Times asked 7 times
6 times
5 times
4 times
3 times
2 times
1 time

TCS Theory Question bank

1.Basic concepts and Finite Automata

1. Explain applications of Finite Automata(FA).
2. Discuss difference in transition function of FA, PDA and TM.
3. Differentiate between FA, PDA and TM.
4. Compare and contrast Moore and Mealy machines.
5. Differentiate between DFA and NFA.
6. Short note on Moore and Mealy machine.
7. Explain Finite State Machine(FSM).

2. Regular Expressions and Languages

1. Explain and give formal definition of pumping lemma for regular language.
2. Short note on Decision properties of Regular language.
3. Explain applications of Regular Expressions.
4. Define Regular language.
5. Short note on Arden's theorem.
6. What are the closure properties of RL.

3. Grammars

1. Write a short note on Chomsky Hierarchy with an example.
2. Steps for converting CFG to CNF.

4. Pushdown Automata (PDA)

1. Explain the ways of acceptance by a PDA.
2. Differentiate between PDA and NPDA.
3. Explain applications for PDA.
4. Write a short note on: Definition and working of PDA.
5. Explain non-deterministic PDA.

5. Turing machine (TM)

1. Write a short note on: Variants of Turing machines.
2. Explain applications for TM.
3. Write a short note on: Universal Turing machine.

6. Undecidability

1. Write a short note on: Post Correspondence problem.
2. Write a short note on: TM Halting problem.
3. Write a short note on: Recursive and Recursively enumerable languages.
4. Write a short note on: Rice's theorem.

TCS Theory Answer bank

1.Basic concepts and Finite Automata

1. Explain applications of Finite Automata(FA).

Finite automata are used for solving several common types of computer algorithms. Some of them are :

(i) Design of digital circuit

(ii) String matching

(iii) Communication protocols for information exchange.

(iv) Lexical analysis phase of a compiler.

Finite automata can work as an algorithm for regular language. It can be used for checking whether a string w∈L, where L is a regular language.

v. Text Search and Pattern Matching
vi. Spell Checkers
vii. Regular Expression Matching
viii. Natural Language Processing (NLP)

2. Discuss difference in transition function of FA, PDA and TM.

Differences in Transition Functions

Aspect	Finite Automaton (FA)	Pushdown Automaton (PDA)	Turing Machine (TM)
Form of Transition Function	$\delta : Q \times \Sigma \to Q$	$\delta : Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$	$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$
Inputs	Current state and input symbol	Current state, input symbol, and top of stack symbol	Current state and tape symbol
Outputs	Next state	Next state and stack operation (push, pop, or no change)	Next state, symbol to write on tape, and head movement (left, right, or stay)
Memory Used	No memory (only states)	Stack memory (last-in-first-out, or LIFO)	Infinite tape (acts as read/write memory)
Operations	State transition based on input symbol	State transition, stack operations based on input symbol and stack top	State transition, tape read/write, and head movement
Example Transition	$\delta(q_0, a) = q_1$	$\delta(q_0, a, X) = (q_1, \gamma)$, where X is popped and γ is pushed	$\delta(q_0, a) = (q_1, b, R)$, where a is overwritten by b and head moves right

3. Differentiate between FA, PDA and TM.

Feature	Finite Automaton (FA)	Pushdown Automaton (PDA)	Turing Machine (TM)
Memory	No additional memory	Has a stack (Last-In-First-Out structure)	Has an infinite tape (both read and write)
Computational Power	Recognizes **regular languages**	Recognizes **context-free languages**	Recognizes **recursively enumerable languages**
Input Alphabet	Finite alphabet, denoted as Σ	Finite alphabet, denoted as Σ	Finite alphabet, denoted as Σ
Transition Function	Depends on the current state and input symbol	Depends on the current state, input symbol, and stack symbol	Depends on the current state and tape symbol
Output	Accepts or rejects based on final state	Accepts or rejects based on final state and stack content	Accepts, rejects, or loops based on tape and state
Tape/Stack Operations	No tape or stack operations	Can push to or pop from the stack	Can read from, write to, and move along the tape
Head Movement	Not applicable	Not applicable	Can move left, right, or stay on the tape
Deterministic vs. Non-Deterministic	Has both Deterministic FA (DFA) and Non-Deterministic FA (NFA)	Has both Deterministic PDA (DPDA) and Non-Deterministic PDA (NPDA)	Turing Machines are usually non-deterministic
Acceptance Criteria	Accepted if reaches an accepting state	Accepted if reaches an accepting state with an empty stack	Accepted if reaches an accepting state
Examples of Use Cases	Used for lexical analysis, pattern matching	Used for parsing expressions in compilers	Used for general computation, simulating algorithms
Limitations	Cannot handle nested structures	Cannot recognize languages requiring multiple stacks	Can perform any computation that a real computer can

4. Compare and contrast Moore and Mealy machines.

Parameter	Mealy Machine	Moore Machine
Output Association	Output is associated with **transitions**	Output is associated with **states**
Equivalence	There is an equivalent Moore machine for every Mealy machine	There is an equivalent Mealy machine for every Moore machine
Number of States	Requires **fewer states** as output is transition-based	Often requires **more states** as each state has a fixed output
Transition Function	Transition function: $\delta : \Sigma \times Q \to Q$ Output function: $O : \Sigma \times Q \to O$	Transition function: $\delta : \Sigma \times Q \to Q$ Output function: $O : Q \to O$
Ease of Implementation	Seldom implemented directly in circuits or programs	Easier to implement in circuits or programs
Output Timing	Output can change **immediately with input changes**, even within the same state	Output changes **only on state transitions**
Design Complexity	Generally **more complex** due to output variations on transitions	**Simpler design** as outputs depend only on states
Flexibility	More flexible in generating different outputs for the same state with different inputs	Less flexible, as outputs are determined solely by states

5. Differentiate between DFA and NFA.

Parameter	NFA	DFA
Transition	Non-deterministic.	Deterministic
No. of states.	NFA has fewer number of states.	More, if NFA contains Q states then the corresponding DFA will have $\leq 2^Q$ states.
Power	NFA is as powerful as a DFA	DFA is as powerful as an NFA
Design	Easy to design due to non-determinism.	Relatively, more difficult to design as transitions are deterministic.
Acceptance	It is difficult to find whether w ∈ L as there are several paths. Backtracking is required to explore several parallel paths.	It is easy to find whether w ∈ L as transitions are deterministic.

	DFA	NFA
1	DFA stands for Deterministic Finite Automata.	NFA stands for Nondeterministic Finite Automata.
2	DFA cannot use Empty String transition.	NFA can use Empty String transition.
3	DFA is more difficult to construct.	NFA is easier to construct.
4	Time needed for executing an input string is less.	Time needed for executing an input string is more.
5	All DFA are NFA.	Not all NFA are DFA.
6	DFA requires more space.	NFA requires less space then DFA.
7	Dead state may be required.	Dead state is not required.
8	Backtracking is allowed in DFA.	Backtracking is not always possible in NFA.
9	Conversion of Regular expression to DFA is difficult.	Conversion of Regular expression to NFA is simpler compared to DFA.

6. Short note on Moore and Mealy machine.

Moore Machine:

- In a Moore machine, the output depends only on the **current state** of the machine.
- Each state in a Moore machine has a fixed output, so the output only changes when the machine transitions to a different state.
- **Example**: Traffic light controller, where each light state (green, yellow, red) has a fixed output.

Mealy Machine:

- In a Mealy machine, the output depends on the **current state** and the **current input**.
- This allows Mealy machines to potentially produce different outputs even while in the same state, based on varying inputs.
- **Example**: Serial data transmission, where the output changes immediately in response to different inputs.

Steps for Converting a Moore Machine to a Mealy Machine

1. **Identify States and Outputs**:
    - List the states and the fixed outputs for each state in the Moore machine.
2. **Associate Outputs with Transitions**:
    - For each state and its outgoing transitions, create corresponding Mealy transitions.
    - The output for each Mealy transition will be the same as the output of the Moore state it's coming from.
3. **Adjust Output for Each Input Condition**:
    - If a state in the Moore machine has different outputs based on input conditions, modify each Mealy transition to have the appropriate output for each input.

4. **Remove Extra States** (if applicable):
    - Since Mealy machines can have fewer states due to their output flexibility, combine states if possible to reduce the machine's size.
5. **Verify**:
    - Ensure that the Mealy machine behaves identically to the Moore machine.

7. Explain Finite State Machine(FSM).

A **Finite State Machine (FSM)** is a computational model used to design and describe the behaviour of systems that have a finite number of states. FSMs are widely used in computer science, digital circuit design, and various fields where systems have predictable, stepwise behaviours.

Types of Finite State Machines

1. **Deterministic Finite Automaton (DFA)**:
    - In a DFA, each state has a unique transition for each possible input.
    - This means that from any given state, an input can lead to only one specific next state.
2. **Non-Deterministic Finite Automaton (NFA)**:
    - In an NFA, a state can have multiple transitions for the same input, leading to different possible next states.
    - NFAs are more flexible but can be more complex to analyze.
3. **Moore Machine**:
    - A type of FSM where the output depends only on the current state, not the input.
    - The output changes only when the state changes.
4. **Mealy Machine**:
    - A type of FSM where the output depends on both the current state and the current input.
    - This allows the output to respond more immediately to input changes.

2. Regular Expressions and Languages

1. Explain and give formal definition of pumping lemma for regular language.

The **Pumping Lemma** is a fundamental property of regular languages, used to prove that certain languages are *not* regular. If a language is regular, it must satisfy the pumping lemma. If we can show that a language does not satisfy the pumping lemma, we can conclude that it is not regular.

Let L be a regular language and $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automata with n-states. Language L is accepted by m. Let $\omega \in L$ and $|\omega| \geq n$, then ω can be written as xyz, where

(i) $|y| > 0$

(ii) $|xy| \leq n$

(iii) $xy^i z \in L$ for all $i \geq 0$ here y^i denotes that y is repeated or pumped i times.

2. Short note on Decision properties of Regular language.

Decision properties of regular languages are properties that allow us to determine certain characteristics or make decisions about regular languages using algorithmic methods. These properties make regular languages particularly useful, as they enable us to analyse and manipulate languages in efficient ways.

A problem that has a decision algorithm is called decidable. A decision algorithm terminates with yes or no answer. Some of the decision problems are given below :

1. Is a regular set empty ? – Emptiness property.
2. Whether a finite automata accepts a finite number of strings ? – Finiteness property.
3. Whether a finite automata accepts an infinite number of strings ? – Infiniteness property.

Decision Algorithm for emptiness :

- Finite automata will fail to accept any string if it does not have a final state.
- Finite automata will fail to accept a string if none of its accepting states is reachable from the initial state.

Decision algorithm for finiteness / infiniteness :

The set of strings accepted by a finite automata M with n states is finite if and only if the finite automata accepts only strings of length less than n.

The set of strings accepted by a finite automata M with n states is infinite if and only if it accepts some string ω such that $n \leq |\omega| < 2n$.

3. Explain applications of Regular Expressions.

1. **Lexical Analysis in Compiler Design**
   - Lexical analysis is the first phase of a compiler, where source code is broken down into tokens, Regular expressions play a critical role in defining the patterns for tokens.

2. **grep in UNIX**
   - The grep command in UNIX is a powerful tool that uses regular expressions to search text in files or streams. Here are some common applications of grep:
3. **Input Validation**:

- Regular expressions are commonly used to validate user input in forms and applications to ensure it meets certain criteria.

4. **Data Extraction**:

- Regex can extract specific information from text by identifying patterns.

5. **Search and Replace Operations**:

- Regular expressions make search-and-replace operations more flexible, allowing bulk modifications based on patterns rather than exact matches.

4. Define Regular language.

A **Regular Language** is a type of formal language that can be recognized by a **finite automaton** and can be defined using **regular expressions**. In simpler terms, it is a language whose strings can be generated by applying a specific set of rules, such as concatenation, union, and repetition (also known as the Kleene star).

Examples of Regular Languages

- The set of all strings over the alphabet {0,1} that contain an even number of 0s.
- The set of all strings over {a,b} that start with a and end with b.

5. Short note on Arden's theorem.

Arden's Theorem is a fundamental theorem in the theory of regular languages and automata, used to solve regular expressions for certain types of equations. It provides a method to express the language accepted by a **finite automaton** in terms of regular expressions, which is useful in designing and analyzing automata.

Statement of Arden's Theorem

For any two regular expressions P and Q over an alphabet Σ if R is a solution to the equation:

$R=Q+RP$

then the solution can be expressed as:

$R=QP*$

where:

- + represents the union of languages,
- · represents concatenation,
- $P*$ is the Kleene star of P, meaning zero or more occurrences of P.

6. What are the closure properties of RL.

If an operation on regular languages generates a regular language then we say that the class of regular languages is closed under the above operation. Some of the important closure properties for regular languages are given below.

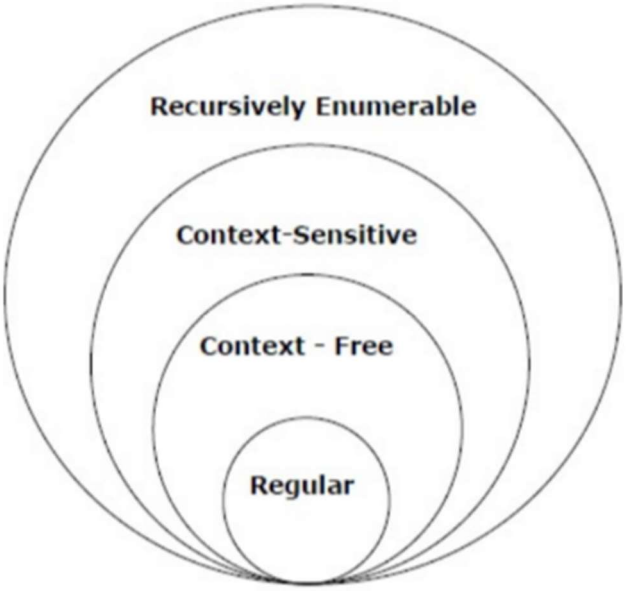
1. Union
2. Difference
3. Concatenation
4. Intersection
5. Complementation
6. Kleene star
7. Transpose or reversal.

3. Grammars

1. Write a short note on Chomsky Hierarchy with an example.

According to Noam Chomosky, there are four types of grammars − Type 0, Type 1, Type 2, and Type 3. The following table shows how they differ from each other −

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton



Type - 3 Grammar

Type-3 grammars generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form **X → a or X → aY**

where **X, Y ∈ N** (Non terminal)

and **a ∈ T** (Terminal)

The rule **S → ε** is allowed if **S** does not appear on the right side of any rule.

Example

```
X → ε
X → a | aY
```

Type - 2 Grammar

Type-2 grammars generate context-free languages.

The productions must be in the form **A → γ**

where **A ∈ N** (Non terminal)

and **γ ∈ (T ∪ N)*** (String of terminals and non-terminals).

These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

Example

```
S → X a
X → a
X → aX
X → abc
X → ε
```

Type - 1 Grammar

Type-1 grammars generate context-sensitive languages. The productions must be in the form

α A β → α γ β

where **A ∈ N** (Non-terminal)

and **α, β, γ ∈ (T ∪ N)*** (Strings of terminals and non-terminals)

The strings **α** and **β** may be empty, but **γ** must be non-empty.

The rule **S → ε** is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

Example

```
AB → AbBc
A → bcA
B → b
```

Type - 0 Grammar

Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.

They generate the languages that are recognized by a Turing machine.

The productions can be in the form of **α → β** where **α** is a string of terminals and nonterminals with at least one non-terminal and **α** cannot be null. **β** is a string of terminals and non-terminals.

Example

```
S → ACaB
Bc → acB
CB → DB
aD → Db
```

2. Steps for converting CFG to CNF.

Steps

1. **Remove Null (Epsilon) Productions**:
   - Eliminate productions of the form $A \to \epsilon$ (where ϵ is the empty string).
   - For each rule that contains A on the right side, create new rules without A by replacing A with ϵ, ensuring all combinations are covered.
2. **Remove Unit Productions**:
   - Remove productions of the form $A \to B$ (where both A and B are non-terminals).
   - Replace these rules by adding new productions from B to A, copying all rules of B to A.
3. **Remove Useless Symbols**:
   - Remove any non-terminals that are unreachable from the start symbol or do not lead to a terminal.
   - This ensures only necessary symbols remain in the grammar.
4. **Convert to Binary Rules**:
   - For productions with more than two non-terminals on the right (e.g., $A \to BCD$), break them down into binary productions by introducing new non-terminals.
   - For instance, replace $A \to BCD$ with $A \to BX$ and $X \to CD$.
5. **Convert Terminals in Mixed Rules**:
   - If a production has a mix of terminals and non-terminals (e.g., $A \to aB$), replace the terminal with a new non-terminal that produces just that terminal.
   - For example, if a is a terminal, introduce $A' \to a$ and replace a in the original rule with A'.

4. Pushdown Automata (PDA)

1. Explain the ways of acceptance by a PDA.

A language L can be accepted by a PDA in two ways :

1. Through final state.
2. Through empty stack.

It is possible to convert between the two classes.

1. From final state to empty stack.
2. From empty stack to final state.

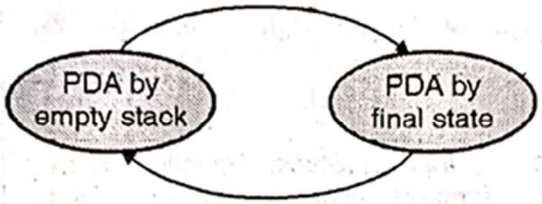


Fig. 6.4.1 : Equivalence of two PDAs

6.4.1 Acceptance by Final State

Let the PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ then the language accepted by M through a final state is given by :

$$L(M) = \left\{ w \mid (q_0, w, z_0) \overset{*}{\underset{M}{\Rightarrow}} (q_1, \varepsilon, \alpha) \right\}$$

Where the state $q_1 \in F$. α, the final contents of the stack are irrelevant as a string is accepted through a final state.

6.4.2 Acceptance by Empty Stack

Let the PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi)$ then the language accepted through an empty stack is given by :

$$L(M) = \left\{ w \mid (q_0, w, z_0) \overset{*}{\underset{M}{\Rightarrow}} (q_1, \varepsilon, \varepsilon) \right\}$$

Where q_1 is any state belonging to Q and the stack becomes empty on application of input string w.

2. Differentiate between PDA and NPDA.

Feature	PDA (Deterministic)	NPDA (Nondeterministic)
Definition	Has a unique transition per input.	Can have multiple transitions per input.
Transitions	Deterministic transitions.	Nondeterministic transitions.
Acceptance	Accepts by final state only.	Accepts by final state or empty stack.
Power	Less powerful; recognizes a subset of context-free languages.	More powerful; recognizes all context-free languages.
Example Languages	Recognizes $a^n b^n$.	Recognizes $a^n b^n c^n$.
Implementation	Simpler and easier to design.	More complex due to nondeterminism.
Closure Properties	Closed under union, but not intersection or complement.	Closed under union, but not intersection or complement.

3. Explain applications for PDA.

1) **Syntax Parsing in Compilers:** PDAs are used to parse context-free grammars, which helps in analyzing the syntax of programming languages to check if code is written correctly.
2) **Language Recognition:** PDAs can recognize context-free languages, like checking if parentheses are balanced in an expression (e.g., in mathematical equations or code blocks).
3) **Natural Language Processing (NLP):** PDAs help model simple structures in human languages, such as basic sentence structures and nested phrases, which aids in understanding and processing human languages.
4) **XML Parsing**: XML documents have a nested structure that can be validated using PDAs, ensuring the correct opening and closing of tags.
5) **Arithmetic Expression Evaluation:** PDAs can be used to evaluate arithmetic expressions by parsing and evaluating expressions with nested structures, such as ((2+3)*4).
6) **Design of Interactive Systems:** PDAs model systems with nested or recursive states, like navigation systems with menus and submenus.

4. Write a short note on: Definition and working of PDA.

A pushdown automata M is defined as 7-tuple :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where,

Q = The set of states

Σ = Input alphabet

Γ = Stack symbols

δ = The transition function is a transition form $Q \times (\Sigma \cup \varepsilon) \times \Gamma$ to $Q \times \Gamma^*$

q_0 = $q_0 \in Q$ is the initial state

F = $F \subseteq Q$ is the set of final states

z_0 = An initial stack symbol

5. **Explain non-deterministic PDA.**

A **Non-deterministic Pushdown Automaton (NPDA)** is a type of automaton that processes input strings using both a stack and non-deterministic transitions. Non-determinism allows the NPDA to make multiple possible moves at each step based on the current input symbol, stack top, and current state. This capability makes NPDAs more powerful than deterministic PDAs (DPDAs) because they can recognize a broader class of languages, specifically all **context-free languages (CFLs)**.

A pushdown automata M is defined as 7-tuple :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where,

Q = The set of states

Σ = Input alphabet

Γ = Stack symbols

δ = The transition function is a transition form $Q \times (\Sigma \cup \varepsilon) \times \Gamma$ to $Q \times \Gamma^*$

q_0 = $q_0 \in Q$ is the initial state

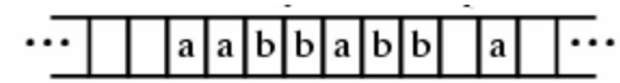
F = $F \subseteq Q$ is the set of final states

z_0 = An initial stack symbol

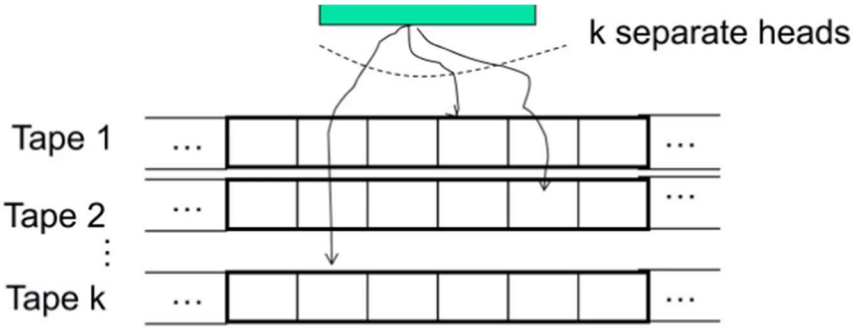
6. Turing machine (TM)

1. Write a short note on: Variants of Turing machines.

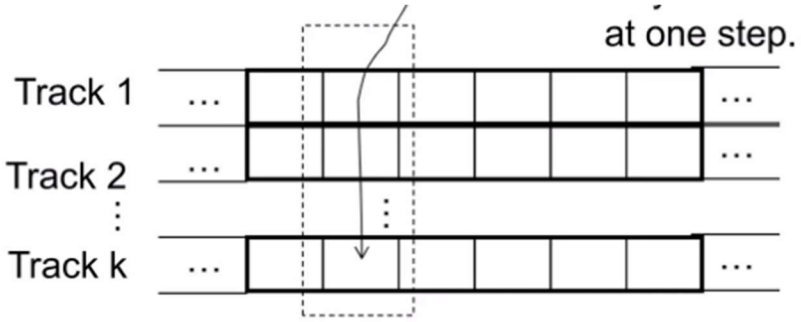
Two-Way Infinite Turing Machine: A Turing Machine with an infinite tape extending in both directions, allowing the head to move left or right indefinitely.



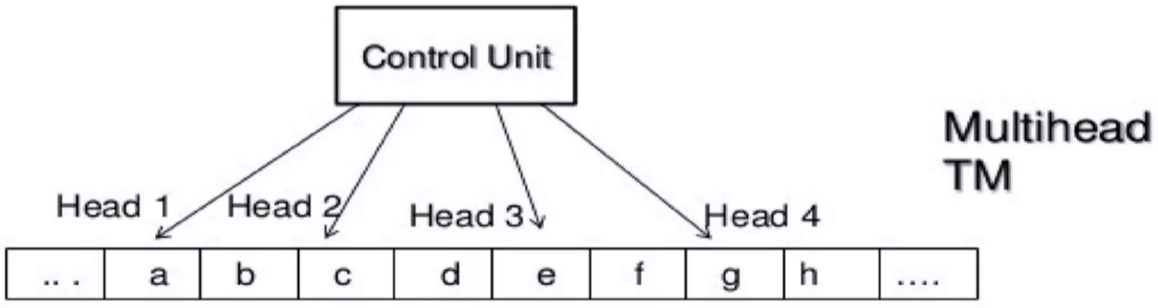
Multitape Turing Machine: Has multiple tapes, each with its own independent tape head, enabling more complex operations by accessing different tapes simultaneously.



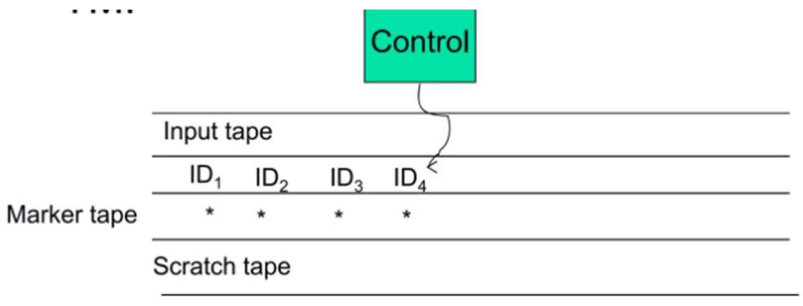
Multitrack Turing Machine: Uses a single tape divided into multiple tracks, allowing it to read or write multiple symbols at the same position on each track.



Multihead Turing Machine: Has multiple heads on a single tape, each head can read or write independently, allowing it to access different parts of the tape simultaneously.



Nondeterministic Turing Machine: Can make multiple possible moves from a given configuration, branching into multiple computation paths to explore all possibilities simultaneously.



2. Explain applications for TM.

1) **Understanding Computability:** TMs help define what problems can be solved by any machine, setting the basis for identifying computable vs. non-computable problems.

2) **Algorithm Design:** TMs offer a basic model for designing and analyzing algorithms, helping to break down complex computations into simple steps.

3) **Complexity Theory:** TMs are used to classify problems based on their time and space requirements, helping identify easy vs. hard problems in terms of resources.

4) **Universal Computation Model:** The Universal Turing Machine concept shows that one machine can simulate any other, which is foundational for general-purpose computing.

5) **Formal Language Processing:** TMs serve as a model for processing languages, aiding in understanding and developing parsers and compilers for programming languages.

3. Write a short note on: Universal Turing machine.

- Universal Turning Machine stimulates a Turning Machine.
- Universal Turing Machine can be considered as a subset of all the Turing machines, it can match or surpass other Turing machines including itself.
- Programmable Turing Machine is called Universal Turing Machine
- Universal Turing Machine is like a single Turing Machine that has a solution to all problems that is computable.
- It minimizes space complexity
- It contains a Turning Machine description as input along with an input string, runs the Turning Machine on the input and returns a result.
- The transition function is $Q \times T \rightarrow Q \times T \times \{L, R\}$, where Q is a finite set of states, T is the tape of the alphabet

6. Undecidability

1. Write a short note on: Post Correspondence problem.

Definition of post correspondence problem (PCP) : Let A and B be two non-empty lists of strings over Σ. A and B are given as below :

$$A = \{x_1, x_2, x_3 \ldots x_k\}$$

$$B = \{y_1, y_2, y_3 \ldots y_k\}$$

We say, there is a post correspondence between A and B if there is a sequence of one or more integers i, j, k …m such that :

The string $x_i\, x_j \ldots x_m$ is equal to $y_i\, y_j \ldots y_m$.

Example 1

Find whether the lists

M = (abb, aa, aaa) and N = (bba, aaa, aa)

have a Post Correspondence Solution.

Solution

	x_1	x_2	x_3
M	abb	aa	aaa
N	bba	aaa	aa

Here,

$x_2x_1x_3$ = 'aaabbaaa'

and $y_2y_1y_3$ = 'aaabbaaa'

We can see that

$x_2x_1x_3 = y_2y_1y_3$

Hence, the solution is i = 2, j = 1, and k = 3.

2. Write a short note on: TM Halting problem.

The halting problem of a Turing machine states :

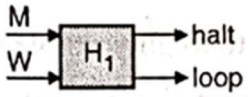
Given a Turing machine M and an input ω to the machine M, determine if the machine M will eventually halt when it is given input ω.

Halting problem of a Turing machine is unsolvable.

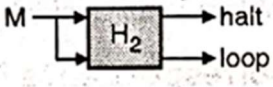
Step 1 : Let us assume that the halting problem of a Turing machine is solvable. There exists a machine H_1(say). H_1 takes two inputs :

1. A string describing M.
2. An input ω for machine M.

H_1 generates an output **"halt"** if H_1 determines that M stops on input ω; otherwise H outputs "loop". Working of the machine H_1 is shown below.

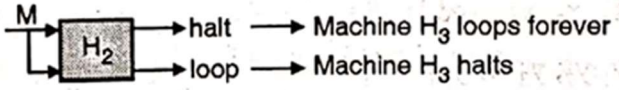


Step 2 : Let us revise the machine H_1 as H_2 to take M as both inputs and H_2 should be able to determine if M will halt on M as its input. Please note that a machine can be described as a string over 0 and 1.



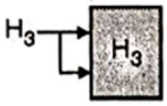
Step 3 : Let us construct a new Turing machine H_3 that takes output of H_2 as input and does the following :

1. If the output of H2 is "loop" than H3 halts.
2. If the output of H_2 is "halt" than H_3 will loop forever.



H_3 will do the opposite of the output of H_2.

Step 4 : Let us give H_3 itself as inputs to H_3.



If H_3 halts on H_3 as input then H_3 would loop (that is how we constructed it).
If H_3 loops forever on H_3 as input H_3 halts (that is how we constructed it).
In either case, the result is wrong.
Hence,
H_3 does not exist.
If H_3 does not exist than H_2 does not exist.

3. Write a short note on: Rice's theorem.

Rice's Theorem is an important result in the theory of computation, which states that **any non-trivial property of the language recognized by a Turing Machine is undecidable**.

Key Points of Rice's Theorem

1. **Non-trivial Property**: A property is considered non-trivial if it is true for some Turing Machines and false for others. In other words, the property does not apply universally to all TMs or none at all.
2. **Undecidability**: Rice's Theorem shows that it is impossible to design an algorithm (or Turing Machine) that can determine any non-trivial property of the language a Turing Machine recognizes. This means that for properties like "the language recognized by a TM is empty," "the language is finite," or "the language includes a specific string," no algorithm can decide them for all possible Turing Machines.
3. **Scope of the Theorem**: Rice's Theorem applies to properties of the **language recognized** by a Turing Machine, not properties of the machine itself (like the number of states or transitions).

Examples of Properties Covered by Rice's Theorem

- Whether a Turing Machine accepts all strings.
- Whether a Turing Machine's language is regular or context-free.

4. Write a short note on: Recursive and Recursively enumerable languages.

Recursive Languages

- **Definition**: A language is **recursive** (or **decidable**) if there exists a Turing Machine that can always determine, in a finite amount of time, whether any given string belongs to the language (accepts) or does not belong (rejects).
- **Properties**:
    - For a recursive language, the Turing Machine will always halt with a definitive answer (accept or reject) for any input.
    - Recursive languages are the class of languages that can be fully "decided" by a Turing Machine.
- **Example**: The set of all even numbers in binary form is recursive since there is a clear algorithm (ending in zero) to determine if a binary number is even.

Recursively Enumerable (RE) Languages

- **Definition**: A language is **recursively enumerable (RE)** if there exists a Turing Machine that will accept any string in the language but may either reject or run indefinitely for strings not in the language.
- **Properties**:
    - For an RE language, the Turing Machine is guaranteed to halt if the string is in the language (accept), but it may never halt if the string is not in the language.
    - RE languages are also known as **semi-decidable** because the Turing Machine may not provide a definitive answer (halt) for strings not in the language.

- **Example**: The **halting problem** (determining if a given Turing Machine halts on a given input) is RE because, if the machine halts, there's a way to verify it, but if it doesn't halt, there's no guaranteed way to detect non-halting.

6 Write Short notes **(Any Four)**

a Chomsky Hierarchy

b Post Correspondence Problem.

c Arden's Theorem

d TM-Halting Problem.

e Variants of Turning Machines

6 Write Short notes **(Any Two)**

a Explain with example Chomsky Hierarchy.

b Post Correspondence Problem.

c Recursive and Recursive enumerable languages.

d TM-Halting Problem.

Q6. Write short note (Solve Any 4)

a) Decision Properties of Regular Languages
b) Post Correspondence Problem
c) Variants of Turing Machine
d) Acceptance by a PDA
e) Conversion of Moore to Mealy Machines

****

6. Write short note on following (Any two) :

a) Chomsky Hierarchy.
b) Decision properties of regular languages.
c) Rice's theorem.
d) Definition and working of PDA.

6. Write detailed note on **(any two):-**

(a) Post Correspondence Problem
(b) Halting Problem.
(c) Rice's Theorem.

******

6. Write short note on following **(any 2)**

(a) Chomsky Hierarchy
(b) Halting Problem
(c) Rice's Theorem
(e) Universal Turing Machine

_____x____

6. Write short note on following (any 4)

(a) Closure properties of Context Free Language
(b) Applications of Regular expression and Finite automata
(c) Rice's Theorem
(d) Moore and Mealy Machine
(e) Universal Turing Machine