Unit-I

Why Study Automata Theory?

Automata theory is the **study** of abstract machines and **automata**, as well as the computational problems that can be solved using them. **Automata theory** is closely related to formal language **theory**. An **automaton** is a finite representation of a formal language that may be an infinite set.

In mathematics, **computer science**, and linguistics, a **formal language** is a set of strings of symbols together with a set of rules that are specific to it. A **formal language** is often defined by means of a **formal** grammar such as a regular grammar or context-free grammar, also called its formation rule.

Important and Previous JNTU Questions:

1) What is Computation? What are the different models of Computation? Explain [5M, Set 1, May 2015]

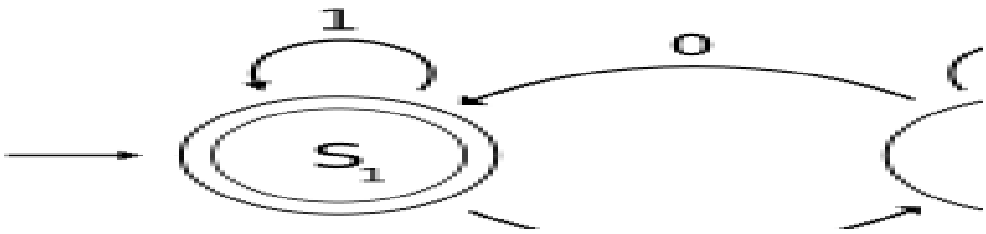
The Central Concepts of Automata Theory

Automation

An automaton is a self-operating machine, or a machine or control mechanism designed to. Not to be confused with automation as a process.

Finite Automation

A **finite automaton** (FA) is a simple idealized machine used to recognize patterns within input taken from some character set (or alphabet) C. The job of an FA is to accept or reject an input depending on whether the pattern defined by the FA occurs in the input.



An automaton with a finite number of states is called a **Finite Automaton**(FA) or **Finite State Machine** (FSM).

Formal definition of a Finite Automaton

An automaton can be represented by a 5-tuple (Q, Σ, δ, q_0, F), where –

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols, called the **alphabet** of the automaton.
- **δ** is the transition function.
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Transition Systems

A finite **automaton** can be seen as a labeled **transition system** whose configurations are its states, whose label set is the input alphabet, whose terminal configurations are its final states and whose **transition** relation corresponds to the **transition** function.

Important and Previous JNTU Questions:

1) What is finite automaton model? How it is useful for the acceptance of strings and languages explain with an example.
2) What is Finite State Machine? What are the elements of FSM?

DFA, Design of DFAs

Finite Automaton can be classified into two types –

- Deterministic Finite Automaton (DFA)
- Non-deterministic Finite Automaton (NDFA / NFA)

Deterministic Finite Automaton (DFA)

In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**. As it has a

finite number of states, the machine is called **Deterministic Finite Machine**or **Deterministic Finite Automaton.**

Formal Definition of a DFA

A DFA can be represented by a 5-tuple (Q, Σ, δ, q_0, F) where –

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols called the alphabet.
- **δ** is the transition function where δ: Q × Σ → Q
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of a DFA

A DFA is represented by digraphs called **state diagram**.

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

Example

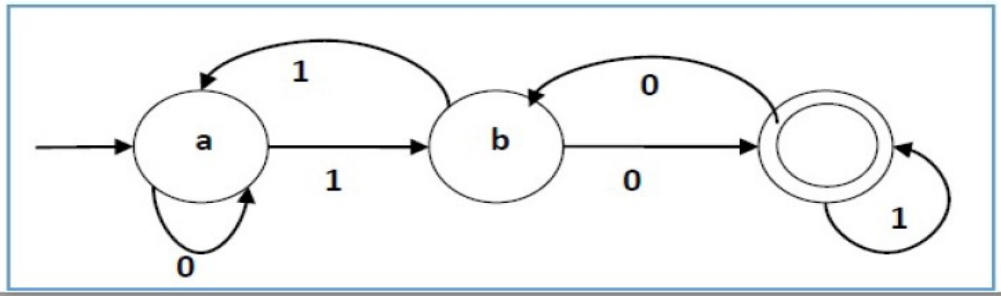
Let a deterministic finite automaton be →

- Q = {a, b, c},
- Σ = {0, 1},
- q_0 = {a},
- F = {c}, and

Transition function δ as shown by the following table –

Present State	Next State for Input 0	Next State for Input 1
a	a	b
b	c	a
c	b	c

Its graphical representation would be as follows –



Advantages and disadvantages

DFAs are one of the most practical models of computation, since there is a trivial linear time, constant-space, online algorithm to simulate a DFA on a stream of input. Also, there are efficient algorithms to find a DFA recognizing:

- The complement of the language recognized by a given DFA.
- The union/intersection of the languages recognized by two given DFAs.

Because DFAs can be reduced to a *canonical form* (minimal DFAs), there are also efficient algorithms to determine:

- whether a DFA accepts any strings
- whether a DFA accepts all strings
- whether two DFAs recognize the same language
- the DFA with a minimum number of states for a particular regular language

DFAs are equivalent in computing power to nondeterministic finite automata (NFAs). This is because, firstly any DFA is also an NFA, so an NFA can do what a DFA can do. Also, given an NFA, using the power-set construction one can build a DFA that recognizes the same language as the NFA, although the DFA could have exponentially larger number of states than the NFA.

Important and Previous JNTU Questions:

1) What is Finite State Machine? What are the advantages of FSM [4M May/June- 2015 Set 2]
2) Give the formal definition of FSM? What are the examples of FSM? [5M May/June- 2015 Set 3]

NFA, Design of NFA

In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton**. As it has finite number of states, the machine is called **Non-deterministic Finite Machine** or **Non-deterministic Finite Automaton**.

Formal Definition of an NDFA

An NDFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where –

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols called the alphabets.
- **δ** is the transition function where $\delta: Q \times \Sigma \rightarrow 2^Q$

  (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of an NDFA: (same as DFA)

An NDFA is represented by digraphs called state diagram.

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.

- The final state is indicated by double circles.

Example

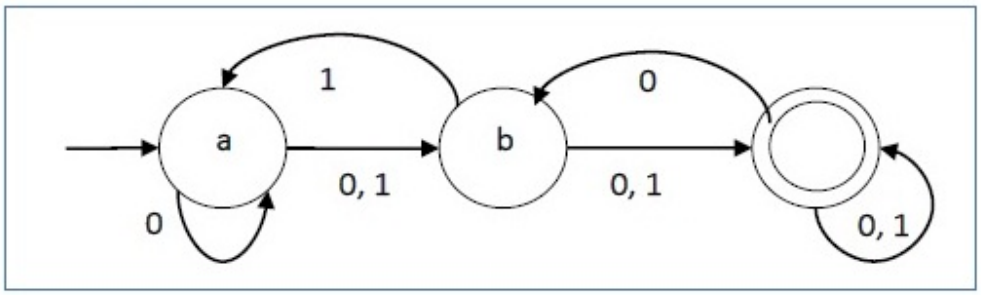
Let a non-deterministic finite automaton be →

- Q = {a, b, c}
- Σ = {0, 1}
- q_0 = {a}
- F = {c}

The transition function δ as shown below –

Present State	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c

Its graphical representation would be as follows –



DFA vs NDFA

The following table lists the differences between DFA and NDFA.

DFA	NDFA
The transition from a state is to a single particular next state for each input symbol. Hence it is	The transition from a state can be to multiple next states for each input symbol. Hence it is called *non-*

called *deterministic*.	*deterministic*.
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
Backtracking is allowed in DFA	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.

Applications of Finite Automata

Application of Finite Automata (FA):

We have several application based on finite automata and finite state machine. Some are given below;

- A finite automata is highly useful to design Lexical Analyzers.
- A finite automata is useful to design text editors.
- A finite automata is highly useful to design spell checkers.
- A finite automata is useful to design sequential circuit design (Transducer).

String matching algorithms :

String Processing

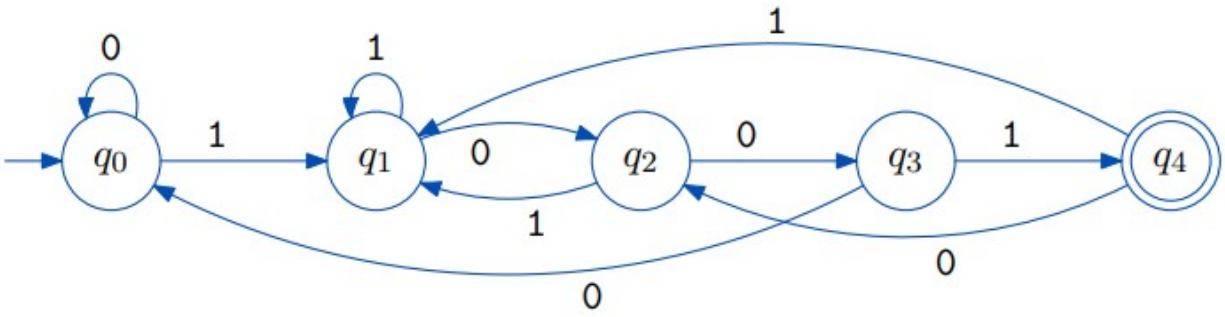
Consider finding all occurrences of a short string (pattern string) within a long string (text string).

This can be done by processing the text through a DFA: the DFA for all strings that end with the pattern string. Each time the accept state is reached, the current position in the text is out- put.

Example:

Finding 1001

To find all occurrences of pattern 1001, con- struct the DFA for all strings ending in 1001.



Limitation of Finite Automata

Examples of Limitation of finite auotmata:

There is no finite automaton that recognizes these strings:

1. The set of binary strings consisting of an equal number of a's and b's.
2. The set of strings over '(' and ')' that have "balanced" parentheses.

The 'pumping lemma' can be used to prove that no such FA exists for these examples.