Father of Modern computation ⇒ Alan Turing | Turing Prize Awarded

Parul University

Faculty of Engineering & Technology

Theory of Computation (Formal Language & Automata Theory)

⇒ "It is the mathematical study of computing machines and their capabilities."

- Mathematical models of a computer → what can't a computer do?

Finite State Automata

FSA/FSM { Ex: forms: email A-Z a-z 0-9 @ ___ . ___

- Push-down Automata (PDA)
  ↳ Code → [compiler] → (verify syntax)
- Turing Machine: (how efficiently can you solve?)
  ↳ Halting Problem

☆ Formal Languages: (linguistics)

*Alphabet: "non-empty finite set of symbols"

Denoted by Σ

Ex:

$\Sigma_{eng} = \{A, B, \dots Z, a, b, \dots z\}$

$\Sigma_{bin} = \{0, 1\}$

$\Sigma_{dec} = \{0, 1, \dots 9, .\}$

✗ $\Sigma_{empty} = \phi$ ✗

* **String / Word**: finite sequence of alphabet

Ex: abc, 010101, 123.65, Application

length = # alphabets in the word

empty string ⇒ length = 0 $|\epsilon| = 0$

$(\epsilon, \lambda, \lambda)$

Σ^* = set of all strings (including empty string) formed using Σ

* **Language**: Subset of Σ^*

or

Set of strings over any Σ

Denoted by 'L'

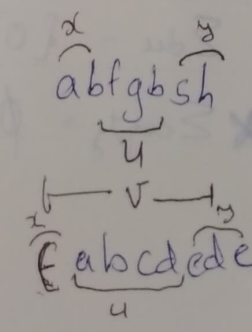
* **Grammar**: Set of rules to produce valid strings in a formal language

* **Substring**: Given Σ, u is a substring of v if $\exists\ xuy = v$

$x, u, v, y \in \Sigma^*$

# of substring of string of length n: $\frac{n(n+1)}{2} + 1$

Ex:



⇒ fgb can be substring, like this there can be many substrings of v

* **Prefix**: u is a prefix of if $\exists x$ s.t. $ux = v$
$u, x, v \in \Sigma^*$

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

$\underbrace{010}_{u}\underbrace{1010}_{x} \} \rightarrow ux = v$

$\underbrace{\quad\quad}_{v}$

* **Suffix**: u is a suffix of if $\exists x$ s.t.
$xu = v$
$u, v, x \in \Sigma^*$

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

$\underbrace{1011}_{x}\underbrace{1011}_{u} \} \rightarrow xu = v$

$\underbrace{\quad\quad}_{v}$

☆ Operations on Alphabets, Strings & Languages

1) **Power of an Alphabet**: Suppose $\exists \Sigma$.
s.t. $\Sigma = \{0, 1\}$
$\Sigma^0 = \epsilon$
$\Sigma^1 = \Sigma = \{0, 1\}$
$\Sigma^2 = \Sigma\Sigma = \{00, 01, 10, 11\}$ (set of strings/words of length 2)
$\Sigma^3 = \Sigma\Sigma\Sigma = \{000, 001, 010, 011, 100, 101, 110, 111\}$
⋮
$\Sigma^k = \{ w \mid |w| = k \}$ & w are words formed using Σ

2) Kleen Closure:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

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

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$$

$$\boxed{\Sigma^+ = \bigcup_{i \geq 0} \Sigma^i}$$

$$\Sigma^* = \{w \mid |w| \geq 0\}$$

3) Positive closure:

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

$$\Sigma = \{0, 1\} \Rightarrow \Sigma^+ = \{0, 1, 00, 01, 10, 11, 000 \dots\}$$

$$\boxed{\Sigma^+ = \bigcup_{i \geq 1} \Sigma^i} \quad \Sigma^+ = \{w \mid |w| \geq 1\}$$

4) Basic properties of Σ^* & Σ^+

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

$$\Sigma^* \cap \Sigma^+ = \Sigma^+$$

$$\Sigma^* \Sigma^* = \Sigma^*$$

$$\Sigma^* \Sigma^+ = \Sigma^+ = \Sigma^+ \Sigma^*$$

□

✱ Example of Languages : (As we know $L \subseteq \Sigma^*$)

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

1) $L_1 = \Sigma^*$: Universal Language

2) $L_2 = \Sigma^+ \subseteq \Sigma^*$

3) $L_3 = \{0^n \mid n \geq 1\}$
$= \{0, 00, 000, 0000 \ldots\} \subseteq \Sigma^*$

4) $L_4 = \{0^n 1^n \mid n \geq 1\}$
$= \{01, 0011, 000111, 00001111, \ldots\} \subseteq \Sigma^*$

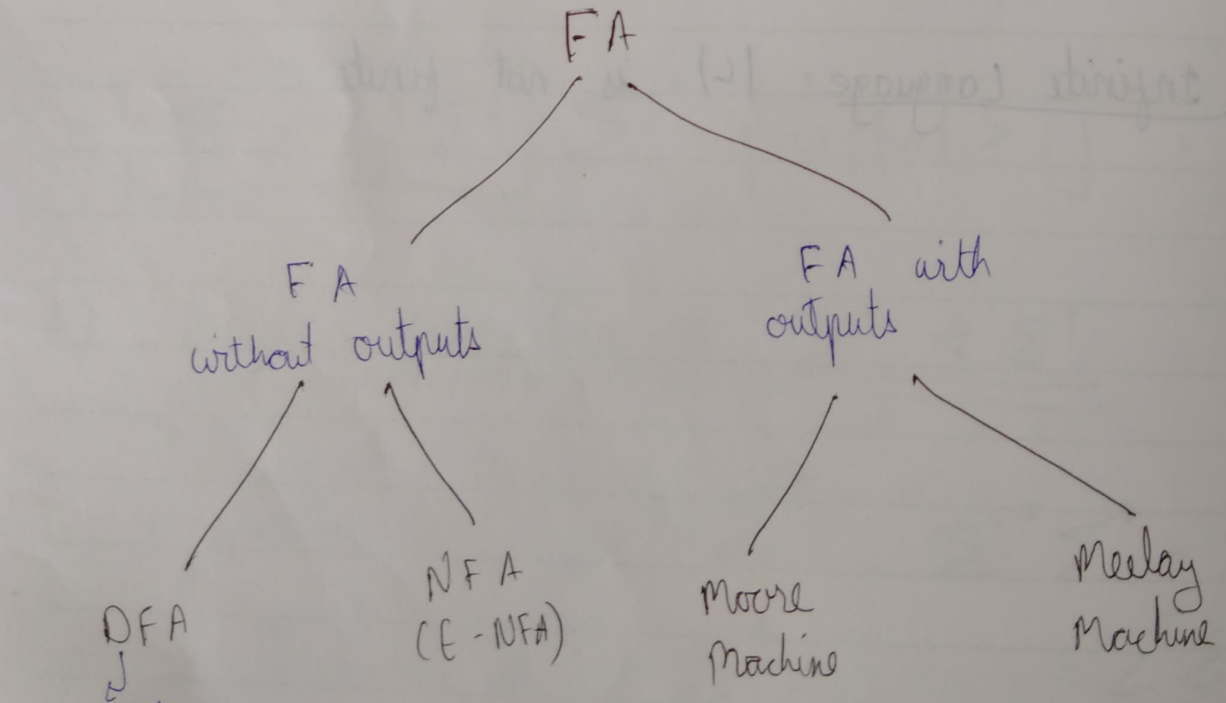
* Empty Language: $L = \{\} = \phi$, doesn't even contain ϵ
$|L| = 0$

* Finite Language : $|L|$ is finite

* Infinite Language : $|L|$ is not finite

☆ Finite State Machine / Automata

- Finite automata is the simplest machine to recognize patterns
- The finite automata or finite state machine is an abstract machine that has five elements or tuples.
  - ☆ Q : Finite set of states
    Σ : set of input symbols
    q : Initial state
    F : set of Final states
    δ : Transition Function



DFA

Applications of FSM / FA or DFA

a) DFA to determine if a given decimal number is even / odd.

b) Simpler email-address verification FSM

c) Turnstile

**a) Design a FSM to determine if a given decimal number is even or odd.

→ : $\Sigma = \{0, 1, 2, \ldots, 9\}$

Input $\in \Sigma^*$

Input word : Ex: $w = 1946$

: Set of states

$Q = \{q_0, q_1, q_2\}$: states finite

: Start state : q_0

: Final state : (0) $\{q_1\}$

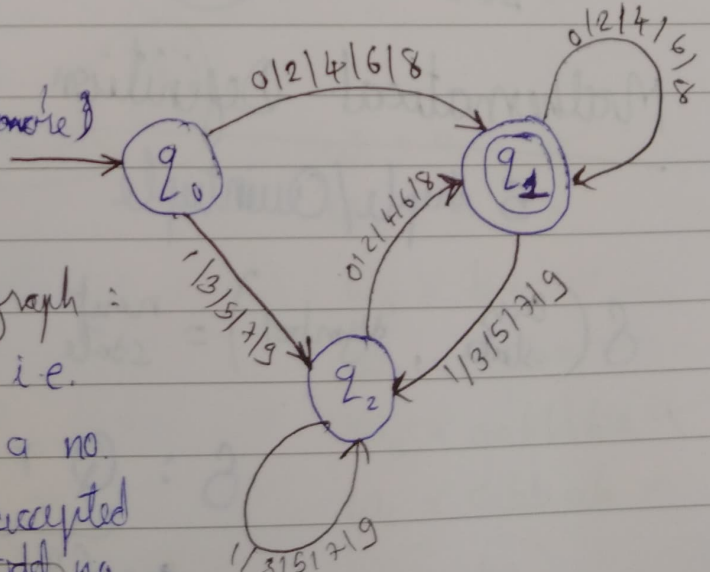
(can be zero or more)

But $w_1 = 123$ is a word

logic : if last digit is 0/2/4/6 or 8 then w_1 is even else it is odd

Input word is always read from left to right i.e. L to

: Transition graph :

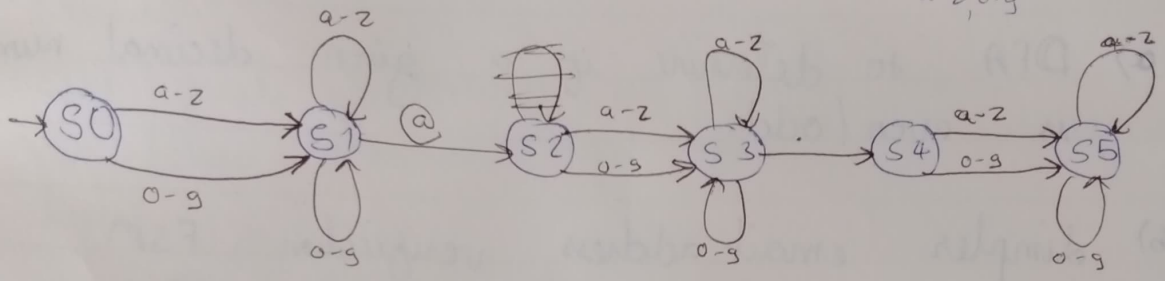


Only accept even numbers i.e. the final state is q_1, if a no. is odd then it won't be accepted which gives us the result as odd no.

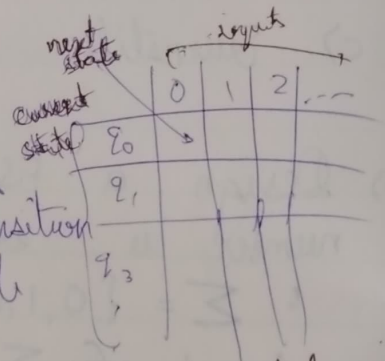
- DFA to determine a valid email address

Considering a format of "____@____.__"

(a-z, 0-9) (a-z, 0-9) (a-z, 0-9)



★ Representation of a DFA :



Transition Diagram
- di-graph

Transition table

Initial state → q_0

Final state → q_2

FSM without final state
Valid & Invalid inputs
$w_i \in \Sigma^*$

* Mathematical Definition of DFA :

5 tuple/Quintuple $M = (\Sigma, Q, q_0, \delta, F)$

(Finite set of i/p symbols; set of final states)

δ (curr state, symbol) = next state

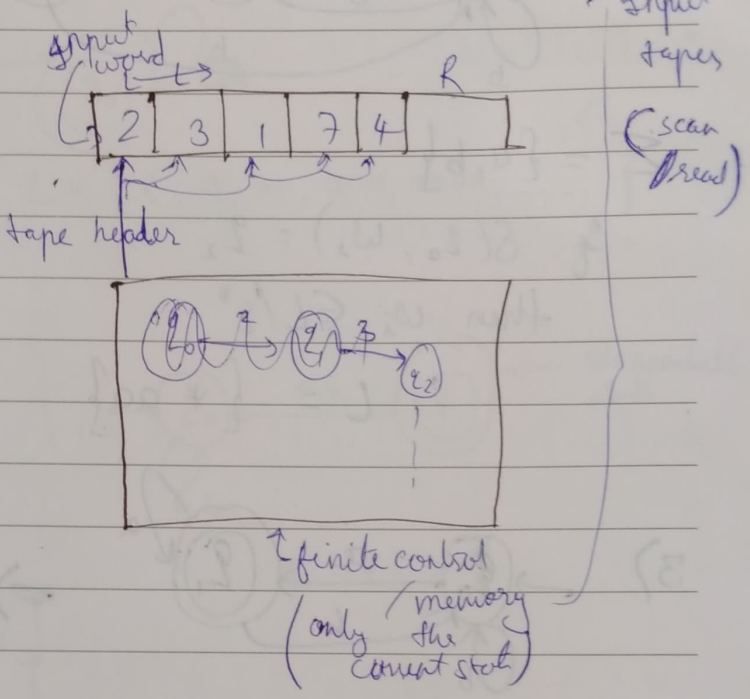
Σ : input alphabet; q_0 : initial state; δ : transition func^n

$\delta : Q \times \Sigma \to Q$

$Q \times \Sigma \to (q, 1)$

q : output state

* Computation Architecture of an FSM:



✡ Language of FA:

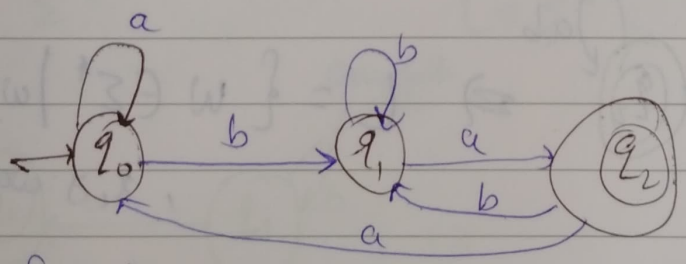
↳ set of all strings accepted by FA

M = machine

$$L(FA) = \{ \omega \in \Sigma^* \mid \omega \text{ is acceptable by } M \}$$

$$= \{ \omega \in \Sigma^* \mid \delta(q_0, \omega) = \text{final state} \}$$

Ex:-



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

If $\delta(q_0, \omega_i) = q_2$ then $\omega_i \in L$

$\omega_1 = a$ ✗

$\omega_2 = b$ ✗

$\omega_3 = ba$ ✓

$\omega_4 = abba$ ✓

$\omega_5 = aabbbba$ ✓

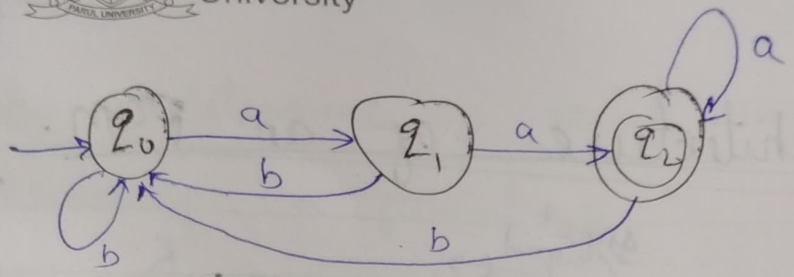
$\omega_6 = abbab$ ✗

$\therefore L = \{ * ba \}$

* in L represent 0 or more symbols from Σ

—ba

2)



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

If $\delta(q_0, w_i) = q_2$

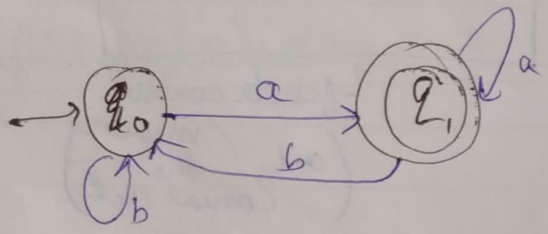
then $w_i \in L / \Sigma^*$

$\therefore L = \{ * aa \}$

$w_1 = ab$ ✗
$w_2 = aa$ ✓
$w_3 = aba$ ✗
$w_4 = abaa$ ✓
$w_5 = abb$ ✗
$w_6 = abbaa$ ✓

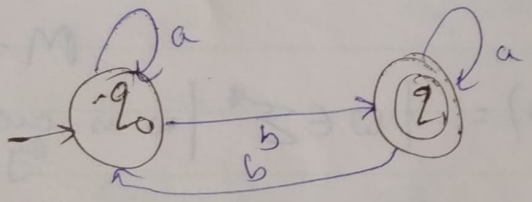
—— aa

3)



$\Rightarrow L = \{ * a \}$

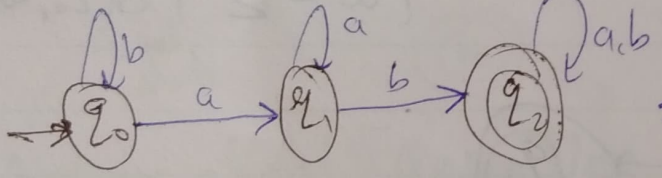
4)



$\Rightarrow L = \{ w \mid |w_b| = odd \}$

1, 3, 5, 7 ... of b

5)

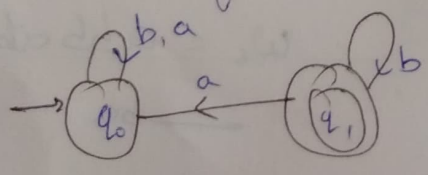


$\Rightarrow L = \{ w \in \Sigma^* \mid w = xaby \}$

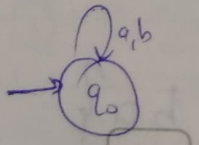
, x, y : words in Σ^*

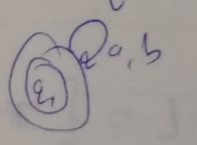
* Unreachable states : The states which are not reachable from the initial state of the DFA

Ex:

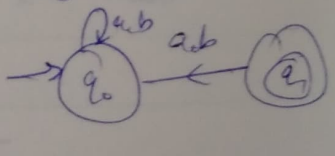


,

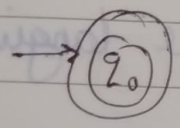




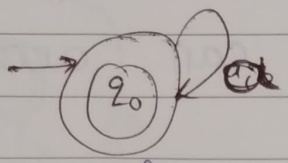
,



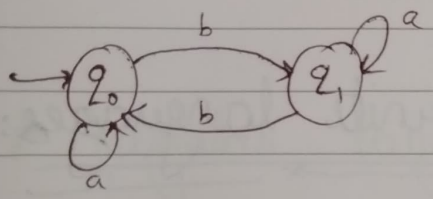
Ex: Calculate the corresponding L for DFAs ($\Sigma = \{a, b\}$)



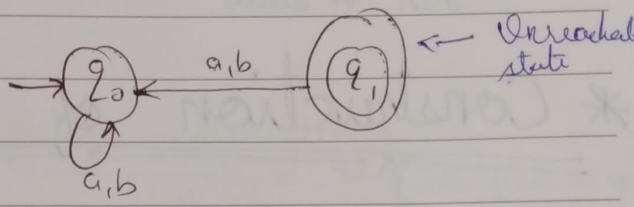
$L = \{\epsilon\}$



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

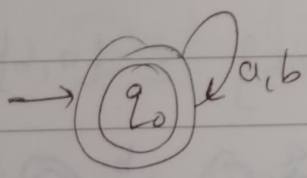


$L = \{\} = \phi$

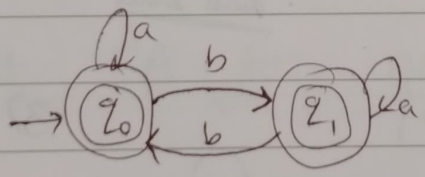


$L = \{\} = \phi$

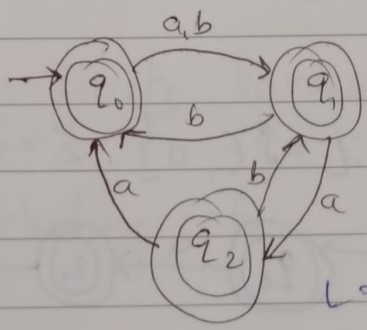
- [- FA without final state accepts ϕ]



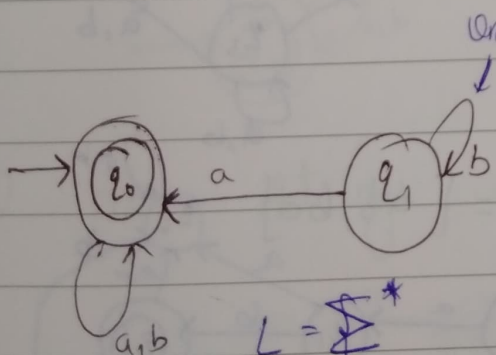
$L = \Sigma^*$



$L = \Sigma^*$



$L = \Sigma$



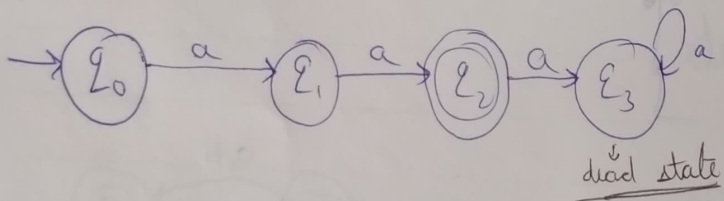
$L = \Sigma^*$

Dead state : The state from which no final state is reachable.

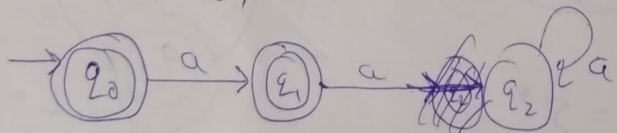
- Every FA accepts exactly one language
- Multiple FA can accept the same language
- Minimal FA for any language is unique
  - Min. # states

* Construction of FA for finite languages:

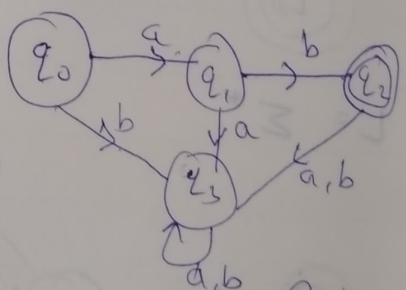
1) $L = \{aa\}$, $\Sigma = \{a\}$



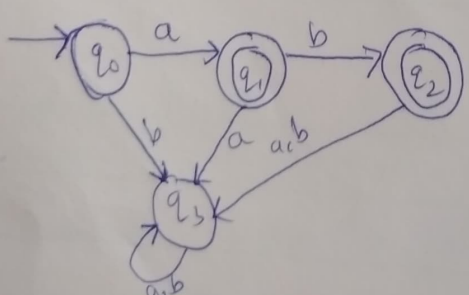
2) $L = \{\epsilon, a\}$, $\Sigma = \{a\}$



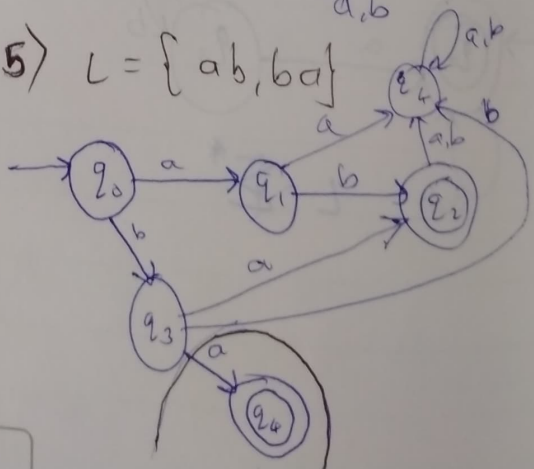
3) $L = \{ab\}$, $\Sigma = \{a, b\}$



4) $L = \{a, ab\}$

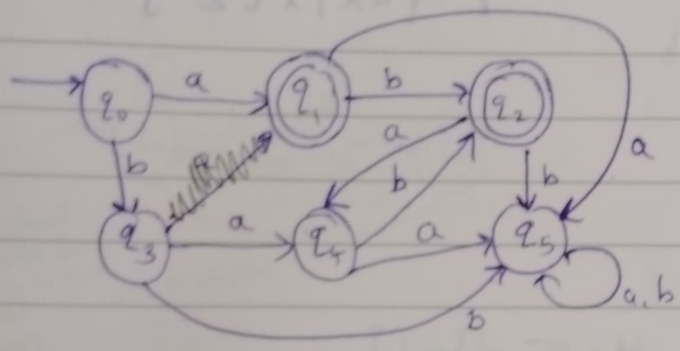


5) $L = \{ab, ba\}$



Nondistinguishable states : The states which cannot be distinguished from one another for any input string. These states can be merged.

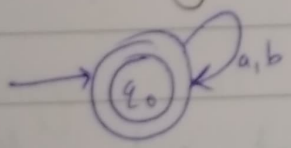
6) $L = \{a, ab, bab, abab\}$



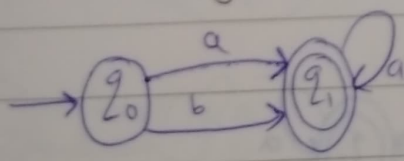
* Regular languages & Finite languages :
  - Finite languages (with finite # of strings) → FA can be constructed
  - Regular languages → Languages for which ∃ FA that accepts the language

* Construction of FA for non-finite languages

1) all strings of a's & b's including ∈

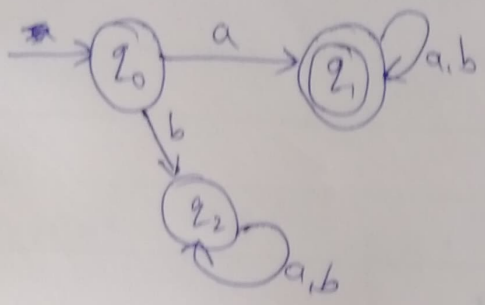


2) all strings of a's & b's excluding ∈



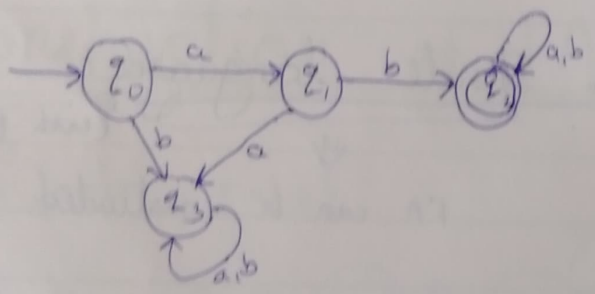
3) Strings that start with 'a' $\Sigma = \{a, b\}$

$L = \{ax \mid x \in \Sigma^*\}$



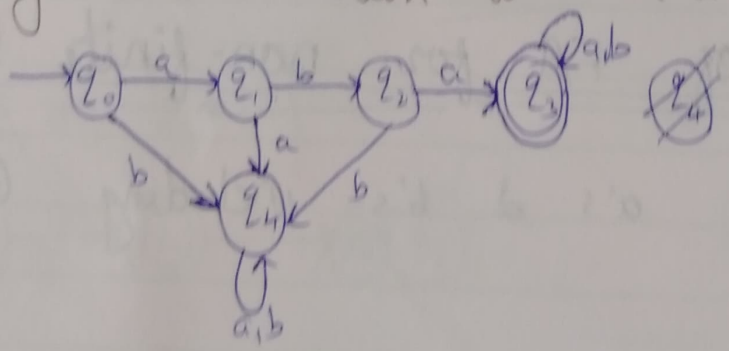
4) Start with 'ab' $\Sigma = \{a, b\}$

$L = \{abx \mid x \in \Sigma^*\}$



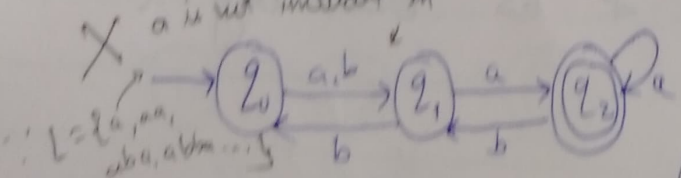
5) Strings that start with aba $\Sigma = \{a, b\}$

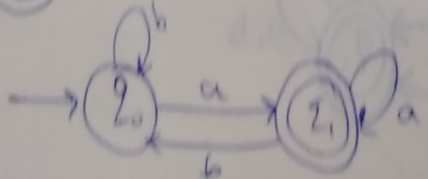
$L = \{abax \mid x \in \Sigma^*\}$



6) Min FA that accepts strings that ends with 'a'

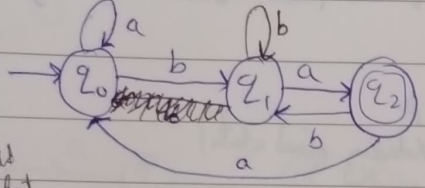
$L = \{xa \mid x \in \Sigma^*\}$





7) Min FA that accepts strings that ends with 'ba'

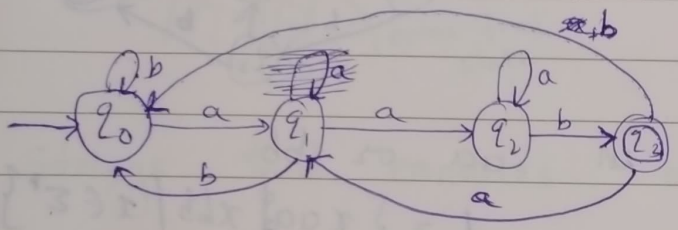
$L = \{ xba \mid x \in \Sigma^* \}$



∵ bba should be accepted but according to that trace it won't be accepted

8) Min FA ..... ends with 'aab'

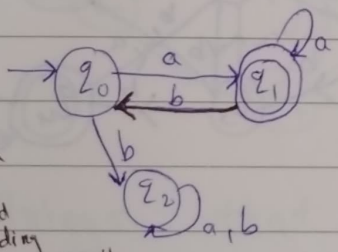
$L = \{ xaab \mid x \in \Sigma^* \}$



9) Strings that start and end in 'a' where $\Sigma = \{a, b\}$

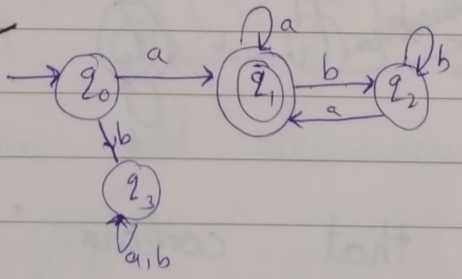
$L = \{ axa \mid x \in \Sigma^* \}$

✗



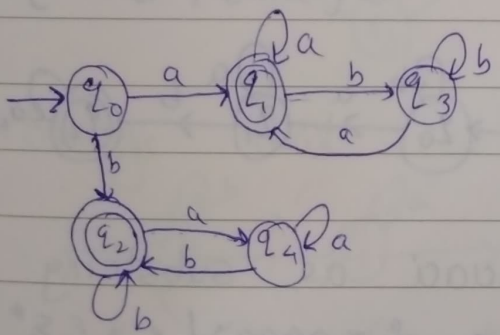
∵ abba must be accepted but according to this DFA it isn't.

✓



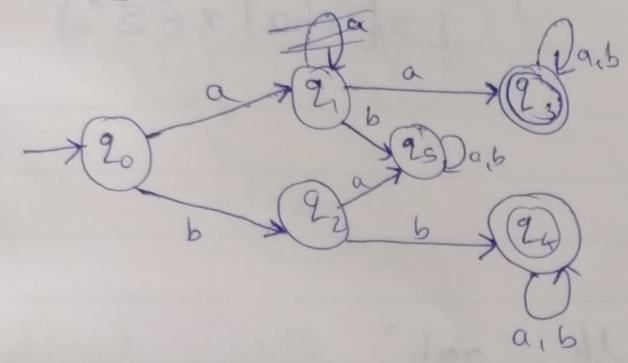
10) Strings that start & end with same symbol

$L = \{ axa, byb \mid x, y \in \Sigma^* \}$

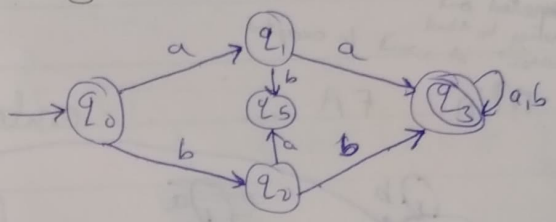


11) String that starts with aa or bb

$L = \{aax, bbx \mid x \in \Sigma^*\}$

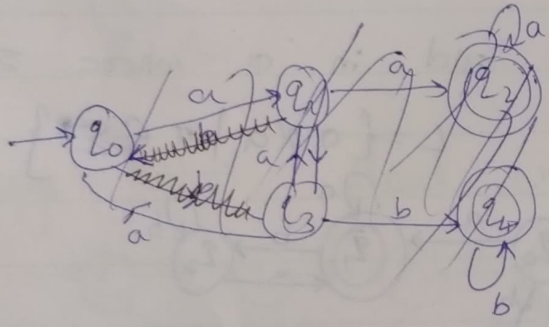


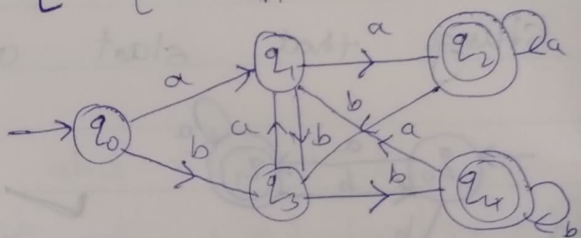
Min FA (Reducing final state)



12) Strings that end in aa or bb

$L = \{xaa, xbb \mid x \in \Sigma^*\}$

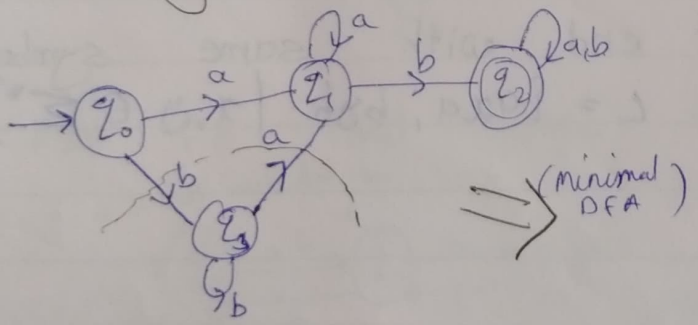




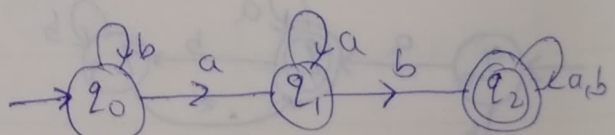
13) Strings that contain 'ab' as a substring

$L = \{*ab*\}$

$= \{xaby \mid x, y \in \Sigma^*\}$

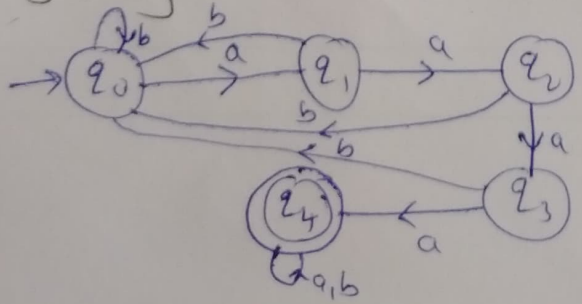


(minimal DFA)



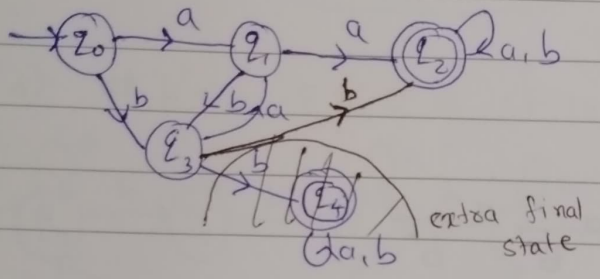
14) String that contain 'aaaa' as substring

$L = \{xaaaay \mid x, y \in \Sigma^*\}$



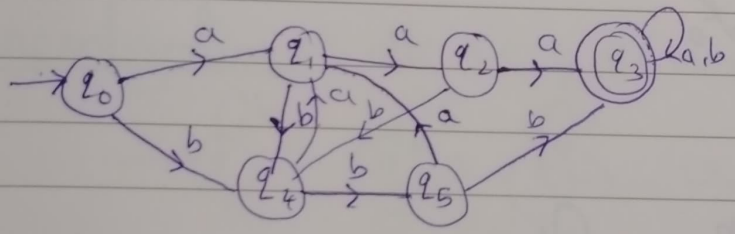
15) Strings that contain 'aa' or 'bb' as substring

$L = \{ xaay, xbby \mid x, y \in \Sigma^* \}$

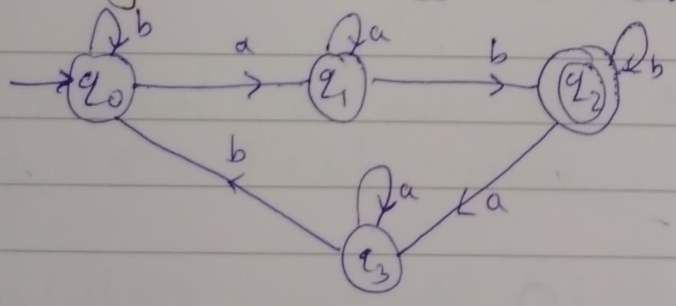


16) Strings that contains 'aaa' or 'bbb' as substring

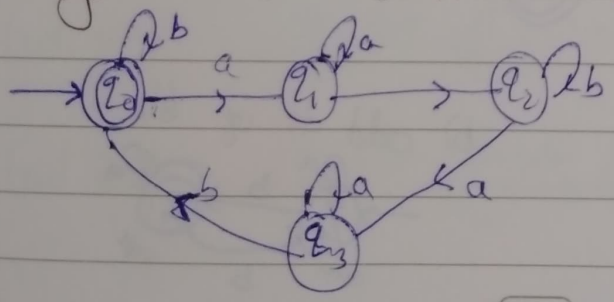
$L = \{ xaaay, xbbby \mid x, y \in \Sigma^* \}$



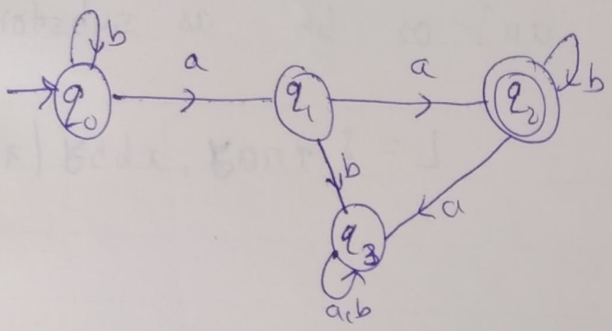
17) Strings that contains odd occurrences of substring 'ab'



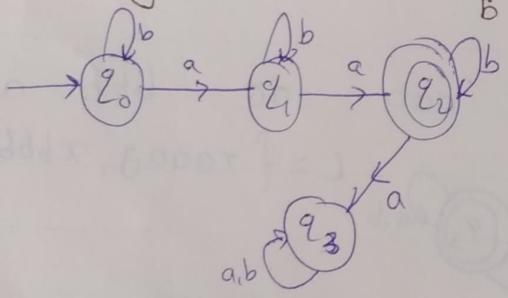
18) Strings that contains even occurrences of 'ab' substring



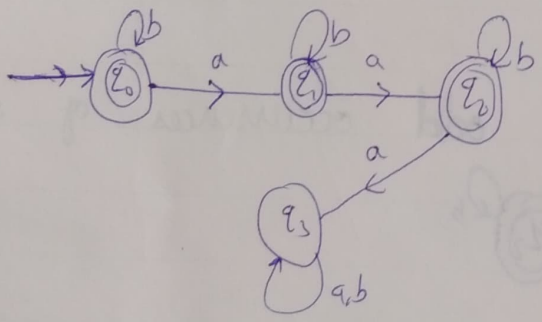
19) $\overline{b^*}\,aa\,\overline{b^*}$ (exactly two consecutive a's only)



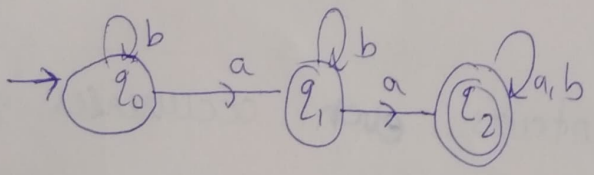
20) Exactly 2 a's : $\overline{b^*}\,a\,\overline{b^*}\,a\,\overline{b^*}$



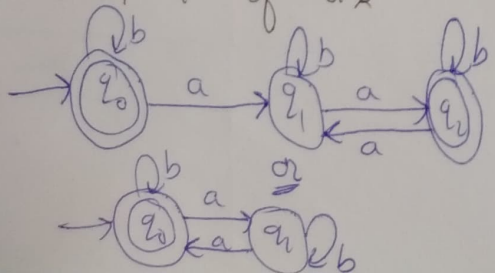
21) atmost 2 a's :



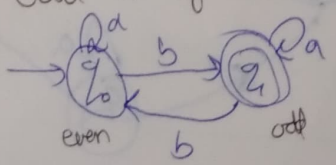
22) atleast 2 a's :



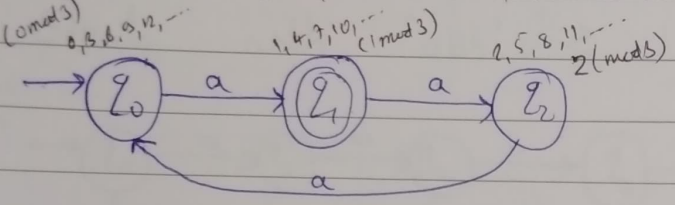
23) a) even # of a's

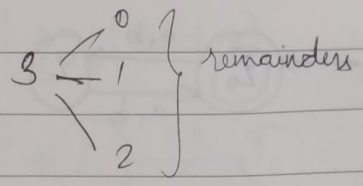


b) odd # of b's

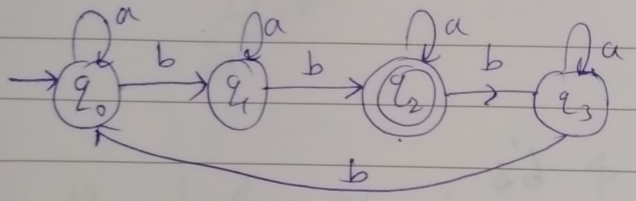


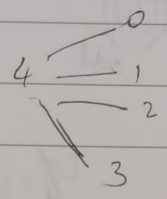
23) c) # a's = 1, 4, 7, ... = 1 (mod 3)



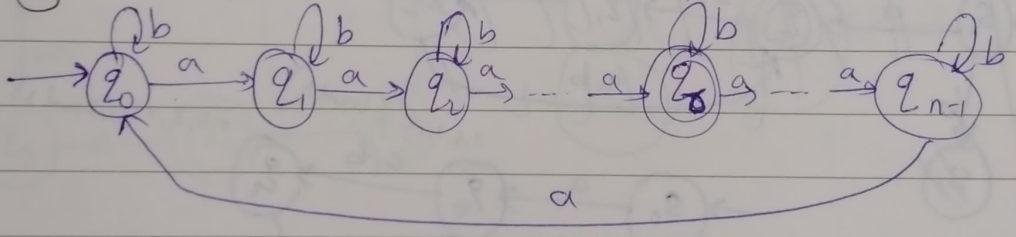


d) # b's = 2, ~~3~~, 6, 10, 14, 18, ... = 2 (mod 4)



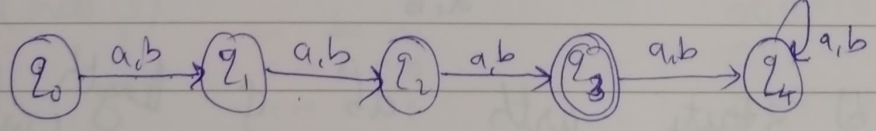


* Generalization: # a's = r (mod n) = $|w_a|$

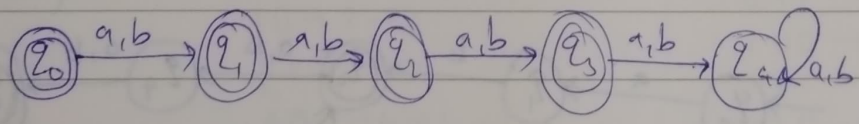


24) Construct a min FA that accepts all strings of $\Sigma = \{a, b\}$ where

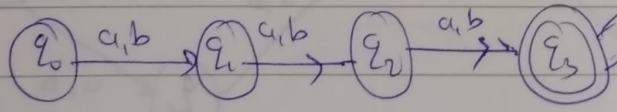
a) $|w| = 3$



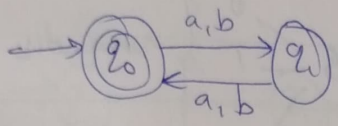
b) $|w| \leq 3$



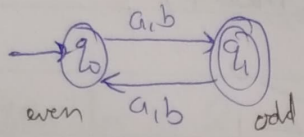
c) $|w| \geq 3$



d) $|w| = 0 \pmod 2$ = even

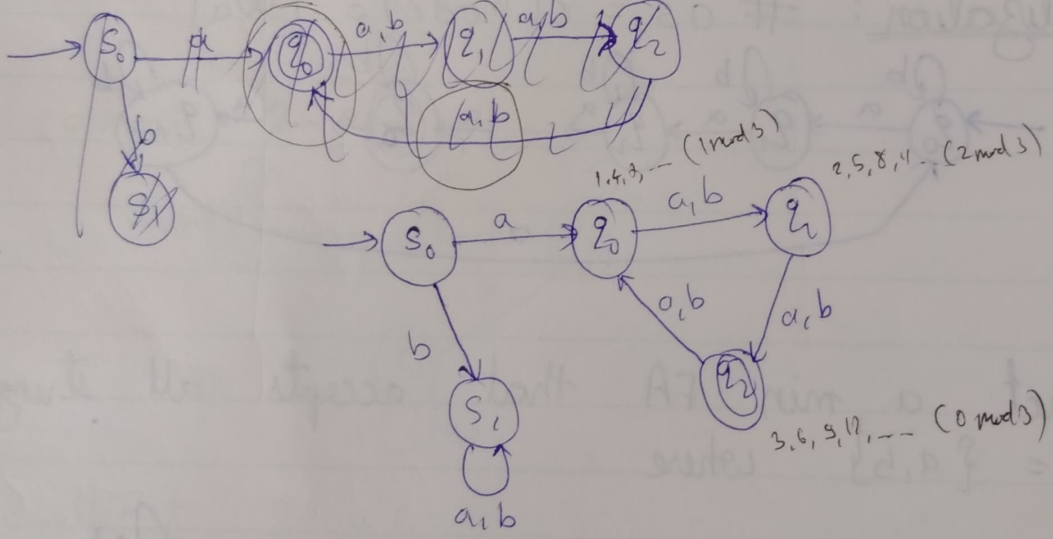


e) $|w| = 1 \pmod 2$ = odd

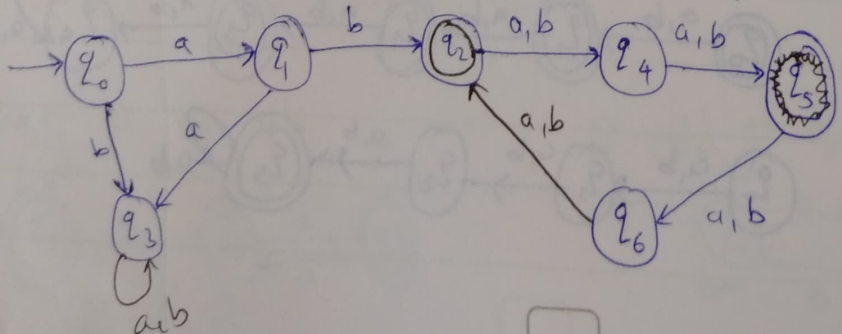


25) Min FA of a's & b's

a) Starts with 'a' & $|w| = 0 \pmod 3$

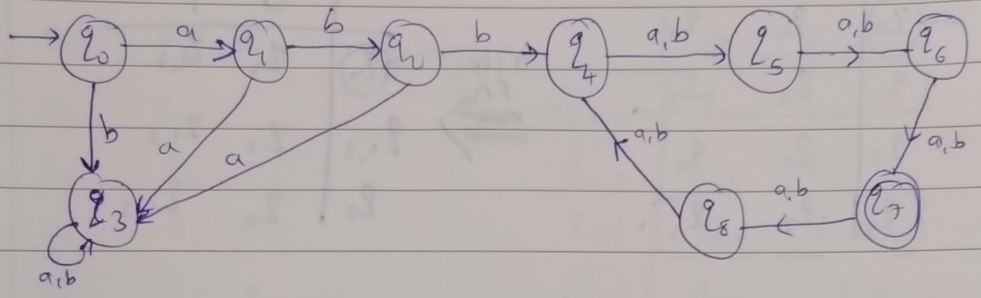


b) Starts with 'ab' & length is $2 \pmod 4$
$|w| = 2 \pmod 4$



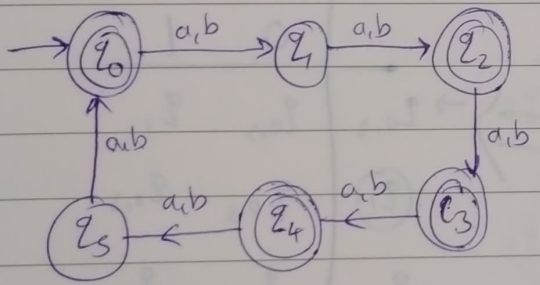
In transition graph → depicts initial state &
O depicts final state

c) Starts with 'abb' & $|w| = 1 \pmod 5$



d) $|w| = 0 \pmod 2$ or $0 \pmod 3$

If $0 \pmod 2$ & $0 \pmod 3$
⇓
$0 \pmod 6$



* DFA's on binary input:

Transition graph

1) Min FA that accepts binary strings whose integer value is $0 \pmod 3$

↓ 3 states

	0	1
→ q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

bin dec rem
0 → 0 → 0
1 → 1 → 1
10 → 2 → 2
11 → 3 → 0
100 → 4 → 1
101 → 5 → q_2

000 ↔ 0 → 0
001 → 1 → 1
010 → 2 → 2
011 → 3 → 0
100 → 4 → 1
101 → 5 → 2
110 → 6 → 0

Fill the cell of transition table one by one from q_0 on 0 then q_0 on 1 and so on with this ~~attempt~~

Ex: 0 we have to accept so q_0 on 0 ⇒ q_0
1 ⇒ remainder is 1 on ÷ with 3 ⇒ $\delta(q_0, 1) = q_1$
Now, for q_1 as we know we reach q_1 with input 1
on 0 ⇒ 10 ⇒ 2 ⇒ $2 \pmod 3$ ⇒ $\delta(q_1, 0) = q_2$

But final ≠ non-final can't be merged

Initial State can be merge with final state each.

All final - final ≠ Non-final non-final can be merged

2) divisible by 4 = (0 (mod 4))

bin dec rem
0 → 0 → 0
1 → 1 → 1
10 → 2 → 2
11 → 3 → 3
100 → 4 → 0
101 → 5 → 1
110 → 6 → 2
111 → 7 → 3

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

q_2 q_3 are similar so we merge them
q_2 q_3
also q_0 q_1 are similar but one of the state is final ✗

Min FA ⟹

	0	1
→ q_0	q_0	$q_{1,3}$
$q_{1,3}$	q_2	$q_{1,3}$
q_2	q_0	$q_{1,3}$

3) Congruent to 1 (mod 6) ≡ 1 (mod 6)

	0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_0	q_1
q_4	q_2	q_3
q_5	q_4	q_5

⟹

	0	1
→ $q_{0,3}$	$q_{0,3}$	q_1
q_1	$q_{2,5}$	$q_{0,3}$
$q_{2,5}$	q_4	$q_{2,5}$
q_4	$q_{2,5}$	$q_{0,3}$

4) ≡ 2 (mod 7)

	0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_6	q_0
q_4	q_1	q_2
q_5	q_3	q_4
q_6	q_5	q_6

$x \pmod y$ ⇒ If y is an even no. we can merge the states & deal with less no. of states but if y is odd then ## of states is $= y$

5) $\equiv 3 \pmod 8$

	0	1	
→q_0	q_0	q_1	✓
q_1	q_2	q_3	*
q_2	q_4	q_5	=
((q_3))	q_6	q_7	
q_4	q_0	q_1	✓
q_5	q_2	q_3	*
q_6	q_4	q_5	=
q_7	q_6	q_7	

Minimizing

	0	1	
→$q_{0,4}$	$q_{0,4}$	$q_{1,5}$	✓
$q_{1,5}$	$q_{2,6}$	q_3	
$q_{2,6}$	$q_{0,4}$	$q_{1,5}$	✓
((q_3))	$q_{2,6}$	q_7	
q_7	$q_{2,6}$	q_7	

Minimizing

	0	1
→q_{0426}	q_{0426}	q_{15}
q_{15}	q_{0426}	q_3
((q_3))	q_{0426}	q_7
q_7	q_{0426}	q_7

* Generalization

Bin num $\equiv r \pmod n$

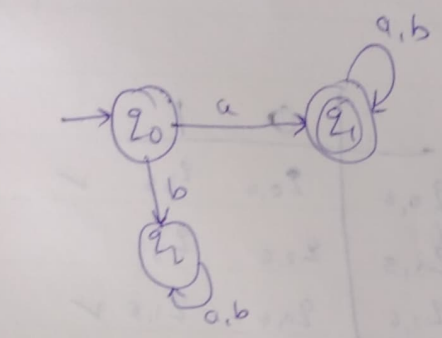
a) n is odd ⇒ n states

b) n ### $= 2^k$ ⇒ $k+1$ states

c) n is even & not 2^k ⇒ no formula, build TT & can merge by simplifying

Complement of FA (Final state ⇔ Non-final)

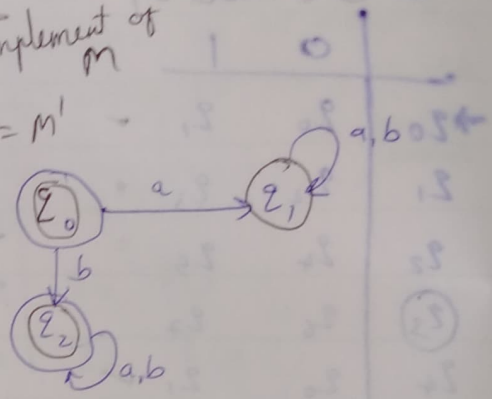
M:



$L(M) = \{ w \in \Sigma^* \mid w = ax, x \in \Sigma^* \}$

Complement of M

$M^c = M'$



$L(M') = \Sigma^* - L(M)$

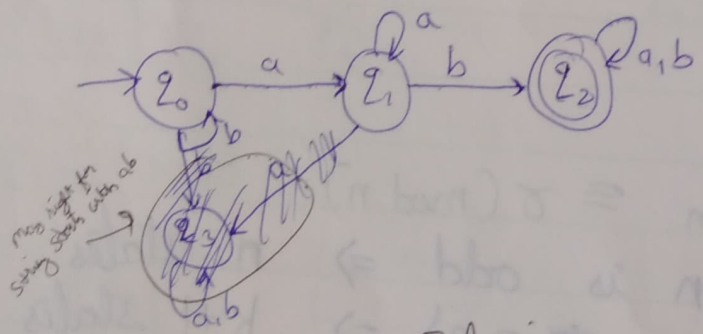
$\Sigma = \{a, b\}$

Application / use of complement

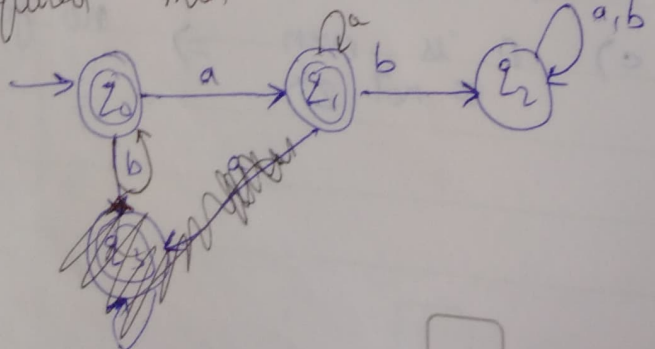
Ex: Construct a min FA that accepts all strings that DONOT contain 'ab' as substring

Let us first construct min FA which contains 'ab' as a substring

$L = \{ xab \mid x, y \in \Sigma^* \}$



Thus, required min FA is,



☆ Compound Automata :

(Unions & Intersections)

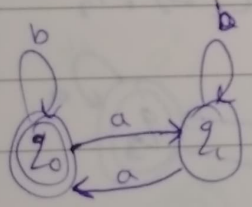
If there are
L_1
L_2 then $L_1 \cup L_2 \cup L_3$ is also possible
L_3 & $L_1 \cap L_2 \cap L_3$ is also possible

Similar for machines but notationally we can't write $M_1 \cup M_2 \cup M_3$ & $M_1 \cap M_2 \cap M_3$

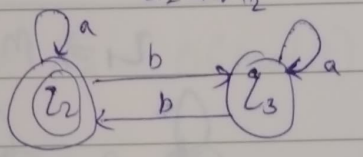
Ex: $\Sigma = \{a, b\}$ # a's is even (and) # b's even:

$L_1 \Rightarrow M_1$ $L_2 \Rightarrow M_2$

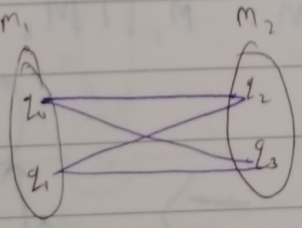
M_1:



M_2:

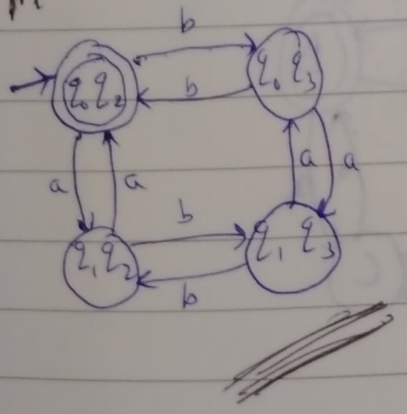


Let $M = M_1 \cap M_2$

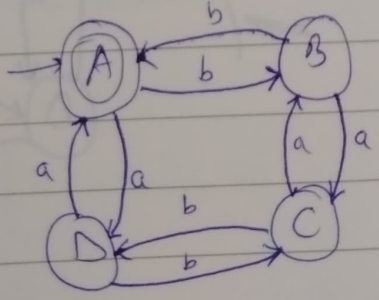


$\Rightarrow$ q_0q_2 q_0q_3
q_1q_2 q_1q_3

M:

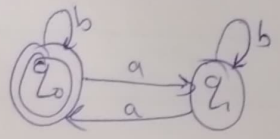


$\Rightarrow$

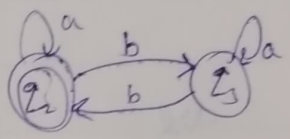


Q: # a's even (or) # b's even

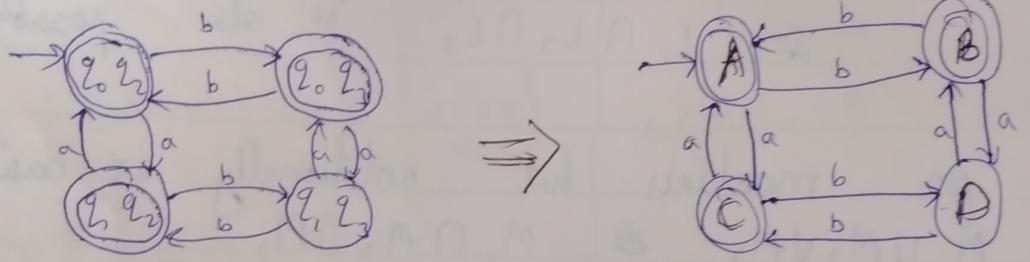
$L_1 \Rightarrow M_1$



$L_1 \Rightarrow M_2$

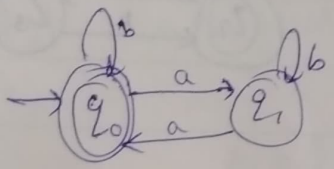


$M: L_1 \cup L_2$

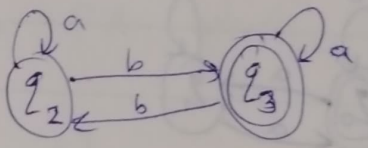


Q: # a's is even (and) # b's is odd

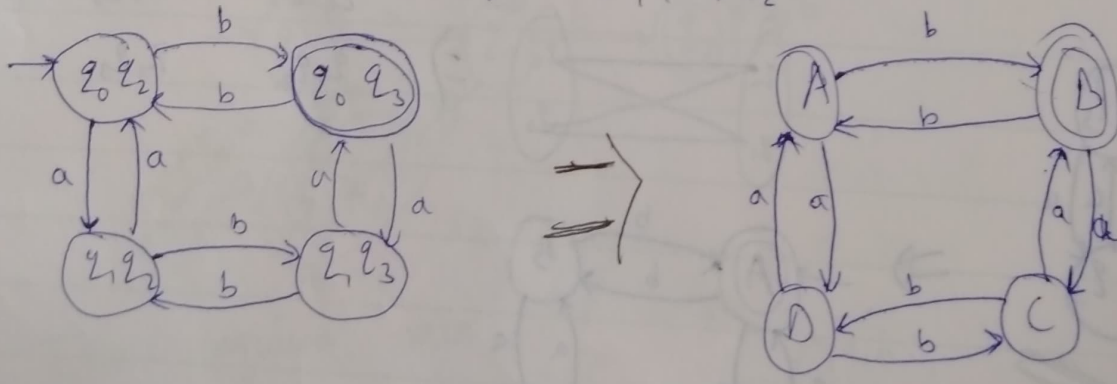
$L_1 \Rightarrow M_1$



$L_2 \Rightarrow M_2$

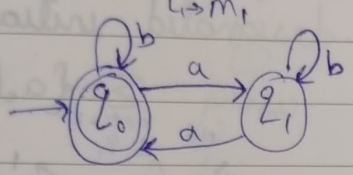


$M: M_1 \cap M_2$



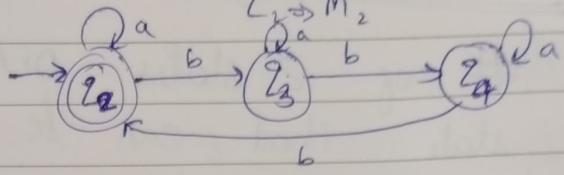
Q: # a's ≡ 0 (mod 2) (and) # b's ≡ 0 (mod 3)

↳ M_1

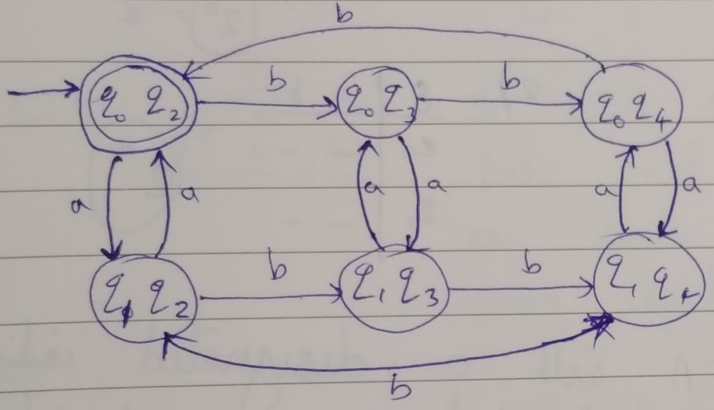


*

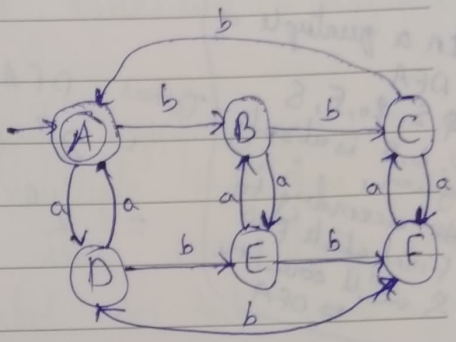
↳ M_2



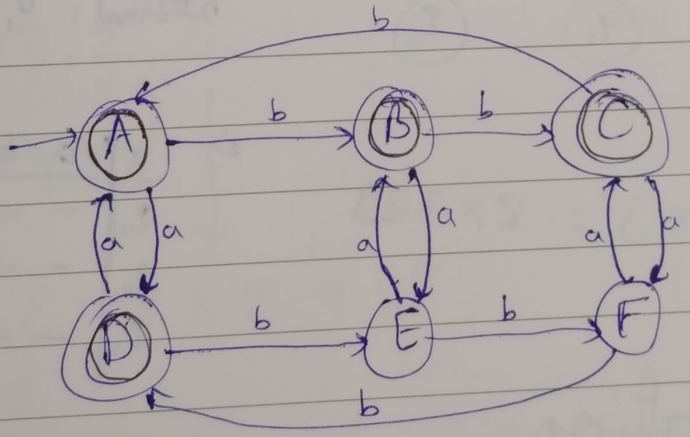
$M: M_1 \cap L_2$



⇒



(1): # a's ≡ 0 (mod 2) (or) # b's ≡ 0 (mod 3)



☆ Counting DFA :

1) # of 2-states DFA with a designated initial state that can be constructed using $\Sigma = \{a, b\}$

Soln

→ q_0 q_1

Points to consider in order to count DFA?

1) # Final : 0, 1, 2 final states

$2^2 = 4$

In a quintuple of DFA
$Q, \Sigma, q_0, F, \delta$
Q, q_0, Σ is already given, so according to Final state F & δ we'll count the # DFA

Total DFAs
$= 2^2 \times 2^4$
$= 2^6 = 64$

2)

δ	a	b
q_0	—	—
q_1	—	—

2^4

2) # of 3-states DFA with a designated initial state that can be constructed using $\Sigma = \{a, b\}$

→ q_0 q_1 q_2

DFAs: 1 3 3 1

1) # final : 0, 1, 2, 3

2^{3} → no. of states

2)

δ	a	b
q_0	—	—
q_1	—	—
q_2	—	—

3^6
↑
No. of states

$2^3 \times 3^6 = 8 \times 729$

* Generalization

$|\Sigma| = m$; # states $= |Q| = n$; initial state is designated

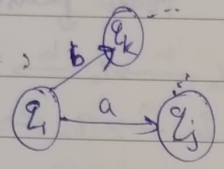
1) Final : 2^n
2) δ : $n^{m \times n}$

→ Total DFAs

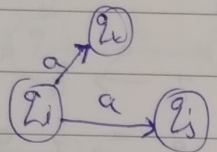
$2^n \times n^{m \times n}$

☆ Non-deterministic FA (NFA)

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

DFA : 

From each state there must be every input symbol transitted (& only once)

NFA : 

In NFA, one input symbol can be take part in multiple transition from a state & it isn't compulsory for every input symbol to take part in transition

NFA

It is 5-tuple (Quintuple);

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

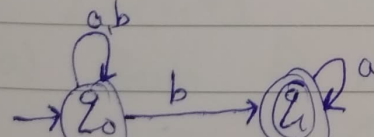
Q : # states; q_0 : initial state; Σ : set of alphabet; F : Final state

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

{ In DFA it was $\delta : Q \times \Sigma \to Q$ }

2^Q is a powerset which means set of subsets of Q

{ Eg: $Q = \{q_0, q_1\}$
$2^Q = \{\phi, \{q_0\}, \{q_1\}, \{q_0, q_1\}\}$ }

Eg: 

$\Sigma = \{a, b\}$ $Q = \{q_0, q_1\}$

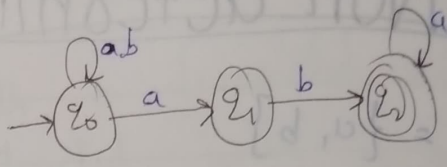
$2^Q = \{\phi, \{q_0\}, \{q_1\}, \{q_0, q_1\}\}$

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

$\delta(q_0, b) = \{q_0, q_1\}$ $\delta(q_1, b) \neq \phi$

* Acceptance by NFA

If any transition path leads to a final state, we accept the word



$w_1 = ab$ (transi^n paths)

(accepted)

$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0$ ✗

$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2$ ✓

- Check for each & every transi^n path of a word

$w_2 = aa$

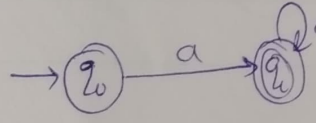
$q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0$ ✗

$q_0 \xrightarrow{a} q_1 \xrightarrow{} \phi$

* Examples (construc^n of NFA)

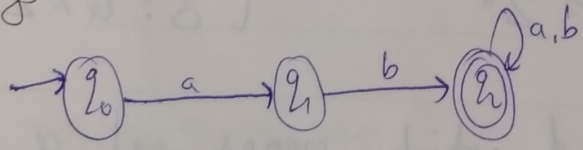
1) $\Sigma = \{a, b\}$

a) string that start with 'a'

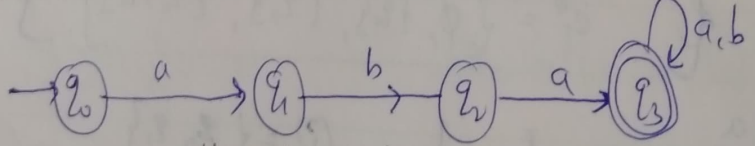


(not DFA : $\because q_0 \xrightarrow{b} \phi$)

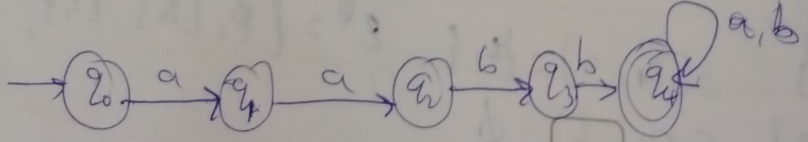
b) strings that start with 'ab'



c) starts with 'aba'



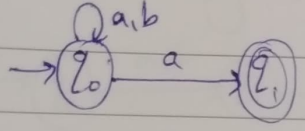
d) starts with 'aabb'



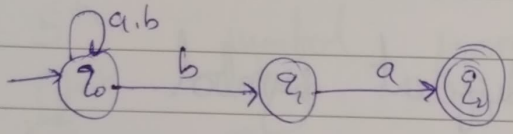
Every DFA is an NFA

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

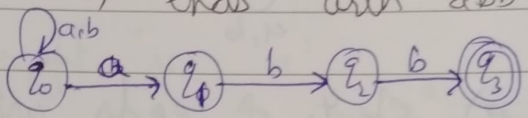
a) ends with 'a'



b) ends with 'ba'

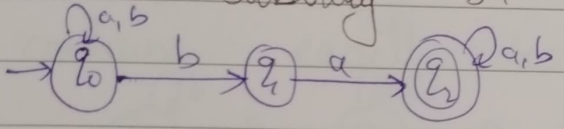


c) ends with abb

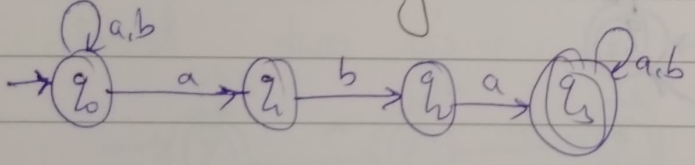


3) $\Sigma = \{a, b\}$

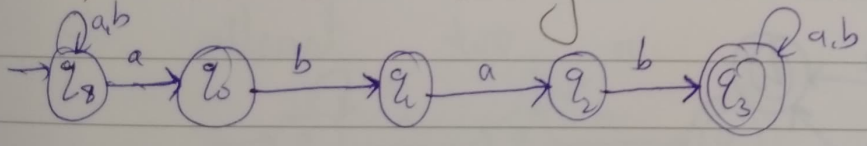
a) contains substring 'ba'



b) contains substring 'aba'

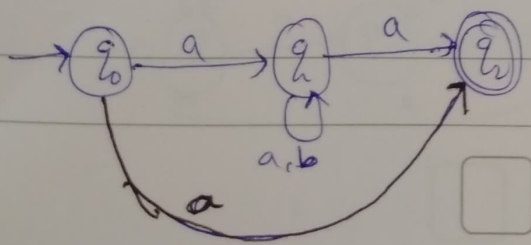


c) contains substring 'abab'

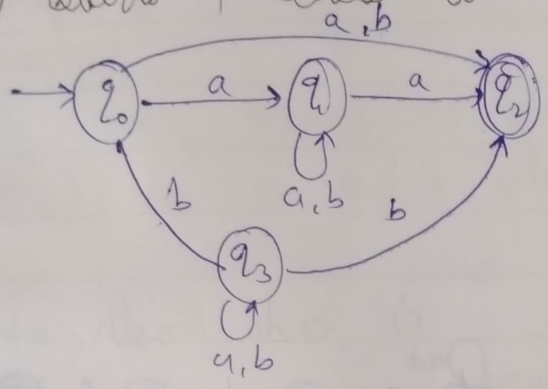


4) $\Sigma = \{a, b\}$

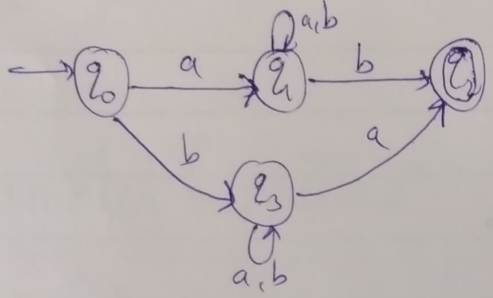
a) starts & ends with 'a'



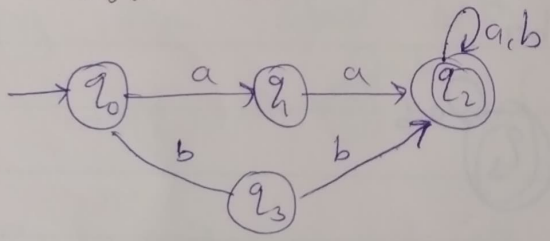
b) starts & ends with same symbol.



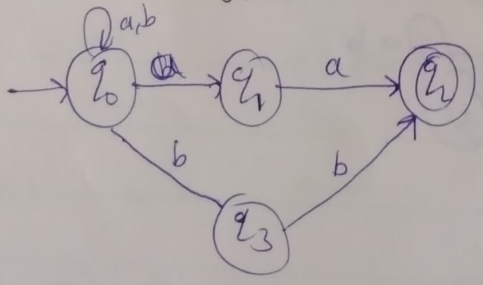
c) Starts & ends with different symbol



d) Starts with 'aa' or 'bb'

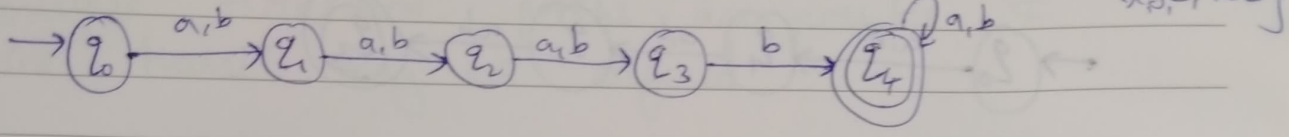


e) ends with 'aa' or 'bb'



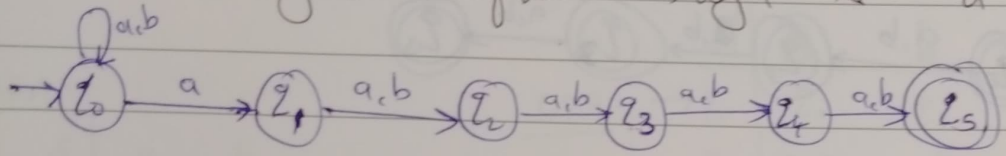
5) $\Sigma = \{a, b\}$

a) 4th symbol from left is 'b' $L = \{xyzaw \mid x,y,z,w \in \Sigma^*\}$



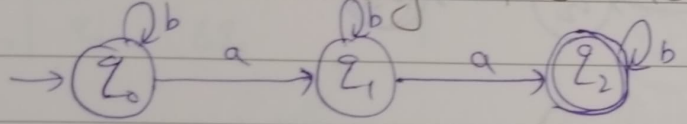
b) 5th symbol from right is 'a'

$L = \Sigma^* a \Sigma^* a + b$

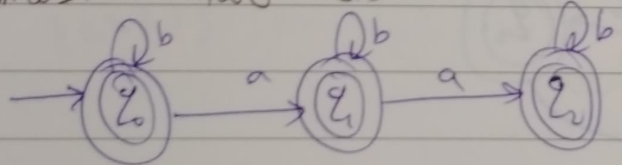


6) $\Sigma = \{a, b\}$

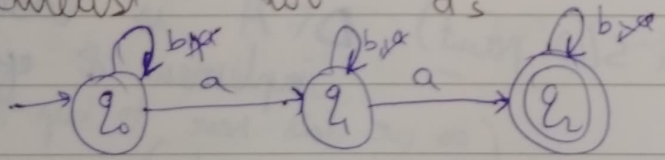
a) contains exactly two a's



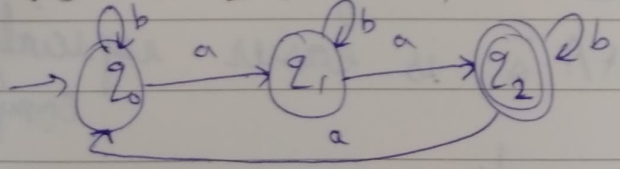
b) atmost two a's



c) atleast two a's

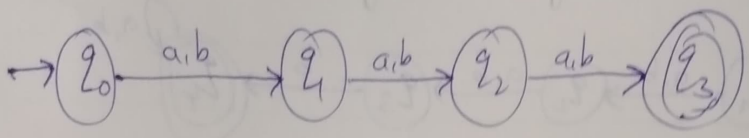


d) # a's $\equiv 2 \pmod 3$

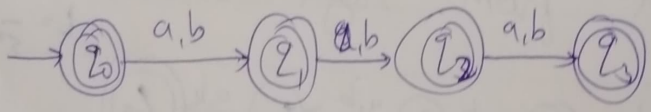


7) $\Sigma = \{a, b\}$

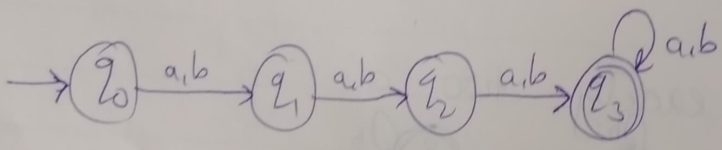
a) $|w| = 3$



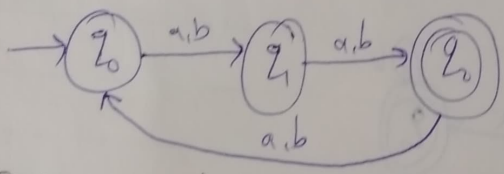
b) $|w| \leq 3$



c) $|w| \geq 3$



d) $|w| \equiv 2 \pmod 3$



* Observation

- Computationally (from s/w point) DFA is easier to implement & efficient (as we don't have to keep a track transit path) 2)
- For eg NFA is there is an equivalent DFA
- Design of NFA is is easier especially for complex cases
- NFA: parallel computing engine { multithreaded }

- No need of dead states
- Multiple transition paths are possible & some transi^n may not defined

☆ Conversion from NFA to DFA:

1) NFA:

δ	a	b
q_0	q_0	q_1
q_1	q_0q_1	q_2
q_2	q_1	q_0q_1

write all the subset of 2^Q except ϕ

$2^Q = \{\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0q_1\}, \{q_0q_2\}, \{q_1q_2\}, \{q_0q_1q_2\}\}$

DFA:

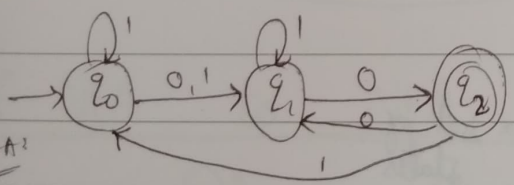
δ'	a	b
→ q_0	q_0	q_1
q_1	q_0q_1	q_2
q_0q_1	q_0q_1	q_1q_2
q_2	q_1	q_0q_1
q_0q_2	q_0q_1	q_0q_1
q_1q_2	q_0q_1	$q_0q_1q_2$
$q_0q_1q_2$	q_0q_1	$q_0q_1q_2$

from NFA:
$q_0 \xrightarrow{a} q_0$, $q_0 \xrightarrow{b} q_1$
$q_1 \xrightarrow{a} q_1$, $q_1 \xrightarrow{b} q_2$
$\Rightarrow q_0q_1 \xrightarrow{a} q_0q_1$, $q_0q_1 \xrightarrow{b} q_1q_2$

$q_0 \xrightarrow{a} q_0$, $q_2 \xrightarrow{a} q_1$ $\Rightarrow q_0q_2 \xrightarrow{a} q_0q_1$
Similar for others

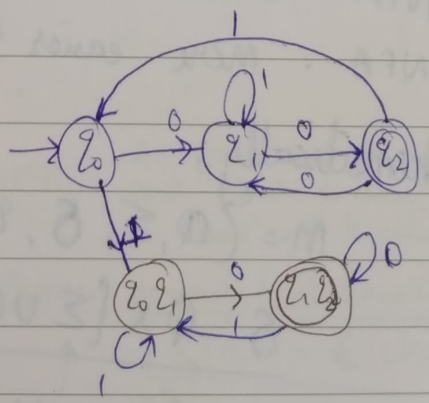
need not be the min DFA

2)



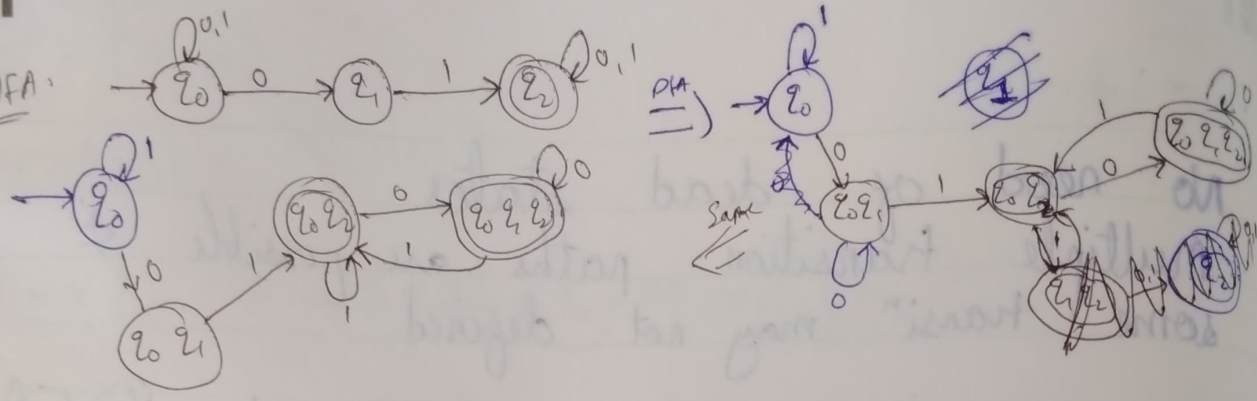
NFA:

DFA ⇒

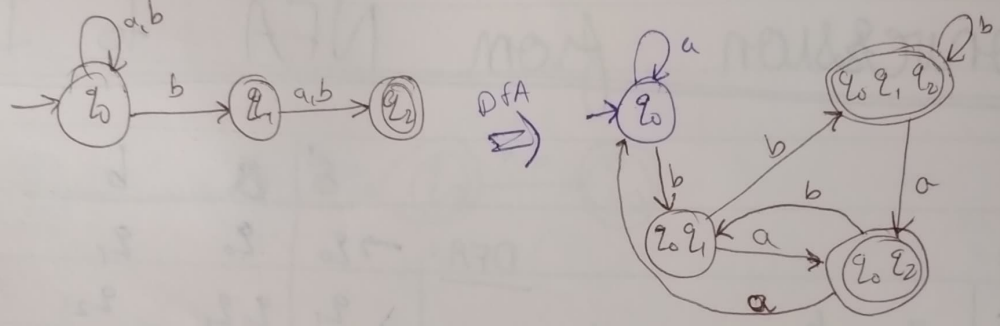


$q_0 \xrightarrow{0} q_1$
$q_1 \xrightarrow{0} q_2$
$\Rightarrow q_0q_1 \xrightarrow{0} q_1q_2$
Similar for others

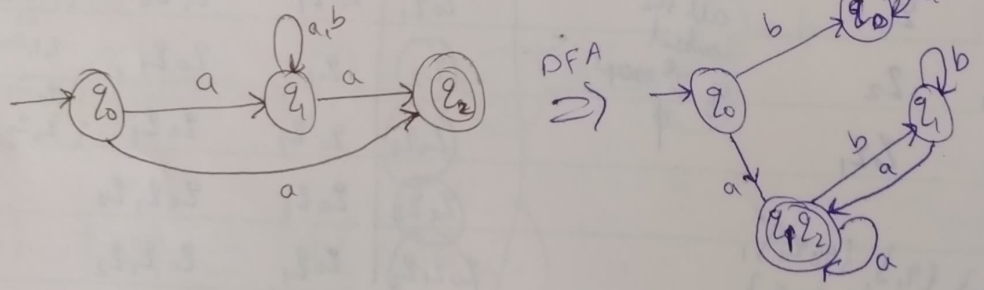
3) NFA:



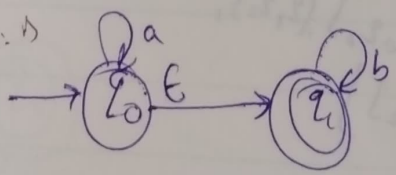
4) NFA:



5)



∈ - NFA ⟶ Ex:



NFA : easier to model

∈-NFA : more easier to model

power : ∈-NFA ≈ NFA ≈ DFA

Mathematically;

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

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

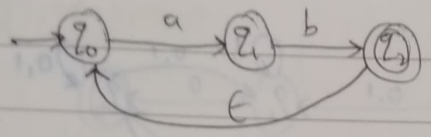
2^Q → set of states

states | symbol | empty string

Universally ; $\delta(q_i, \epsilon) = q_i$

write the languages correspond to ENFA

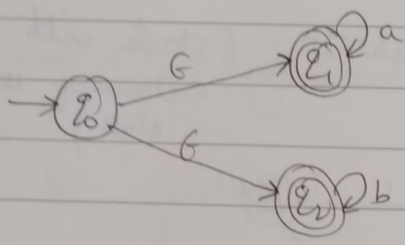
1)



$\Rightarrow$ $L = \{ ab, abab, ababab, \dots \}$

$L = \{ (ab)^n \mid n \geq 1 \}$

3)

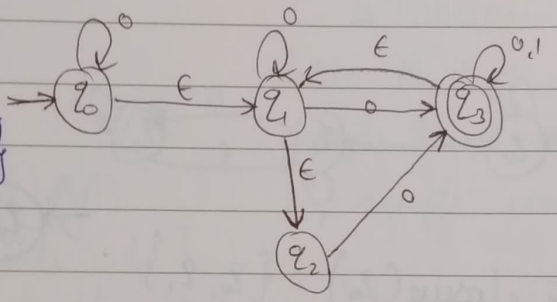


$\Rightarrow$ $L = \{ \epsilon, a, aa, aaa, aaaa, \dots$
$b, bb, bbb, bbbb, \dots \}$

$= \{ a^n \mid n \geq 0 \} \cup \{ b^n \mid n \geq 0 \}$

* ϵ - Closure :

↳ All states that can be reached from this state using ϵ transi"



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

ϵ - closure $(q_1) = \{ q_1, q_2 \}$

ϵ - closure $(q_2) = \{ q_2 \}$

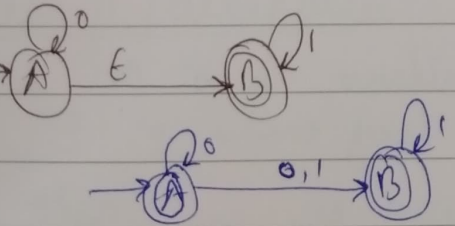
ϵ - closure $(q_3) = \{ q_3, q_1, q_2 \}$

ϵ - NFA $\xrightarrow{\text{(Thompson construction)}}$ NFA $\xrightarrow{\text{(subset construction)}}$ DFA

* ϵ - NFA to NFA conversion :

- All states will be as it is
- Just transi" is change (particular ϵ transi" will be removed)

Ex: 1)



$A \xrightarrow{0} A \xrightarrow{\epsilon} B$; $A \xrightarrow{\epsilon} B \xrightarrow{1} B$

$\therefore$ $0 = 0\epsilon$

$A \xrightarrow{\epsilon 1} B$

$A \rightarrow B$

General formula (notationally incorrect) :

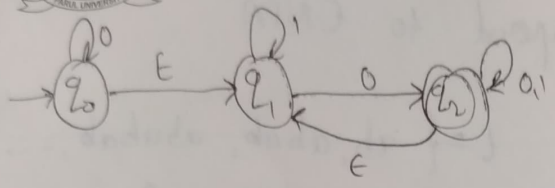
$\delta'(A, 0) = \epsilon$ - closure $(\delta(\epsilon\text{-closure}(A), 0))$

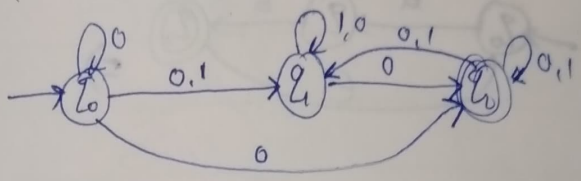
NFA — ϵ NFA

$\therefore$ ϵ can also reach final state
A must be final state $(A \xrightarrow{\epsilon} B)$

ϵ - closure $(\phi) = \{\phi\}$

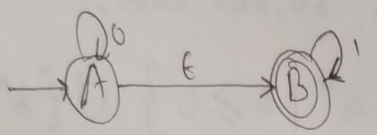
2)



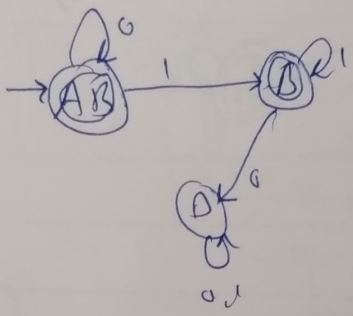


* ε-NFA to DFA conversion: (starts with ε-closure of initial state)

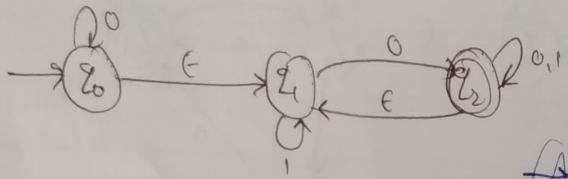
1)



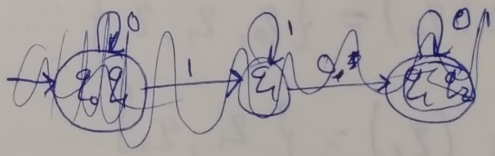
ϵ-closure(A) = {A, B} ⇒

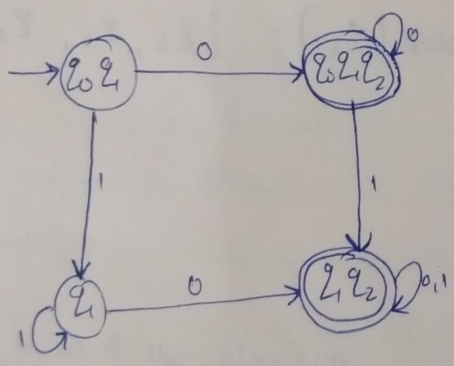


2)



ϵ-closure(q_0) = {q_0, q_1}





$q_0 \xrightarrow{0} q_0 q_2 q_1$ (ϵ0, ϵ0ϵ)

$q_0 \xrightarrow{1} q_1$

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

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

$q_1 \xrightarrow{0} q_2 q_1$

$\quad \xrightarrow{1} q_1$

$q_1 q_2 \xrightarrow{0} q_1 q_2$

$\quad \xrightarrow{1} q_1 q_2$

☆ Decision properties of FA

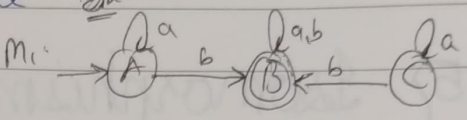
1) Emptiness: does the FA accept empty or non-empty L?

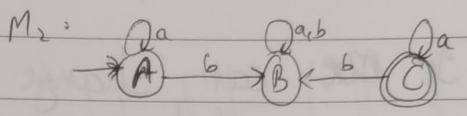
a) delete/ignore all unreachable states

b) atleast one final state ⇒ Non-empty L

no final state ⇒ empty L

Ex:





C is unreachable state in both M_1 & M_2 but M_1 accepts non-empty L ≠ M_2 accepts empty (reason b)

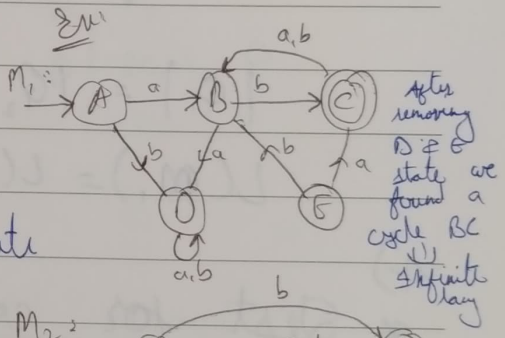
2) Finiteness: does the FA accept finite L?

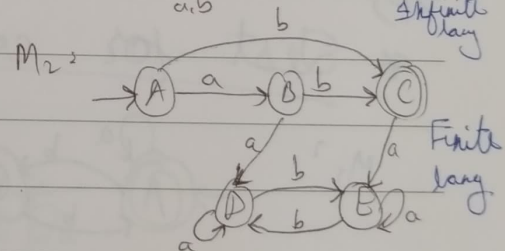
a) ignore/del unreachable states ignore/del dead states and

b) all of the other states from which we can't reach a final state

c) loops/cycles ⇒ Infinite lang.
no loops/cycles ⇒ Finite lang.

Ex:





3) Membership/acceptance

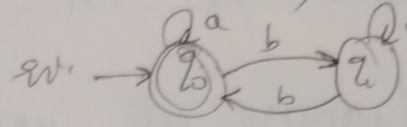
Let M is a machine & 'w' is a word

If $q_0 \xrightarrow{w}$ final state (q0 → some transition path → final state)

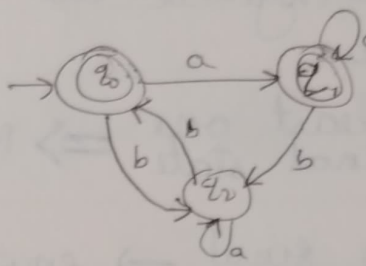
then w is accepted by M

4) Equality of M_1 & M_2:

$M_1 = M_2$ iff $L(M_1) = L(M_2)$

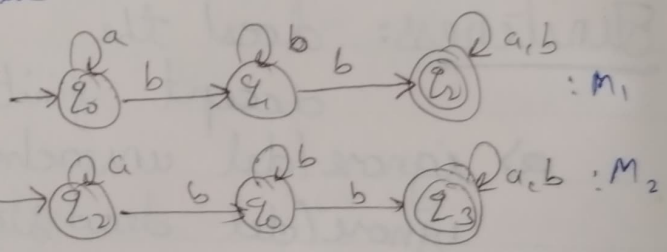
Ex: 

5) Isomorphism:



If we can change the name of states

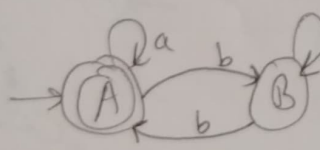
If we can derive any machine M_2 by just changing the name of states of a machine M_1, then M_1 & M_2 are isomorphic.

Ex: 

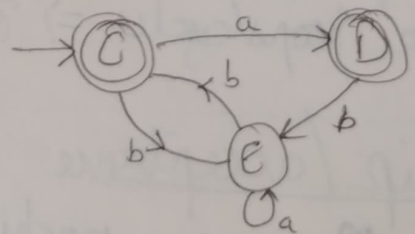
$|Q_1| = |Q_2|$ ⇐

$L(M_1) = L(M_2)$

4) * Test for equality of M_1 & M_2:

M_1: 

=

: M_2 

	a	b
AC	AD	BE
AD	AD	BE
BE	BE	AC

Write all final-final pairs & non-final-non final pairs

As with each transition we get corresponding FF or NFNF state

$M_1 = M_2$

If we get FNF or NFF any one of the transition then $M_1 \neq M_2$

Parul University

Faculty of Engineering & Technology

Equivalence Theorem

Myhill Nerode theorem (Table Filling method)

☆ Minimization of FA

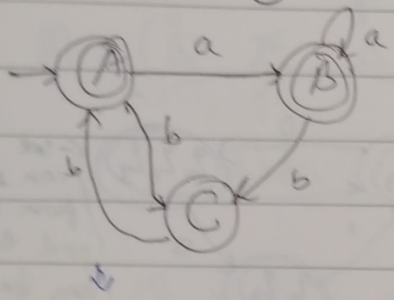
Rules

1) Delete unreachable states

2) Merge final states or NF states

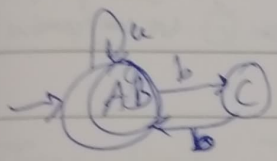
Indirect

1)



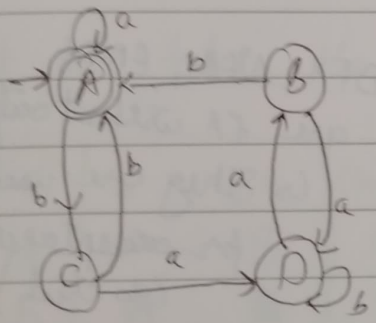
δ	a	b
→ A	B	C
B	B	C
C	–	A

⇓



δ'	a	b
← AB	AB	C
C	–	AB

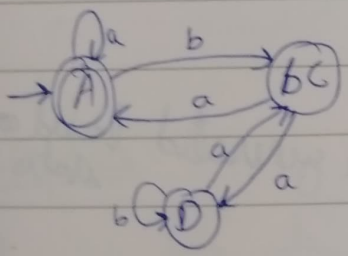
2)



δ	a	b
→ A	A	C
B	D	A
C	D	A
D	B	D

⇒

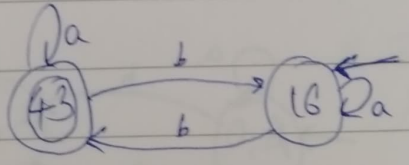
δ'	a	b
→ A	A	BC
BC	D	A
D	BC	D



3)

δ	a	b
→ 1	6	4
2	5	2
③	4	6
④	3	1
5	2	5
6	1	3

We can merge 4 & 3 as both are final & transⁿ corresponding to a & b are also pair of NF & NF

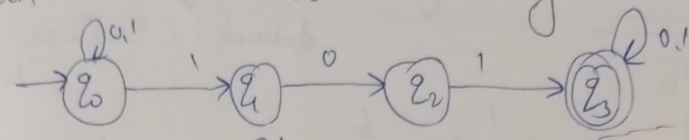


25 & a/b unreachable state

4) Build min FA that accepts strings that contain '101' as substring

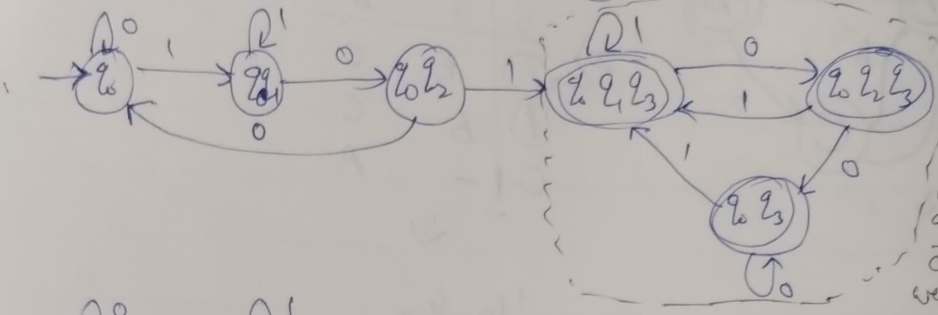
Sol^n

NFA :

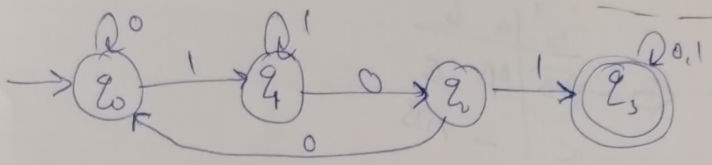


⇩

DFA :



← we can see from this part there is any transi... other part has we can combine...



☆ FA with outputs

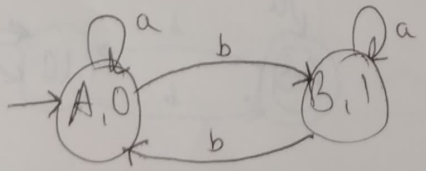
DFA, NFA, ε-NFA are FA without outputs
⇒ They are used for acceptance (to check)

- Moore machine
- Mealy machine

- No final states

☆ Moore Machine (Outputs are generated based on states)

$\Sigma = \{a, b\}$ (input)
$\Delta = \{0, 1\}$ (output)
$Q = \{A, B\}$ (states)



δ : Transition func^n
λ : Output func^n

If Input : n - length
Output : (n+1) length

Ex: $w = b\,a\,b$

Output = 0 1 1 0

As initial state has output symbol 0, first 0 will definitely come. Then according to input symbol (a, b) the state on which it transitted, corresponding output will come.

If $w = \varepsilon$
Output = $\lambda(q_0)$

Mathematically, Moore Machine,

(set of states, set of output, output func")

$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$

(Σ: set of alphabets; δ: transi" func"; q_0: initial state)

$\delta : Q \times \Sigma \to Q$ (same as DFA)

$\Delta : Q \to \Delta$

Transition Table for Moore machine:

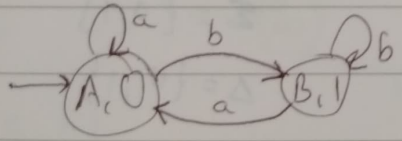
Let us take the same previous transi" graph of Moore machine,

δ	a	b	Output (Δ)
A	A	B	0
B	B	A	1

(Δ ↑ Output; δ ↑ Next state)

Examples

1)



w = abab

output = 0 0 1 0 1 → Count the number of 1's

= 2 ⇒ # of b's in the input string

← This moore machine can be used as a counter (counts the # of b's)

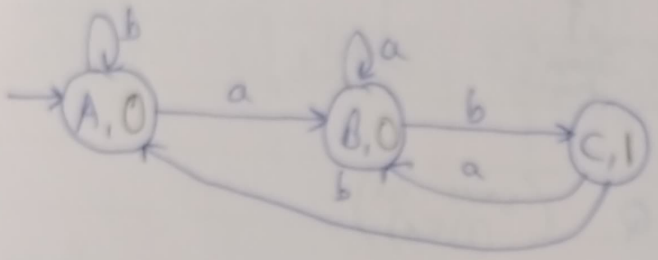
In transition table of moore machine is same as DFA we have to construct moore machine such that each state has every "transition" of Σ

Parul University — Faculty of Engineering & Technology

2) Construct a moore machine that counts the # of occurrences of substring 'ab'. $\Sigma = \{a, b\}$

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

- First make the simple case i.e. (ab) for this example
- Whenever there is ab it should reach the state with 1 as output



Verification

ab → 001

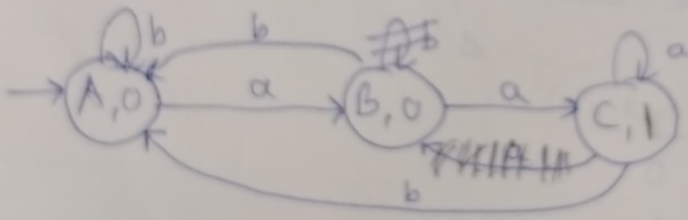
bab → 0001

aab → 0001

abab → 00101

abb → 00100

3) Counts # occurrences of two consecutive a's



Verification

aa → 001

baa → 0001

bbb → 0000

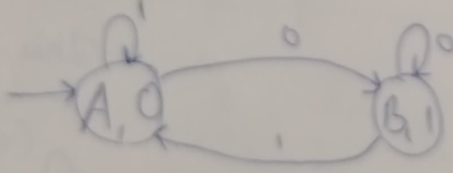
aaa → 0011

aab → 0010

4) Output 1's complement of a binary input string

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

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



Assumption: first character letter should be ignored as for every string it will output 0

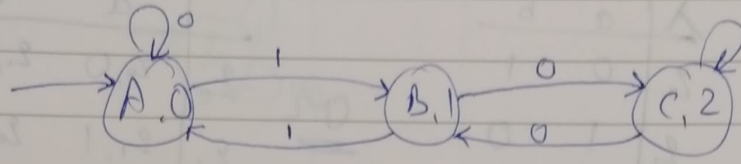
This state is for 0 & 1

This is for 1 to 0

5) input: binary string, output = (input) mod 3

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

$\Delta = \{0, 1, 2\}$



Assumption: Only accept the last letter/char of output & ignore the rest

Firstly mod 3 = rem can be 0 or 1 or 2 ⇒ Three diff states for each

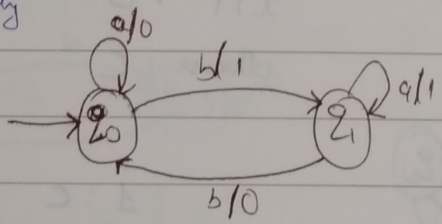
Steps to compute moore machine

Input	dec	Rem
0	0	0
1	1	1
10	2	2
11	3	0
100	4	1
101	5	2

☆ Mealy Machine (Outputs are generated based on transition)

mathematically

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$



Ex: $w = ababa$

outputs = 0 1 1 0 0

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

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

$Q = \{q_0, q_1\}$

$q_0 = q_0$

$\delta : Q \times \Sigma \to Q$

$\lambda : Q \times \Sigma$

If Input = n - length then Output = n - length

If $w = \epsilon$ output = ϵ

Representation using Transition Table

δ	a	b
q_0	q_0	q_1
q_1	q_1	q_0

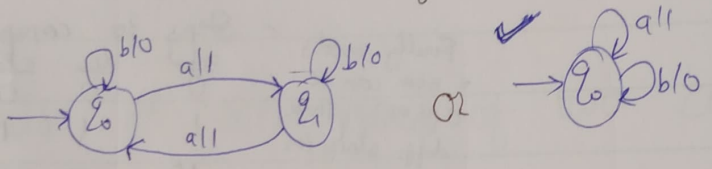
λ	a	b
q_0	0	1
q_1	1	0

or

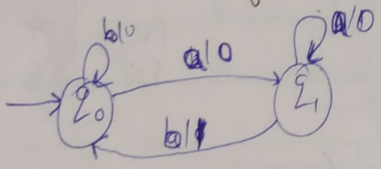
	a	b
q_0	q_0, 0	q_1, 1
q_1	q_1, 1	q_0, 0

Examples

1) Count # occurences of a's in a string, $\Sigma = \{a, b\}$, $\Delta = \{0, 1\}$

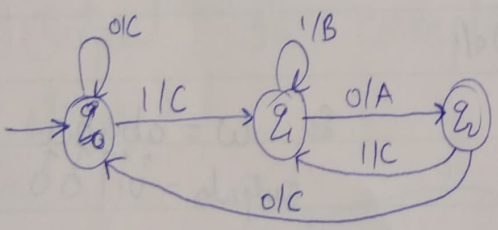


2) Count the # of occurences of 'ab'.



3) $\Sigma = \{0, 1\}$, $\Delta = \{A, B, C\}$

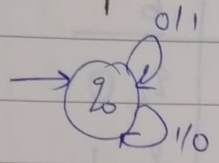
x10 : A
x11 : B
others : C



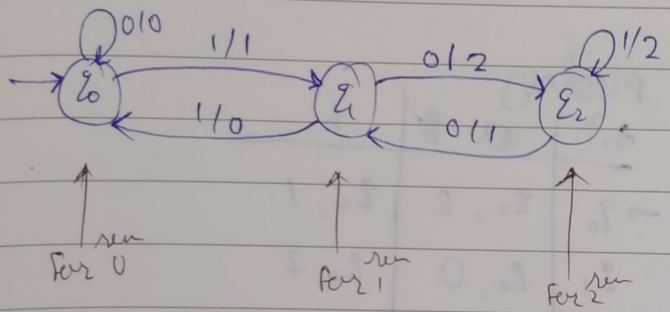
Better to first draw for any one condi" later decide the rest transi" left

1 : C
0 : C
10 : CA
11 : CB
110 : CBA
101 : CAC
1011 : CACB
100 : CAC

4) 1's complement mealey machine



5) $\Sigma = \{0,1\}$, $(input)_2 \bmod 3$

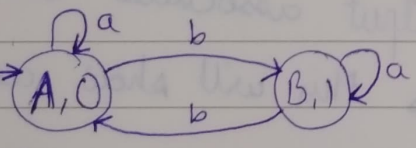


bin	dec	rem
0	0	0
1	1	1
10	2	2
11	3	0
100	4	1
101	5	2

☆ Conversion (Moore ⇌ Mealey)

* Moore Machine → Mealey Machine

Ex: (Moore)



TT for moore:

S	a	b	λ
A	A	B	0
B	B	A	1

⇒

S	a	b
A	A,0	B,1
B	B,1	A,0

In moore A corresponds to 0 ⇔ In mealey A,0 corresponds

- It is easier to convert moore to mealey using TT.
- We just have to copy transition as same and assign the output value from moore TT i.e. whatever the output corresponds to some state same will be there in mealey with each transition

while mealy → more # of states need not be

2)

Moore

δ	a	b	λ
→ q_0	q_1	q_2	0
q_1	q_0	q_3	1
q_2	q_3	q_0	0
q_3	q_2	q_1	1

⇒

Mealy

δ, λ	a	b
→ q_0	q_1, 1	q_2, 0
q_1	q_0, 0	q_3, 1
q_2	q_3, 1	q_0, 0
q_3	q_2, 0	q_1, 1

3)

δ	a	b	λ
→ q_0	q_2	q_4	0
q_1	q_0	q_2	1
q_2	q_3	q_1	2
q_3	q_4	q_0	0
q_4	q_1	q_3	1

⇒

Mealy

δ, λ	a	b
→ q_0	q_2, 2	q_4, 1
q_1	q_0, 0	q_2, 2
q_2	q_3, 0	q_1, 1
q_3	q_4, 1	q_0, 0
q_4	q_1, 1	q_3, 0

* Mealy → Moore Machine

First find what are the output associated with each state of moore machine as this will show as the states of mealy machine

Ex.

1)

δ, λ	a	b
→ q_0	q_2, 0	q_1, 0
q_1	q_0, 0	q_2, 1
q_2	q_1, 1	q_0, 0

→

δ	a	b	λ
→ q_0	$q_{2,0}$	$q_{1,0}$	0
⇒ $q_{1,0}$	q_0	$q_{2,1}$	0
$q_{1,1}$	$q_{1,1}$	$q_{2,1}$	1
$q_{2,0}$	$q_{1,1}$	q_0	0
$q_{2,1}$	$q_{1,1}$	q_0	1

$q_0 \to 0$

$q_1 \Rightarrow 0$

$q_2 \to 1$

Mealy

2)

δ, λ	0	1
$\rightarrow q_0$	$q_2, 1$	$q_3, 1$
q_1	$q_3, 1$	$q_4, 1$
q_2	$q_0, 1$	$q_1, 0$
q_3	$q_4, 0$	$q_2, 0$
q_4	$q_1, 0$	$q_0, 0$

Initial state can be q_{00} or q_{01} (any one)

$\Rightarrow$

δ	0	1	λ
$\rightarrow q_{00}$	q_{21}	q_3	0
$\rightarrow q_{01}$	q_{21}	q_3	1
q_1	q_3	q_{41}	1
q_{20}	q_{01}	q_{10}	0
q_{21}	q_{01}	q_{10}	1
q_3	q_{40}	q_{20}	1
q_{40}	q_1	q_{00}	0
q_{41}	q_1	q_{00}	1

$q_0 \leftarrow 0$, $q_1 \leftarrow 1$, $q_2 \leftarrow 0$, $q_3 \leftarrow 1$, $q_4 \leftarrow 0$

☆ Regular Languages (RL)

Here FA focused only on FA without output

→ A language is regular iff ∃ FA that accepts every word in Lang.

⇒ L is RL if ∃ FA, M (name) s.t. L(M) = L (Lang accepted by M)

- Every finite language is a regular language
- L = ϕ = { } is also a regular language

Examples: Σ = {a, b}

1) L = {ab, abb, aab} ⇒ Finite ⇒ RL

2) L = {$a^m b^n$ | m, n ≥ 0} ⇒ It is non-finite, but FA is possible for this ⇒ RL

3) L = {$a^m b^n$ | m ≥ 2, n ≥ 3} ⇒ RL

4) L = {$a^m b^n$ | m * n = const.} ⇒ RL {const can be any no. other than ∞ = }

5) L = {$a^n b^n$ | m = n = const} ⇒ RL

6) L = {$a^m b^n$ | 1 ≤ m = n ≤ const} ⇒ RL

7) $L = \{ a^m b^n \mid 1 \le m \le n = const \} \Rightarrow RL$

8) $L = \{ a^n b^n \mid n \ge 1 \} \Rightarrow L = \{ ab, aabb, aaabbb, \ldots \}$
FA isn't possible for this as it needs
NRL $\Leftarrow$ ∞ states & FA can't have ∞ states

9) $L = \{ a^m b^n \mid m = 10, 20, 30, \ldots \; n = 5, 10, 15, 20, \ldots \} \Rightarrow m \pmod{10} = 0 \Rightarrow$ 10 states
$n \pmod{5} = 0 \Rightarrow$ 5 states
RL $\Leftarrow$ Finite states

10) $L = \{ a^m b^n \mid m < n \} \Rightarrow NRL$ (it is infinite L but also we have to keep track of # a's & b's which isn't possible using Finite states)

11) $L = \{ a^m b^n \mid m > n \} \Rightarrow NRL$ "

12) $L = \{ a^m b^n \mid m \neq n \} \Rightarrow NRL$ "

13) $L = \{ a^m b^n \mid m = 2n \} \Rightarrow NRL$ "

14) $L = \{ a^m b^n \mid m = n^2 \} \Rightarrow NRL$

15) $L = \{ a^m b^n \mid m + n = 10 \} \Rightarrow$ Finite $\Rightarrow RL$

16) $L = \{ a^m b^n \mid gcd(m, n) = 1 \} \Rightarrow$ store remainders (Not possible by FA)
NRL

17) $L = \{ a^m b^n \mid m + n = even \} \Rightarrow RL$

18) $L = \{ w w^R \mid w \in \Sigma^* \}$ (Reverse of word) $\Rightarrow L = \{ abba, abbbba, \ldots \}$

(as we have to keep a track of w) NRL $\Leftarrow$

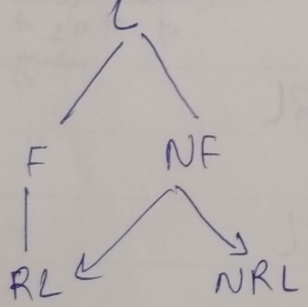
19) $L = \{ w \in \Sigma^* \mid |w_a| = |w_b| \} \Rightarrow L = \{ aabbab, ababba, \ldots \}$

NRL $\Leftarrow$

* Properties of RL:

1) Finite Lang. $\Rightarrow$ RL

2)



(Finite, Non-finite)

3) RL = F or NF

4) Every NRL has to be Infinite language

5) Subset of a RL need not be regular

Ex. $L = \{ a^m b^n \mid m, n \geq 0 \} \Rightarrow$ RL

$L' = \{ a^m b^n \mid m = n \geq 0 \} \Rightarrow$ NRL $\quad L' \subset L$

6) Every finite subset of RL is RL

A) Every finite subset of NRL is RL

8) Subset of NRL need not be NRL

9) Superset of RL need not be R
" " NR " " " NRL

10) Finite Union of Regular Language is a RL

11) Finite Intersect of RL is a RL

☆ Regular Expressions (RE)

(Used in some programming lang. to find substring)

→ One way to represent a RL.

→ Operators: * , . , + (OR/Union)
* ↓ Kleen closure
. ↓ Concatenation
+ in superscript ↓ $r = a^+ = \{a, aa, aaa, \ldots\}$
$r = a^+ = a^* - \{\epsilon\}$

Ex:
1) $r = a^* = \{\epsilon, a, aa, aaa, aaaa, \ldots\}$
2) $r = a+b = \{a, b\}$
3) $r = ab = \{ab\}$
4) $a^* + ba = \{\epsilon, a, aa, \ldots\} \cup \{ba\}$
5) $(ab)^* + b^*a^* = \{\epsilon, ab, abab, ababab, \ldots\}$
$\cup \{\{\epsilon, b, bb, bbb \ldots\} . \{\epsilon, a, aa, aaa, \ldots\}\}$