Page No.
Date

TOC Module 1.

* Automata

→ An automata is an abstract model of a digital computer.

→ An automation has a mechanism to read the input tape.

→ A Machine with intermediate stages which on receival of inputs, chooses a path from the initial state to the final state to produce a valid output. such a machine having a finite number of intermediate states is called finite automata

* Computability

→ Deals with question of how problems can be solved algorithmically.

→ Finite automata is an example of a simple computational model.

Page No.
Date

* Complexity

→ Complexity focuses on the resources (time and space) required to solve computational problems.

→ Time complexity of a finite automata is usually measured in the no. of steps taken from initial state to final state.

* Common Terminologies.

① Alphabet : set of input symbols

② String : Sequential arrangement of input symbols.

③ Empty string : ε or null.

④ Languge : Collection of strings which satisfy the finite automata.

* Finite State Machine (FSM)

→ It is a mathematical model of computation used to design and describe systems with discrete states.

Date

→ Consists of finite set of states 's which on receiving an Input provides an output.

* Formal definition of Finite Automata

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

Q = set of states
Σ = Alphabetic set of all input symbols
δ = Transition Function.

$\delta = Q \times \Sigma \longrightarrow Q$ (used to find the next state)

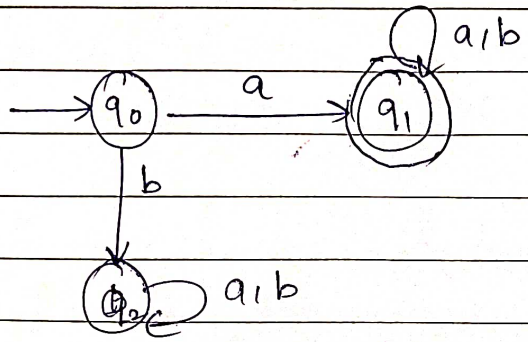
q_0 - Initial state.
F = Set of final state.

* Deterministic Finite Automata. (DFA)

→ DFA is a type of Finite state Machine (FSM) that recognises a set of strings in a langugge, where each string is composed of symbols from a finite alphabet.

→ DFAs have a deterministic behaviour, meaning that for each combination of current state and an input symbol, there is exactly one state which is next.

Eg. String starting with 'a' for $\Sigma(a, b)$



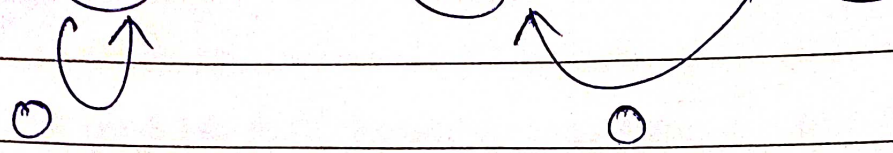
$Q = \{ q_0, q_1, q_2 \}$

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

$F = \{ q_1 \}$

$q_0 = q_0$

Q / Σ	a	b
q_0	q_1	q_2
q_1	q_1	q_1
q_2	q_2	q_2



* NFA (Non - Deterministic Finite Automata.

→ NFA is another type of FSM that recognizes languages. similar to DFA

→ However, NFA can have multiple possible transitions from a given state for a particular input symbol.

Page No.
Date

* Properties of NFA.

① Can have more than one transition for a symbol.

② Not necessary to perform transition for every symbol.

③ No dead state

④ Practical Implementation is not possible.

⑤ No need to complete NFA.

DFA

$\delta : Q \times \Sigma = Q.$

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

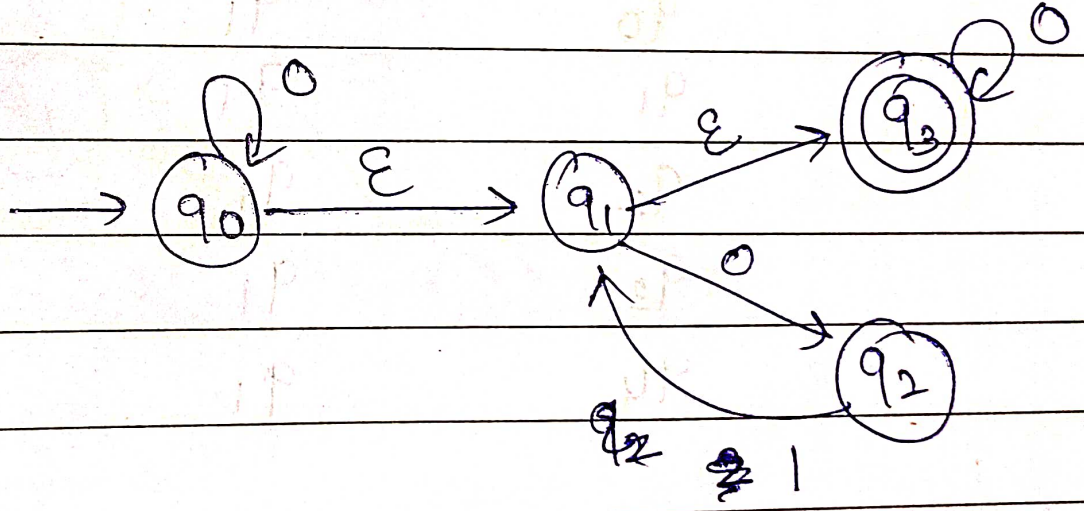
DFA	NFA
① There is only one transition for an input	① Allows multiple transitions from a state for one input.
② Transition function $\delta: Q \times \Sigma \rightarrow Q$	② Transition function. $\delta: \Sigma \times Q \rightarrow 2^Q$
③ Accepts a string if the computation path leads to an accepting state	③ Accepts the string if there atleast one accepting path.
④ Simpler to implement and analyze	④ More complex due to multiple transition.
⑤ No need for conversions, already deterministic.	⑤ Can be converted to equivalent DFA.

* Epsilon - NFA.

→ It is an extension of NFA which contains epsilon NFA transitions.

→ It has the notion where transition can occur without consuming any input symbol.

Eg: Example :



Date

* Applications of NFA / DFA / ε-NFA.

① Lexical Analysis in compilers: Used for lexical analysis to recognise tokens in programming languages

② Spell checkers: NFA/DFA are employed in spell checkers to identify and correct misspelled words.

③ Pattern Matching: NFAs can be used for pattern matching in text processing application States represents patterns, and transitions occur based on matching input characters

Page No.
Date

Module 2.

* Regular Expression.

→ Regular Expressions are used to describe regular languages, which are a class of languages recognised by finite automata.

Some operations which can be performed on regular expressions

① Union (|) : If you have two regex R, S then union is denoted by R|S. This represents the strings can be from either R or S.

② Concatenation (.) : For R and S, concatenation denotes RS and represents the language formed from strings concatenated from both R and S.

③ Kleen Closure: If R is a regular expression, then R* represents the language formed by zero or more (ten concatenations from string R.

④ Intersection (∩) : If you have two regular expressions R and S, then R∩S represents strings present in both R and S.

Page No.
Date

(5) Complement (') : If R is a regex, then R' (complement of R) represents the language of all strings over the alphabet that are not in language R.

* Identities of Regular Expressions.

(1) $\phi + R = R + \phi = R$.

(2) $\phi . R = R . \phi = \phi$.

(3) $\varepsilon R = R \varepsilon = R$.

(4) $\varepsilon^* = \varepsilon$

(5) $\phi^* = \varepsilon$

(6) $\varepsilon + RR^* = RR^* + \varepsilon = R^*$

Page No.
Date

* Conversion of FA to RE

Method 1: Arden's Theorem.

→ If P & Q are two RE over Σ and if if P does not contain ε then the following equation in R given by

$R = Q + RP$ has a unique soln $R = QP^*$

$= Q + (Q + RP)P$

$= Q + QP + RP^2$

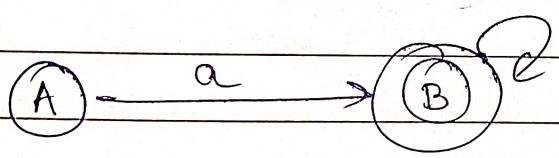
$= Q + QP + (Q + RP)P^2$

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

$= Q(P^0 + P^1 + P^2 + P^3 \ldots$

$\therefore \boxed{R = QP^*}$

Example



$A = \varepsilon$

$B = Aa + Ba + Bb$

$= \varepsilon a + B(a+b)$

$B = a + B(a+b)$

$\downarrow \quad \downarrow \quad \downarrow$

$R = Q + R \quad P$

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

Page No.
Date

* Pumping Lemma for Regular Languages.

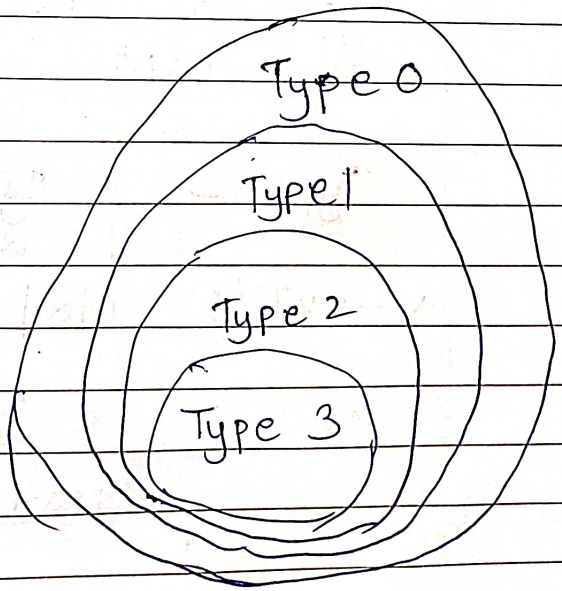
→ The Pumping Lemma states that 'For every regular language L, there exists a constant p (the pumping length) such that for any string s in L with length at least p, s can be divided to three parts xyz such that :-

① Length condition: $|xy| \leq p$

② Pumping condition: $|y| \geq 0$

③ Closure condition: For all $i \geq 0$, $xy^i z$ is in L.

* Chomsky Hierarchy



Type 0: Unrestricted Grammar.

Type 1: Context-Sensitive Grammar

Type 2: Context-Free Grammar

Type 3: Regular Grammar.

Page No.
Date

→ Type 0 Grammar includes all formal grammar. Type 0 grammar languages are recognised by Turing Machine

→ Also known as Recursively Enumerable Languages.

→ Type 1 grammar is recognised by Linear Bound Automata.

→ Type 2 is context free grammar recognised by PDA.

→ Type 3 is regular grammar recognised by FSM.

Type 3	Type 2 $\alpha \rightarrow \beta$	Type 1	Type 0
$V \rightarrow VT/T$	$\lvert\alpha\rvert = 1$	$\alpha \rightarrow \beta$	$\alpha \rightarrow \beta$
or	β has no restriction	$\lvert\alpha\rvert <= \lvert\beta\rvert$	$\alpha = (V+T)^* V (V+T)^*$
$V \rightarrow TV/T$		β cannot be ε	$\beta = (V+T)^*$
		$\beta \in (V+T)^+$	

Page No.
Date

* Context Free Grammar (CFG)

$$S \rightarrow (S) \mid \lambda$$

$$S \rightarrow (S)S$$

$$S \rightarrow ((S)S)S$$

A formal grammar used to describe the syntax of languages in a form that is convenient for passing

$$CFG = (V, T, S, P)$$

V → Set of Non-Terminal Symbols

T → Set of Terminal Symbols

S → Start Symbol

P → Production Rules.

CNF	GNF
Production Rule : $A \rightarrow BC$ $A \rightarrow a$	(1) Production Rule: $A \rightarrow a\alpha$ $\alpha \Rightarrow ABC \cdots$
More restrictive, allows only binary productions	(2) More general, allows productions with a terminal symbol followed by string of non-terminal
(3) Conversion is complex as it involves introducing new productions	(3) Simpler as it requires adjusting existing productions.
(4) Used in CYK (Cocke-Younges-Kasami) Algorithm.	(4) Used in NLP and analysis of programming languages

Page No.
Date

* Applications of CNF and GNF

① Parsing Algorithms : CNF is used in parsing algorithms like CYK algorithm

② NLP : CNF is applied in the analysis of natural language sentences and syntax.

③ Compiler Design: GNF is used in code generation in compiler design due to its concise nature.

④ Compact Representation: GNF provides a more compact representation of certain languages compared to CNF

——— X ——— X ———

* Pumping Lemma for CFL.

For every CFL L, there exists a constant p (the pumping length) such that for any string s in L with length at least p,

s can be divided into five parts, s = uvwxy

* Pushdown Automata.

→ PDA is a finite automata with extra memory called stack, which helps the PDA recognise CFL.

- Q = set of states
- Σ = set of input symbols
- Γ = set of pushdown symbols.
- q_0 = Initial state.
- Z = Initial stack symbol.
- F = set of final states.
- δ = transition function

$$Q \times \{\Sigma \cup \varepsilon\} \times \Gamma \longrightarrow Q \times \Gamma^*$$

Date

① Length Condition : $|vwx| \leq p$

② Pumping condition : $|vx| \geq 1$

③ Closure Condition : for all $i \geq 0$,

$uv^i wx^i y$ is in L.

---

* Idea of Universal Turing Machine

→ Idea of universal Turing Maching is a fundamental concept in the field of science.

→ It represents a theoritical model of computation that can simulate the behaviour of any other Turing Machine/computational device, given the appropriate instructions.

→ Takes the description of the type of Turing Machine it wants to simulate. Then it can simulate that Turing Machine

→ UTM is an extended idea of Turing Machine introduced by Alan Turing.

→ In a theoritical sense, a single machine can perform computation for any problem which can be solved algorithmically.

→ Thus UTM serves as basis of digital computers and lets us know about the limits of computation and algorithms.

× ×

* Church's Thesis-

→ "Every effectively calculable function is a computable function".

→ This thesis asserts that any function that is intuitively computable (can be solved algorithmically), can also be solved by a turing machine.

→ This thesis is a hypothesis about the nature of computability and the limits of what can be algorithmically solved. given by Alonzo Church in 1930s.

Page No.
Date

* Halting Problem. (Alan Turing - 1936)

→ The halting problem is a classic computability theory and the theory of algorithms

→ The problem states that we have to check whether a program will halt or will run indefinitely.

→ Alan Turing proved that it is impossible to create a general algorithm that can solve the halting problem for all possible inputs.

→ The proof is based on diagonalisation, where we assume the existence of a universal halting problem solver, but then construct a program which contradicts the existence.

---

* Hilbert's Problems.

→ Hilbert's problems are a set of 23 mathematical problems posed by German mathematician David Hilbert in 1900.

→ These problems spanned various areas of mathematics and were influential in shaping the course of mathematics today.

→ Hilbert's problems did not directly affect the computation field.

① The Continuum Hypothesis (1st)
② Riemann's Hypothesis (8th)
③ Axiomization Of Physics (6th)
④ Quadratic Reciprocity (9th)

These are some examples of Hilbert's problems

---

* Classes P and NP

P (Polynomial Time)

→ P is a class of decision problems that can be solved using a deterministic Turing Machine in polynomial time.

Date

→ P problems are considered to be easy and efficient in terms of computation

② NP (Non-deterministic Polynomial Time)

→ NP is a class of decision problems for which a solution, once proposed, can be checked quickly (in polynomial time) by a deterministic Turing Machine.

→ Problems in NP are considered 'verifyable' in polynomial time, while finding a solution may be difficult, verifying a proposed solution is easier.

NP-Complete: A problem is NP-Complete, if it is both in NP and at least as hard as the hardest problem in NP

If a solution to an NP-complete problem exists, then we can solve all NP problems

NP-Hard:- A problem is NP-hard if solving it efficiently would allows to solve all problems in NP, may not be NP itself.

Page No.
Date

* Post's Correspondence Problem

→ The Post's correspondence Problem is a classic problem in the theory of computation and formal languages.

→ Problem Statement

→ Given a set of dominoes, each with two strings written on it, (one on top and one on the bottom), can we arrange these dominoes in such a way that the top strings and bottom strings match.

→ There is no general algorithm that can determine whether solution exists for an arbitary value.

→ So the Post correspondence problem is undecidable.

X X

Date

Variations of Turing Machine

① Multi-Tape TM: Has multiple tapes that can move independently.

② Multi-Head TM: Has multiple heads on a single tape used to change (R/W) multiple values at once.

③ Non-Deterministic TM: Allows for multiple possible transitions for a given set of instructions

④ Multi-Dimensional TM: Has 4 possible movement instead of 2, (Up, Down, Left, Right), so it can move accordingly in various direction

⑤ Two-Way Infinite Tape: Infinite tape on both ways left and right

---

* Undecidability.

→ Undecidability refers to the property of certain problems for which there is no algorithmic procedure to determine a yes/no answer.

Page No.
Date

→ Recursive Languages: Languages which can have a Turing machine and accepts all strings in L and rejects all strings not in L.

→ Turing machine gives a decision and halts without going into loop forever.

→ Recursively Enumerable language: Languages which can have a Turing Machine and accepts all strings in L and halts.

Machine ~~doesn't halt~~ may or may not halt for rejection of strings.

→ For an undecidable language, turing Machine cannot be made.

* Terminologies and components of Turing Machine

Q = set of states
X = Tape symbol
Σ = input alphabet set
δ = Transition function $\delta: Q * X \rightarrow Q * X * \{L, R\}$
q_0 = initial state
B = Blank symbol
F = set of final states.

Components

(1) Tape: Infinite 1-D tape divided into cells. Each cell on the tape can contain a finite symbol including blank.

(2) R/W Head: The read/write head is a mechanism that moves left and right along the tape.

(3) Finite control: Brain of the Turing Machine controls the state management of the Turing machine.

(4) State Register: Contains the current state of the TM.