

Chapter 1: Finite automata and their application in text processing

Himpriya Kumari, Sunil Kumar

Abstract: This chapter investigates the key concepts of finite automata and then finite-state machines. It comprehensively analyses its mathematical and theoretical foundations, varied classes, and extensive applications across multiple domains. The paper aims to cover the notion of regular languages, the differences between the types of finite state machines, and their role in language theory and computability. The prerequisites and constraints of such machines are also cited in detail. This paper focuses on examining its diverse applications in fields such as Software Development, Robotics, Communications, Artificial Intelligence and Cybersecurity.

Keywords: alphabet, automata, language, regular expression, search engine

1 Introduction

Automata Theory is a constituent of theoretical computer science and mathematical analytics that closely explores abstract machines and their computational capacities. The early foundations of this theory date back to the pre-20th century. An engineer named Hero of Alexandria built mechanical devices that could mimic human and animal actions in 50 AD. Later on, in the 17th century, Gottfried Wilhelm Leibniz pictured a rational device capable of reasoning mathematically, an idea that foreshadowed formal logic.

Himpriya Kumari

Department of Mathematics, Chandigarh University, Punjab, India.

Sunil Kumar

Department of Mathematics, Chandigarh University, Punjab, India.

Following him, in the 19th century, George Boole developed Boolean algebra, which later became a crucial tool in formalising automata. These innovations eventually led to the formalisation of automata in the 1930s-1940s. In 1936, an English mathematician and computer scientist named Alan Turing presented the idea of a theoretical computing machine that is today known as the Turing Machine, which evolved to be the foundation for computability theory. Alonzo Church's work on lambda calculus and Norbert Wiener's idea of cybernetics led to the contribution of influencing automata and language theory (M. Sipser, 2013).

The inception of Finite Automata occurred in the 1950s with the invaluable contribution of several prominent and designated scientists and mathematicians. Claude Shannon's Switching Theory formed the basis of digital automata. In 1956, an American mathematician, Stephen Cole Kleene, introduced finite automata and regular expressions (DFA/NFA models). An American professor, Noam Chomsky, introduced the Chomsky Hierarchy in 1956, which types languages into regular, context-free, context-sensitive, and recursively enumerable. Two years later, John McCarthy, an American computer scientist, developed a language based on formal grammar and recursion and named it LISP. The second half of the 20th century and then the 21st century witnessed the extension of the Automata Theory. Pushdown Automata, Context-free Languages, Buchi Automata, Applications in Computability and Complexity and Automata in Verification and Artificial Intelligence were developed. (A. Maheshwari and M. Smid, 2019)

Automata theory is the basis for the theory of formal languages. Formal languages are treated as mathematical sets and usually have several components, such as symbols, alphabets, and words, which later form a language. Automata theory has 3 components - inputs, states and outputs. A finite automaton describes a regular language. Based on the type of work, it is of two types - acceptor and transducer. The acceptor accepts or rejects given strings, and transducers produce an output string for the given input string (K. Verma, S. Kourav, M. Jangid, U. Sahu, and N. Shivhare, 2023). Based on the output, finite automata are of two types - with output and without output, which are discussed in detail later on in the paper. This mathematical model of computation with a finite number of states that possess input symbols and transition between states in accordance with some predefined rules, is a significant notion in automata theory and formal language theory, laying the ground for lexical analysis in compilers, text-search algorithms, weather forecasting and communications, robotics and artificial intelligence and in digital circuit design. (D. Lee and M. Yannakakis, 1996)

2 PRELIMINARIES

This section consists of some definitions that are required for the complex concepts (Hopcroft, J. E., Motwani, R., & Ullman, J. D. 2006).

2.1 Alphabet - It is a finite set of symbols used to construct a string. These form the basic building blocks. For example, the English alphabet = (a, b, ..., z), the Binary alphabet = (0,1), and the Decimal alphabet = (0, 1, 2, ..., 9).

2.2 String - It is a sequence of symbols defined over a given alphabet.

2.3 Language - It is a set of strings formed from an alphabet Σ , according to specific rules.

- Σ^* = Collection of all strings.
- Language is a subset of Σ^* .

2.4 Automata Theory - It is a study of abstract machines and computational problems. It is a theory in theoretical computer science. The word 'automata' comes from a Greek word that means 'self-willing/ self-acting/ self-moving.'

2.5 Automata/ Automation - It refers to a system that transforms and transmits information, which is used for some action without any human intervention, such as an automatic coffee machine, washing machine and many more.

2.6 Components of Automation -

- Inputs - It indicates the value taken from the input alphabet and applied to the automation at a discrete instant of time. It is denoted by I.
- Outputs - It refers to the response of the automation based on the input taken and is denoted by O.
- State - At a specific instant of time, the automation can take any state, and it is represented by q.
- State relation - It refers to the next stage of automation based on the current state and current input.
- Output relation - Output is related to either the state only or both the state and input.

3 MATHEMATICAL AND THEORETICAL FOUNDATIONS

This section comprises diversified concepts, definitions, examples and rules related to finite automata and their components.

3.1 Formal Definition of Finite Automata - A finite automaton is represented formally using a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q = a finite, non-empty set of states
- Σ = a finite, non-empty set of input symbols called an alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ = transition function
- q_0 = initial state
- F = set of final states ($F \subseteq Q$) [4][6]

Here, it is essential to comprehend the attributes of each of these elements to apprehend their role.

- Here, Q , which is the set of states, consists of all states, such as the initial, intermediate and final. The initial state is the state from which the automation starts. The intermediate state, as the name indicates, falls in the middle of the automation. The final state is the state where the automation terminates, and no further transition takes place.
- The transition function is denoted by delta (δ). It has two parameters: the current state and the input symbol. It returns a state, which is then called the successive state.

The representation of the transition function has a syntax:

$$\delta (\text{current state, current input symbol}) = \text{next state} \rightarrow \delta : Q \times \Sigma \rightarrow Q$$

For example, assume q_0 is the current state and 0 is the current input symbol. Then, the transition function is denoted by $\delta (q_0, 0) = q_1$. In this transition, the current state q_0 , on receiving input 0, moves to the next state q_1 [5].

3.2 Finite Automata Model - A finite automaton model conceptually consists of 3 parts. They are:

- **Input Tape** - A tape to carry the input string. The tape is divided into a limited number of cells. Each cell bears a symbol from Σ .
- **Tape Head** - The input tape consists of a tape head for reading symbols from the tape. It reads the cells one by one from left to right, and at a time, only one input symbol is read.

- **Finite Control** - This control has 3 parts to it. It has a finite number of states that the machine is allowed to be in. Secondly, it has a current state, initially set to an initial state. Lastly, it has a state transition function for changing the current state (Prajana, Mamidi et. al 2024).

An automaton processes a string on the tape by replicating the subsequent steps until the tape head has read the complete string:


- The tape head starts reading the first tape cell. It reads the current symbol and sends it to the control. Then, the tape head moves to the following cell.
- The control takes the current state symbol and applies the state transition function to reach the successive state, which eventually evolves to be the new present state.
- Once the complete string is processed, the state in which the automaton enters is reviewed. If the state is accepted, the input string is accepted; otherwise, it is rejected (Deshmukh, Mrugakshi. Et. Al. 2022)

3.3 Transition Systems (Transition Graph and Transition Diagram) - A finite automaton is usually portrayed diagrammatically via a transition system known as a transition graph. This graph is subsequently transformed into a transition diagram, which is a directed graph in which the vertices correspond to the states of the given finite automata. There are certain rules to represent a finite automaton through a transition graph and then convert it to a transition diagram.

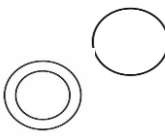
Rules -

- The initial state is represented by an arrow accompanied by a single circle.
- The intermediate states are represented by single circles.
- The final state is displayed by a double circle.
- The circles that represent different states are connected via arrows that represent the state relation, i.e., the relation between the states.

Elements of the Transition Graph -

- Initial State (say q_0) \rightarrow 

Intermediate State (say q_1)

- Final State (say q_2)
 - State Relation \rightarrow
- 

3.4 Transition Table - The transition function of a particular finite automaton needs to be represented in a tabular form. A transition table is a tabular representation of the transition function. It takes two major arguments - a state and a symbol, and then returns the value, i.e. the successive state. Certain rules need to be followed to construct the transition table. They are as follows:

- The rows in the table correspond to states.
- The columns in the table represent the input symbols.
- The entries in the table correspond to the successive states.
- The initial state in the table is marked with an arrow.
- The final state in the table is marked with an asterisk symbol.

4 CLASSES OF FINITE AUTOMATA

Finite Automata are mainly classified into two types -

1. **Finite Automata without output** - This type of finite automata does not return any value/output. These machines talk about acceptance, i.e., giving input on whether machines accept them. It is further divided into three parts:
 - Deterministic Finite Automata (DFA)
 - Non-deterministic Finite Automata (NFA/NDFA)
 - Non-deterministic Finite Automata with epsilon moves (NFA-E)
2. **Finite Automata with Output** - This type of Finite Automata returns an output. These machines produce output for a given input, i.e., they talk about functionality. This is further classified into two kinds:
 - Moore Machine

- Mealy Machine

4.1 Deterministic Finite Automata (DFA) - A Deterministic Finite Automaton is a finite automaton that reads an input symbol at a time. The word “deterministic” refers to the uniqueness of a computation. A machine for which a deterministic code can be formulated, and if there is only one unique way to formulate the code, then the machine is called a deterministic finite automaton.

4.1.1 Formal Definition of DFA - A deterministic finite automaton is represented formally using a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q = a finite, non-empty set of states
- Σ = a finite, non-empty set of input symbols called an alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ = transition function
- q_0 = initial state
- F = set of final states ($F \subseteq Q$)

4.1.2 Properties of DFA -

- DFAs have only one path from the present state to the next state for a particular input.
- DFAs cannot change state in the absence of an input character.
- DFAs do not accept null moves.
- DFAs can contain more than one final state.

4.1.3 Acceptance of Strings - A DFA accepts a string $S = a_1, a_2, \dots, a_n$ and a sequence of states $Q = q_1, q_2, \dots, q_n$ such that

1. q_0 is the start state
2. $\delta(q_i, a_{i+1}) = q_{i+1}$ for all $0 < i < n$
3. q_n belongs to F (where F is the set of final states)

4.1.4 Language recognised by a DFA - The language accepted by a DFA - M is the set of all strings accepted by M and represented by $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$ [5].

4.1.5 Example - Construct a DFA where $Q = \{q_e, q_0\}$. If the FA is in state q_e , then it reads an even number of 1s, and if it is in state q_0 , then it reads an odd number of 1s and $\Sigma = \{0, 1\}$.

- Here, q_e will be the start state as the number of 1s read by FA is zero, and zero is even.
- So, we'll construct a transition table.
- Now, we'll construct the transition diagram according to this transition table.
- In the diagram we'll see that, q_e makes a self-loop and stays in q_e only when it receives input 0, whereas when it receives 1, it goes to the state q_0 . Then q_0 makes a self-loop and stays in q_0 when it receives 0, whereas when it receives 1, it goes back to q_e . Here, q_0 is the final state, so it is denoted by double circles.

4.2 Non-deterministic Finite Automata (NFA/NDFA) - Non - Non-deterministic finite automata are a type of finite automata that have multiple paths for one particular input for transition from the current state to the following state.

4.2.1 Formal Definition of NFA/NDFA - A non-deterministic finite automaton is represented formally using a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q = a finite, non-empty set of states
- Σ = a finite, non-empty set of input symbols called an alphabet
- $\delta : Q \times \Sigma \rightarrow 2^Q$ = transition function
- q_0 = initial state
- F = set of final states ($F \subseteq Q$)

4.2.2 Properties of NFA/NDFA -

- In comparison with DFA, the construction of NFA is easier.

- NFA is a generalisation of DFA. Every DFA is an NFA, but not vice versa.
- Every NFA can be translated into its corresponding DFA.
- In NFA, multiple next states exist.
- In NFA, null moves or ϵ -transitions also exist.

4.3 Non-deterministic Finite Automata with ϵ -moves (ϵ -NFA) - Non-deterministic finite automata with ϵ -moves are a type of finite automata that have multiple paths for one particular input for the transition from the current state to the following state.

4.3.1 Formal Definition of ϵ - NFA - A non-deterministic finite automaton with ϵ - moves is a regular NFA and is represented formally using a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q = a finite, non-empty set of states
- Σ = a finite, non-empty set of input symbols called an alphabet
- $\delta : Q \times \Sigma \cup \epsilon \rightarrow 2^Q$ = transition function
- q_0 = initial state
- F = set of final states ($F \subseteq Q$)

4.4 Mealy Machine - A Mealy Machine is a type of finite automaton that returns an output. It is a finite-state machine where the output is determined by both the current state and the current input. George H. Mealy, an American mathematician and computer scientist, invented the Mealy Machine in 1955, and so the machine was named after him.

4.4.1 Formal Definition of Mealy Machine - A Mealy machine is formally described by 6-tuples $(Q, \Sigma, O, \delta, X, q_0)$, where

- Q = a finite, non-empty set of states
- Σ = a finite, non-empty set of input symbols called an alphabet
- O = a finite set of symbols called the output alphabet
- $\delta: Q \times \Sigma \rightarrow Q$ = input transition function

- $X: Q \times \Sigma \rightarrow O =$ output transition function
- $q_0 =$ initial state

4.4.2 Properties of Mealy Machine -

- This machine talks about functionality.
- The output is associated with the transition.
- This machine operates within a limited number of states.
- Input affects output in this machine.
- The output of a Mealy Machine changes as soon as the input changes, even before the state transition.
- Each transition between states is associated with an output function that defines the output established on the existing state and input.
- This machine has a compact design as it requires fewer states as compared to the other type.

4.5 Moore Machine - A Moore Machine is another type of finite automaton that returns an output. It is a finite-state machine where the output depends only on the current state. An American professor of Mathematics and Computer Science, Edward Forrest Moore, introduced this machine in 1956, and so it was named after him.

4.5.1 Formal Definition of Moore Machine - A Moore machine is formally described by 6-tuples $(Q, \Sigma, O, \delta, X, q_0)$, where

- $Q =$ a finite, non-empty set of states
- $\Sigma =$ a finite, non-empty set of input symbols called an alphabet
- $O =$ a finite set of symbols called the output alphabet
- $\delta : Q \times \Sigma \rightarrow Q =$ input transition function
- $X: Q \times \Sigma \rightarrow O =$ output transition function
- $q_0 =$ initial state

4.5.2 Properties of Moore Machine -

- This machine talks about functionality.
- Output depends only on the state.
- The output of a Moore Machine changes only on a state transition.
- It has a stable output.
- Since outputs are associated with states, it has more number of states.
- This machine requires more logic to decode the output, resulting in more circuit delays.

5 Minimization of Deterministic Finite Automata - Deterministic Finite Automata can be minimized. Minimization means reducing the number of states from a given finite automaton.

The steps are as follows:

1. Remove all unreachable states from the initial state.
2. Construct the transition table for all pairs of states.
3. Split the table into two different tables - one that contains all final states and the other with all non-final states.
4. Find the two states which have the same value of input and next state, and then remove them.
5. Repeat step 4 until no similar rows are found in the first table.
6. Now repeat steps 4 and 5 for the second table as well.
7. Combine the two tables and construct a final transition table of a minimised DFA.

6 Finite State Machines in Language Theory and Computability

This section talks about different concepts related to regular languages and regular expressions.

6.1 Regular Language - The language accepted by Finite Automata can be easily described by a simple expression called a regular expression. The language accepted by

some regular expressions is referred to as regular language. It is the most effective way to represent any language. It can also be described as a sequence of patterns that defines a string.

6.2 Operations on Regular Language - Several operations are closed in Regular Languages.

1. **Union** - If L and M are two regular languages, then $L \cup M$ is also a regular language.

$$L \cup M = \{s: s \text{ is in } L \text{ or is in } M\}$$

2. **Intersection** - If L and M are two regular languages, then $L \cap M$ is also a regular language.

$$L \cap M = \{s: s \text{ is in both } L \text{ and } M\}$$

3. **Kleen Closure** - L^* - If L is a regular language, then L^* is also a regular language. It consists of all strings that can be formed by concatenating zero or more strings from L.
4. **Concatenation** - If L and M are regular languages, then their concatenation is also a regular language. It consists of all strings formed by taking a string from L followed by a string from M.

6.3 Example - Write a regular expression for the language accepting all strings containing any number of a's and b's.

Solution -

$$a^* = \{\epsilon, a, aa, aaa, \dots\}$$

$$b^* = \{\epsilon, b, bb, bbb, \dots\}$$

$$\text{Regular Expression} = (a+b)^*$$

6.4 Conversion of Regular Expression to Finite Automata - To convert a regular expression to a finite automaton, we use a method called the Subset Algorithm .

The steps are as follows:

1. Design a transition diagram for a given regular expression using an NFA with null moves.
2. Convert NFA with null moves to NFA without null moves.

3. Convert the obtained NFA to its equivalent DFA.

6.6 Pumping Lemma - It is used to show that a language is not regular.

6.6.1 Statement - If the substring of a string is repeated many times and if the resultant string is also available in language L, then we say that the language is regular.

6.6.2 Steps -

1. Consider the language 'L' as regular.
2. Assume a constant 'c' and select a string 'w' from L such that $|w| \geq c$.
3. Divide the string 'w' as 'xyz' such that $|y| > 0$ and $|xy| \leq c$ for $i \geq 0$, every string $xy^i z \in L$.

6.6.3 Example -

Prove that $L = \{a^i b^i \mid i \geq 0\}$ is not regular.

Solution -

- At first, we assume that L is regular and n is the number of states.
- Let $w = a^n b^n$. Thus $|w| = 2n \geq n$.
- By pumping lemma, let $w = xyz$, where $|xy| \leq n$.
- Let $x = a^p$, $y = a^q$, and $z = a^r b^n$, where $p + q + r = n$, $p \neq 0$, $q \neq 0$, $r \neq 0$. Thus $|y| \neq 0$.
- Let $k = 2$. Then $xy^2z = a^p a^{2q} a^r b^n$.
- Number of as = $(p + 2q + r) = (p + q + r) + q = n + q$
- Hence, $xy^2z = a^{n+q} b^n$. Since $q \neq 0$, xy^2z is not of the form $a^n b^n$.
- Thus, xy^2z is not in L. Hence L is not regular.

7 Applications of Finite Automata in Text Processing:

The most important part of human life is language, which can be represented in either spoken or textual content. Language has a significant role in document writing. The misspelling makes the content more difficult to read and reduces the text's informational

value (Kumar, S., & Kour, G, 2025). Many web apps have a spell checker, which is a highly valuable tool for users. It may offer options in a dropdown menu so that the tool will type the rest of the word that the user intends to type (Kumar, S., 2024). This tool assists the user in correcting errors in document text writing. Most of the spell checker systems which have been designed use Deterministic Finite Automata. (Singh, M. K., & Kumar, S., 2024)

Pattern/text recognition plays an important role in various fields such as information retrieval, data mining and bioinformatics. There are many matching algorithms for string machines. (Tiwari, K. K., Singh, A., & Kumar, S.,2025)

7.1 Text Processing - Searching for a word or regular expression in a document (e.g., Ctrl+f function). Finite automata are the theoretical basis of regular expressions. They efficiently recognize patterns in strings by transitioning through states. Regular expressions use FA to match string patterns efficiently. DFA and NFA are used in this (Kumar, S., Rampal, S., Gaur, M., & Gaur, M, 2024).

7.2 Key applications in text processing -

- Search engines break text into tokens and use RE to find matching patterns.
- Spell Checkers - Underlining misspelt words in Word or Google Docs. Dictionary words are represented in the form of an NFA.
- Regular Expression Engines - Tools like grep, Python, Java, JavaScript – built-in regex modules. Regex patterns are compiled into NFAs.

7.3 Example 1 - Here is a simple FA simulation to match the regex $a(b|c)^*d$ using Python.

Here, an online compiler is used.

The Python code is -

```

def match_string(s):
    state = 'q0'
    i = 0

    while i < len(s):
        char = s[i]

        if state == 'q0':
            if char == 'a':
                state = 'q1'
            else:
                return False
        elif state == 'q1':
            if char == 'b' or char == 'c':
                # stay in q1 for (b|c)*
                state = 'q1'
            elif char == 'd':
                state = 'q2'
            else:
                return False
        elif state == 'q2':
            # d must be the final character
            return i == len(s) - 1

```

```

        i += 1

    return state == 'q2'
test_strings = ["ad", "abd", "abcbcd", "a", "acd", "abcbcbcbcd", "abcd"]

for test in test_strings:
    result = match_string(test)
    print(f"{test}: {'Accepted' if result else 'Rejected'}")

```

The generated output is -

Output
ad: Accepted
abd: Accepted
abcbcd: Accepted
a: Rejected
acd: Accepted
abcbcbcbcd: Accepted
abcd: Accepted

Explanation - Here in the previous code, the input string is broken into tokens (characters) and then each character is checked to see if it meets the requirements of the user. This same technique is used in search engines and spell checkers, wherein the compiler checks the input and matches it with the existing data to generate the desired output for the user.

7.4 Example 2 - Construct a DFA and then a Python code to detect if a string ends with “ing”.

Problem Statement - To check if the input word (like "running", "eat", "singing") ends with the suffix "ing" — a common check in language processing or grammar detection.

DFA –

q0: Start

q1: The last character was 'i'.

q2: The Last two characters were 'in'

q3: Last three characters were 'ing' → Accepting state

Python Code using an online Python compiler:

```
def ends_with_ing(word):
    state = 'q0'

    for char in word:
        if state == 'q0':
            state = 'q1' if char == 'i' else 'q0'
        elif state == 'q1':
            state = 'q2' if char == 'n' else ('q1' if char == 'i' else 'q0')
        elif state == 'q2':
            state = 'q3' if char == 'g' else ('q1' if char == 'i' else 'q0')
        elif state == 'q3':
            state = 'q1' if char == 'i' else 'q0'

    return state == 'q3'

print(ends_with_ing("running"))
print(ends_with_ing("singing"))
print(ends_with_ing("run"))
print(ends_with_ing("kingdom"))
```


Generated Output:

Output
True
True
False
False

The previous example is mainly used in grammar checkers, search filters and auto-suggestions in writing tools.

8 Some Other Applications of Finite State Machines (FSM): Automata have been incredibly triumphant in comparing distinct sorts of complicated designs on sequences, with applications in numerous dimensions, from text retrieval to bioinformatics, from multimedia databases to signal processing. This section consists of the varied applications of finite-state machines in diverse domains of human existence.

8.1 FSM in software engineering:

1. Compiler Structure and Lexical Analysis

- Lexical Analysis: DFA is directed to scan and tokenise the input source code.
- Parsing: FA alters source code into tokens (such as keywords, operators, and identifiers).
- Syntax checking: FA guarantees an appropriate format before parsing.
- For example, Lex & Flex tools develop lexical analysers utilising regular expressions, which are converted into a DFA.
- Example: `int t = 9;`

Lexical analyser using FA:

- `int` → Keyword
- `t` → Identifier
- `=` → Assignment operator

- 9 → Integer constant

8.2 FSM in Artificial Intelligence:

1. Oration Recognition-

- FA is operated by voice assistants like Siri, Google Assistant, and Alexa to process said words into text.
- Finite-state transducers (FSTs) are used to map phonemes to words in speech-to-text conversion.
- Google Speech API uses FA to identify distinct accents and articulations.
- IBM Watson Speech-to-Text uses weighted FA to enhance speech recognition precision.

2. Morphological Research and Word Segmentation-

- FA enables the breakdown of words into significant elements (prefixes, roots, suffixes).
- Useful for machine translation (Google Translate) and chatbots (GPT-based models).
- Arabic and Japanese NLP systems utilise FA to segment expressions and specify core essences.

3. Neural Networks and Automata Learning

- FA is used to introduce recurrent neural networks (RNNs).
- Automata learning permits AI to comprehend language practices.
- AI-powered chatbots (Chatgpt, Bard, Watson) use FA for context tracking.

8.3 FSM in Communication Network and Cybersecurity:

1. Network Protocol Design

- FA is employed in the creation of transmission protocols like TCP/IP, HTTP, and Bluetooth.

- It aids in specifying accurate sequences of messages and error-handling strategies.

2. Data Compression

- FA is utilised in compression techniques such as Huffman coding and Lempel-Ziv-Welch (LZW).
- JPEG image compression uses FA for efficient encoding of picture data.

8.4 FSM in Digital Circuit Design:

- FA is used to develop sequential circuits like flip-flops, counters, and memory controllers.
- FSMs regulate electronic devices like elevators and vending machines.
- Traffic light controllers operate using FA:
 - RED → GREEN → YELLOW → RED

8.5 FSM in Database Management System (DBMS):

1. Query Processing and Optimization-

- FA is used in SQL query implementation to parse and validate commands.
- Regular expressions and FA help in text searching within databases.
- Mysql employs FA for query execution plans and string-matching operations.

9 Limitations and Challenges in FSM:

Though FSMs are widely used and have varied applications across domains, they also have some limitations and challenges, such as -

1. Finite Memory Constraint: FA has a finite number of states, making it unfit for recalling a random amount of data.
2. FA cannot process context-free languages (CFL).
3. FA fails to crack problems demanding recursion or unbounded memory access.

4. FA blunders with real-world applications that mandate probabilistic decision-making.
5. Different FAs can denote identical language, making optimization difficult and thus resulting in ambiguity.
6. Checking the language equivalency of two NFAs demands exponential time, thus showcasing the complexity of FAs.

Conclusions

Finite state machines (FSMs) propose several useful features that make them practical tools in a combination of industries and applications, as discussed in the paper. This paper deeply analyses the classes, configuration, applications and limitations of FSMs across domains. FAs have diverse applications in the fields of software engineering, AI, cybersecurity, communication networks and digital circuit design.

Despite these challenges, persistent study is expanding the ability of FA. Probabilistic and quantum finite automata are being investigated for applications in machine learning, cryptography, and quantum computing. Further, hybrid models incorporating FA with deep learning are paving the way for more competent and adaptive computing systems.

An FSM's elemental concept is to hold a string of individual states and transition between them based on the weights of the inputs and the machine's current state. Finite-state machines are a fundamental topic in computer science and engineering. They supply a methodical framework for depicting and assessing systems with discrete conditions and transitions, making them beneficial instruments for handling an expansive scope of computational and control matters.

References

- M. Sipser, *Introduction To The Theory of Computation - Michael Sipser*. 3rd ed., USA: Cengage Learning, 2013. [Online]. E-book.
- A. Maheshwari and M. Smid, "Introduction to Theory of Computation," in *Introduction to Theory of Computation*, 2019, pp. 1-128.
- K. Verma, S. Kourav, M. Jangid, U. Sahu, and N. Shivhare, "Research on Finite State Machine and Its Real Life Time Applications ", vol. 9. *Journey of Information Technology and Sciences*, Dec. 19, 2023.
- D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines survey." *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090-1123, 1996, doi: 10.1109/5.533956.

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to automata theory, languages, and computation* (3rd ed.). Pearson.

Handa, N., Kumar, S., Kumar, J., Development of a closed-loop supply chain system with exponential demand and multivariate production/remanufacturing rates for deteriorated products, *Materials Today: Proceedings*, Volume 47, Part 10, 2021, Pages 2560-2564, ISSN 2214-7853, <https://doi.org/10.1016/j.matpr.2021.05.055>.

Prajana, Mamidi & Rajkumar, Harsha & Sannidhi, Akepati & Vidyasri, Kamhari & Panda, Niharika. (2024). *Pattern Recognition for Identifying a String Within a Text File Using Finite Automata*. 1-7. 10.1109/ICCCNT61001.2024.10723912.

Tyagi, V. K., Goel, R., Singh, M., Kumar, S., (2020). Modeling and Analysis of a Closed Loop Supply Chain With uncertain Lead Time in the Perspective of Inventory Management. *International Journal of Scientific and Technology Research Sciences*, 9(1), 3643-3650.

Deshmukh, Mrugakshi & Deshmukh, Shreyash & Jangid, Devansh & Dewalkar, Dhanashree & Dighole, Rushikesh. (2022). *A Review Paper on Text Corrector Using Finite Automata*.

Kumar, S., & Kour, G. (2025, March). Advanced Machine Learning Approaches for Fastag Fraud Detection. In *2025 International Conference on Automation and Computation (AUTOCOM)* (pp. 149-154). IEEE.

Kumar, S. (2024, May). Advancements in meta-learning paradigms: a comprehensive exploration of techniques for few-shot learning in computer vision. In *2024 International conference on intelligent systems for cybersecurity (ISCS)* (pp. 1-8). IEEE.

Singh, M. K., & Kumar, S. (2024, April). Stress Detection During Social Interactions with Natural Language Processing and Machine Learning. In *2024 International Conference on Expert Clouds and Applications (ICOECA)* (pp. 297-301). IEEE.

Tiwari, K. K., Singh, A., & Kumar, S. (2025, February). A Comprehensive Analysis of CNN-Based Deep Learning Models: Evaluating the Impact of Transfer Learning on Model Accuracy. In *2025 2nd International Conference on Computational Intelligence, Communication Technology and Networking (CICTN)* (pp. 62-67). IEEE.

Kumar, S., Rampal, S., Gaur, M., & Gaur, M. (2024, March). Advanced ensemble learning approach for asthma prediction: Optimization and evaluation. In *2024 International Conference on Automation and Computation (AUTOCOM)* (pp. 283-288). IEEE.