INTRODUCTION

INDEX:

1. Mathematical Preliminaries

Automata theory relies heavily on some basic mathematics: sets, relations, functions, graphs, logic, and proofs. These tools give us the language to define machines formally. Let's start with Sets.

1.1 Sets 📦

A **set** is an unordered collection of distinct objects. The objects in a set are called **elements** or **members**. The key characteristics of a set are that its elements are **unique** (no duplicates) and the **order doesn't matter**.

We usually denote sets with capital letters and enclose their elements in curly braces {}.

Key Concepts:

- **Element of (∈):** The symbol ∈ means "is an element of." For the set A={1,3,5}, we can say 3∈A.
- **Subset (⊆):** Set A is a **subset** of set B if every element of A is also an element of B. For example, {1,3} is a subset of {1,2,3,4,5}.
- **Cardinality (|A|):** The number of elements in a set. For A={a,b,c}, |A|=3.
- **Empty Set (∅ or {}):** The unique set with no elements. Its cardinality is 0.

Examples in Automata Theory:

- **An Alphabet (Σ):** A set of symbols. Σ={a,b}
- **A set of States (Q):** A set representing the states in a machine. Q={q0 ,q1,q2}

Set Operations

These are common ways to combine or modify sets.

- **Union (A∪B):** A new set containing all elements that are in A, or in B, or in both.
- **Intersection (A∩B):** A new set containing only the elements that are in both A and B.

- **Difference (A−B):** A new set containing elements that are in A but not in B.
- **Power Set (P(A) or 2A):** The set of all possible subsets of A. This is crucial for converting a Nondeterministic Finite Automaton (NFA) to a DFA, where a single DFA state represents a *set* of NFA states.

1.2 Relations 🔗

A **relation** describes a connection between elements of sets. A **binary relation** is a set of **ordered pairs** (a, b), where the first element a is from a set A (the **domain**) and the second element b is from a set B (the **codomain**).

Unlike sets, the order in an ordered pair matters: (a, b) is not the same as (b, a).

Example: Let A={Student1, Student2} and B={CS101, MATH203}. A relation "is enrolled in" could be
R={(Student1, CS101),(Student1, MATH203),(Student2, CS101)}.

Example in Automata Theory:

- The **transition function (δ)** is a relation. It relates a state and an input symbol to a next state. For example, the transition q0 goes to q1 on input a can be represented by the ordered pair ((q0, a), q1).

Properties of Relations

When a relation is defined from a set to itself (e.g., from A to A), it can have several important properties:

- **Reflexive:** Every element is related to itself. For all a in A, (a, a) is in the relation.
- **Symmetric:** If a is related to b, then b is related to a. If (a, b) is in the relation, then (b, a) must also be in it.
- **Transitive:** If a is related to b and b is related to c, then a is related to c. If (a, b) and (b, c) are in the relation, then (a, c) must also be in it.

A relation that is reflexive, symmetric, and transitive is called an **equivalence relation**. Equivalence relations are used to partition sets into distinct classes, a concept used in algorithms like DFA minimization.

2. Types of Automata

Automata theory is the study of abstract computing devices, or "machines." These machines are grouped into a hierarchy based on their computational power and the complexity of the languages they can recognize. The three fundamental types are Finite Automata, Pushdown Automata, and Turing Machines.

2.1 Finite Automata (FA) ⚙️

A **Finite Automaton** (also called an FSM or Finite State Machine) is the simplest type of automaton. It serves as a model for systems that have a finite number of states and no memory other than their current state.

- **How it Works:** An FA reads an input string one symbol at a time and moves from state to state based on a predefined **transition function**. A string is "accepted" if the machine ends in a designated

accepting state after reading the entire string.

- **Memory:** It has no external memory. Its only "knowledge" of the past is encoded in the state it is currently in.
- **Formal Components:** A deterministic FA (DFA) is formally defined as a 5-tuple: $A=(Q,\Sigma,\delta,q0,F)$, where:
    1. Q is a finite set of **states**.
    2. Σ is a finite set of **input symbols** (the alphabet).
    3. δ is the **transition function** that maps a state and an input to a next state.
    4. q0 is the **start state**.
    5. F is a set of **final or accepting states**.
- **Language Class:** FAs recognize **Regular Languages**. They are ideal for tasks involving simple pattern matching, such as finding keywords in text or the lexical analysis phase of a compiler.

2.2 Pushdown Automata (PDA) 📚

A **Pushdown Automaton** is essentially a Finite Automaton augmented with a **stack** for memory. This stack allows the PDA to be more powerful than a simple FA.

- **How it Works:** A PDA's transition from one state to another depends not only on the current state and input symbol, but also on the symbol at the top of the stack. In a single move, a PDA can modify the stack by pushing a new symbol on top or popping the top symbol off.
- **Memory:** It has a single **stack**, which provides a Last-In, First-Out (LIFO) memory structure. This memory is more powerful than an FA's but is still restricted; it can only access the top element of the stack.
- **Power:** The stack allows PDAs to recognize languages that require some form of counting or matching of nested symbols, which FAs cannot. A classic example is the language of balanced parentheses or the language $L=\{0^n1^n | n \geq 0\}$.
- **Language Class:** PDAs recognize **Context-Free Languages**. They are fundamental to parsing in compilers, which deals with the recursively nested structures of programming languages.

2.3 Turing Machines 💻

The **Turing Machine** is a more powerful and general model of computation. It was studied by A. Turing in the 1930s and is considered a model with all the capabilities of a modern computer.

- **How it Works:** A Turing machine consists of a finite control (like an FA) that can read and write symbols on an **infinite tape**. The machine can also move the tape left or right one cell at a time.
- **Memory:** Its memory is an **infinite tape**. This provides unlimited storage that can be accessed in any order (though not as quickly as random-access memory).
- **Power:** The Turing machine is the theoretical basis for modern computing. It is used to define precisely the boundary between what a computing machine can and cannot do. Any problem that can be solved by an algorithm can be solved by a Turing machine.

- **Language Class:** Turing Machines recognize **Recursively Enumerable Languages**. They are used to study "decidability" (what problems can be solved at all) and "intractability" (what problems can be solved efficiently).

Hierarchy (Power of Automata):

$$FA < PDA < TM$$

Every FA is a PDA (without using the stack).

- Every PDA is a TM (with restrictions).
- But not every TM is an FA.

---

3. Finite Automata (FA)

A **Finite Automaton** is the most basic model of computation. It has:

- A **finite set of states** (like checkpoints).
- An **input alphabet** (symbols it can read).
- A **transition function** (rules for moving between states).
- A **start state** (where it begins).
- A set of **accept states** (where it "accepts" input).

Formally, an FA is a 5-tuple:

$M=(Q,\Sigma,\delta,q0,F)$

where:

- Q: set of states
- Σ: input alphabet
- δ: transition function $Q\times\Sigma\rightarrow Q$
- q0: start state ($q0 \in Q$)
- F: set of final/accept states ($F \subseteq Q$)

3.1 Behaviour of FA

- **How it works:**
    1. The FA starts in the **start state**.
    2. Reads the input string symbol by symbol.
    3. For each symbol, it **moves to a new state** according to the transition function.
    4. After reading the whole string:
        - If the FA ends in an **accept state**, the string is **accepted**.
        - Otherwise, the string is **rejected**.

Example:
Alphabet: Σ={0,1}
FA that accepts strings ending with 1.

- States: Q={q0,q1}
- Start: q0
- Accept: q1
- Transition:
    - From q0 : read 0 → stay in q0, read 1 → go to q1
    - From q1: read 0 → go to q0, read 1 → stay in q1

👉 So:

- Input 101 → ends in q1 → accepted.
- Input 100 → ends in q0 → rejected.

Why is this important?

- FA are used in **text search, pattern recognition, and compilers (lexical analysis)**.
- They help us define **regular languages** formally

---

4. Deterministic Finite Automaton (DFA)

A **DFA** is a special type of finite automaton where:

- For **every state** and **every input symbol**, there is **exactly one transition**.
- No guessing, no multiple options → everything is **deterministic**.

Formally:

M=(Q,Σ,δ,q0,F)M = (Q, \Sigma, \delta, q_0, F)M=(Q,Σ,δ,q0,F)

where:

- Q: finite set of states

- Σ: finite input alphabet
- δ: transition function $Q \times \Sigma \to Q$
- $q0 \in Q$: start state
- $F \subseteq Q$: set of final (accepting) states

4.1 Formal Definition

For a string $w=a1a2a3...an$, where each $ai \in \Sigma$:

- Start at the initial state q0.
- Apply the transition function repeatedly:

  $\delta*(q0,w)=\delta(...\delta(\delta(q0,a1),a2)...,an)$
- If the resulting state is in F, then **w is accepted**.

4.2 Simplified Notations

Since 5-tuples are abstract, we usually describe DFAs in **two simpler ways**:

4.2.1 State Transition Diagram

- A directed graph:
  - **Circles** = states
  - **Arrow →** = transition
  - **Double circle** = accept state
  - **Arrow from nowhere →** start state

Example: DFA for "strings ending with 1" over {0,1}\{0,1\}{0,1}:

- States: q0,q1
- Start: q0
- Accept: q1
- Diagram (sketch in notes):
  - From q0: 0 → stay in q0, 1 → go to q1

- From q1: 0 → go to q0, 1 → stay in q1

4.2.2 Transition Table

Same DFA in tabular form:

State	Input 0	Input 1
→ q0	q0	q1
*q1	q0	q1

- → = start state
- * = accept state

4.3 Language of a DFA

The **language of a DFA** is the set of all strings it accepts:

$L(M)=\{w \in \Sigma* | \delta*(q0,w) \in F\}$

Example: For the DFA above (strings ending with 1):

$L=\{w \in \{0,1\}* | w \text{ ends with } 1\}$

---

5. Non-Deterministic Finite Automaton (NFA)

An **NFA** is like a DFA but with more freedom:

- For a given state and input, it may have **zero, one, or many transitions**.
- It can even move **without consuming input** (ε-transitions).
- The machine "guesses" a path, and if **any path leads to an accept state**, the string is accepted.

5.1 Formal Definition

An NFA is a 5-tuple:

$M=(Q,\Sigma,\delta,q0,F)$

where:

- Q: finite set of states
- Σ: input alphabet
- δ: transition function

  $\delta:Q\times\Sigma\rightarrow P(Q)$

(Instead of a single state, it returns a **set of states**.)

- $q0 \in Q$: start state
- $F \subseteq Q$: set of accept states

Key Difference:

In DFA, $\delta : Q\times\Sigma \rightarrow Q$.
In NFA, $\delta : Q\times\Sigma \rightarrow P(Q)$.

5.2 Language of an NFA

The **language of an NFA** is the set of all strings for which **at least one possible path** leads from the start state to an accept state.

Formally:

$L(M) = \{w \in \Sigma^* \mid \delta^*(q0,w) \cap F \neq \emptyset\}$

Example:
Alphabet: $\Sigma=\{0,1\}$.
NFA that accepts strings ending with 01.

- States: q0,q1,q2
- Start states: q0
- Accept states: q2
- Transitions:
    - From q0: on 0 → q0 or q1
    - From q1: on 1 → q2
    - q2: accept state

Path example:

- Input 1101:
    - From q0 follow the 0 → go to q1.
    - Then on 1 → go to q2.
    - Ends in accept state → accepted.

Important Fact:
Even though NFAs look more powerful, **NFA and DFA recognize exactly the same class of languages (Regular Languages)**.

- For every NFA, there is an equivalent DFA.

6. Finite Automata with Output

So far, FA, DFA, and NFA were **language recognizers** (only say yes/no).
But sometimes we need automata that **generate outputs** while processing input.

Two main models exist:

- **Moore Machine**
- **Mealy Machine**

6.1 Moore Machine

Definition:
A **Moore Machine** is a 6-tuple:

$M=(Q,\Sigma,\Delta,\delta,\lambda,q0)$

where:

- Q: set of states
- Σ: input alphabet
- Δ: output alphabet
- $\delta:Q\times\Sigma\to Q$ → transition function
- $\lambda:Q\to\Delta$ → output function (depends only on states)
- $q0 \in Q$: start state

Key Idea:

- Output depends **only on the current state**.
- Each state has a fixed output.

Example:
Suppose input alphabet $\Sigma=\{0,1\}$
output alphabet $\Delta=\{A,B\}$

- State q0: output A
- State q1: output B
- Transition:

- From q0: on 1 → go to q1
- From q1: on 0 → go to q0

👉 If input = 10, output sequence = AB.

6.2 Mealy Machine

Definition:
A **Mealy Machine** is a 6-tuple:

$M=(Q,\Sigma,\Delta,\delta,\lambda,q0)$

where:

- Same components as Moore, except:
- $\lambda: Q \times \Sigma \to \Delta$ → output function depends on **state + input**.

Key Idea:

- Output depends on **current state and current input**.
- Faster reaction (output changes immediately when input changes).

Example:
Input alphabet $\Sigma=\{0,1\}$

output alphabet $\Delta=\{X,Y\}$

- In state q0:
  - On input 0 → output X, go to q0
  - On input 1 → output Y, go to q1

👉 If input = 10, output sequence could be YX.

Moore vs Mealy (Key Difference):

Feature	Moore Machine	Mealy Machine
Output depends on	Current state	Current state + input
Output changes	On state change	Immediately with input
States required	More	Fewer (compact design)

---

7. Applications in IT

Finite automata and their variants are not just math toys — they're **core tools in computer science**. Let's break them down:

7.1 Lexical Analysis

- **Where?** In compilers.
- **What happens?** Source code → tokens (keywords, identifiers, numbers, operators).
- DFA is used to scan input one character at a time and decide if it forms a valid token.
- Example: To recognize identifiers in C (like hello123), an FA checks if the first character is a letter, then accepts letters/digits afterward.

7.2 Protocol Design

- **Where?** Networking (communication protocols).
- Protocols can be modeled as states + transitions.
- Example: TCP connection states (LISTEN, SYN_SENT, ESTABLISHED, CLOSED) → represented as an FA.
- Ensures communication follows rules, and errors (invalid transitions) are detected.

7.3 Text Search

- **Where?** Search engines, editors, pattern matching.
- FA can be built for a given pattern and used to scan large text efficiently.
- Example: Searching for "cat" in a document → FA with states that track c, then a, then t.

7.4 Traffic Light Control

- **Where?** Embedded systems.
- Traffic lights cycle through finite states (Red → Green → Yellow).
- The automaton ensures correct sequence and timing.
- Can also handle conditions like pedestrian crossing buttons or sensor signals.

7.5 Medical Diagnosis

- **Where?** Expert systems & AI.
- Symptoms can be modeled as inputs; states represent possible health conditions.
- Example: If fever + cough → move to “flu state”, else if chest pain → move to “cardiac check state”.
- Used in **rule-based medical diagnostic systems**.

---

8. JFLAP Software

What is JFLAP?

- **JFLAP (Java Formal Languages and Automata Package)** is an educational software tool.
- It helps students **build, test, and visualize automata** and other formal language concepts.
- Developed at Duke University by Susan H. Rodger and her team.

8.1 Introduction to JFLAP

Features:

1. **Finite Automata (FA, DFA, NFA):**
    - You can draw states, transitions, and simulate inputs.
    - Helps visualize acceptance/rejection of strings.
2. **Pushdown Automata (PDA):**
    - Models with stack → test context-free languages.
3. **Turing Machines:**
    - Design and simulate multi-tape TMs.
4. **Grammars & Languages:**
    - Work with Regular Expressions, CFGs, and conversions between them.
5. **Conversions:**
    - NFA → DFA
    - DFA → Regular Expression
    - CFG → PDA
6. **Step-by-Step Simulation:**
    - You can **trace** execution of automata with input strings.

    - Good for debugging and learning.

Why JFLAP is Useful:

- Removes abstract difficulty → makes automata concepts **visual and interactive**.
- Lets you **experiment**: draw automata, feed inputs, and immediately see results.
- Great for assignments, projects, and exam prep.

Example Use Case:

- Build an NFA for strings ending with 01.
- Feed input 1001.
- JFLAP will show possible state paths and highlight the accept state if successful.