**1.Types Of Automata And It’s Application**

Automata theory is the basis for the theory of **formal languages**. It is the study of abstract machines and the computation problems that can be solved using these machines. The abstract machine is called the automata.The main motivation behind developing the automata theory was to develop methods to describe and analyse the dynamic behaviour of discrete systems. A Proper treatment of formal language theory begins with some basic definitions:♣

* A symbol is simply a character, an abstraction that is meaningless by itself.♣
* An alphabet is a finite set of symbols.♣
* A word is a finite string of symbols from a given alphabet.
* Finally, a language is a set of words formed from a given alphabet

Types of Automata and its applications

**Finite Automata**

Finite automata are use to recognize patterns.An automaton with a finite number of states is called a Finite automaton. It takes the string of symbol as input and changes its state accordingly. When the desiring symbol is search, then the transition occurs. At the time of transition, the automata can either move to the next state or stay in the same state. Finite automata have two states, Accept state or Reject state. When the input string is processing successfully, and the automata reached its final state, then it will accept.

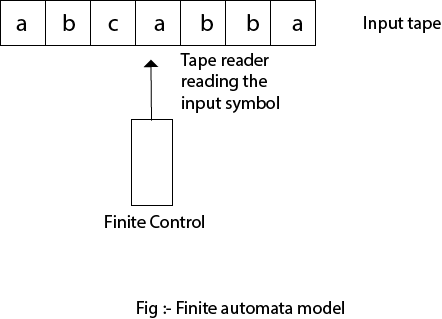
**Finite Automata Model:**

A finite automaton is a collection and machine of 5-tuple (Q, ∑, δ, q0, F), where:

1. Q: finite set of states
2. ∑: finite set of the input symbol
3. q0: initial state
4. F: **final** state
5. δ: Transition function

Note: **Input tape** is a linear tape having some number of cells. Each input symbol is place in each cell.

**Finite control:** The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one to one from left to right, and at a time only one input symbol is read.



**Types of Automata:**

There are two types of finite automata:

1. DFA(deterministic)
2. NFA(non-deterministic)

**1.DFA**

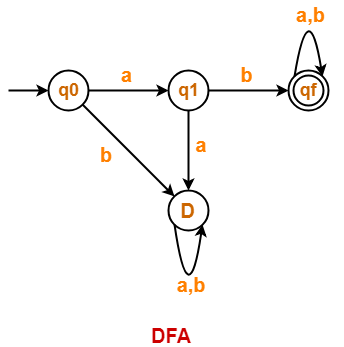


Image by Gate Vidyalaya

DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

**2. NFA**

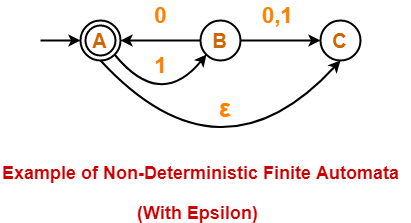


Image by Gate Vidyalaya

NFA stands for non-deterministic automata. It is used to transmit any number of states for a particular input. It can accept the null move.

**Some important points about DFA and NFA:**

\* Every DFA is NFA, but NFA is not DFA.  
\* There can be multiple final states in both NFA and DFA.  
\* DFA is used in Lexical Analysis in Compiler.  
\* NFA is more of a theoretical concept.

**Applications of Finite Automata (FA):**

* Software for designing and checking the behavior of digital circuits
* It is highly useful to design Lexical Analyzers
* This automata is highly useful to design spell checkers.
* For this automata is useful to design sequential circuit design (Transducer).
* Another application of finite automata is programming simple agents to retort to inputs and produce actions in how. you’ll be able to write a full program, but a DFA is commonly enough to try to to the duty. DFAs are easier to reason about and easier to implement.

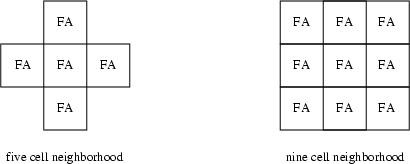
**Cellular Automata**

Cellular automata is a deterministic rewriting dynamic systems that evolves in discrete time and discrete space, the latter usually a grid. It consists of a grid of cells that are locally but synchronously updated across the grid according to a global time scale and a global recursive rule governing the evolution of the state of each cell as a function of the state of neighboring cells.

While the model of cellular automata is of the same computational power than other Turing universal models and, therefore, fundamentally equivalent as they can emulate each other.

Typically, a CA neighborhood is defined as an automaton and its immediate neighbors. The two most common neighborhood templates used for a two-dimensional lattice are:

**Figure:** Two most common 2-D CA neighborhood templates.



Since the automata are using state configurations of a given template as their input source, the set of all possible configurations for a given template is isomorphic to the set of input symbols that the automata need recognize.

To simplify construction of CA systems, the following conventions are universally employed:

1. All FA are defined to use the same set of state symbols

https://miro.medium.com/v2/resize:fit:14/0*ISYPe76D1Zd3pzOW.png

2. All FA use the same neighborhood template

https://miro.medium.com/v2/resize:fit:16/0*CyKRm9-IASeoqHC4.png

3. As a consequence of 1 and 2, all FA will be recognizing the exact same set of state configurations. Thus, it its natural to define all automata to use the same input symbol alphabet

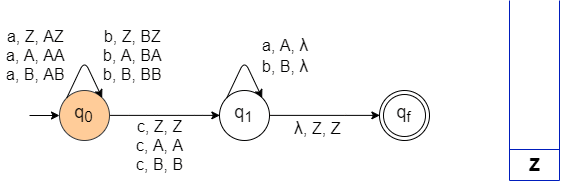
https://miro.medium.com/v2/resize:fit:14/0*79z-KBklwlWmeZhB.png

**Applications of Cellular Automata (CA)**

Let us just list some examples of physical phenomena and CA that exhibit similar behavior:

* Patterns on sea shells *Cymbiola innexa* are similar to patterns generated by rule 30 1D CA
* Growth of crystals especially patterns in snowflakes can be modeled by simple 2D CA
* Excitable media in biology (predator-prey dynamics) and chemistry (Belousov-Zhabotinsky reaction) produce spiral patterns that can be seen in Greenberg-Hastings Models
* Self-replication is seen in the *Langton loop* (which is a small component of a simplified version of the original von Neumann self-replicating machine)
* Evolution (self-replication + mutation + selection) can be seen on the Evoloop model, a modified Langton loop
* Fractal growth of biological organisms
* Models of *universal computation* include rule 110 and the Game of Life
* The *Game of Life* exhibits interesting patterns that can’t really be seen in the real world but they excite our imagination about what CA are capable of

**Pushdown Automata**



Gif by Dev Community

The formal notation for a pushdown automata (PDA) involves seven tuples. We write the specification of a PDA ‘M’ as follows;

**M = (Q, ∑, Γ, δ, q0 , z0, F)**

These tuples have the following meaning;

1. **` Q`:** A finite set of states, like the states of a finite automation.
2. **`∑`:** A finite set of input alphabet/ symbols, also analogous to the corresponding component of a finite automaton.
3. **`Γ`:** A finite stack alphabet. This tuple, which has no finite-automaton analog, it the set of symbols that we are allowed to push onto the stack.
4. **`δ`:** The transaction function. As for a finite automation, δ governs the behavior of the automaton. Formally, δ takes as argument a triple δ(q, a, X), where;
5. q is a state in Q.
6. a is either an input symbol in ∑ or a = ε, the empty string, which is assumed not to be an input symbol.
7. X is a stack symbol, that is a member of Γ.

Basically the complete transaction function of pushdown automata is as;

**δ: (Q x (Σ ∪{ε}) x X) → (p, γ)**

Here the output of δ is a finite set of pair (p, γ), where p is the new state, and γ is the string of stack symbols that replaces X at the top of the stack. For instance, if γ = ε, then the stack is popped, if γ = X, then the stack is unchanged, and if γ = YZ, then X is replaced by Z and Y is pushed onto the stack.

1. **q0:** The initial state. The PDA is in this state before making any transitions.
2. **z0:** The stack start symbol. Initially, the PDA’s stack consists of one instance of this symbol, and nothing else.
3. **F:** The set of final states or accepting states.

Pushdown Automata has a Stack mechanism to accept languages which aren’t possible in a Finite Automata. The problem comes when before a Push operation we need to check for Overflow Condition, or before a Pop Operation we check for the Underflow Condition.

We solve this problem by assuming that the Stack is infinite, while the empty condition is overcome by pre introducing an element Z0 into the stack before working on the string.

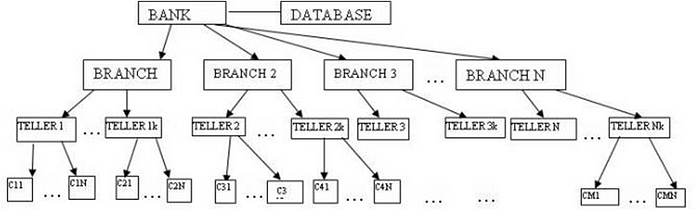
**Applications of Pushdown Automata**

* Whenever your smartphone calculator calculates an arithmetic expression, it uses an implementation of the push down automata to evaluate it. It verifies that the parentheses are balanced. In the absence of parentheses, it makes sure that the multiplication operator has more precedence than the addition or subtraction operator. All of it can be summarized in three lines of grammar:

E→E+T|E−T|TE→E+T|E−T|TT→T∗F|T/F|FT→T∗F|T/F|FF→x|y|(E)|−F

* Online Transaction Processing system

In this work an attempt is made to model the online transaction processing system of a banking organization with timed automata. In Figure the organization of the bank is depicted. It consists of multiple branches with multiple tellers at each branch.



* **Tower Of Hanoi (Recursive Solution)**

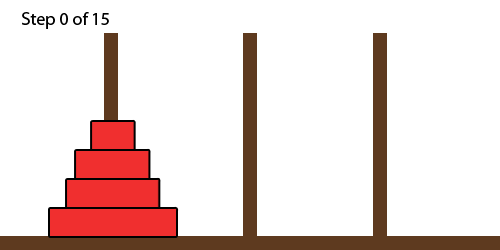


Image by Viblo

**Linear Bound Automata**

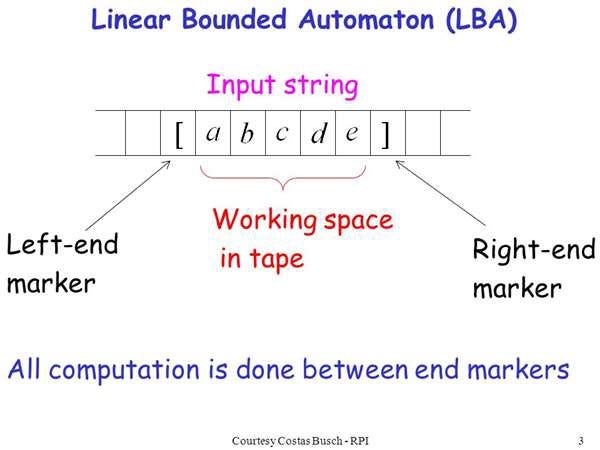
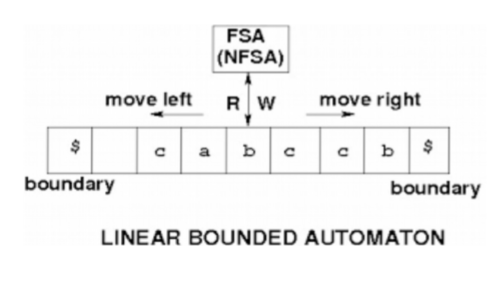


Image by Youth4work

A Turing machine that uses only the tape space occupied by the input is called a linear-bounded automaton (LBA).An LBA differs from a Turing machine in that while the tape is initially considered to have unbounded length, only a finite contiguous portion of the tape, whose length is a linear function of the length of the initial input, can be accessed by the read/write head; hence the name linear bounded automaton.



Linear bounded automata is a non-deterministic Turing machine

**M = (Q , T , E , q0 , ML , MR , S , F),** where,   
**Q** -> A finite set of transition states  
**T** -> Tape alphabet  
**E** -> Input alphabet  
**q0** -> Initial state  
**ML** -> Left bound of tape  
**MR** -> Right bound of tape  
**S** -> Transition Function  
**F** -> A finite set of final states

* There are two special tape symbols (the left end marker and right end marker).
* The TM begins in the configuration (s, , 0).
* The TM cannot replace with anything else, nor move the tape head left of .

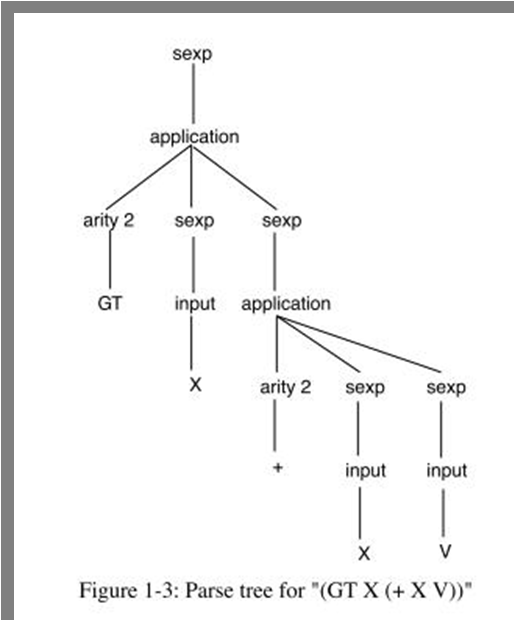
An equivalent definition of an Linear Bound Automata(LBA) is that it uses only k times the amount of space occupied by the input string, where k is a constant fixed for the particular machine. Possible to simulate k tape cells with a single tape cell, by increasing the size of the tape alphabet.

Examples: {a^n | n is a perfect square }

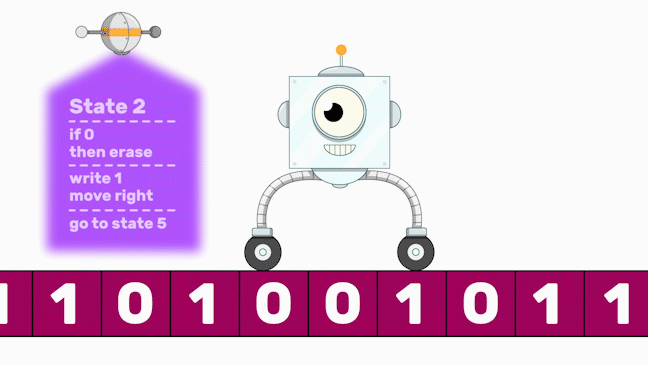
Used as a model for actual computers rather than models for the computational process.

**Applications of Linear Bound Automata**

* Genetic Programming
* For constructing syntactic parse trees for semantic analysis of the compiler



**Turing Machine**



**Turing machine**is a theoretical device that manipulates symbols on a strip of tape according to a table of rules. Despite its simplicity, a Turing machine can be adapted to simulate the logic of any computer algorithm, and is particularly useful in explaining the functions of a CPU inside a computer. It consists of an infinitely long tape, which has been divided up into cells. Each cell can contain either a 1, a 0, or an empty space. Above one cell of the tape is a **head**, which can either move left or right, and can read the symbols written in the cells. The head is also capable of erasing symbols and writing new symbols into the cells.

The Turing machine is not intended as a practical computing technology, but rather as a thought experiment representing a computing machine. Turing machines help computer scientists understand the limits of mechanical computation.

A Turing Machine can be formally described as :

**7-tuple** (Q, X, ∑, δ, q0, B, F) where −  
\* Q is a finite set of states  
\* X is the tape alphabet  
\* ∑ is the input alphabet  
\* δ is a transition function; δ : Q × X → Q × X × {Left\_shift, Right\_shift}.  
\* q0 is the initial state  
\* B is the blank symbol  
\* F is the set of final states

**2.Prove that √2. is an irrational number by contradiction method**

**Solution**

## Let √2 be a rational number then √2 = p/q squaring both the sides we get 2=p2/q2 (2p)2=q2 {equation 1} this implies that q32 is divisible by 2 and then can also be said that q is divisible by 2 hence can be written as q=2k where k is an integer squaring both sides q2 = (2k)2 from equation 1 (2k)2=(2p)2 and p2 = 2k2 hence we can say 2 is the common factor in p and q and this is a contradiction to the fact that p and q are co prime numbers hence √2 cannot be expressed as p/q hence √2 is an irrational number.

**3.To determine the properties of the relation R defined on the set A={1,2,3,4}, where R={(1,1),(2,2),(3,3),(4,4),(1,2),(1,3),(3,2)}, we will check if R is reflexive, symmetric, and transitive.**

**Step 1: Check for Reflexivity**  
A relation R is reflexive if every element in set A is related to itself. This means we need to check if the pairs (1,1),(2,2),(3,3),(4,4) are all present in R.

- The pairs in R are:  
- (1,1) - present  
- (2,2) - present  
- (3,3) - present  
- (4,4) - present

Since all these pairs are present in R, we conclude that R is reflexive.

**Step 2: Check for Symmetry**  
A relation R is symmetric if for every pair (a,b) in R, the pair (b,a) is also in R.

- Check the pairs in R:  
- (1,2) is in R but (2,1) is not.  
- (1,3) is in R but (3,1) is not.  
- (3,2) is in R but (2,3) is not.

Since there are pairs in R for which the corresponding reverse pairs are not present, we conclude that R is not symmetric.

**Step 3: Check for Transitivity**  
A relation R is transitive if whenever (a,b) and (b,c) are in R, then (a,c) must also be in R.

- Check the pairs in R:  
- From (1,2) and (2,3), we need (1,3) which is present.  
- From (1,3) and (3,2), we need (1,2) which is present.  
- From (3,2) and (2,1), we need (3,1) which is not relevant since (2,1) is not in R.

Since all necessary conditions for transitivity are satisfied, we conclude that R is transitive.

**Conclusion**  
Thus, we have shown that the relation R is:  
- Reflexive: Yes  
- Symmetric: No  
- Transitive: Yes

**4.To determine the minimum number of ordered pairs required to be added to the relation R={(1,2),(2,3)} so that it becomes an equivalence relation on the set {1,2,3}, we need to ensure that the enlarged relation satisfies the properties of reflexivity, symmetry, and transitivity.**

1. **Identify the Current Relation**:  
The current relation R is given as:  
R={(1,2),(2,3)}

2. **Check for Reflexivity**:  
An equivalence relation must be reflexive, meaning every element must relate to itself. For the set {1,2,3}, we need to add:  
(1,1),(2,2),(3,3)  
So, we add 3 pairs for reflexivity.

3. **Check for Symmetry**:  
The relation must also be symmetric. This means that if (a,b) is in R, then (b,a) must also be in R.  
- From (1,2), we need to add (2,1).  
- From (2,3), we need to add (3,2).  
  
Thus, we need to add 2 more pairs for symmetry.

4. **Check for Transitivity**:  
The relation must be transitive. This means if (a,b) and (b,c) are in R, then (a,c) must also be in R.  
- We have (1,2) and (2,3), so we need to add (1,3).  
- We also have (2,1) and (1,2), which requires adding (2,2) (already added).  
- We have (3,2) and (2,1), which requires adding (3,1).

5. **Count the Total Ordered Pairs Added**:  
Now, let's summarize the pairs added:  
- Reflexive pairs: (1,1),(2,2),(3,3) → 3 pairs  
- Symmetric pairs: (2,1),(3,2) → 2 pairs  
- Transitive pairs: (1,3),(3,1) → 2 pairs

Total pairs added:  
3+2+2=7

6. **Final Count**:  
The minimum number of ordered pairs required to be added to R to make it an equivalence relation is **7**.

**Answer:**  
The minimum number of ordered pairs required to be added in R is **7**.