Theory of Computation (TOC)

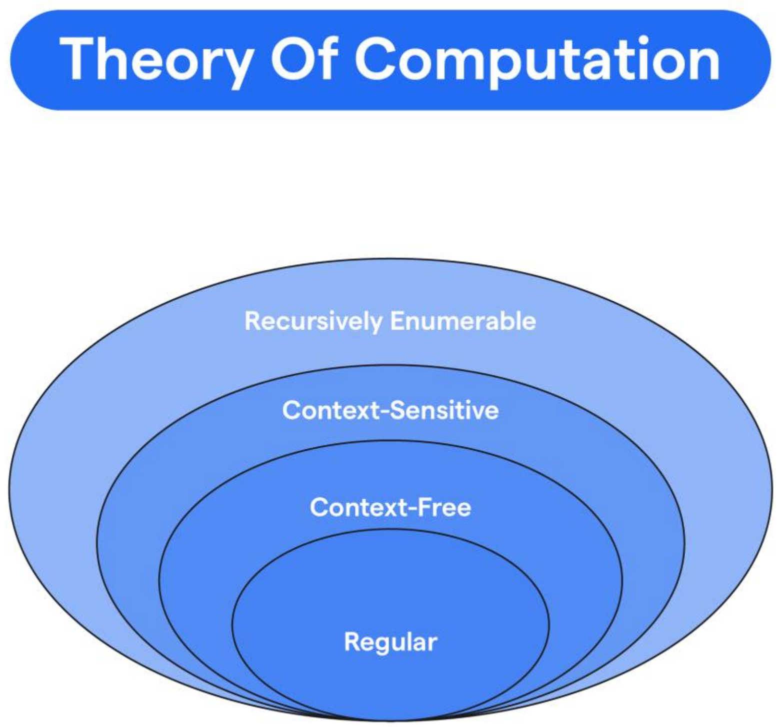
CSE 0613-2101

WEEK-01 Introduction

What is Theory of Computation?

The **Theory of Computation (TOC)** is a core theoretical discipline of computer science that studies the **nature of computation itself**.

Instead of focusing on how to program, TOC focuses on **what problems can be solved using algorithms and machines, and what problems fundamentally cannot be solved**.



Regular Languages

Languages that can be accepted by a **Finite Automaton (DFA/NFA)**.

Example Language:

$$L_1 = \{a^n b^m \mid n, m \geq 0\}$$

Example Strings:

- ε (empty string)
- aabbb
- Bbb

Why Regular?

There is **no dependency** between the number of as and bs. A finite automaton can recognize this pattern.

Context-Free Languages (CFL)

Languages generated by **Context-Free Grammar (CFG)** and accepted by **Pushdown Automata (PDA)**.

Example Language:

$$L_2 = \{a^n b^n \mid n \geq 0\}$$

Example Strings:

- ab
- aabb
- aaabbb

Why Context-Free?

The number of as must equal the number of bs. This requires **stack memory**, which finite automata do not have.

Context-Sensitive Languages (CSL)

Languages accepted by **Linear Bounded Automata (LBA)**.

Example Language:

$$L_3 = \{a^n b^n c^n \mid n \geq 1\}$$

Example Strings:

- abc
- aabbcc
- aaabbbccc

Why Context-Sensitive?

This language needs **three equal counts**.

- Not Context-Free
- Requires more memory than a PDA but limited by input size

TOC answers three fundamental questions

1. **What problems are computable?**

   → Can a problem be solved by *any* algorithm at all?

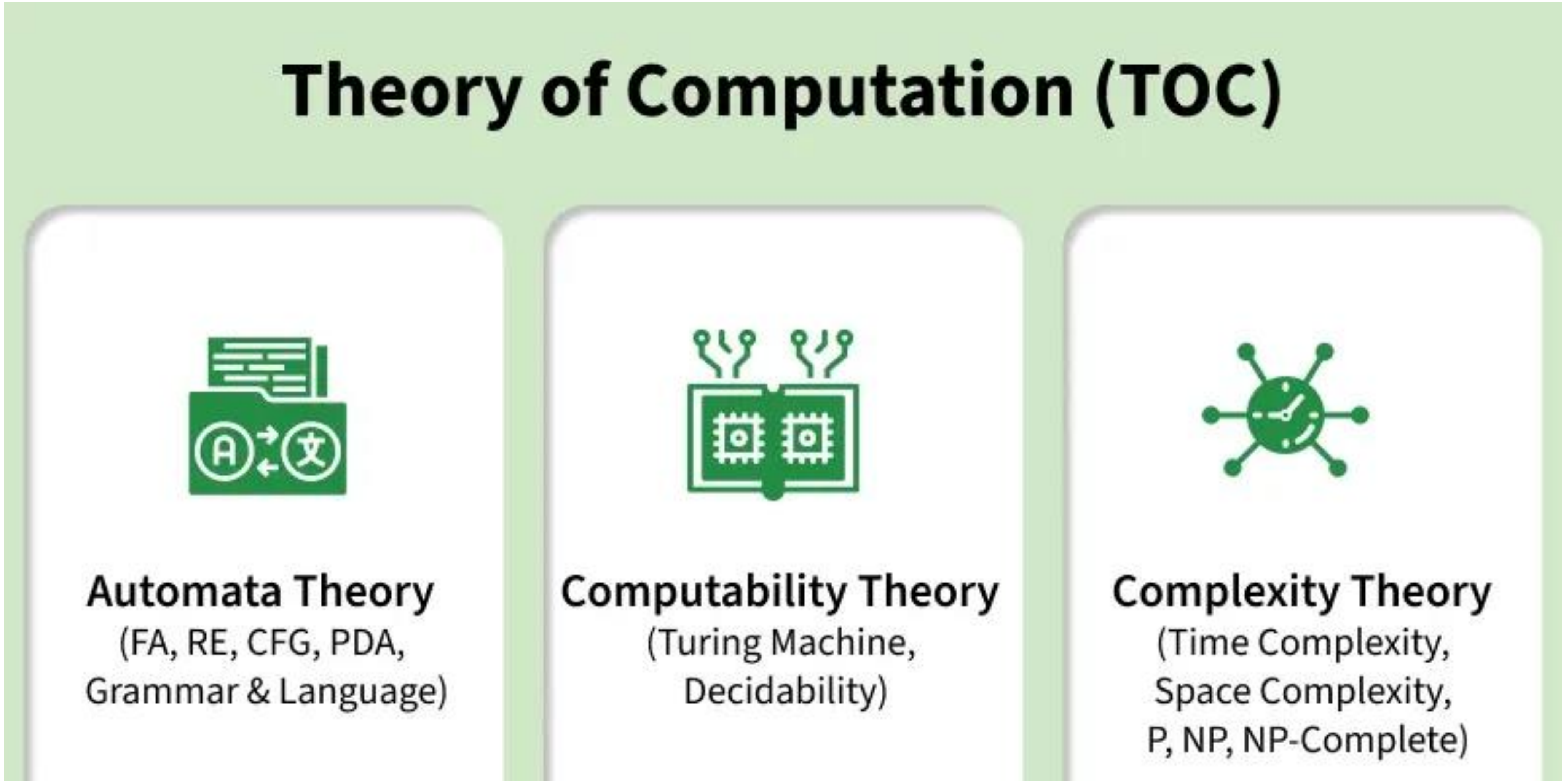
2. **How efficiently can problems be solved?**

   → What is the minimum time or memory required?

3. **What abstract machines define computation?**

   → What models best represent computation?

TOC has Three Main Parts:



Theory of Computation (TOC)

Automata Theory
(FA, RE, CFG, PDA, Grammar & Language)

Computability Theory
(Turing Machine, Decidability)

Complexity Theory
(Time Complexity, Space Complexity, P, NP, NP-Complete)

TOC has Three Main Parts.......

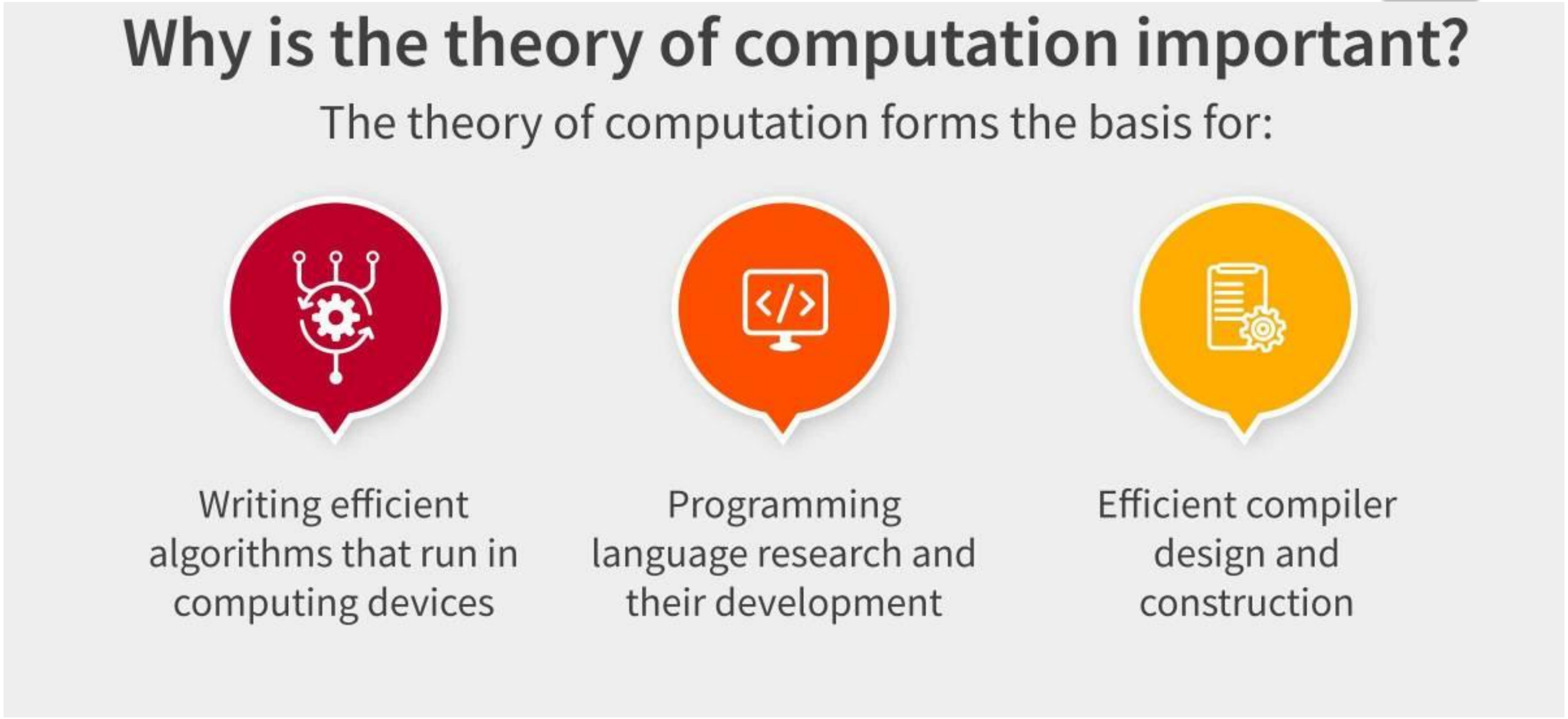
- **Automata Theory** – Studies abstract machines that work like simple computer models and helps in designing compilers, parsers, and other language processing tools.
- **Computability Theory** – Explores which problems can or cannot be solved by a computer, revealing the boundaries of algorithmic solutions and helping us identify unsolvable problems.
- **Complexity Theory** – Examines how much time, memory, or other resources are needed to solve problems and classifies problems based on their difficulty, efficiency, and resource requirements.

Why is the Theory of Computation Important?

- **Limitations and Potential of Computation:** It allows us to comprehend the limitations and potential of computation, which forms the bedrock of virtually every technology today.
- **Classification of Problems:** The Theory of Computation plays a vital role in classifying problems based on their inherent complexity and computational resources required.
- **Decision Making and Optimization:** It guides us in making decisions and optimizations, shedding light on algorithmic efficiencies.
- **Pioneering Advances in Computation:** By understanding the basics of computation, we can pioneer advances in computation, such as quantum computing.

Why TOC is Important……………..

- It provides the **theoretical foundation** of all computing systems
- It explains why **some problems have no algorithmic solution**
- It helps determine **efficient vs inefficient algorithms**
- It prevents wasting effort on **impossible problems**
- It supports many advanced CS areas



Why is the theory of computation important?

The theory of computation forms the basis for:

- Writing efficient algorithms that run in computing devices
- Programming language research and their development
- Efficient compiler design and construction

Fields Influenced by TOC:

- **Compiler Design** → lexical and syntax analysis
- **Artificial Intelligence** → decision problems
- **Cryptography** → hardness of problems
- **Operating Systems** → resource allocation
- **Database Systems** → query optimization

Practical Applications of TOC

Example 1: Manufacturing

In a production line, one machine processes parts slower than others, causing a backlog. TOC helps identify this machine as the constraint and optimize its use, resulting in higher overall output.

Example 2: Service Industry

In a hospital, patient delays may occur due to limited diagnostic equipment. TOC helps focus improvement efforts on scheduling and utilization of that equipment.

Example 3: Project Management

In projects, TOC is applied through Critical Chain Project Management (CCPM) to reduce delays and improve completion times

“ WEEK-02 Computational Model ”

What is a Computational Model?

A computational model is an abstract mathematical representation of a computer used to describe how computation is carried out.

It does not describe physical hardware; instead, it focuses on:

- How input is processed
- How memory is used
- How decisions are made
- What problems can be solved

Computational models allow us to formally define algorithms, compare their power, and understand the limits of computation.

Note: In Theory of Computation, machines define what it means to compute.

Major Computational Models in TOC

Theory of Computation mainly studies three fundamental computational models:

1. **Finite Automata (FA)**
2. **Pushdown Automata (PDA)**
3. **Turing Machines (TM)**

Finite Automata (FA)

A **Finite Automaton (FA)** is a **mathematical model of computation** used to recognize **regular languages**. It reads an input string symbol by symbol and moves between a **finite set of states** according to defined rules (transitions).

Formal Definition of Finite Automata

A finite automaton can be defined as a tuple:

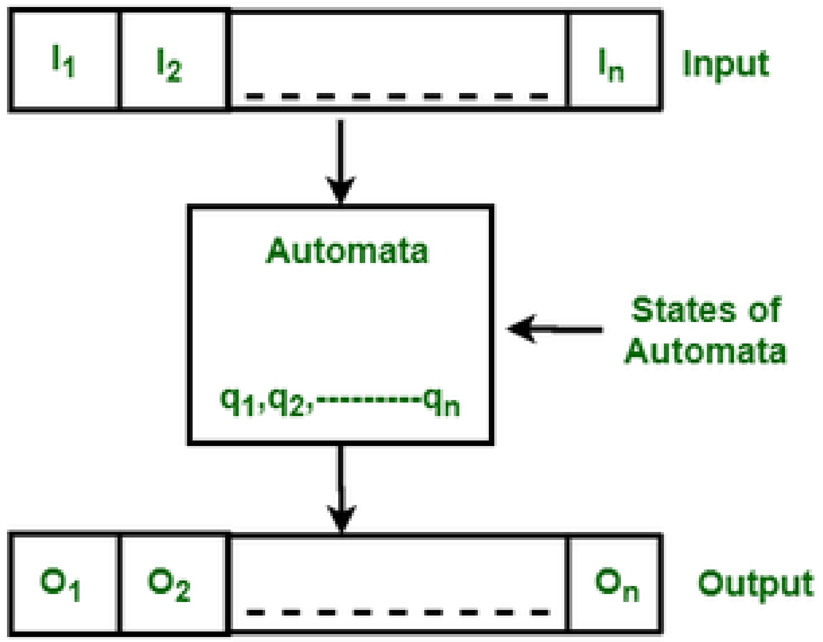
{ Q, Σ, q, F, δ }, where:

- Q: Finite set of states
- Σ: Set of input symbols
- q: Initial state
- F: Set of final states
- δ: Transition function

- Consist of states, transitions, and input symbols, processing each symbol step-by-step.
- If ends in an accepting state after processing the input, then the input is accepted; otherwise, rejected.
- Finite automata come in deterministic (DFA) and non-deterministic (NFA), both of which can recognize the same set of regular languages.
- Widely used in text processing, compilers, and network protocols.

Features of Finite Automata

- **Input:** Set of symbols or characters provided to the machine.
- **Output:** Accept or reject based on the input pattern.
- **States of Automata:** The conditions or configurations of the machine.
- **State Relation:** The transitions between states.
- **Output Relation:** Based on the final state, the output decision is made.



Components of a Finite Automaton

A **Finite Automaton** is formally defined as a 5-tuple:
FA = $(Q, \Sigma, \delta, q_0, F)$

Component	Description
Q	Finite set of states
Σ	Input alphabet
δ	Transition function ($Q \times \Sigma \rightarrow Q$)
q_0	Start state ($q_0 \in Q$)
F	Set of accepting states ($F \subseteq Q$)

Types of Finite Automata

There are two types of finite automata:

- Deterministic Fintie Automata (DFA)
- Non-Deterministic Finite Automata (NFA)

Deterministic Finite Automata (DFA)

A Deterministic Finite Automaton (DFA) is a FA where each state has exactly one transition for each input symbol.

- A Deterministic Finite Automaton (DFA) is a type of finite automaton in which:
- For each state, there is exactly one transition for each input symbol from the alphabet Σ.
- The machine starts in a unique start state.
- It accepts a string if, after reading all input symbols, it ends in an accepting (final) state.

A DFA is a **5-tuple**: **DFA = $(Q, \Sigma, \delta, q_0, F)$**

Symbol	Meaning
Q	Finite set of states
Σ	Input alphabet
δ	Transition function: $\delta(Q \times \Sigma \rightarrow Q)$
q_0	Start state ($q_0 \in Q$)
F	Set of accepting (final) states ($F \subseteq Q$)

How DFA Works

1. Start at **q_0.**
2. Read the **input string symbol by symbol**.
3. Move to the **next state** according to δ.
4. If the **last state** after reading all symbols is in **F**, the string is **accepted**; otherwise, **rejected**.