UNIT-1

Theory of Computation

Introduction:

In theoritical computa science and mathematics, the theory of Computation is the branch that deals with how efficiently problems can be solved on a model of computation, using an algorithm.

This field is divided into three major branches:

→ Automata theory and Language

→ Computability theory

→ Complexity theory

Which are linked by the question: "What are the fundamental capabilities and limitations of Computers?"

Automata Theory:

Automata theory is the study of abstract machines and the computational problems that can be solved using these machines. These abstract machines are called Automata.

An Automata can be a finite representation of a formal language that may be an infinite set.

Automata are used as theoritical models for computing machines, and are used for proofs about computability.

Ex:

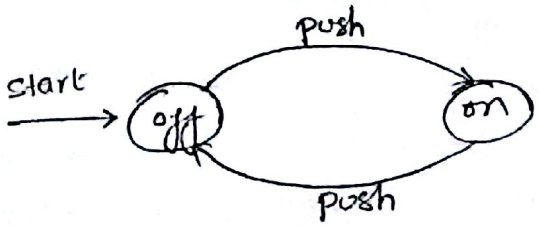


Fig: A finite automaton modeling an on/off switch.

Ex:2

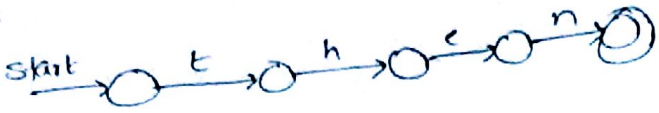


Fig: A finite automaton modeling recognition of then.

Formal Language theory: Language theory is a branch of mathematics concerned with describing languages as a set of operations over an alphabet. It is closely linked with automata theory, as automata are used to generate and recognize formal languages.

There are several classes of formal Languages, each allowing more complex language specification than the one before it, i.e chomsky hierarchy, and each corresponding to a class of automata which recognizes it.

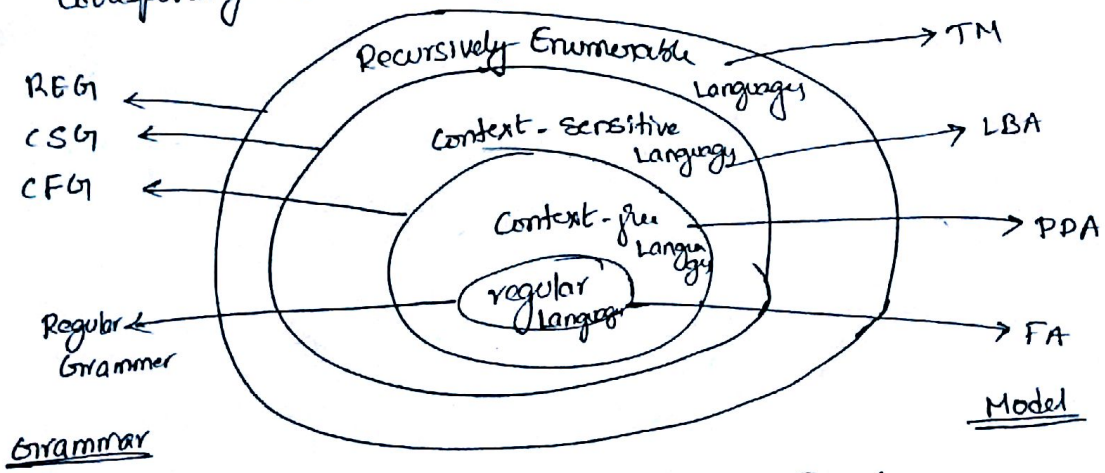


Fig: Chomsky hierarchy of Formal Languages

Note:
The no. of languages accepted by an automata is called expressive power or accepting power of automata.

Computability Theory:
Computability theory deals primarily with the question of the extent to which a problem is solvable on a computer.

Ex: Unsolvable problems
1. The post correspondence problem
2. Determining if a context-free grammar generates all possible strings or if it is ambiguous

③ Halting problem of a Turing Machine.

Complexity Theory:

Complexity theory considers not only whether a problem can be solved at all on a computer, but also how efficiently the problem can be solved.

Two major aspects are considered: Time Complexity and Space Complexity, which are respectively, how many steps does it take to perform a computation, and how much memory is required to perform that computation.

Fundamentals:

Alphabet: Non-empty finite set of symbols is called as an Alphabet.

It is denoted by Σ.

Ex: $\Sigma = \{a, b, \ldots z\}$

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

$\Sigma = \{$ ASCII symbols $\}$

String: The sequence of symbols from the alphabet Σ is called as string.

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

$w = 10, 001, 010 \ldots$

Length of the string:

If w is any string over Σ then the no. of symbols involved in the sequence of string is called the length of the string and it is denoted by $|w|$

Ex: $w = 10 \Rightarrow |w| = 2$

$w = 1010 \Rightarrow |w| = 4$

Empty String: A string of length zero (0) a string without any symbol is known as empty string and it is denoted by ϵ.

$$w = \epsilon \text{ iff } |w| = 0$$

Note:

$$w \cdot \epsilon = \epsilon \cdot w = w$$

Ex: $TOC \cdot \epsilon = \epsilon \cdot TOC = TOC$

Substring: Let u, w be the two strings over the same alphabet Σ then u is said to be a substring of w, if u is obtained from w.

Note: If u is a substring of w, then $|u| \leq |w|$

Every string is a substring to itself

Empty string ϵ is a substring for every string.

Ex: $w = TOC$

possible Sub Substrings: $\epsilon, T, O, C, TO, OC, TOC$

Prefix: The sequence of ~~starting~~ of ~~leading~~ symbols is called as prefix.

Suffix: The sequence of ~~ending~~ of ~~trailing~~ symbols is called as suffix.

Ex:

$w = TOC$

Prefixes	Suffixes
ϵ, T, TO, TOC	TOC, OC, C, ϵ

Power of Alphabets:

If Σ is any alphabet then Σ^k is the set of all strings of length k.

Eg: $\Sigma = \{0,1\} \Rightarrow \Sigma^1 = \{0,1\}$

$\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

$\vdots$

$\Sigma^k = \underbrace{X \cdot X \cdots X}_{k \text{ bit}}$

Kleen closure

Note: The set of all the strings over an alphabet Σ is conventionally denoted by Σ^*

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

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

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

positive closure:

The set of non-empty strings from the set of an alphabet Σ is denoted by Σ^+

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

Language: The collection of strings from the input alphabet Σ is called as Language.

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

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

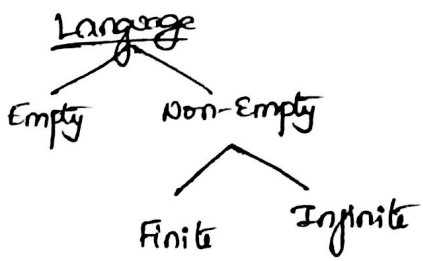
$L = \{(01)^n / n \geq 1\}$

$L = \{0^n 1^m / m, n \geq 1\}$

① If Σ is any alphabet, then Σ^* is called universal language.

② If L is any language defined over Σ then $L \subseteq \Sigma^*$

③



Empty Language:
The language which does not contain any string, even empty string ϵ is called empty language, and is denoted by ϕ.

Non-Empty Language:
The Language L which contains atleast one string is called as Non-Empty Language.

Eg: $L = \{\epsilon\} \Rightarrow |L| = 1$

$L = \{a\} \Rightarrow |L| = 1$

$L = \{a, \epsilon\} \Rightarrow |L| = 2$

$L = \{a^n / n \geq 1\}$

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

$|L| = \infty$

Finite Language: The Language which contains finite no. of strings but length of each and every string is finite called as Finite Language.

Eg: $L = \{\epsilon\} \Rightarrow |L| = 1$

$L = \{a^n b^n / \ 1 \le n \le 10\} \Rightarrow |L| = 10$

$L = \{a^n / \ 1 \le n \le |TOC|\} \Rightarrow |L| = 3$

Infinite Language: The Language which contains infinite no. of strings but the length of each & every string is finite is called as Infinite language.

Eg: $L = \{a^n / \ n \ge 1\}$

$L = \{(ab)^n / \ n \ge |TOC|\}$

$L = \{a^m b^n / \ m > 0, n > 0\}$

Eg: ① The Language of all strings consisting of n 0's followed by n 1's for some $n \ge 0$

$L = \{\epsilon, 01, 0011, 000111 \ldots\}$

② The set of strings of 0's and 1's with equal number of each

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

③ The set of binary numbers whose value is prime.

$L = \{10, 11, 101, 111, 1011 \ldots\}$

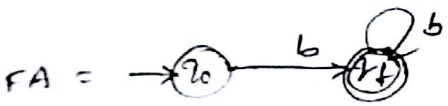
④ Σ^* is a Language for any alphabet Σ

⑤ $\emptyset$ is the empty Language, is a Language over any alphabet

⑥ $\{\epsilon\}$, the Language consisting of only the empty string is also a Language over any alphabet.

Regular Language: The Language which is accepted by FA or generated by RE or generated by RG is called as Regular Language.

Ex: $L = \{ b^n / n \geq 1 \}$

FA = 

$RE = b^+$

RG : $S \rightarrow Sb / b$

The Language which is not accepted by FA is called Non-regular Language.

Finite Automata: The Mathematical representation of a regular Language is called FA.

(or)

The Mathematical system that process the string is called FA

(or)

Finite Automata is 5-tuple

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

Where

$Q \longrightarrow$ set of all states

$\Sigma \longrightarrow$ i/p alphabet

$\delta \longrightarrow Q \times \Sigma = Q$ Transition function

$q_0 \longrightarrow$ Initial state

$F \longrightarrow$ Final state

① In general the FA is DFA

② Every DFA has only one initial state

→ (q0)

③ FA can be constructed with (0) without final state

o(0) | (0) more.

◎ → Final state

④ DFA is a complete system that can process both valids as well as Invalids.

⑤ The system is complete iff a transition a transistion function is defined for each and every input symbol at each and every state.

⑥ DFA Moves to exactly one state after taking the i/p symbol from Σ

⑦ No. of transitions at a state = |Σ| → no. of input symbols

⑧ Total no. of transitions in DFA = |Σ| x |Q|

Instantaneous Description (ID):

→ ID Describes the movements of FA.

→ The Movements of FA depends on two entities

current state current i/p symbol

$$\delta(q_i, a) \rightarrow q_j$$

Representation of FA:

FA can be represented in two ways

→ Transistion Diagram

→ Transistion Table

Transition diagram:

A Transition diagram for a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is a graph defined as follows.

(a) For each state in Q there is a node

(b) For each state in Q and each input symbol a in Σ, let $\delta(q,a) = p$. Then transition diagram has an arc from node q to node p, labeled a.

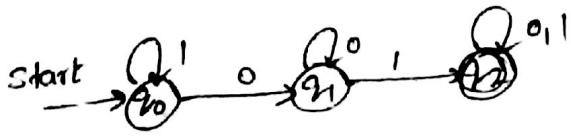
(c) There is an arrow into the start state q_0, labeled start. This arrow does not originate at any node.

(d) Nodes corresponding to accepting states are marked by a double circle. States not in F have a single circle.

Ex:

Construct the FA that accepts all the strings of 0's & 1's where every string contains the substring 01

$\Sigma = \{0,1\}$ $\quad L = \{01, 001, 0010 \cdots\}$



Transition Tables:

A transition table is a conventional, tabular representation of a function like δ, that takes two arguments and return a value.

→ The rows of the table corresponding to the states, and columns corresponding to the inputs.

→ The entry for the row corresponding to state q and the column corresponding to input a is the state $\delta(q,a)$

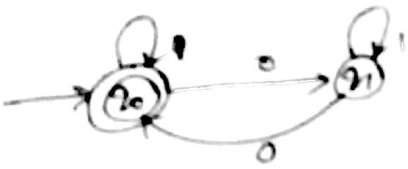
Ex: Transition table for the above example

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

Ex2: Construct the FA, that accepts all the strings of 0's & 1's where no of 0s in a string is

(i) even (ii) odd

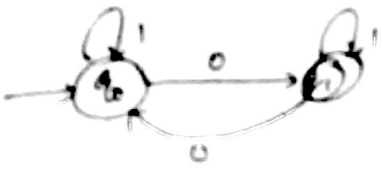
(i)



	0	1
*q_0	q_1	q_0
q_1	q_0	q_1

Transition table

(ii)

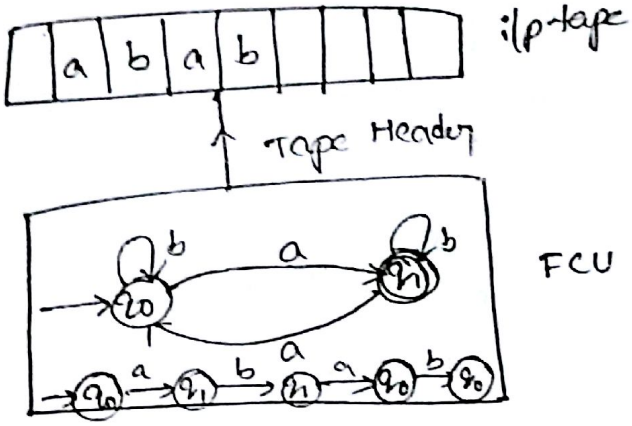


	0	1
→ q_0	q_1	q_0
q_1	q_0	q_1

Block diagram of FA:

FA consists of 3 components

1. Input Tape 2. Tape Header 3. Finite Control unit



Input tape: Input tape is divided into finite no. of cells where each cell hold only one i/p symbol.

→ At any point of time the i/p tape contains only one i/p string.

Tape Header:

Tape Header scans the i/p symbol from the i/p tape starting from left most end.

After scanning the i/p symbol from the tape, the header moves to exactly one state in the right direction.

Finite Control unit:

→ FCU takes care of the movements of FA, to process the i/p string.

→ FCU implements the movement of FA by ID and Complete the process of the i/p string.

Acceptance by FA:

Let w be any string from Σ, corresponding to w if there exists a transistion path which starts at initial state and ends in any one of the final state, then the string w is accepted by FA.

The set of all the strings which are accepted by FA is called as Language of FA.

$$L(FA) = \{ w \in \Sigma^* \mid \delta(q_0, w) = FS \}$$

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

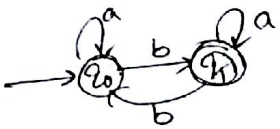
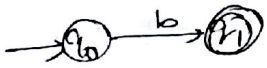


Fig: FA for accepting all strings of a's & b's, where $|w_b| = odd$.

① $w = b$

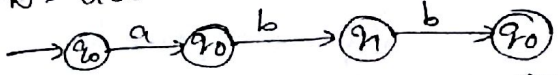


Accepted.

② $w = ab$

Accepted

③ $w = abb$



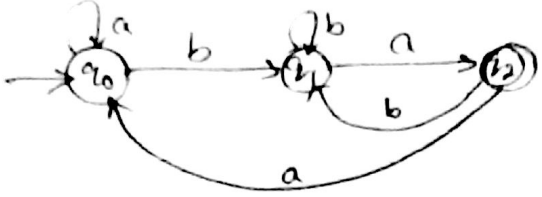
Non-Final state - Not Accepted

④ $w = abbabb$

$$\delta(q_0, abbabb) = \delta(q_0, bbabb)$$
$$= \delta(q_1, babb)$$
$$= \delta(q_0, abb)$$
$$= \delta(q_0, bb)$$
$$= \delta(q_1, b)$$
$$= \delta(q_0, q_0 - \text{Non-Final state}$$

Rejected

Ex: Which of the following strings accepted by FA



$L = \{ w \in \Sigma^* \mid w \text{ is ending with } ba \}$

(a) ab (b) ba (c) aba (d) baba (e) ababa (f) abbba

$$\delta(q_0, ab) = \delta(q_0, b)$$
$$= q_1 \quad \text{Rejected}$$

Note: FA accepts empty string $\in$ iff the initial state = Final state

Productive state:

The state which are involved in the process of any i/p string is called productive state.

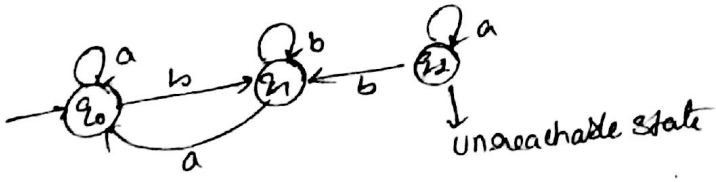
Non-productive state: The state which is not involved in the process of any valid i/p string is called as non-productive state.

Types of Non-productive state:

1. Unreachable state
2. Dead state
3. Equal state

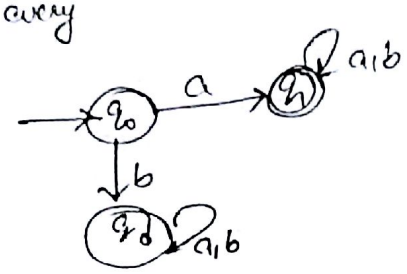
Unreachable state: The state that cannot be reached from initial state is called unreachable state.

Ex:



Dead state: A non-final state from which we cannot come back and goes to itself by i/p is called as dead state.

Ex:

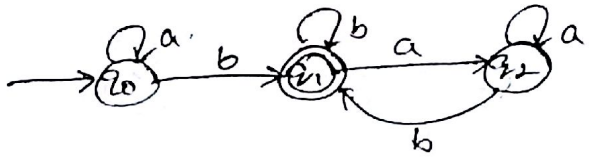


Equal states: Two states P, q are said to be equal if

$$\delta(P, x) = \delta(q, x) \quad \forall\ x \in \Sigma^*$$

i.e both of them goes to final or non-final state.

Ex:



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

$w = xb$

$\delta(q_0, ab) = \delta(q_2, ab) = q_1$ Final state

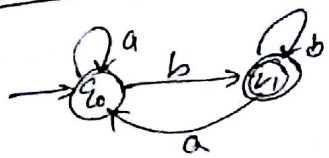
$\delta(q_0, a) = \delta(q_2, a) = q_0, q_2$ Non-final state

$\delta(q_0, aba) = \delta(q_2, aba) = q_2$ Non-final state

$\Rightarrow \delta(q_0, x) = \delta(q_2, x) \quad \forall\ x \in \Sigma^*$

$\therefore q_0 = q_2$

Minimal DFA:



Note: The FA which is free from unreachable & equal states is called minimal FA.

① Construct the minimal FA that accepts all the strings of 0's & 1's

(a) including ϵ

(b) excluding ϵ

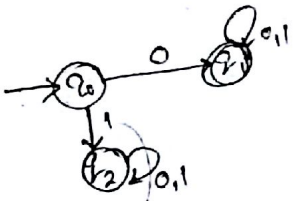
(a) 

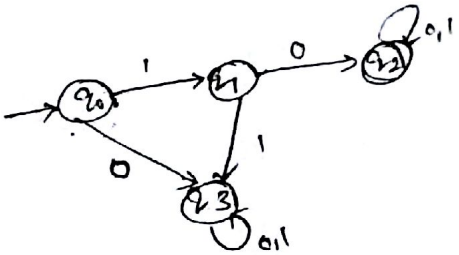
(b) 

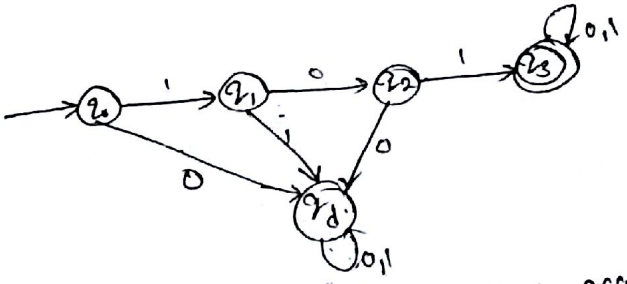
② Construct the minimal FA that accepts all strings of 0's & 1's where every string starts with

(a) 0 (b) 10 (c) 101

(a) Sol:

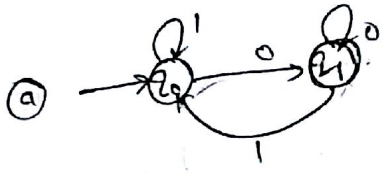


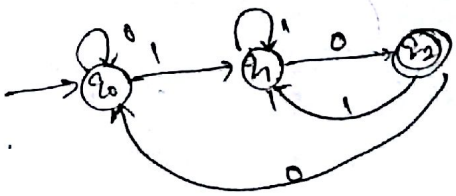
(b) 

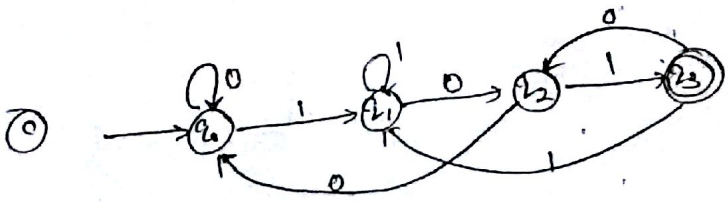
(c) 

③ Construct the minimal FA that accepts all strings of 0s & 1's where every string ends with

(a) 0 (b) 10 (c) 101

(a) 

(b) 

(c) 

Note: ① Let L is a set of all strings of 0's & 1's where every string starts with a substring s of length n

$|w| = sx, \quad |s| = n$

∴ The no. of states in minimal FA to accept $L = n+2$

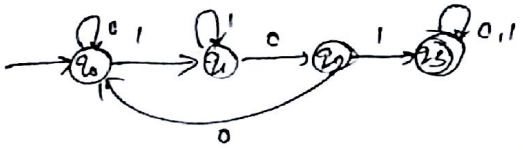
② Let L is a set of all strings of 0's & 1's where every string ends with a substring s of length n, then the no. of states in minimal FA to accept L is $n+1$

Q Construct the minimal FA that accepts all the strings of 0's & 1's where every string contain the substring.

ⓐ 0 ⓑ 10 ⓒ 101

ⓐ 

ⓑ 

ⓒ 

Note: Let L is a set of all strings of 0's & 1's where every string Contains the substring s of length n, then the no of states in minimal FA will be $n+1$

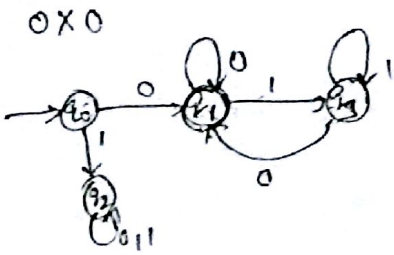
$w = x[s]x$

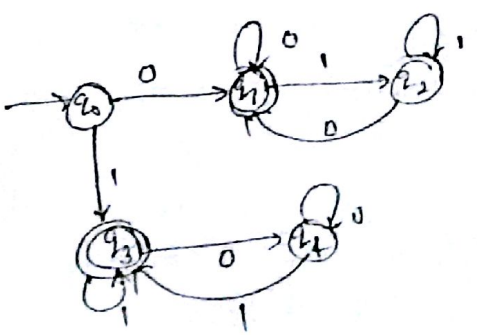
Q Construct the minimal FA that accepts all the strings of 0's & 1's where every string ⓐ starts and ends with 0

ⓑ Starts and ends with same symbol

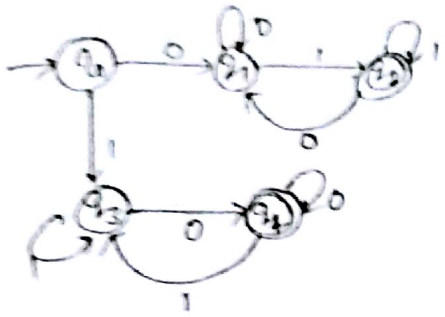
ⓒ Starts and ends with different symbol

ⓐ 0x0



ⓑ 

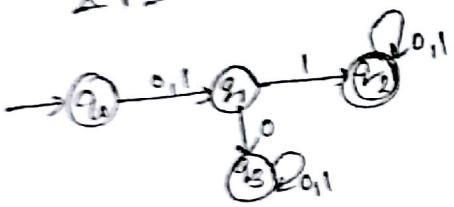
(c)



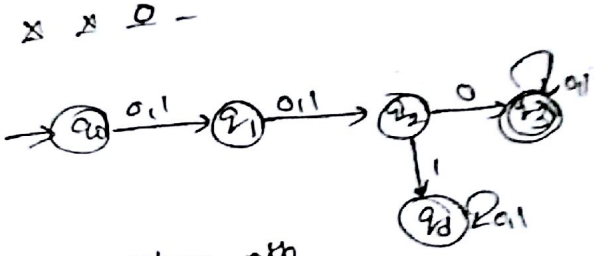
(5) Construct the min FA that accepts all the strings of 0's & 1's where

(a) 2nd symbol from left hand side is 1 (b) 3rd symbol from left hand side is 0

(a)



(b)

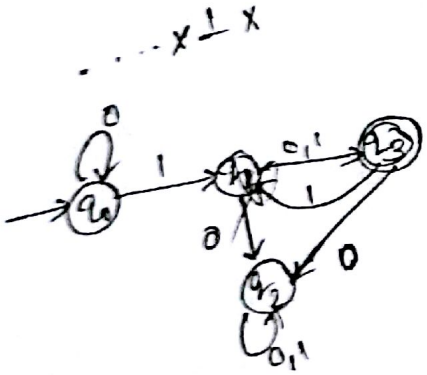


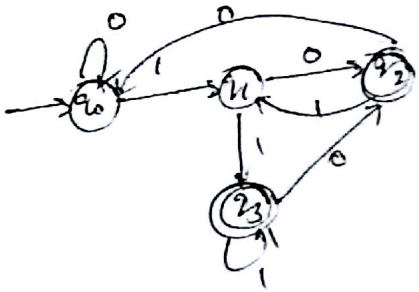
Note: The minimal FA that accepts all the strings of 0's & 1's where nth Symbol from LHS is fixed contains exactly $n+2$ states.

(6) Construct the min FA that accepts all strings of 0s & 1's where

(1) 2nd symbol from R.H.S is 1

(2) 3rd symbol from R.H.S is 0

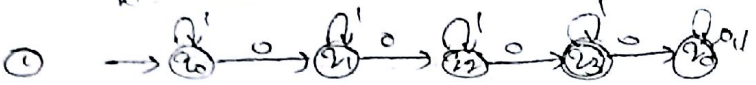




Construct the min FA that accepts all the strings of 0's & 1's where no of zeros in a string is

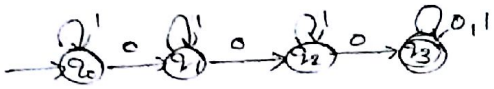
1. Exactly 3
2. Almost 3
3. Atleast 3
4. even
5. odd
6. $0 \pmod 3$
7. $\geq 2 \pmod 3$

i) 0X0X0X



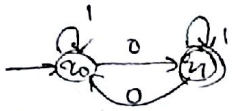
ii) $|w| \leq 3$

iii) $|w| \geq 3$

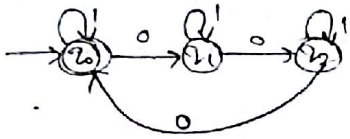


iv)

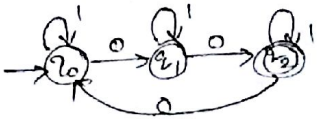
v)



vi)



vii)

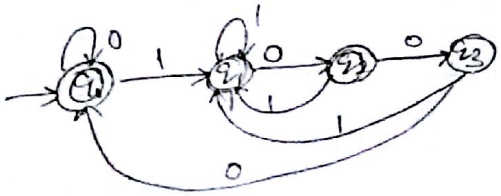


Complement of Language L:

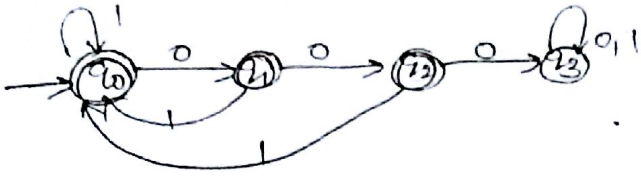
If L is any regular language then the complement of L is $L' = \Sigma^* - L$ is also regular language.

The DFA for L' can be obtained by interchanging Final and Non-Final States from the DFA of L.

Ex: Construct the MFA that accepts all the strings of 0's & 1's where every string doesnot ends with 100.



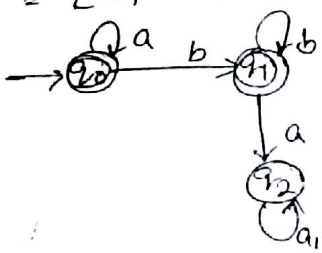
① Construct the MFA that accepts all the strings of 0's & 1's where every string does not contains the substring 000.



② Construct the MFA for the following language.

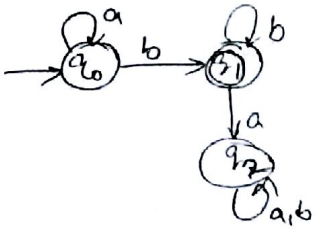
① $L = \{a^m b^n \mid m, n \geq 0\}$

$= \{\epsilon, a, b, aaa, bbb \cdots\}$



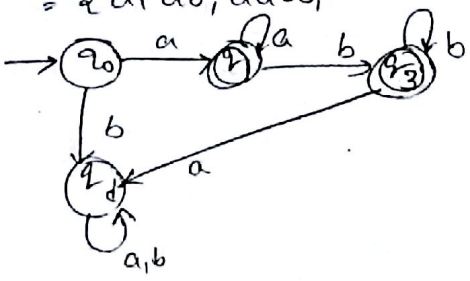
② $L = \{a^m b^n \mid m \geq 0, n \geq 1\}$

$= \{b, ab, aabb \cdots\}$



③ $L = \{a^m b^n \mid m \geq 1, n \geq 0\}$

$= \{a, ab, aabb,$



④ $L = \{a^m b^n \mid m+n = \text{even}\}$

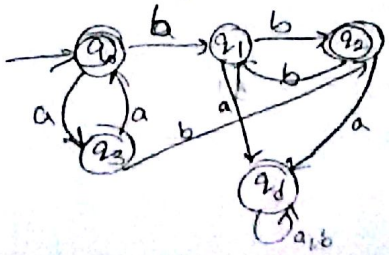
ϵ, ab, aabb, aaabbb, ...

m+n

↓ ↓

even even

odd odd



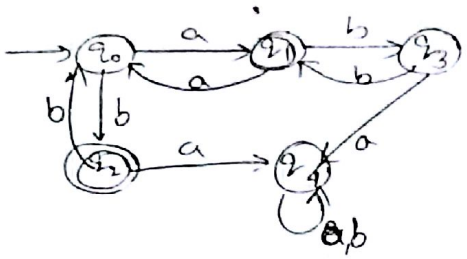
$L = \{a^m b^n / m+n = \text{odd}\}$

↓ ↓

even odd

odd even

- $\{b, a, aab, abb\}$ aaaabbb



Ex: Construct the NFA that accepts all strings of 0's & 1's whose integer equivalent

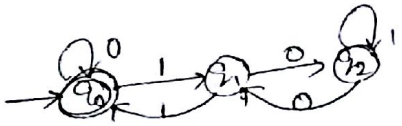
① Divisible by 3 ② $0 \pmod 4$, ③ $2 \pmod 5$

① Sol: 0, 1, 2, 3, 4, 5, 6 ...

$L = \{\epsilon, 11, 101, 110, 111, 1000, 1001 \ldots\}$

$= 4 \pmod 3$

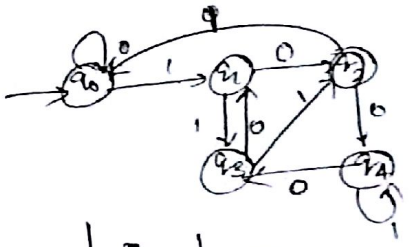
$= 1$



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

③ Sol: 0, 1, 2, 3, 4, 5, 6, 7 ... (mod 5)

$L = \{10, 111, 1100 \ldots\}$



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

② Sol: 0, 1, 2, 3, 4, 5 ...

q_0, q_1, q_2, q_3

$L = \{\epsilon, 0, 100, 1000, 1100, 10000 \ldots\}$

$1 = 5 \pmod 4$

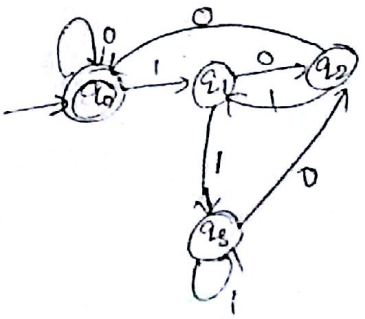
101

$2 = 6 \pmod 4$

110

$3 = 7 \pmod 4$

$0 = 8 \pmod 4$



	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