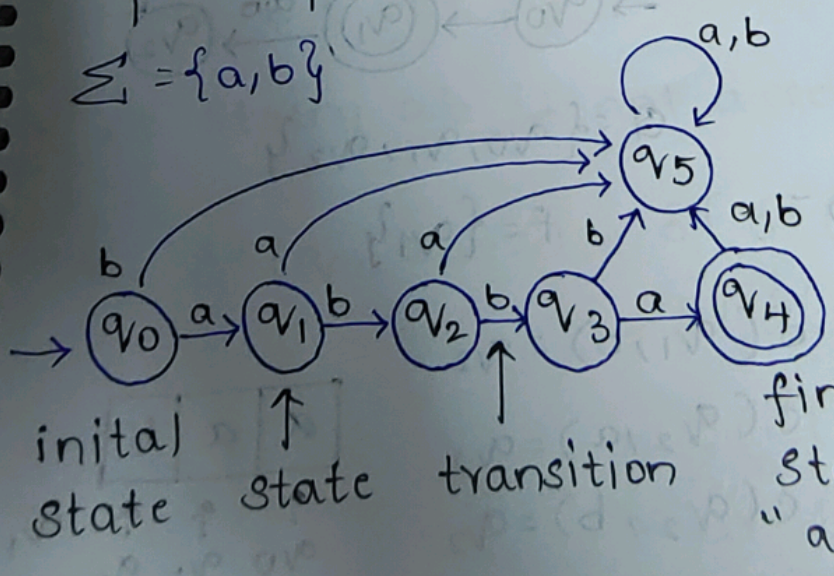
Q : set of states
Σ : input alphabet
δ : transition state function → defined as $\delta : Q \times \Sigma \to Q$
q_0 : initial state
F : set of final states.

20-12-2024

- DFA is defined as finite tuple

Input Alphabet Σ

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



$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

inital state
state
transition
final state "accept"

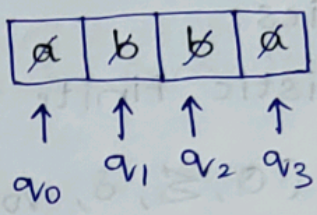
Abba - Finite Acceptor:

δ	a	b
→ q_0	q_1	q_5
q_1	q_5	q_2
q_2	q_5	q_3
q_3	q_4	q_5
q_4	q_5	q_5
q_5	q_5	q_5

$\delta(q_0, a) = q_1$

$\delta(q_0, b) = q_5$

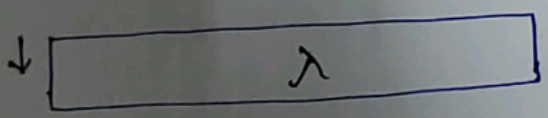
$F = \{q_4\}$



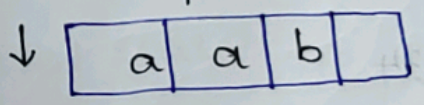
q_4 EF

*abba is accepted by the given DFA or the given DFA accepts the string "abba"

Another Rejection

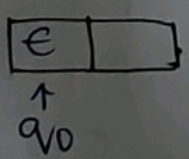


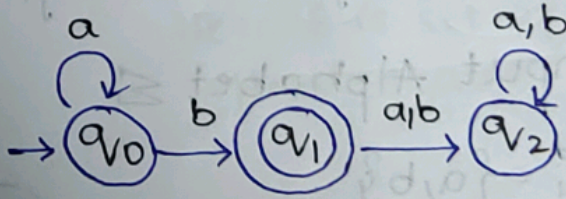
Example:



$\delta(q_{01}, \epsilon) = q_0$

λ or ϵ





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

$\Sigma = \{a, b\}$ q_0 ... $F = \{q_1\}$

$\delta(q_0, a) = q_0$

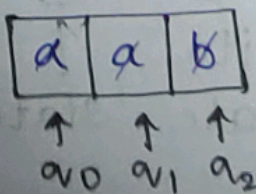
$\delta(q_0, b) = q_1$

$\delta(q_1, a) = q_2$

$\delta(q_1, b) = q_2$

$\delta(q_2, a) = q_2$

$\delta(q_2, b) = q_2$



q_1

• δ*

• δ*

• δ*

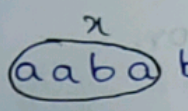
Ex:

*It

Ex:

- $\delta^* : Q \times \Sigma^* \to Q$
- $\delta^*(q_0, \epsilon) = q_0$
- $\delta^*(q_0, w) = \delta^*(q_0, xa)$
  $= (\delta(\delta^*(q_0, x), a))$

Ex:



(aab)

*It is a extended version of δ.

Ex: $\delta^*(q_0, ab) = \delta(\delta^*(q_0, a), b)$

↓ expand

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

till we get epsilon we need to expand.

$= \delta(\delta(q_0, a), b)$

$= \delta(q_1, b)$

$= q_2 \notin F$ (q_2 is not finite acceptor.)

- The String ab is not accepted by given DFA.

Ex: $\delta^*(q_0, abba) = \delta(\delta^*(q_0, abb), a)$

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

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

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

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

$= \delta(\delta(\delta^*(q_1, b), b), a)$

$= \delta(\delta^*(q_2, b), a)$

$= \delta(q_3, a) = q_4 \in F$

Here q_4 is finite acceptor.

$\delta^*(q_0, a) = \delta(\delta^*(q_0, \epsilon), a)$

$\delta^*(q_0, w) = \delta^*(q_0, xa)$

$= (\delta(\delta^*(q_0, x), a))$

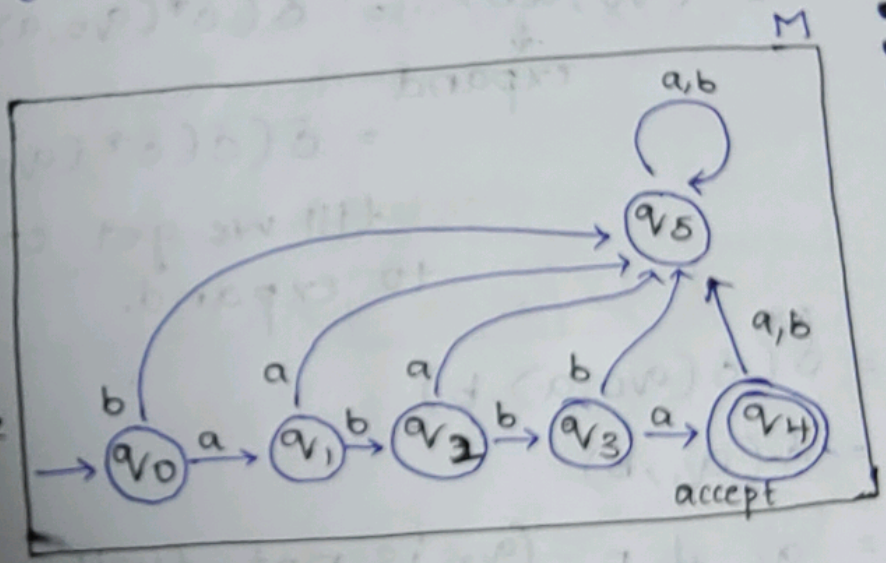
21-12-2024

Example:

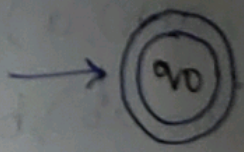
$L(M) = \{abba\}$

Design a DFA that accepts the language

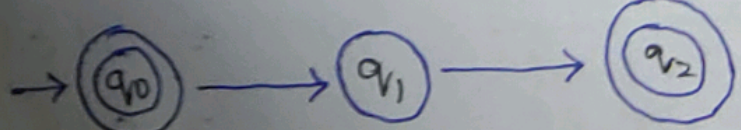
$L = \{\epsilon, ab, abba\}$



(i) ϵ acceptance



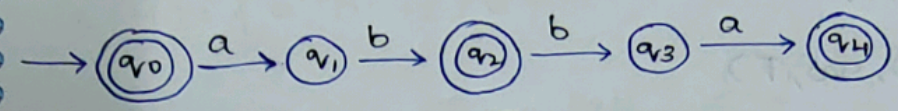
(ii) ab acceptance



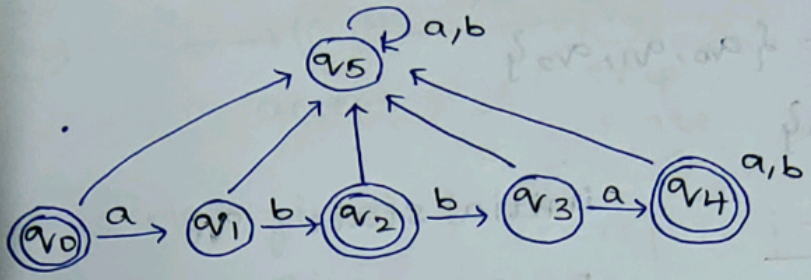
(iii) abb

(iv)

(iii) abba acceptance



(iv)

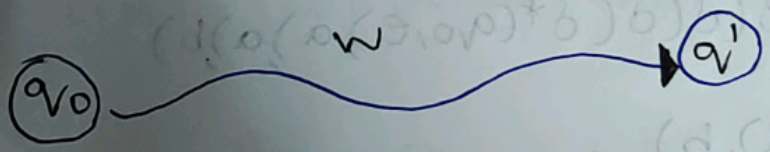


Observation:

Language rejected by M:

$$\overline{L(M)} = \{ w \in \Sigma^* : \delta^*(q_0, w) \notin F \}$$

↳ set of final states.

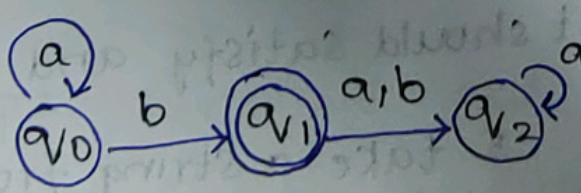


$q' \notin F$

(i) Design the language accepted by that DFA:

$L = \{ b, ab, aab, aaab, \ldots \}$

transition state diagram



(ii) The DFA accepting the given language is

Tuple form:

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

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

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

initial state = q_0

F = set of final states

$= \{q_1\}$

δ	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2

(iii) Justification

$\delta^*(q_0, aab) = \delta(\delta^*(q_0, aa)b)$

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

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

$= \delta(\delta(\delta(q_0, a), a), b)$

$= \delta(\delta(q_0, a), b) = \delta(q_0, b) = q_1 \in F$

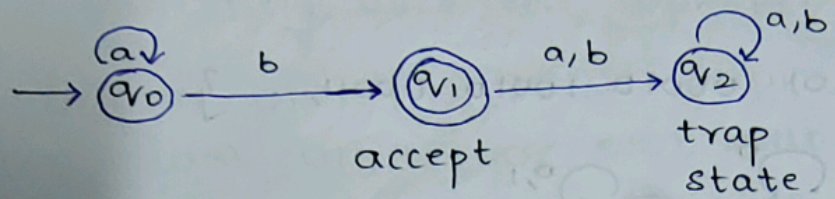
Therefore aab is accepted

Steps for DFA:

1. expand language
2. start with first string it should satisfy and
3. construct DFA
4. construct into tuple form
5. take a string from set and justify.

Ex:

$L(M) = \{ a^n b : n \geq 0 \}$



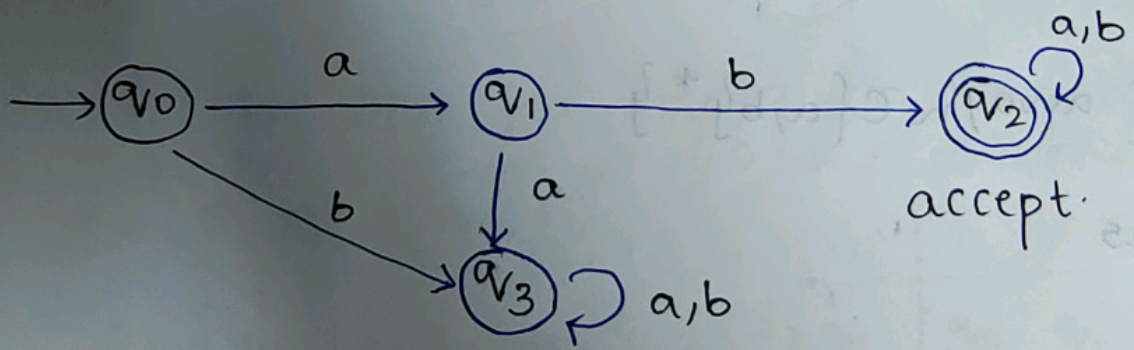
q_0 'b' is not given
q_1 'a' is not given
} Therefore it is not a DFA.

(i) Design a DFA that accepts the language

$L = \{$ set of all strings of prefix ab $\}$

(complete the next steps)

Ex: $L(M) = \{$ all strings with prefix ab $\}$

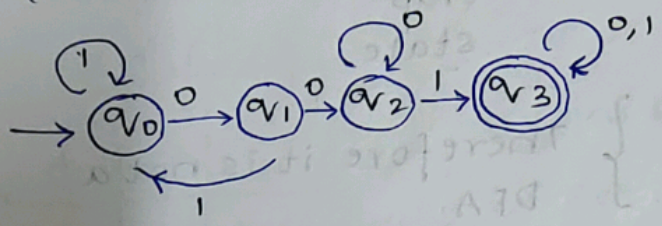


$L = \{$ set of all strings without substrings 001 $\}$

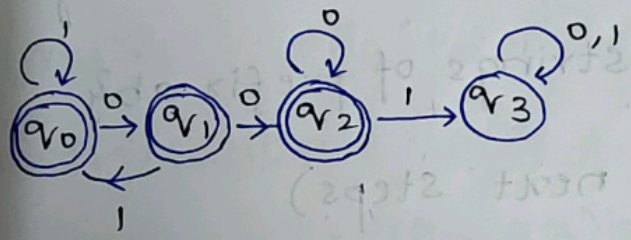
Step 1: First we need to design a DFA of $L(M)$ for the languages that consists of all

strings with 001 substring then we can complement it to get the DFA for the given language.

{001 0001 1001 00010 10010 00011, ...}



Step2: The DFA accepting the given language is



Ex: Σ^* {b, a, b, aa, ab, ba, abb...}

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

$L = \{ awa / w \in \{a, b\}^* \}$

03-01-2025

Design a DFA for the language awa

$L = \{ awa / w \in \{0, 1\}^* \}$

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

prefix

$L = x v w$

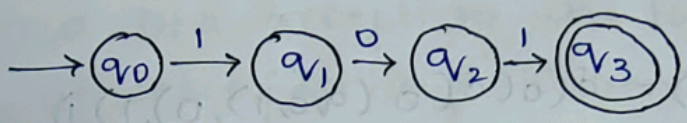
prefix suffix

10m

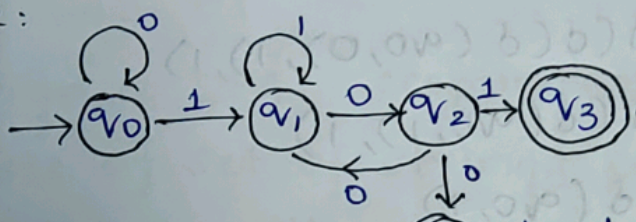
• Design a DFA accepting the language that consists of set of all strings with '010' as a substring over the alphabet {0, 1}.

L = {101, 0101, 1101, 00101, 01101, 10101, 11101, 1010, 1011, 10100, 10101, 10110, 10111, ...}

Step1:

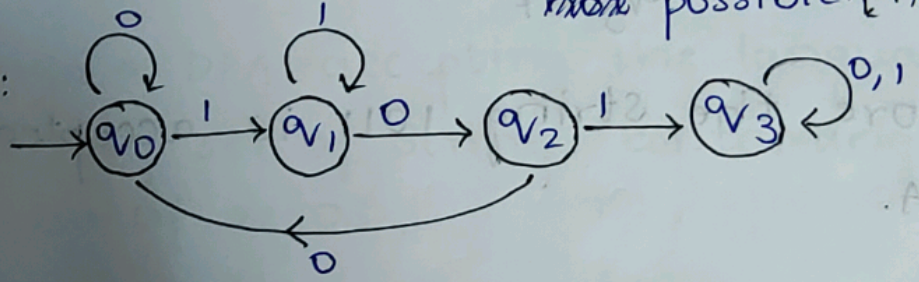


Step2:



dead state possible. [But no need here]

Step3:



Step4: The DFA for the given language is

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

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

$\Sigma = \{0, 1\}$ ↓ input symbols

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

Initial state = q_0 $F = \{q_3\}$

Step5:

$\delta^*(q_0, 1011) = \delta(\delta^*(q_0, 101), 1)$

$= \delta(\delta(\delta^*(q_0, 10), 1), 1)$

$= \delta(\delta(\delta(\delta^*(q_0, 1), 0), 1), 1)$

$= \delta(\delta(\delta(\delta(\delta^*(q_0, \epsilon), 1), 0), 1), 1)$

Step6:

$\delta^*(q_0, 1011) = \delta(\delta(\delta(\delta(q_0, 1), 0), 1), 1)$

$= \delta(\delta(\delta(q_0, 0), 1), 1)$

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

$= \delta(q_0, 1)$

$= q_3 \in F$

Therefore the string 1011 is accepted by the DFA.

Step7:

$\delta^*(q_0, 100) = \delta(\delta^*(q_0, 10), 0)$ $F = \{q_3\}$

$= \delta(\delta(\delta^*(q_0, 1), 0), 0)$

$= \delta(\delta(\delta(\delta^*(q_0, \epsilon), 1), 0), 0)$

$= \delta(\delta(\delta(q_0, 1), 0), 0)$

$= \delta(\delta(q_0, 0), 0)$

$= \delta(q_0, 0) = q_0 \notin F$

Therefore 100 is not accepted by the designed DFA.

1. Design a DFA accepting the language consists of set of all strings containing 000 as substring over {0,1}

2. Design a DFA accepting the language consists of set of strings containing that start with 01.

3. Design a DFA accepting the language consists of set of strings containing strings that start with 11.

4. Design a DFA accepting the language consists of set of strings containing with even no. of zeros.

5. Design a DFA accepting the language consists of set of strings containing with even no. of zeros only with two states.

6. Design a DFA accepting the language consists of set of strings. containing no. of zeros divisible by '3'

7. Design a DFA accepting the language consists of set of strings. containing even

a's and even b's.

8. Design a DFA accepting the language consists of all set of strings containing even a's and odd b's.

9. Design a DFA accepting the language consists of all set of strings containing even b's and odd a's.

10. Design a DFA accepting the language consists of set of strings containing odd a's and odd b's.

11. Binary number starting with '1' divisible by '5'.

12. Ending with 01.

04-01-2025

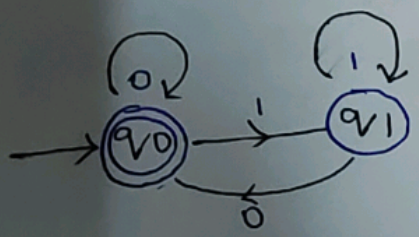
Q. Design a DFA that accept the strings that are divisible by 2 over the binary alphabet.

Sol:

n Decimal	Binary	(n%2) Remainder
0	0	0/2 = 0
1	1	1
2	10	0
3	11	1

$L = \{ \epsilon, 0, 10, 100, 0110, 1000, 1010 \ldots \}$

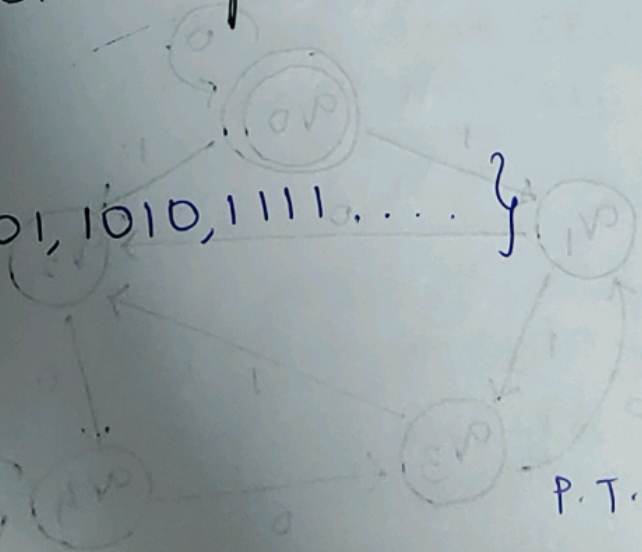
Decimal number	Binary number	remainder	
0	0	0	$q_0 \vdash^0 q_0$
1	01	1	$q_0 \vdash^0 q_0 \vdash^1 q_1$
~~2~~	~~010~~	~~2~~	
2	10	0	$q_0 \vdash^1 q_1 \vdash^0 q_0$
3	011	1	~~$q_0 \vdash^1 q_1 \vdash^1 q_1$~~
		no. of states	$q_0 \vdash^0 q_0 \vdash^1 q_1 \vdash^1 q_1$
16	10000		$q_0 \vdash^1 q_1 \vdash^0 q_0 \vdash^0 q_0 \vdash^0 q_0 \vdash^0 q_0$



→ Final Transition state

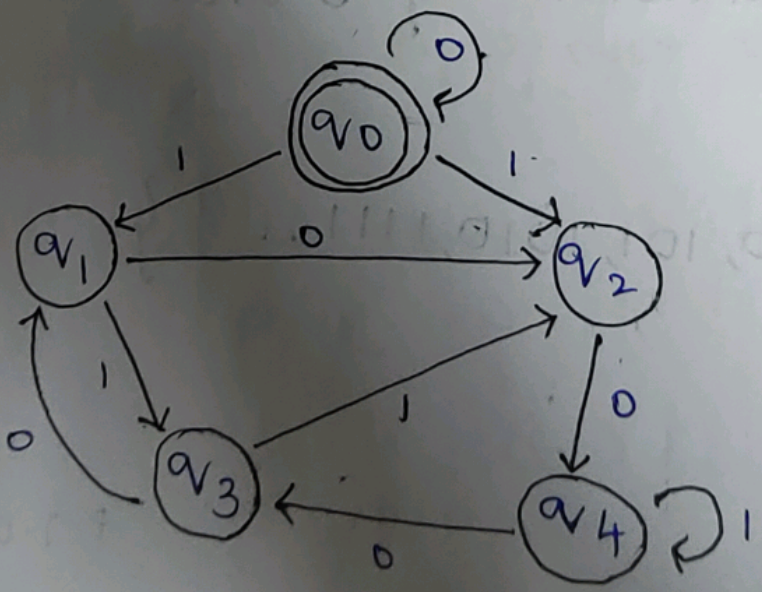
Q. Design a DFA that accepts the strings that are divisible by 5 over the binary alphabet.

Sol: $L = \{ \epsilon, 0, 101, 1010, 1111 \ldots \}$



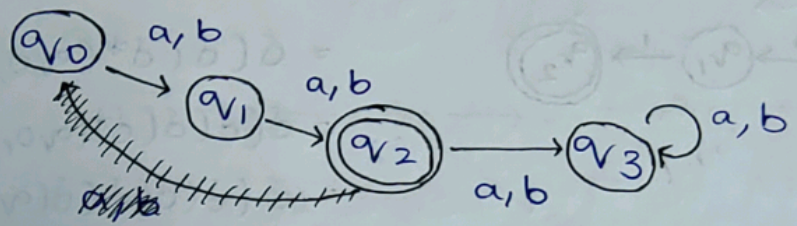
P.T.O

Decimal number	Binary number	Remainder	States
0	0	0	$q_0 \xrightarrow{0} q_0$
1	01	1	$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1$
2	10	2	$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$
3	011	3	$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_3$
4	100	4	$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_4$
5	101	0	$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_0$
6	110	1	
7	0111	2	
8	1000	3	
9	1001	4	
10	1010	0	

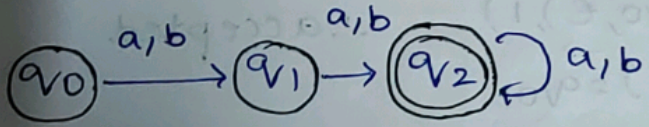


1. any no. of one's followed by any no. of 0's is followed by any no. of 2's.
2. over alpha $\Sigma = \{a, b\}$ length of string is 2.

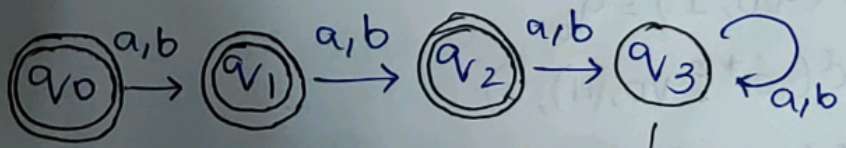
$L = \{ aa, ab, ba, bb \}$ [length ~~atleast~~ is 2] ≥ 2



atleast 2



atmost 2



for dead state we can give self loop

07-01-2025

Non deterministic Finite Automata (NFA)

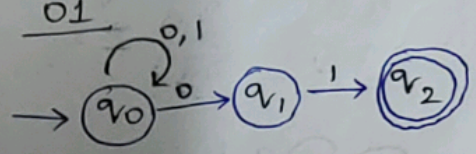
- It is represented essentially as DFA.

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

Ex: Design a DFA using NFA accepting all strings that end with '01'.

$L = \{01, 001, 101, 0001, 0101, 1001, 1101, 00001, \ldots\}$

01



$\delta^*(q_0, \epsilon) = q_0$

$\delta^*(q_0, 1) = \delta(\delta^*(q_0, \epsilon), 1)$

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

$\delta^*(q_0, 11) = \delta(\delta^*(q_0, 1), 1)$

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

$\delta^*(q_0, 110) = \delta(\delta^*(q_0, 11), 0)$

$= \delta(q_0, 0)$

$= (q_0, 0)\delta$

$= q_1$

$\delta^*(q_0, 100) = \delta(\delta^*(q_0, 10), 0)$

$= \delta(\delta(\delta^*(q_0, 1), 0), 0)$

$= \delta(\delta(\delta(\delta^*(q_0, \epsilon), 1), 0), 0)$

$\delta^*(q_0, 1101) = \delta(\delta^*(q_0, 110), 1)$

$= \delta(\delta(\delta^*(q_0, 11), 0), 1)$

$= \delta(\delta(\delta(\delta^*(q_0, 1), 1), 0), 1)$

$= \delta(\delta(\delta(\delta(\delta(q_0, \epsilon), 1), 1), 0), 1)$

$= \delta(q_1, 1) = q_2 \in F$

$\therefore$ accepted

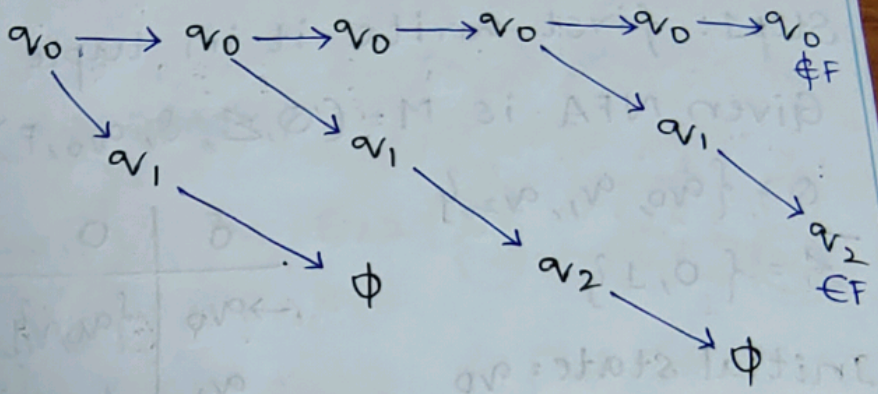
$= \delta(\delta(\delta(q_0,1),0)0)$

$= \delta(\delta(q_0,0),0) = \delta(q_1,0) = \phi$ ∴ not accepted

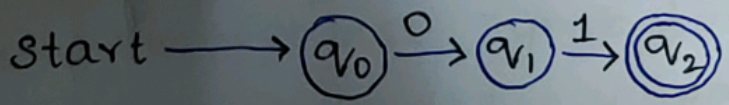
Transition Table:

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

The states an NFA is in during the processing of in-put sequence 00101



Ex: Design an NFA accepting all strings with '01' as substring.



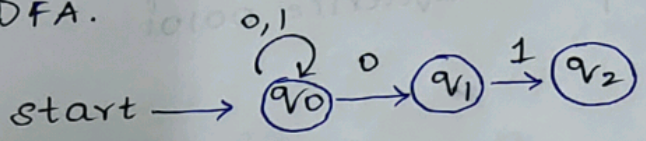
$L = \{01, 001, 010, 101, 011, 0001, 0010, 0100 \ldots\}$

$\Sigma^* = \{\in, 0, 1, 00, 01, 10, 11 \ldots\}$

Conversion of NFA to DFA:

- If NFA has n states, then DFA will have at most 2^n starts.

Example: Convert the following NFA to DFA.



Step1: first write it in tuple form.

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

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

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

Initial state: q_0

$F = \{q_2\}$

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

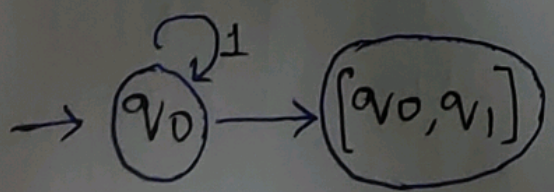
Step2: To construct equivalent DFA using Step1.

Initial state = q_0

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

$\delta'([q_0], 0) = \delta(q_0, 0) = \{q_0, 0\} = \{q_0, q_1\} = [q_0, q_1]$

$\delta'([q_0], 1) = \delta(q_0, 1) = [q_0]$



$\delta'([q_0, q_1], 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$

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

$= [q_0, q_1, q_2]$

$\delta'([q_0, q_1], 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = q_0 \cup q_2$

$= [q_0, q_2]$

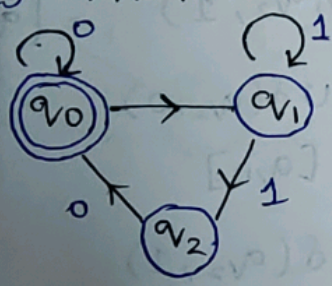
Transition table:

δ'	0	1
$[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_2]$
$[q_0, q_2]$	$[q_0 q_1]$	$[q_0]$

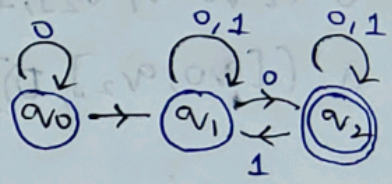
09-01-2025

NFA to DFA:

Ex1:

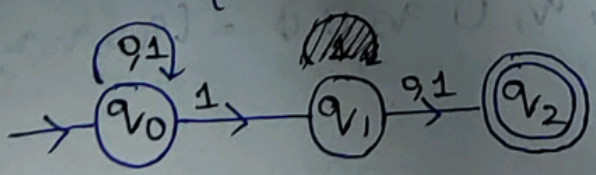


Ex2:



Example: Design an NFA for the language that accepts all strings over {0, 1} in which the second last symbol is always '1' then convert it to its equivalent DFA.

Sol: $L = \{10, 010, 110, 111, 11, 01, 0010, 0110, \ldots\}$.



$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$

Transition table

	0	1
A	A	AB
B	C	C
C	ϕ	ϕ

To design the equivalent DFA

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

$\Sigma = \{0, 1\}$, $q_0' = [q_0]$

$\delta'([q_0], 0) = \delta(q_0, 0) = [q_0]$

$\delta'([q_0], 1) = \delta(q_0, 1) = \{q_0, q_1\} = [q_0, q_1]$

$\delta'[[q_0, q_1], 0] = \delta(q_0, 0) \cup \delta(q_1, 0) = [q_0, q_2]$

$\delta'[[q_0, q_1], 1] = \delta(q_0, 1) \cup (q_1, 1) = [q_0 q_1 q_2]$

$\delta'([q_0 q_1 q_2], 0) = \delta(q_0 q_1 q_2, 0) = [q_0, q_2]$

$\delta'([q_0 q_1 q_2], 1) = \delta(q_0 q_1 q_2, 1) = [q_0, q_1 q_2]$

$\delta'([q_0, q_2], 0)] = \delta(q_0, 0) \cup \delta(q_2, 0)$

$= q_0 \cup \phi = [q_0]$

$\delta'[[q_0, q_2] 1] = \delta(q_0, 1) \cup \delta(q_2, 1)$

$= \{q_0, q_1\} \cup \phi = [q_0 q_1]$

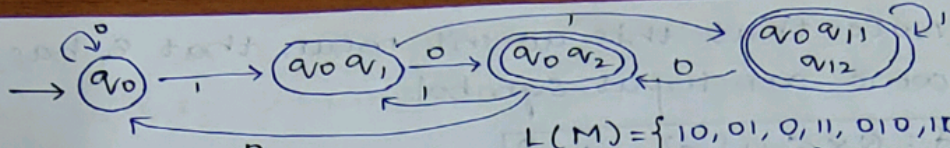
$\delta'([q_0, q_1, q_2], 1) = q_0, 0 \cup q_1, 0 \cup q_2, 0$

$q_0 \cup q_2 \cup \phi = [q_0, q_2]$

$\delta'([q_0, q_1, q_2], 1) = q_0, 1 \cup q_1, 1 \cup q_2, 1$

$q_0, q_1 \cup q_2 \cup \phi = [q_0, q_1, q_2]$

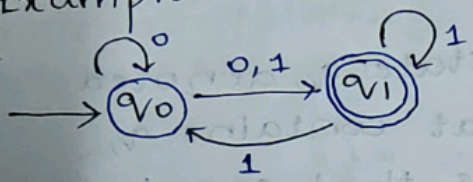
δ	0	1
$[q_0]$	$[q_0]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_2]$	$[q_0, q_1, q_2]$
$[q_0, q_2]$	$[q_0]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_2]$	$[q_0, q_1 q_2]$



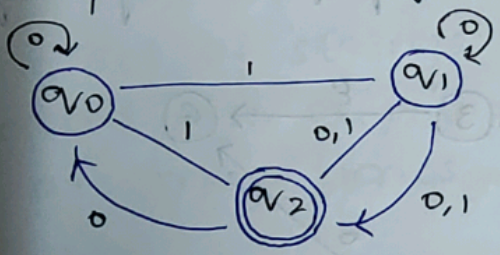
$L(M) = \{10, 01, 0, 11, 010, 1D110, 1111\}$

$L(M) = L(M')$

Example: Convert given NFA to DFA.



Example: Convert given NFA to DFA.



10-01-2025

Finite Automata with Epsillon transitions (ε-NFA)

*Epsillion NFA is defined as

- $M(Q, \Sigma, \delta, q_0, F)$
- $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$

* We extend the class of NFA's by allowing instantaneous (ϵ) transitions:

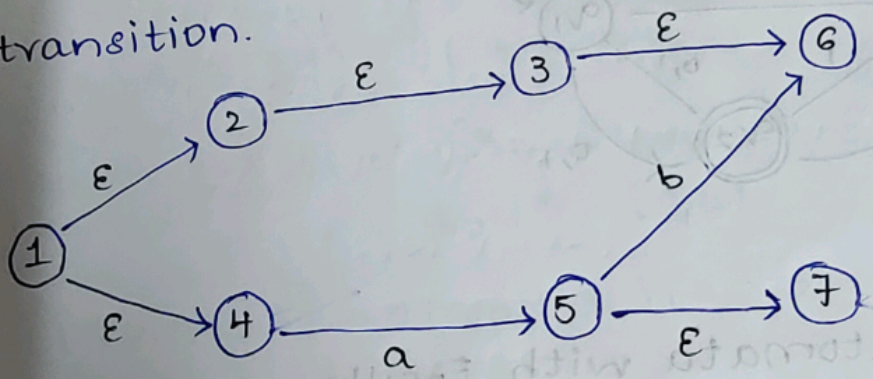
1. In diagrams such transitions are depicted by labelling the appropriate arcs with 2.

2. Note that this doesn't mean that ε has become an input symbol.

$$\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$$

ε - closure:

- The ε-closure of the state q, denoted ECLOSE(q) is the set that contains q, together with all states that can be reached starting at q by following only ε-transition.



ε - closure (1) = {1,2,3,6,4}

ε - closure (2) = {2,3,6}

ε - closure (3) = {3,6}

ε - closure (5) = {5,7}

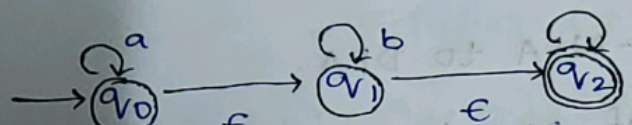
ε - closure (4) = {4}

ε - closure (6) = {6}

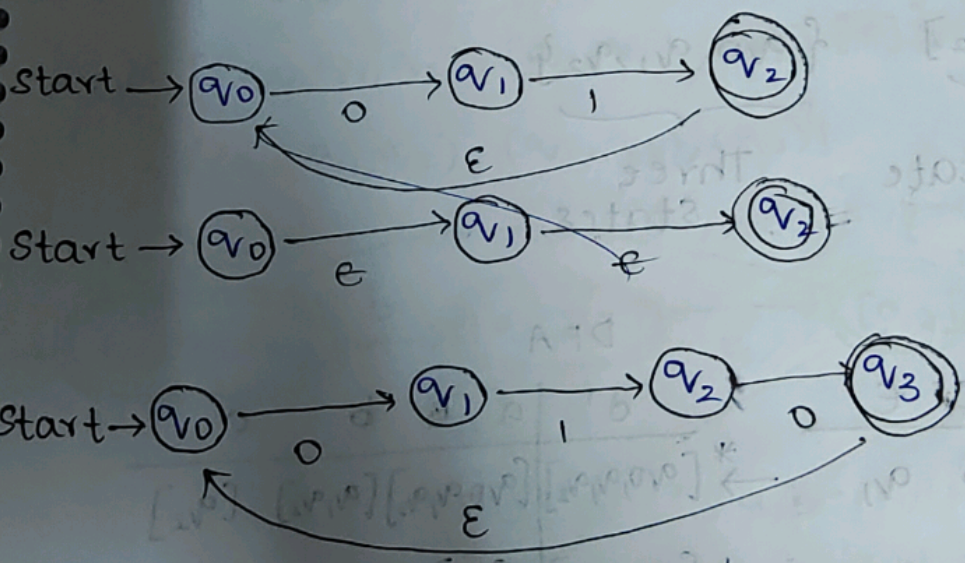
ε - closure (7) = {7}

Q) Design an ε-NFA for the following languages. Try to use ε-transitions to simplify the design.

- Set of strings that consists of zeros or more a's followed by zero or more b's followed by zero or more c's.

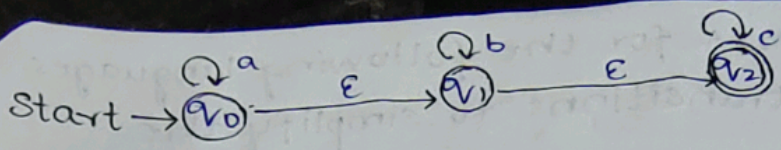


- The set of strings that consists of 01 repeated one or more times or 010 repeated one are more times.



One or more times '010' repeated.

- The set of strings consists of zero or more a's followed by zero or more b's followed by zero or more c's.



ϵ closure $(q_0) = \{q_0, q_1, q_2\}$
ϵ closure $(q_1) = \{q_1, q_2\}$
ϵ closure $(q_2) = \{q_2\}$

δ	a	b	c	ϵ
$\rightarrow q_0$	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
$*q_2$	ϕ	ϕ	q_2	ϕ

Conversion of ϵ-NFA to DFA:

* The start state of DFA is the ϵclose(start state of the ϵ-NFA)
* ϵclose (q_0) is the start state of DFA.
* ϵ close $(q_0) = \{q_0, q_1, q_2\}$ new state.

$[q_0\ q_1\ q_2]$ $\{q_0, q_1, q_2\}$

$[q_0\ q_1\ q_2]$ → Single state

$\{q_0, q_1, q_2\}$ → Three states..

ϵ-NFA

δ	a	b	c	ϵ
$\rightarrow q_0$	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
$*q_2$	ϕ	ϕ	q_2	ϕ

DFA

δ'	a	b	c
$\rightarrow *[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_1, q_2]$	$[q_2]$
$*[q_1 q_2]$	$[\phi]$	$[q_1, q_2]$	$[q_2]$
$*[q_2]$	$[\phi]$	$[\phi]$	$[q_2]$
ϕ	$[\phi]$	$[\phi]$	$[\phi]$

$\delta'(\{q_0, q_1, q_2\}, a) = \epsilon\text{closure}\{\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)) = \epsilon\text{closure}\{q_0\}$

$= \epsilon$ closure $(q_0) = \{q_0, q_1, q_2\}$

ϵ-closure $(\phi \cup q_1 \cup \phi) = \epsilon$ closure $(q_1) = [q_1 q_2]$ new state

ϵ-closure $(\phi \cup \phi \cup q_2) = \epsilon$ closure $(q_2) = [q_2]$ new state

$\delta'([q_1, q_2], a) = \epsilon$ closure $(\delta(q_1, a) \cup \delta(q_2, a))$

$= \epsilon$ closure $(\phi \cup \phi) = \phi$

$\delta'([q_1, q_2], b) = [q_1, q_2]$

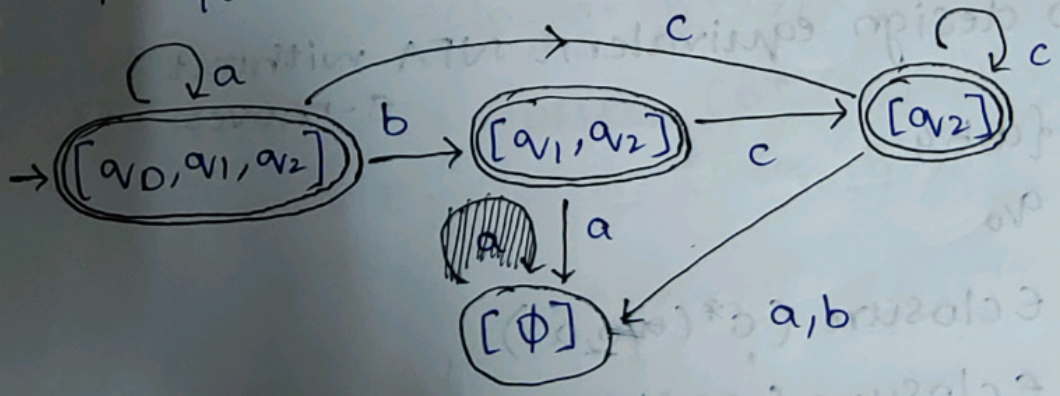
$\delta'([q_1, q_2]), c) = [q_2]$

DFA :

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

$Q' = \{[q_0, q_1, q_2], [q_1, q_2], [q_2] [\phi]\}$

$F = \{(q_0, q_1, q_2), (q_1, q_2), (q_2)\}$



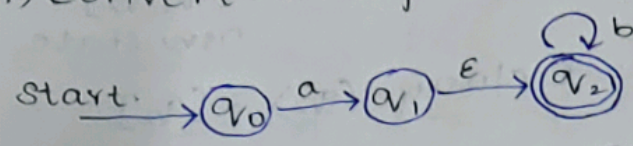
$L = \{ e, a, b, c, ab, ac, bc, \ldots \}$

$L(M) = L(M')$

So, it is correct.

11-01-2025

9) Convert the following ε-NFA to NFA.



Sol: ε-NFA is $M = (Q, \Sigma, \delta, q_0, F)$

Step 1: $Q = \{q_0, q_1, q_2\}$

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

ε-NFA

δ	a	b	c
→ q_0	q_1	ϕ	ϕ
q_1	ϕ	ϕ	q_2
* q_2	ϕ	q_2	ϕ

$F = \{q_2\}$

ε closure $(q_0) = \{q_0\}$

ε closure $(q_1) = \{q_1, q_2\}$

ε closure $(q_2) = \{q_2\}$

Step 2:

NFA - To design equivalent NFA without ε-moves.

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

$q_0' = q_0$

$\delta'(q_0, a) = \varepsilon\text{closure}(\delta^*(q_0, a))$

$= \varepsilon\text{closure}(\delta(\delta^*(q_0, \varepsilon), a)$

$= \varepsilon\text{closure}(\delta(\varepsilon\text{closure}(q_0), a)$

$= \varepsilon\text{closure}(\delta(q_0, a))$

$= \varepsilon\text{closure}(q_1) = \{q_1, q_2\}$

$\delta'(q_0, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), b)$

$= \epsilon\text{-closure}(\delta(q_0, b) = \epsilon\text{-closure}(\phi) = \phi$

NFA

δ'	a	b
q_0	$\{q_1, q_2\}$	ϕ
q_1	ϕ	q_2
q_2	ϕ	q_2

$\delta'(q_1, a) = \epsilon$ closure $(\delta^*(q_1, a))$

$= \epsilon$ closure $(\delta(\delta^*(q_1, \epsilon), a))$

$= \epsilon$ closure $(\delta(\epsilon\text{-closure}(q_1)), a)$

$= \epsilon$ closure $(\delta(\{q_1, q_2\}, a))$

$= \epsilon$ closure $(\delta(q_1, a) \cup \delta(q_2, a))$

$= \epsilon$ closure $(\phi \cup \phi) = \phi$

$\delta'(q_1, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), b))$

$= \epsilon\text{-closure}(\delta(q_1, b) = \epsilon\text{-closure}(\delta(\{q_1, q_2\}, b)$

$\delta'(q_2, a) = \epsilon\text{-closure}(\delta^*(q_2, a))$

$= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), a))$

$= \epsilon\text{-closure}\ \delta(q_2, a).$

$\delta'(q_2, b) = \epsilon\text{-closure}(\delta^*(q_2, b))$

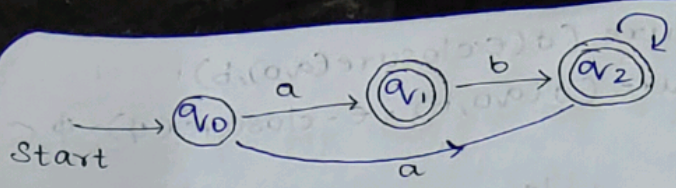
$= \epsilon\text{-closure}(\delta(q_2, b))$

$= q_2.$

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

$F = \{q_1, q_2\}$

∵ Wherever there is epsillion present in the end of the states take the Corresponding states as final states.

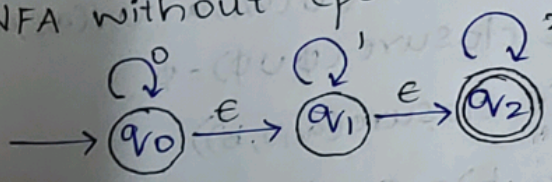


$L(M') = \{a, ab, abb, abbb, \ldots\}$

$= \{ab^n / n \geq 0\}$

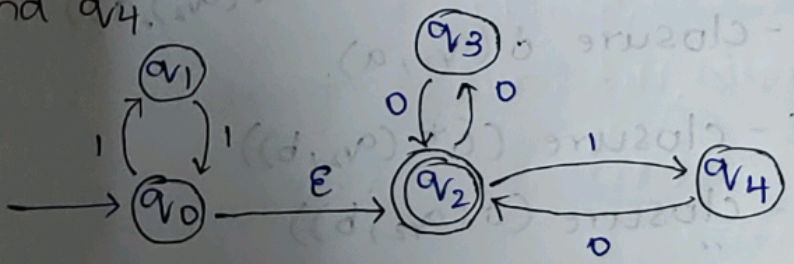
∵ Now we can convert this into DFA.

Q) Convert the given NFA with epsillon to NFA without epsillon.

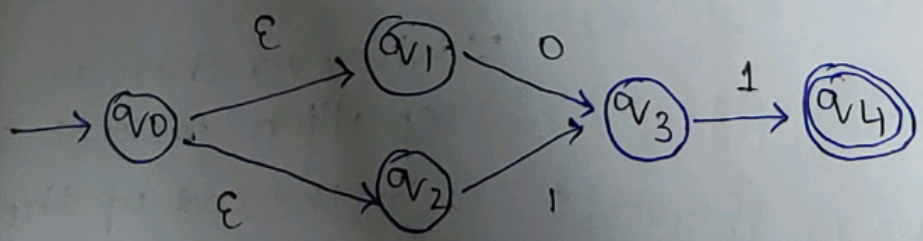


Q) Convert the epsillon - NFA to NFA.

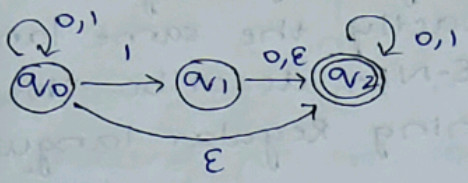
Consider the example having states q_0, q_1, q_2, q_3 and q_4.



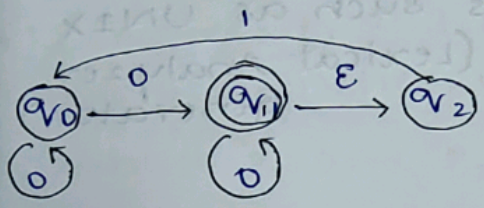
Q) Convert the epsillon-NFA to DFA.



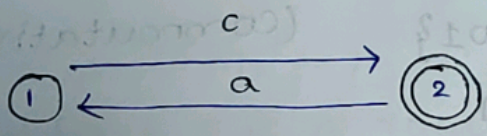
Q) Convert the following ϵ-NFA to NFA and then to DFA.

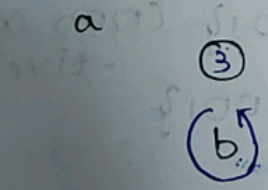


Q) Convert the given NFA with epsillon to DFA.



Q) Convert the given ϵ-NFA to DFA





16-01-2025

MODULE-2

- Regular Expression
- Operators of RE
- Building RE
- Languages of RE
- Properties of RL

} Overview of Mod 2

→ FA → RE RE → FA.

What is RE?

*It is type of language defining notations. They can define exactly the same languages that NFA, DFA or ε-NFA describe. They are capable of defining Regular languages.

* Applications of RE:
  - Search commands such as UNIX
  - LEX and FLEX. (Lexical Analyzer Generator.)

Operators of RE:

1. Union: LUM

   $L = \{001, 10, 11\}$ $M = \{\varepsilon, 01\}$

   $LUM = \{\varepsilon, 01, 10, 11, 001\}$ (Commutative)

2. Concatenation: L.M

   $L.M = \{10, 11, 001, 1001, 1101, 00101\}$ (non commutative).

   $M.L = \{10, 11, 001, 0110, 0111, 01001\}$

3. Star (or) Kleene closure:

   If $L = \{0, 1\}$

   $L^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$

   $L^* = \{L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots\}$

   $\phi^* = \{\varepsilon\}$ $\phi^0 = \{\varepsilon\}$ (ϕ^i, $i \geq 1$ is ϕ)

   $\phi^+ = \phi$ *ε- null string / empty string

* ϕ - empty set: 0-length string

Definition of a Regular Expression:

R is a regular expression if it is:

1. a for some a in the alphabet Σ, standing for the language $\{a\}$
2. ε, standing for the language $\{\varepsilon\}$
3. ϕ, standing for the empty language.
4. $R_1 + R_2$ where R_1 and R_2 are regular expressions and + signifies union
5. $R_1 R_1$ where R_1 and R_2 are regular expression and this signifies concatenation.
6. R^* where R is a regular expression and signifies closure.
7. (R) where R is a regular expression then the parenthesized R.

Precedence:

Tightest — ( )

Star ("*")

Concatenation (".", " ")

Union ("∪", "+", "|")

Loosest

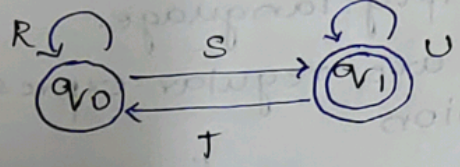
priority order

Example:

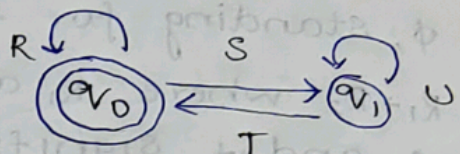
$R_1 * R_2 \cup R_3 = ((R_1 *) R_2) \cup R_3$

Write RE for the given language:

- Beginning with 1: 1(0+1)*
- Ends with 1: (0+1)*1
- Substring 1: (0+1)*1(0+1)*
- Even number of a's: (b+ab*a)*



(R+SU*T)*SU*



(R+SU*T)*

Beginning with 1:

L={1, 10, 11, 100, 101, 110, 111...}

1{either 0 or 1}

↑ any no. of times.

1(0+1)*

Ends with 1:

L={1, 01, 11, 001, 011, 101, 111...}

{either 0 or 1}1

(0+1)*1

Substring 1:

L={010, 011, 110, 111, ...}

{either 0 or 1}1{either 0 or 1}

(0+1)*1(0+1)*

Even no. of a's:

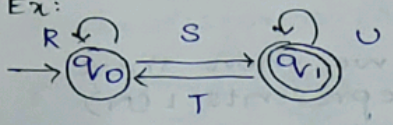
L={ϵ, b, aa, aab, baa, aba, aaaa, abaa, aaba....}

(b+ab*a)*

↓

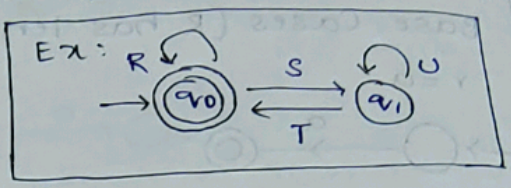
for ϵ.

Ex:



L = { S, RS, RRS, SU, RSU, RRSU, RRSUU, ... STS, STRS, STRSU, ... }

$(R + SU^*T)^*SU^*$



Write RE for the given language:

1. { w/w has exactly a single 1 } $R = 0^*10^*$
2. The set of all strings of 0's and 1's ending with 00 $R(0+1)^*00$
3. The set of all strings beginning with 0 and ending with 1. $R = 0(0+1)^*1$
4. $L = \{ \epsilon, aa, aaaa, aaa\,aaa, \ldots \}$ $R(aa)^*$
5. Define RE to denote any no. of 0's followed by any number of 1's followed by any no. of 2's $R = 0^*1^*2^*$
6. { w/w has length ≥ 3 and its 3rd symbol is 0 }. $R = 000(0+1)^* + 010(0+1)^* + 100(0+1)^* + 110(0+1)^* = (0+1)(0+1)0(0+1)^*$

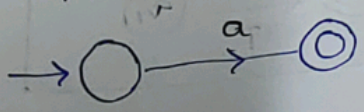
17-01-2025

*Given regular expression R, we show there exits NFA N such that R represents L(N)

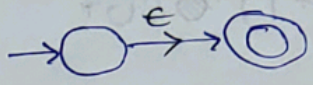
Induction on the length of R:

Base Cases (R has length 1):

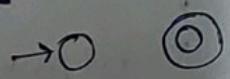
r = a



r = ϵ

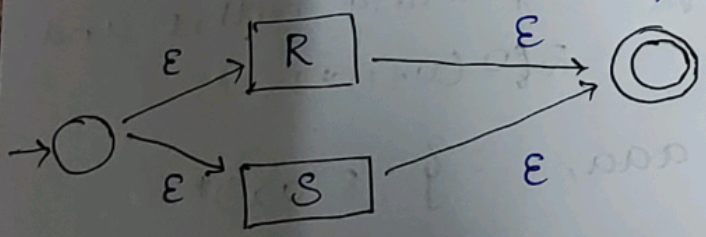


r = ϕ

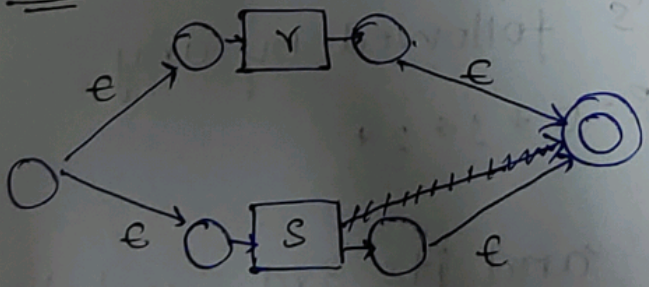


Thompson's Construction:

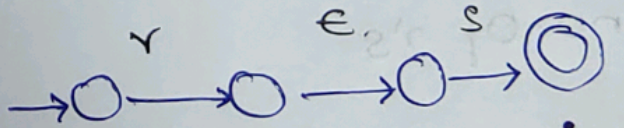
UNION THEORM - (R+S)



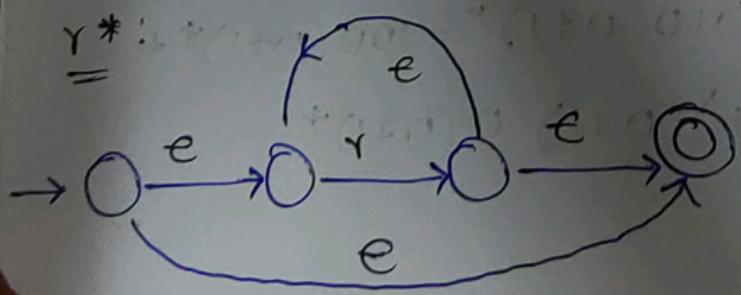
r+s : L(r) U L(s)



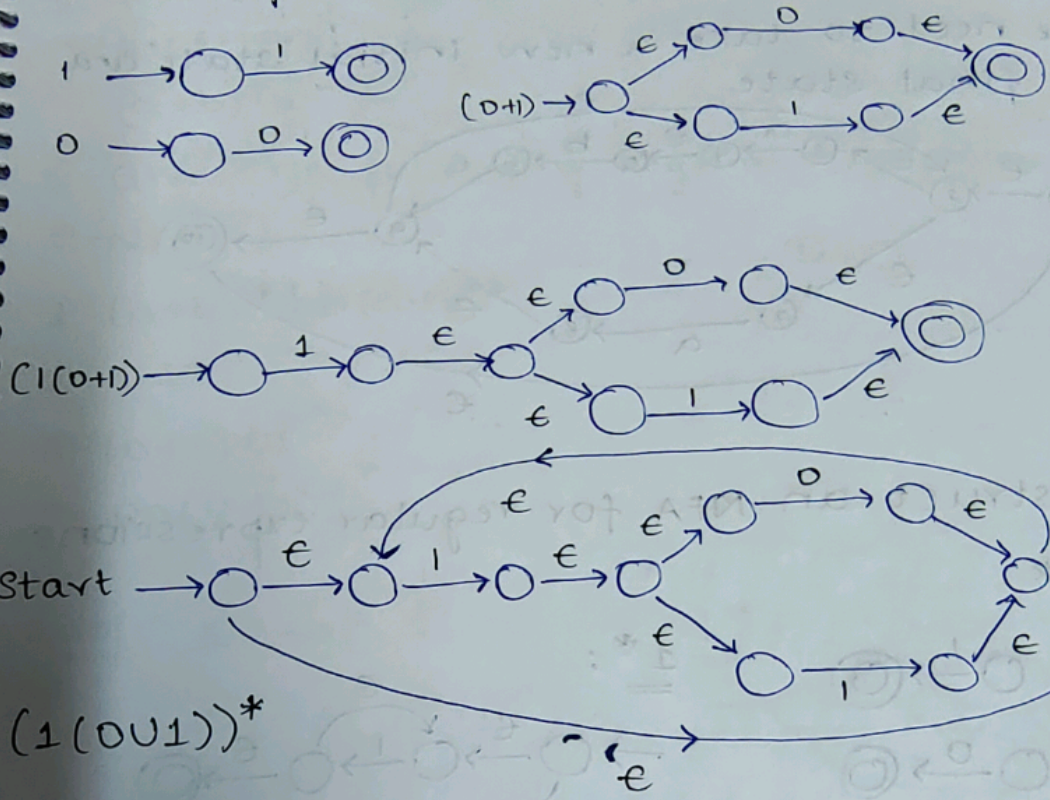
rs :



r*:

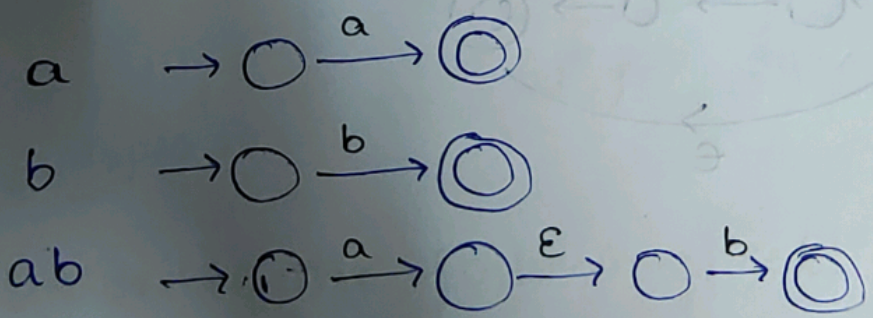


Q. Transform $(1(0 \cup 1))^*$ to an NFA.

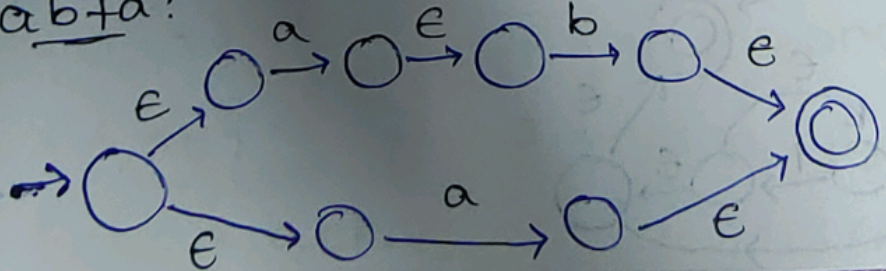


Q. RE to ε-NFA example

Convert $R = (ab+a)^*$ to an NFA.

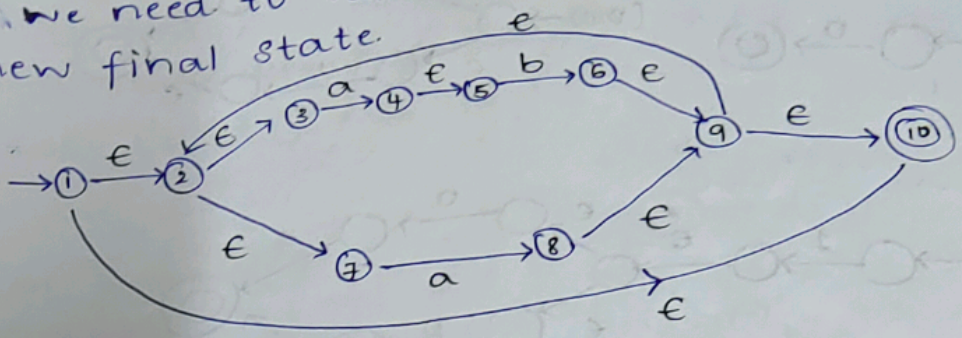


ab+a:



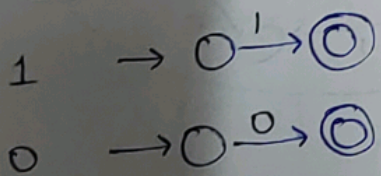
(ab+a)*

We need to take a new initial state and new final state.

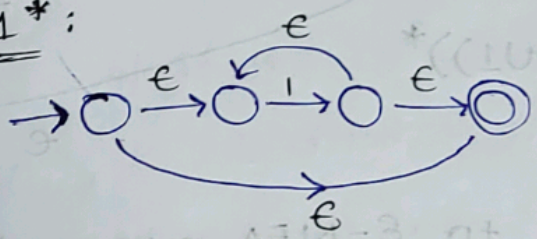


Q. Construct an NFA for regular expressions

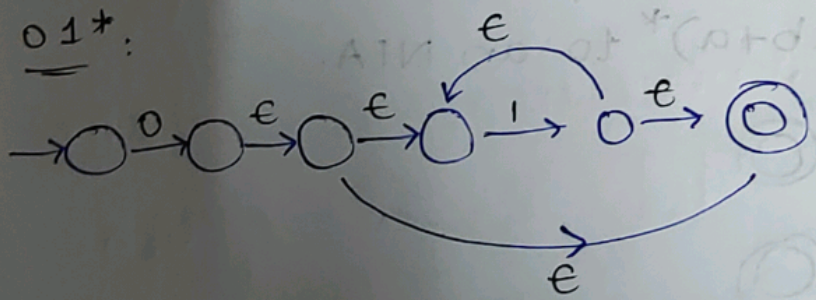
1+01*



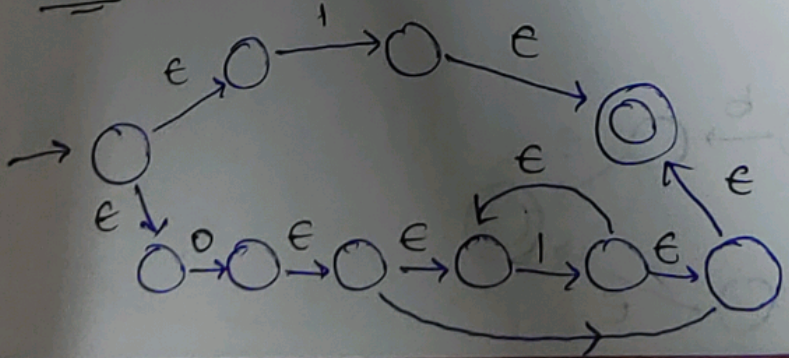
1*:



01*:



1+01*:



Q. Give
expres
(a+b

Q. Give
ε-NFA

1. (a
2. O
3. (
4.

2.

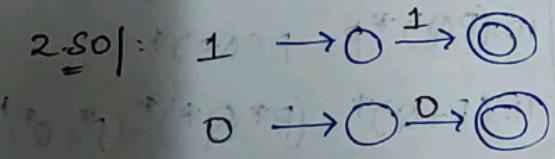
Q. Give a ϵ-NFA for the following regular expression.

$(a+b)^*a$

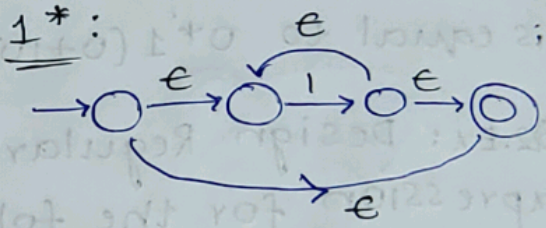
Q. Give the following regular expressions ϵ-NFA.

1. $(a+b)^*bb(a+b)^*$
2. 01^*
3. $(0+1)01$
4. $00(0+1)^*$

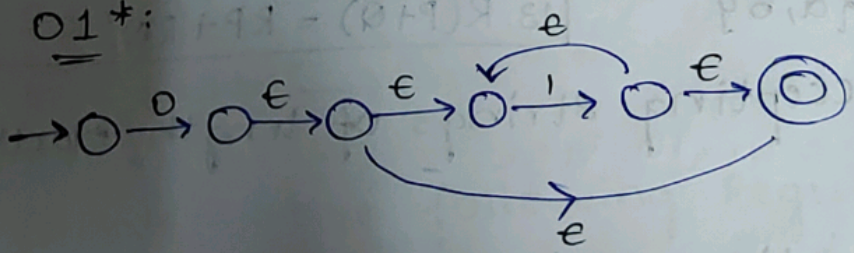
2. 1:



1*:



01*:



18-01-2025

Describe the following sets as Regular expressions:

1. $\{0,1,2\}$
2. $\{\wedge, ab\}$
3. $\{abb, a, b, bba\}$
4. $\{\wedge, 0, 00, 000, \ldots\}$
5. $\{1, 11, 111, 1111, \ldots\}$

1 Sol: 0+1+2

2 Sol: ϵ+ab (or) ϵab

3 Sol: abb + a + b + bba

4 Sol: $\{0^*\}$

5 Sol: $\{1\}^+$

Identities of Regular expressions:

1. $\phi + R = R$
2. $\phi R + R\phi = \phi$
3. $\epsilon R = R\epsilon = R$
4. $\epsilon^* = \epsilon$ and $\phi^* = \epsilon$
5. $R + R = R$
6. $R^* R^* = R^*$
7. $RR^* = R^*R$
8. $(R^*)^* = R^*$
9. $\epsilon + RR^* = \epsilon - R^*R = R^*$
10. $(PQ)^*P = P(QP)^*$
11. $(P+Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$
12. $(P+Q)R = PR + QR$ and
13. $R(P+Q) = RP + RQ$.

Q1. An example proof using identities of Regular expressions:

Prove that $(1+00^*1) + (1+00^*1)(0+10^*1)(0+10^*1)$ is equal to $0^*1(0+10^*1)^*$

Q2. Ex: Design Regular expression for the following languages over $\{a, b\}$

1) Language accepting strings of length exactly 2.

2) Language accepting strings of length atleast 2.

3) Language accepting strings of length atmost 2.

1 Sol: aa, bb, ab, ba = (a+b)(a+b)

2 Sol: aa, bb, ab, ba, aaa... => (a+b)(a+b)$(a+b^*)$

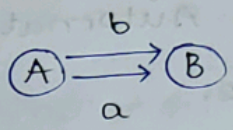
3 Sol: ϵ, a, b, aa, ab, ba, bb

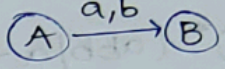
$= \epsilon + a + b + aa + ab + ba + bb$

$\Rightarrow (\epsilon + a + b)(\epsilon + a + b)$.

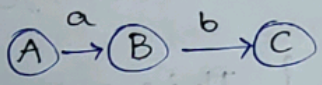
Conversion of Regular expressions to finite Automata.

- $(a+b)$

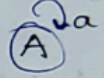


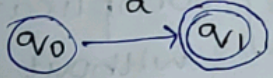


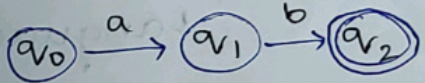
- (ab)

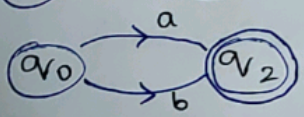


- a^*

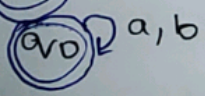


RE = a → 

RE = ab → 

RE = a+b → 

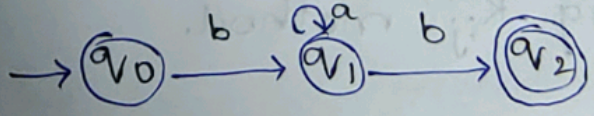
RE = (a^*) → 

RE = $(a+b)^*$ → 

Convert the following Regular expressions to their equivalent Finite Automata.

1. ba^*b
2. $(a+b)c$
3. $a(bc)^*$

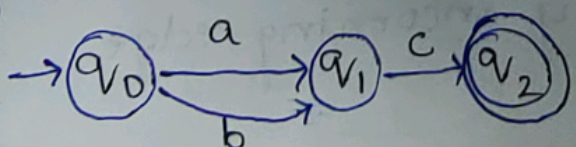
Sol: $L = \{bb, baab, baaab, \ldots\}$



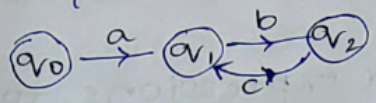
Sol: $L = \{ac, bc\}$

$ac + bc$

$(a+b)c$ ∵ By using distributive.



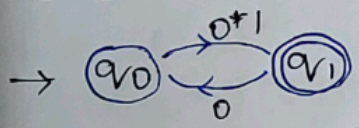
Sol: $L = \{a, abc, abcbc, \ldots\}$

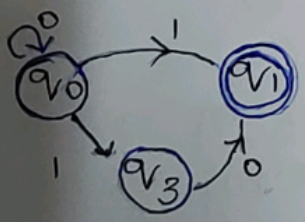


Convert the following regular expression into equivalent finite Automata.

1. $(a/b)^*(abb/a+b)$
2.
3. $0^*1 + 10$
4. $10+(0+11)0^*1$

Sol: $0^*1 + 10$





} This way is for complicated structures without using Languages.

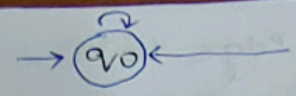
21-01-2025

Finite Automata to Regular expression.

1. State Elimination method
2. using Arden's Theorm
3. using $R_{ij}^{(k)}$ method.

1. State Elimination method:

* There should not be any incoming edge to the initial state.



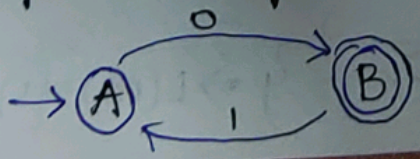
* If there is an incoming edge to the intial state (q_0) then include a new initial state (q_i) and give ϵ-transition from $q_i \xrightarrow{\epsilon} q_0$

* There should not be multiple final states. If there are multiple final states then make them as non-final states and include a new final state and give ϵ-transition to this new final state from the previous final states.

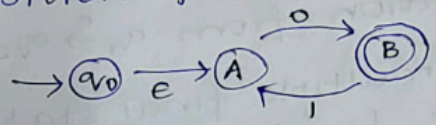
* There should not be any outgoing edge from the final state. If there is an out going edge from the final state (q_f) include new final state q_{final} and give ϵ-transition from $q_f \longrightarrow q_{final}$.

* Eliminate all intermediate states one by these states may be eliminated in any order at the end only one initial state going one final state will be left the cost of transition is called regular expression.

Problem: find the Regular expression for the following finite Automata.

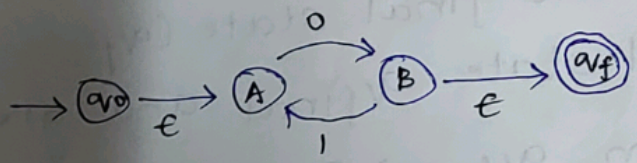


Step1: There is an incoming edge to the initial state we have to take an ε-transition from new initial state q_0 to A.

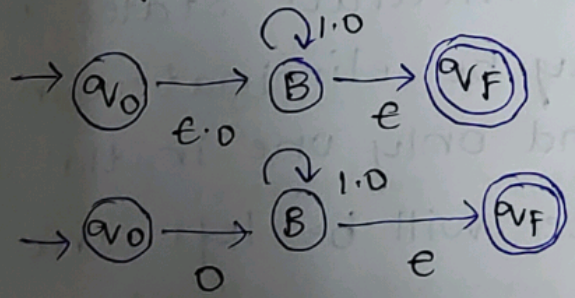


Step2: There are no multiple ~~states~~ final states.

Step3: See if there is an outgoing edge from the final state. Here it is there, make it non final and make ε transition from it to the new final state.

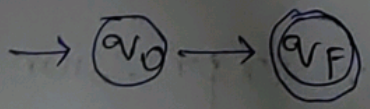


Step4: By eliminate A



By elimate B.

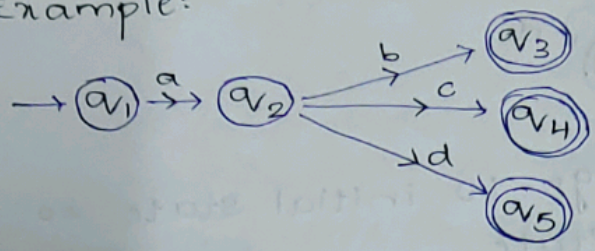
$0(10)^*\varepsilon$



final RE = $0(10)^*$

$L(r) = \{0, 00, 0101 0, 0101010 \ldots\} \Rightarrow L(m) = \{0, 010 \ldots\}$

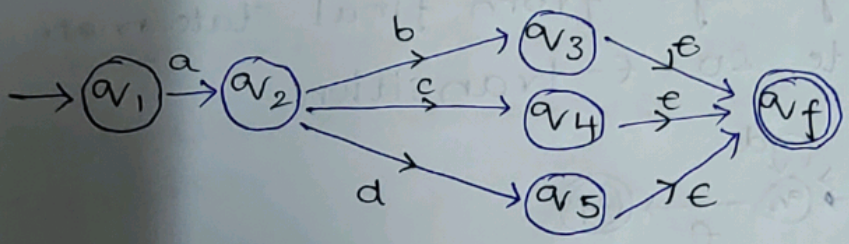
Example:



Step 1: no incoming edge to initial state.

Step 2: check for the multiple final state.

Yes, we have multiple final state. So, we need to make one final state.



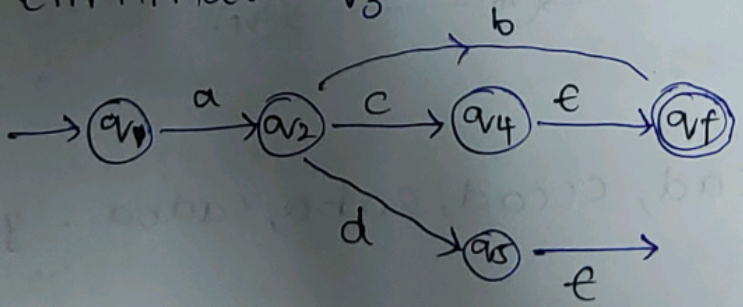
RE = a·(b+c+d)

L(r) = {a, b, ac, ad}

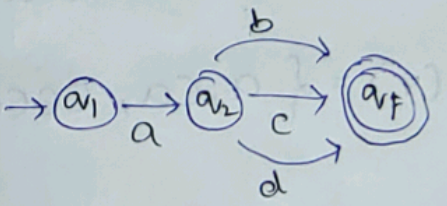
Step 3: no outgoing edge from final state so no edge.

Step 4: eliminate all intermediate states

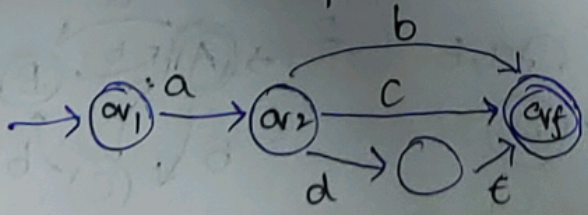
eliminate q_3.



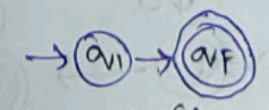
eliminate q_5.



eliminate q_4

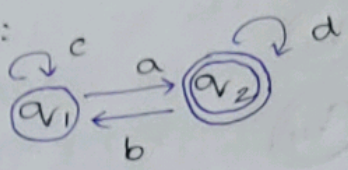


eliminate q_2.

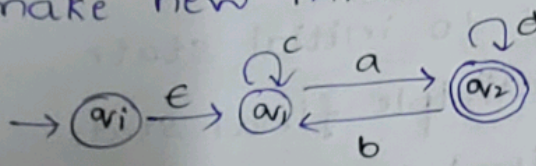


a·(b+c+d)

Example:

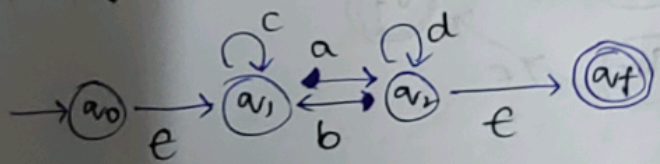


Step 1: incoming edge to initial state so make new initial state.



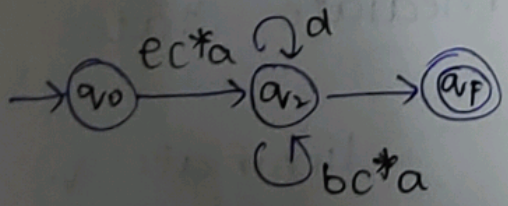
Step 2: no multiple final states no change.

Step 3: outgoing edge from final state make new final state so ϵ-transition.

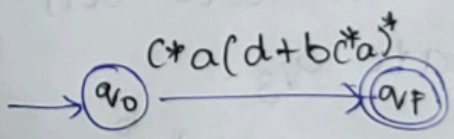


Step 4: eliminate all intermediate states

eliminate q_1.



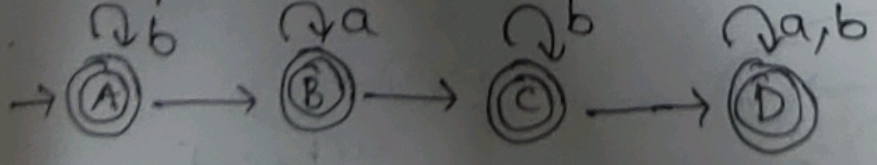
eliminate q_2



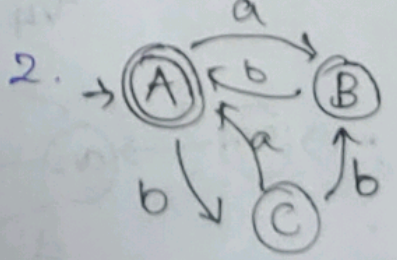
$L(r) = \{ a, ca, ccca, cad, cccad, caba, cadba \ldots \}$

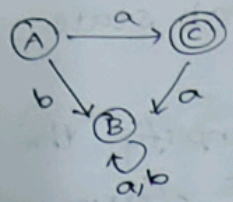
Example:

1.



2.



3. 

Converting DFA to Regular expression

(Methods)

Arden's method

State Elimination.

Arden's Theorm:

- It is used to convert DFA to its regular expression.
- It states that P and Q be two regular expressions over Σ. If P doesnot contain a null string ϵ, then $R = Q + RP$ has a unique solution i.e., $R = QP^*$

Steps:

To convert a given DFA to its regular expression using Arden's theorm.

Step 1:

(i) Form an equation for each state considering the transitions which comes towards the state (i.e., based on incoming edges).

(ii) Add ϵ in Equation of initial state.

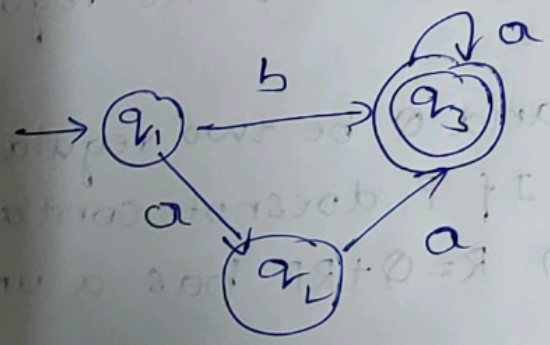
Step 2:
Use Arden's theorem to simplify the Eq's & bring final state in this form

$R = Q + RP$ { R = final state }

then req $\boxed{R = \emptyset QP^*}$

→ If there are multiple final states find R's individually $[R_1, R_2 \cdots]$ and add them for required Reg EXP.

→



$q_1 = \epsilon$ —① { for initial state ε is added

$q_2 = q_1 a$ —②

$q_3 = q_1 b + q_2 a + q_3 a$ —③

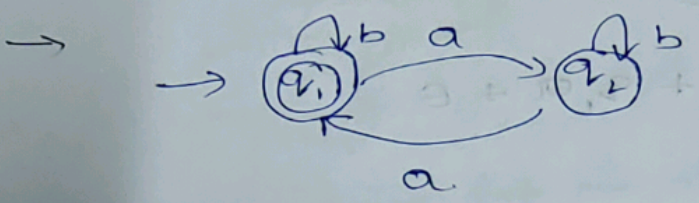
Solving ①, ② & ③

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

$= b + aa + q_3 a$ [$R = Q + RP$

$q_3 = (b+aa)\,a^*$ $\quad [\because R = Qp^*]$

$L(M) = \{b, ba, baa \cdots aa, aaa \cdots\}$



$q_1 = q_2a + q_1b$

$q_1 = q_1b + q_2a + \epsilon$ ~ (1) { ϵ is added to initial state

$q_2 = q_1a + q_2b$ ~ (2)

Simplify (1) & (2) using Arden's theorem

$\Rightarrow$ Applying Arden's for EQ (2) Alone

~~$q_1 = (q_1a + q_2b)\,a + q_1$~~

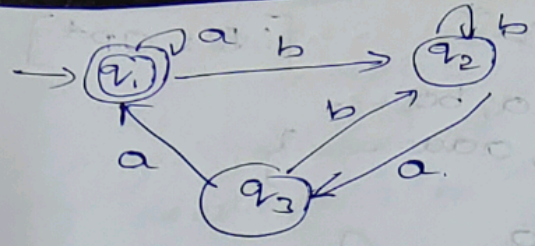
$q_2 = q_1ab^*$ — (3)

from (1) & (3)

$q_1 = q_1b + q_1ab^*a + \epsilon$

$q_1 = ~~q_1\phi~~ ~~(ab^*a + \epsilon)\,b^*~~ q_1(b + ab^*a) + \epsilon$

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



$$q_1 = q_3a + q_1a + \epsilon$$

$$q_3 = q_2a$$

$$q_2 = q_1b + q_2b + q_3b$$

$$q_2 = q_1b + q_2b + q_2ab$$

$$= q_2(b+ab) + q_1b$$

$$\boxed{q_2 = q_1b(b+ab)^*}$$

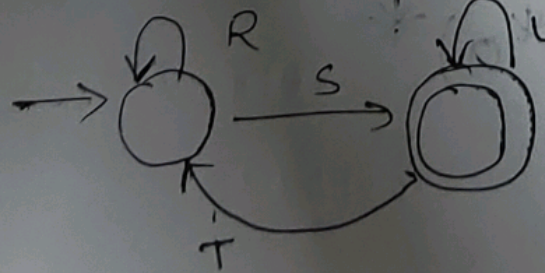
$$\boxed{q_3 = q_1b(b+ab)^*a}$$

$$q_1 = q_1b(b+ab)^*aa + q_1a + \epsilon$$

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

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

1)



$$\boxed{(R+SU^*T)^*SU^*}$$

2) →

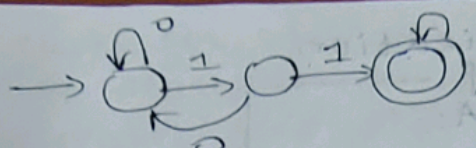
3)

4)

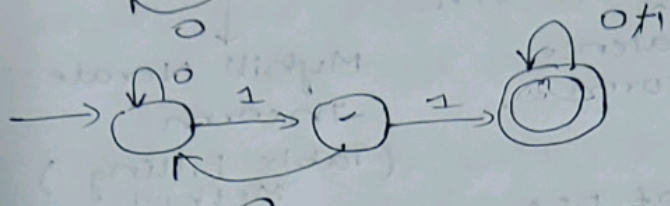
5)

6

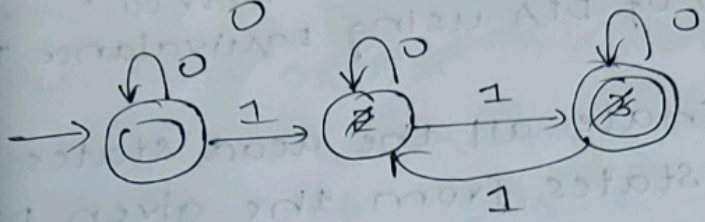
2)



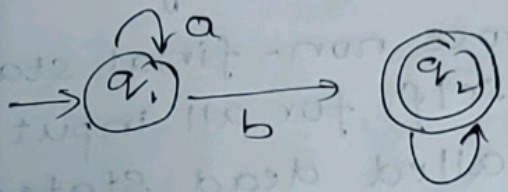
3)



4)

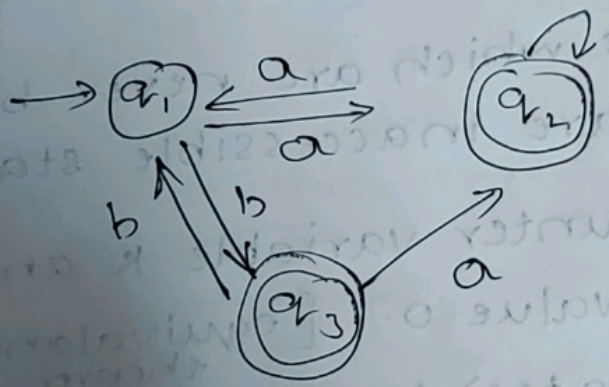


5)



$(a^*b)(a \cup b)^*$

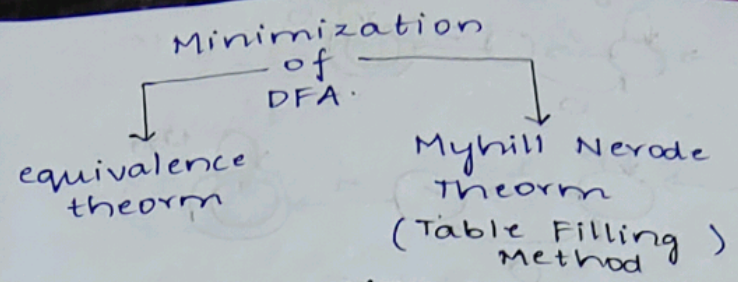
6)



~~7)~~

24-01-2025

Minimization of DFA:



Minimization of DFA using Equivalance Theorm

Steps:

Step 01: Eliminate all the dead states and in accessible states from the given DFA.

- Dead state: All those non-final states which transit to itself for all input symbols in Σ are called dead states.

Step 02:

- All those states which are never be reachable those are inaccessible states.

Step 03: - Take a counter variable k and initialize it with value 0. [equivalance theorm]

- Divide Q (set of states) into two sets such that one set contains all the non-final states and other set contains all final states.
- This partition is called P_0.

Step 04: - Increment k by 1.

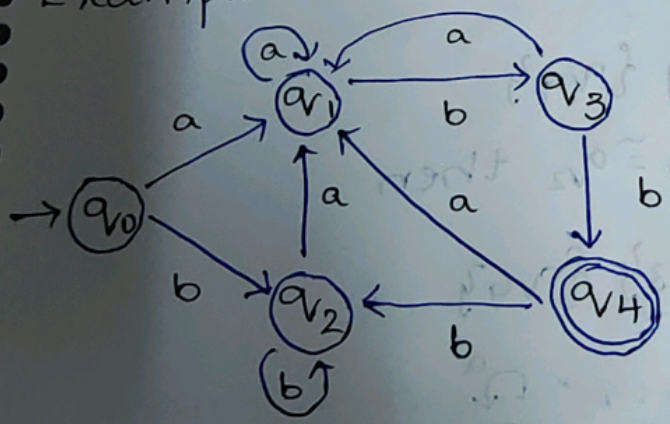
- Find P_k by partitioning the different sets of P_{k-1}.

- In each set of P_{K-1}, consider all the possible pair of states within each set and if the two states are distinguishable, partition the set into different sets in P_K.
- Two states q_1 and q_2 are distinguishable in partition P_K for any input symbol 'a', if $\delta(q_1, a)$ and $\delta(q_2, a)$.

Step 05 : • All those states which belong to the same set of equivalent.
- The equivalent states are merged to form a single state in minimal DFA.

* no. of states in minimal DFA = no. of sets in P_K.

Example: Minimize the following DFA.



Sol:

δ	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
* q_4	q_1	q_2

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$P_0 = \{q_0, q_1, q_2, q_3\}\ \{q_4\}$

$(q_0, q_1) = (\delta(q_0, a), \delta(q_1, a)) = (q_1\ q_1)$ $\because q_0 \cong q_1$

$(\delta(q_0, b), \delta(q_1, b)) = (q_2, q_3)$

$(q_0, q_2) \Rightarrow (\delta(q_0, a), \delta(q_2, a)) = (q_1, q_1)$

$(\delta(q_0, b), \delta(q_2, b)) = (q_2, q_2)$

same set $\Rightarrow q_0 \cong q_2$

$(\delta(q_0, a)), \delta(q_3, a)) = (q_1, q_1)$

$(\delta(q_0, b)), \delta(q_3, b)) = (q_2, q_4)$

$q_0 \not\cong q_3$ then

$P_1 = \{q_0\ q_1\ q_2\}\{q_3\}\{q_4\}$

again compare then

$(\delta(q_0, b), \delta(q_1, b)) = (q_2, q_3)$

not equivalent $q_0 \not\cong q_1$

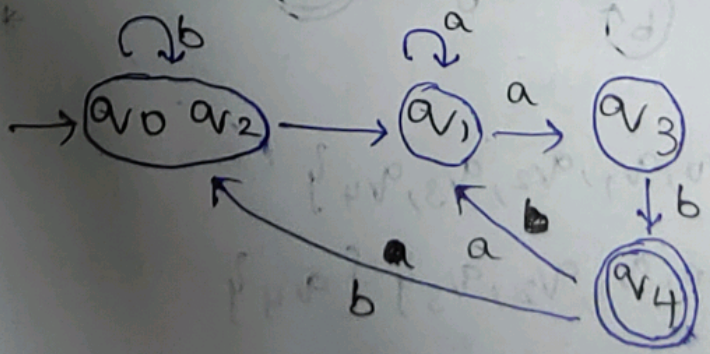
and $q_0 \cong q_2$

$P_2 = \{q_0\ q_2\}\{q_1\}\{q_3\}\{q_4\}$

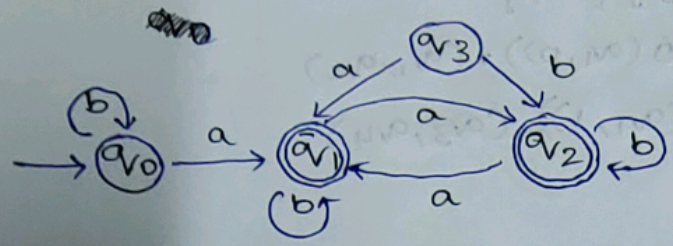
then compare $q_0 \cong q_2$ then

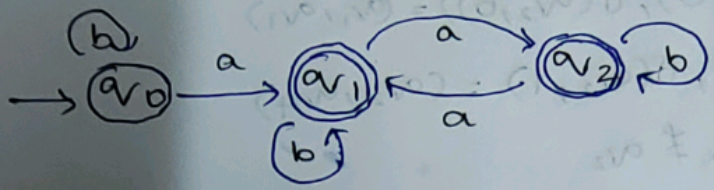
$P_3 = \{q_0\ q_2\}\{q_1\}\{q_3\}\{q_4\}$.

$P_2 = P_3$.



Q. The resulting DFA is





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

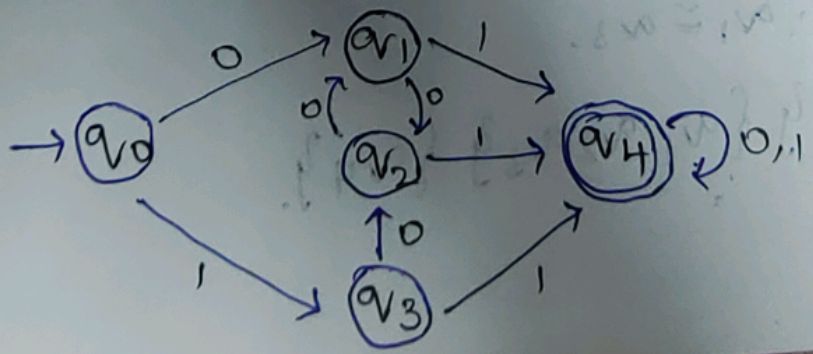
$P_0 = \{q_0\}\{q_1, q_2\}$

Sol: This DFA contains an inaccessible state. So remove the state.

Transition table:

δ	a	b
$\rightarrow q_0$	q_1	q_0
$* q_1$	q_2	q_1
$* q_2$	q_1	q_2

Q. Minimize the given DFA.



Sol: Transition table:

δ	0	1
$\rightarrow q_0$	q_1	q_3
q_1	q_2	q_4
q_2	q_1	q_4
q_3	q_2	q_4
$* q_4$	q_4	q_4

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$P_0 = \{q_0, q_1, q_2, q_3\}\{q_4\}$

$(q_0 q_1) \Rightarrow (\delta(q_0, a0), \delta(q_1, 0)) = (q_1, q_2)$

$(\delta(q_0, 1), \delta(q_1, 1)) = (q_3, q_4)$

$\therefore q_0 \not\equiv q_1$

$(q_0, q_2) \Rightarrow (\delta(q_0, 0), \delta(q_2, 0)) = (q_1, q_1)$

$(\delta(q_0, 1), \delta(q_2, 1) = (q_2, q_4)$

$\therefore q_0 \neq q_2$

$(q_0, q_3) \Rightarrow \delta(q_0, 0), \delta(q_3, 0) = (q_1, q_2)$

$\delta(q_0, 1), \delta(q_3, 1) = (q_3, q_4)$

$\therefore q_0 \not\equiv q_3$

$P_1 = \{q_0\}\{q_1 q_2 q_3\}\{q_4\}$

$(q_1, q_2)\ (\delta(q_1, 0), \delta(q_2, 0)) = (q_2, q_1)$

$(\delta(q_1, 1), \delta(q_2, 1) = (q_4, q_4).$

$\therefore q_1 \not\equiv q_2$

$(q_1, q_3) = (\delta(q_1, 0), \delta(q_3, 0) = q_2 q_2$

$(\delta(q_1, 1), (\delta(q_3, 1) = q_4 q_4$

$\therefore q_1 \not\equiv q_3.$

$P_2 = \{q_0\}\{q_1 q_2 q_3\}\{q_4\}.$

Q.