Introduction:

Computer Science is a cluster of related scientific and engineering disciplines concerned with the study and application of computations. These disciplines range from the pure and basic scientific discipline concerned with the foundations (or theory) of computer science (or of computation) to engineering disciplines concerned with specific applications.

The foundations (or theory) of computer science can be partitioned into two sub-disciplines: one concerned with the Theory of Computation, and the other concerned with the Theory of Programming.

The theory of computation aims at understanding the nature of computation, and specifically the inherent possibilities and limitations of efficient computations. The Theory of Programming is concerned with the actual task of implementing computations (i.e., writing computer programs).

Strings, Alphabet, Language, Operations:

In computer science, in the area of formal language theory, frequent use is made of a variety of string functions; however, the notation used is different from that used oncomputer programming, and some commonly used functions in the theoretical realm are rarely used when programming. This article defines some of these basic terms.

Alphabets

Any finite set of symbols {0,1} is a set of binary alphabets, {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F} is a set of Hexadecimal alphabets, {a-z, A-Z} is a set of English language alphabets.

Strings

Any finite sequence of alphabets is called a string. Length of the string is the total number of occurrence of alphabets, e.g., the length of the string tutorialspoint is 14 and is denoted by |tutorialspoint| = 14. A string having no alphabets, i.e. a string of zero length is known as an empty string and is denoted by ε (epsilon).

The classical theory of computation traditionally deals with processing an input string of symbols into an output string of symbols. Note that in the special case where the set of possible output strings is just {‘yes’, ‘no’}, (often abbreviated {T, F} or {1, 0}), then we can think of the string processing as string (pattern) recognition.

We should start with a few definitions. The first step is to avoid defining the term ‘symbol’ – this leaves an open slot to connect the abstract theory to the world . . .

Special Symbols

A typical high-level language contains the following symbols:-

Arithmetic Symbols	Addition(+), Subtraction(-), Modulo(%), Multiplication(*), Division(/)
Punctuation	Comma(,), Semicolon(;), Dot(.), Arrow(->)
Assignment	=
Special Assignment	+=, /=, *=, -=
Comparison	==, !=, <, <=, >, >=
Preprocessor	#
Location Specifier	&
Logical	&, &&, |, ||, !
Shift Operator	>>, >>>, <<, <<<

Language

The language is considered as a finite set of strings over some finite set of alphabets. Computer languages are considered as finite sets, and mathematically set operations can be performed on them. Finite languages can be described by means of regular expressions.

Operations

The various operations on languages are:

- Union of two languages L and M is written as
  L U M = {s | s is in L or s is in M}
- Concatenation of two languages L and M is written as
  LM = {st | s is in L and t is in M}
- The Kleene Closure of a language L is written as
  L* = Zero or more occurrence of language L.

Notations

If r and s are regular expressions denoting the languages L(r) and L(s), then

- **Union** : (r)|(s) is a regular expression denoting L(r) U L(s)
- **Concatenation** : (r)(s) is a regular expression denoting L(r)L(s)
- **Kleene closure** : (r)* is a regular expression denoting (L(r))*
- (r) is a regular expression denoting L(r)

An alphabet is a finite set of symbols. A string over an alphabet A is a finite ordered sequence of symbols from A. Note that repetitions are allowed. The length of a string is the number of symbols in the string, with repetitions counted. (e.g., |aabbcc| = 6).

The empty string, denoted by, is the (unique) string of length zero. Note that the empty string is not the same as the empty set ∅. If S and T are sets of strings, then ST = {xy| x ∈ S and y ∈ T}.

Given an alphabet A, we define A 0 = {} A n+1 = AAn A ∗ = [∞ n=0 A n 6. A language L over an alphabet A is a subset of A∗ . That is, L ⊂ A∗ . We can define the natural numbers, N, as follows: We let 0 = ∅ 1 = {∅} 2 = {∅, {∅}} and in general n + 1 = {0, 1, 2, . . . , n}. Then N = {0, 1, 2, . . .}.

Sizes of Sets and Countability:

1. Given two sets S and T, we say that they are the same size (|S| = |T|) if there is a one-to-one onto function f : S → T.

2. We write |S| ≤ |T| if there is a one-to-one (not necessarily onto) function f : S → T.

3. We write |S| < |T| if there is a one-to-one function f : S → T, but there does not exist any such onto function.

4. We call a set S (a) Finite if |S| < |N| (b) Countable if |S| ≤ |N| (c) Countably infinite if |S| = |N| (d) Uncountable if |N| < |S|.

Finite State Machine

A finite-state machine (FSM) or finite-state automaton (plural: *automata*), or simply a state machine, is a mathematicalmodel of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of *states*. The machine is in only one state at a time; the state it is in at any given time is called the *current state*. It can change from one state to another when initiated by a triggering event or condition; this is called a *transition*.

A particular FSM is defined by a list of its states, and the triggering condition for each transition. The behaviour of state machines can be observed in many devices in modern society that perform a predetermined sequence of actions depending on a sequence of events with which they are presented.

Simple examples are vending machines, which dispense products when the proper combination of coins is deposited, elevators, which drop riders off at upper floors before

going down, traffic lights, which change sequence when cars are waiting, and combination locks, which require the input of combination numbers in the proper order.

Finite-state machines provide a simple computational model with many applications. Recall the definition of a Turing machine: a finite-state controller with a movable read/write head on an unbounded storage tape. If we restrict the head to move in only one direction, we have the general case of a finite-state machine.

The sequence of symbols being read can be thought to constitute the input, while the sequence of symbols being written could be thought to constitute the output. We can also derive output by looking at the internal state of the controller after the input has been read.

Finite-state machines, also called finite-state automata (singular: automaton) or just finite automata are much more restrictive in their capabilities than Turing machines. For example, we can show that it is not possible for a finite-state machine to determine whether the input consists of a prime number of symbols.

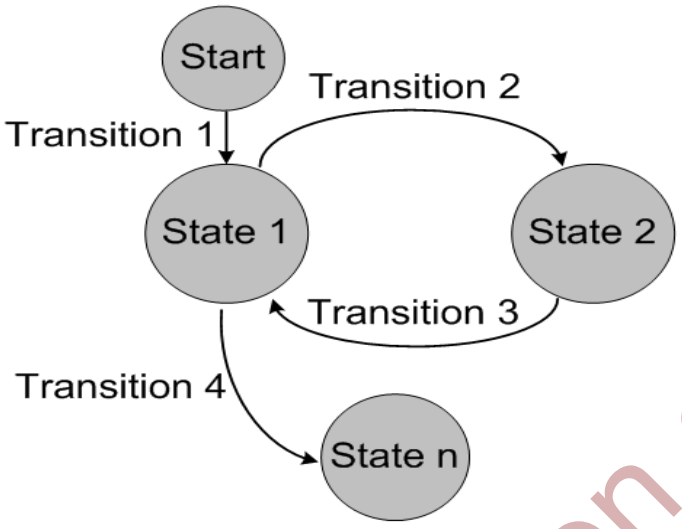
Much simpler languages, such as the sequences of well-balanced parenthesis strings, also cannot be recognized by finite-state machines. Still there are the following applications:

• Simple forms of pattern matching (precisely the patterns definable by "regular expressions", as we shall see).

• Models for sequential logic circuits, of the kind on which every present-day computer and many device controllers is based.

• An intimate relationship with directed graphs having arcs labeled with symbols from the input alphabet. Even though each of these models can be depicted in a different setting, they have a common mathematical basis. The following diagram shows the context of finite-state machines among other models we have studied or will study.

The interrelationship of various models with respect to computational or representational power. The arrows move in the direction of restricting power. The bi-directional arrows show equivalences.



Automata Theory:

Automata Theory is an exciting, theoretical branch of computer science. It established its roots during the 20th Century, as mathematicians began developing - both theoretically and literally - machines which imitated certain features of man, completing calculations more quickly and reliably. The word automaton itself, closely related to the word "automation", denotes automatic processes carrying out the production of specific processes. Simply stated, automata theory deals with the logic of computation with respect to simple machines, referred to as automata.

Through automata, computer scientists are able to understand how machines compute functions and solve problems and more importantly, what it means for a function to be defined as *computable* or for a question to be described as *decidable.*

Automatons are abstract models of machines that perform computations on an input by moving through a series of states or configurations. At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration. As a result, once the computation reaches an accepting configuration, it accepts that input. The most general and powerful automata is the Turing machine.

The major objective of automata theory is to develop methods by which computer scientists can describe and analyze the dynamic behavior of discrete systems, in which signals are sampled periodically. The behavior of these discrete systems is determined by the way that the system is constructed from storage and combinational elements. Characteristics of such machines include:

- **Inputs:** assumed to be sequences of symbols selected from a finite set *I* of input signals. Namely, set *I* is the set $\{x_1, x_{,2}, x_3 ... x_k\}$ where *k* is the number of inputs.
- **Outputs:** sequences of symbols selected from a finite set *Z*. Namely, set *Z* is the set $\{y_1, y_2, y_3 ... y_m\}$ where *m* is the number of outputs.
- **States:** finite set *Q*, whose definition depends on the type of automaton.

There are four major families of automaton**:**

- Finite-state machine
- Pushdown automata
- Linear-bounded automata
- Turing machine

The families of automata above can be interpreted in a hierarchal form, where the finite-state machine is the simplest automata and the Turing machine is the most complex. The focus of this project is on the finite-state machine and the Turing machine. A Turing machine is a finite-state machine yet the inverse is not true.

Finite State Machines

The finite-state machines, the Mealy machine and the Moore machine, are named in recognition of their work. While the Mealy machine determines its outputs through the current state and the input, the Moore machine's output is based upon the current state alone. An automaton in which the state set Q contains only a *finite* number of elements is called a Finite-State Machine (FSM).

FSMs are abstract machines, consisting of a set of states (set Q), set of input events (set I), a set of output events (set Z) and a state transition function. The state transition function takes the current state and an input event and returns the new set of output events and the next state. Therefore, it can be seen as a function which maps an ordered sequence of input events into a corresponding sequence, or set, of output events.

State transition function: $I \rightarrow Z$

Finite-state machines are ideal computation models for a small amount of memory, and do not maintain memory. This mathematical model of a machine can only reach a finite number of states and transitions between these states. Its main application is in mathematical problem analysis. Finite-machines are also used for purposes aside from general computations, such as to recognize regular languages.

In order to fully understand conceptually a finite-state machine, consider an analogy to an elevator:

An elevator is a mechanism that does not remember all previous requests for service but the current floor, the direction of motion (up or down) and the collection of not-yet satisfied requests for services. Therefore, at any given moment in time, an elevator in operated would be defined by the following mathematical terms:

- **States:** finite set of states to reflect the past history of the customers' requests.
- **Inputs:** finite set of input, depending on the number of floors the elevator is able to access. We can use the set I, whose size is the number of floors in the building.
- **Outputs:** finite set of output, depending on the need for the elevator to go up or down, according to customers' needs.

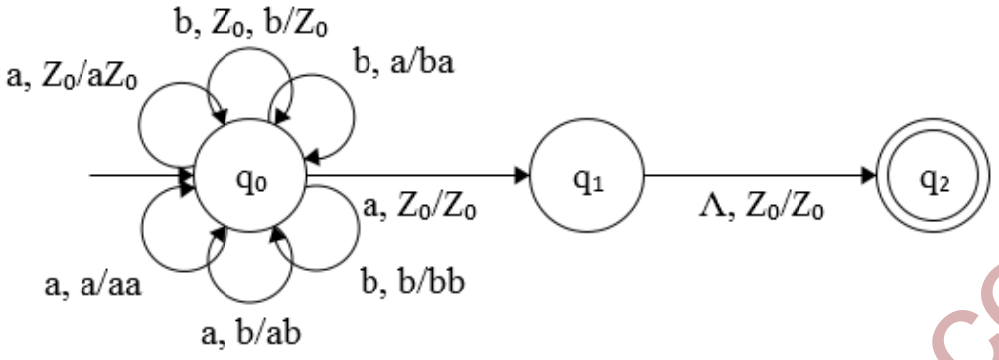
A *finite-state machine* is formally defined as a 5-tuple (Q, I, Z, ∂, W) such that:

- Q = finite set of states
- I = finite set of input symbols
- Z = finite set of output symbols
- ∂ = mapping of I x Q into Q called the state transition function, i.e. I x Q $\rightarrow$ Q
- W = mapping W of I x Q onto Z, called the output function
- A = set of accept states where F is a subset of Q

From the mathematical interpretation above, it can be said that a finite-state machine contains a finite number of states. Each state accepts a finite number of inputs, and each state has rules that describe the action of the machine for ever input, represented in the state transition mapping function. At the same time, an input may cause the machine to change states. For every input symbol, there is exactly one transition out of each state. In addition, any 5-tuple set that is accepted by nondeterministic finite automata is also accepted by deterministic finite automata.

When considering finite-state machines, it is important to keep in mind that the mechanical process inside the automata that leads to the calculation of outputs and change of states is not emphasized or delved into detail; it is instead considered a "black box", as illustrated below:

Having finite, constant amounts of memory, the internal states of an FSM carry no further structure. They can easily be represented using state diagrams, as seen below:



The state diagram illustrates the operation of an automaton. States are represented by *nodes* of graphs, transitions by the arrows or *branches*, and the corresponding inputs and outputs are denoted by symbols. The arrow entering from the left into q_0 shows that q_0 is the initial state of the machine. Moves that do not involve changes of states are indicated by arrows along the sides of individual nodes. These arrows are known as *self-loops* .

There exist several types of finite-state machines, which can be divided into three main categories:

- **acceptors**: either accept the input or do not
- **recognizers**: either recognize the input or do not
- **transducers**: generate output from given input

Applications of finite-state machines are found in a variety of subjects. They can operate on languages with a finite number of words (standard case), an infinite number of words, various types of trees, and in hardware circuits, where the input, the state and the output are bit vectors of a fixed size.

Finite State vs. Turing Machines

The simplest automata used for computation is a finite automaton. It can compute only very primitive functions; therefore, it is not an adequate computation model. In addition, a finite-state machine's inability to generalize computations hinders its power.

The following is an example to illustrate the difference between a finite-state machine and a **Turing machine:**

Imagine a Modern CPU. Every bit in a machine can only be in two states (0 or 1). Therefore, there are a finite number of possible states. In addition, when considering the parts

of a computer a CPU interacts with, there are a finite number of possible inputs from the computer's mouse, keyboard, hard disk, different slot cards, etc. As a result, one can conclude that a CPU can be modeled as a finite-state machine.

Now, consider a computer. Although every bit in a machine can only be in two different states (0 or 1), there are an infinite number of interactions within the computer as a whole. It becomes exceeding difficult to model the workings of a computer within the constraints of a finite-state machine. However, higher-level, infinite and more powerful automata would be capable of carrying out this task.

World-renowned computer scientist Alan Turing conceived the first "infinite" (or unbounded) model of computation: the Turing machine, in 1936, to solve the*Entscheindungsproblem*. The Turing machine can be thought of as a finite automaton or control unit equipped with an infinite storage (memory). Its "memory" consists of an infinite number of one-dimensional array of cells. Turing's machine is essentially an abstract model of modern-day computer execution and storage, developed in order to provide a precise mathematical definition of an algorithm or mechanical procedure.

While an automaton is called *finite* if its model consists of a finite number of states and functions with finite strings of input and output, infinite automata have an "accessory" - either a stack or a tape that can be moved to the right or left, and can meet the same demands made on a machine.

A *Turing machine* is formally defined by the set [Q, Σ, Γ, δ, q_0, B, F] where

- Q = finite set of states, of which one state q_0 is the initial state
- Σ = a subset of Γ not including B, is the set of *input symbols*
- Γ = finite set of allowable tape symbols
- δ = the *next move function* , a mapping function from Q x Γ to Q x Γ x {L,R}, where L and R denote the directions left and right respectively
- q_0 = in set Q as the *start* state
- B = a symbol of Γ, as the *blank*
- F ⊆ Q the set of *final states*

Therefore, the major difference between a Turing machine and two-way finite automata (FSM) lies in the fact that the Turing machine is capable of changing symbols on its tape and simulating computer execution and storage. For this reason, it can be said that the Turing Machine has the power to model all computations that can be calculated today through modern computers.

Regular Languages

• A Regular Language is a set of Strings.

• Two ways to describe sets of strings S – Enumerate the strings: S = {s1, s2, s3, …} – Write a predicate – p: p(x)=True if x is in the set S.

• Problems – Enumeration is hard if set is infinite – Writing predicate varies depending upon how the set S is described (RegExp, DFA, NFA, etc) Enumeration.

• Enumeration is easy to write.