Theory of automata is a theoretical branch of computer science and mathematical. 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.

> **Languages:** "A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols"
>
> **Grammars:** "A grammar can be regarded as a device that enumerates the sentences of a language" - nothing more, nothing less

An alphabet is a finite, non-empty set of symbols

We use the symbol ∑ (sigma) to denote an alphabet

String:

It is a finite collection of symbols from the alphabet.

Finite Automata

- Finite automata are used to recognize patterns.
- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, 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 processed successfully, and the automata reached its final state, then it will accept.

A finite automaton is a collection 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

Some Applications of FA

Software for designing and checking the behavior of digital circuits

Lexical analyzer of a typical compiler

Software for scanning large bodies of text (e.g., web pages) for pattern finding

Software for verifying systems of all types that have a finite number of states (e.g., stock market transaction, communication/network protocol)

On Theorems, Lemmas and Corollaries

A major result as a "**theorem**"

An intermediate result that we show to prove a larger result as a "**lemma**"

A result that follows from an already proven result as a "**corollary**"

Proving techniques

By contradiction

By induction

(3 steps) Basis, inductive hypothesis, inductive step

By contrapositive statement

DFA

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

NFA stands for non-deterministic finite 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:

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

Transition Table

The transition table is basically a tabular representation of the transition function. It takes two arguments (a state and a symbol) and returns a state (the "next state").

A transition table is represented by the following things:

- Columns correspond to input symbols.
- Rows correspond to states.
- Entries correspond to the next state.
- The start state is denoted by an arrow with no source.
- The accept state is denoted by a star.

DFA (Deterministic finite automata)

- DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time.
- In DFA, there is only one path for specific input from the current state to the next state.
- DFA does not accept the null move, i.e., the DFA cannot change state without any input character.
- DFA can contain multiple final states. It is used in Lexical Analysis in Compiler.

A DFA can be represented by digraphs called state diagram. In which:

1. The state is represented by vertices.
2. The arc labeled with an input character show the transitions.
3. The initial state is marked with an arrow.
4. The final state is denoted by a double circle.

NFA (Non-Deterministic finite automata)

- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition.

ε Transitions

NFA with ε can be converted to NFA without ε, and this NFA without ε can be converted to DFA. To do this, we will use a method, which can remove all the ε transition from given NFA. The method will be:

1. Find out all the ϵ transitions from each state from Q. That will be called as ϵ-closure{q1} where qi Q.
2. Then δ' transitions can be obtained. The δ' transitions mean a ε-closure on δ moves.
3. Repeat Step-2 for each input symbol and each state of given NFA.
4. Using the resultant states, the transition table for equivalent NFA without ε can be built.

Conversion from NFA to DFA

In this section, we will discuss the method of converting NFA to its equivalent DFA. In NFA, when a specific input is given to the current state, the machine goes to multiple states. It can have zero, one or more than one move on a given input symbol. On the other hand, in DFA, when a specific input is given to the current state, the machine goes to only one state. DFA has only one move on a given input symbol.

et, = (Q, ∑, δ, q0, F) is an FA which accepts the lan ua e ( ). There should be equi alent FA denoted by ' = (Q', ∑', q0', δ', F') such that ( ) = (').

Steps for converting NFA to DFA:

Step 1: Initially Q' =

Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

Step 3: In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.

Step 4: In DFA, the final state will be all the states which contain F(final states of NFA)

Steps for converting NFA with ε to DFA:

Step 1: We will take the ε-closure for the starting state of NFA as a starting state of DFA.

Step 2: Find the states for each input symbol that can be traversed from the present. That means the union of transition value and their closures for each state of NFA present in the current state of DFA.

Step 3: If we found a new state, take it as current state and repeat step 2.

Step 4: Repeat Step 2 and Step 3 until there is no new state present in the transition table of DFA.

Step 5: Mark the states of DFA as a final state which contains the final state of NFA.

Minimization of DFA

Minimization of DFA means reducing the number of states from given FA. Thus, we get the FSM(finite state machine) with redundant states after minimizing the FSM.

We have to follow the various steps to minimize the DFA. These are as follows:

Step 1: Remove all the states that are unreachable from the initial state via any set of the transition of DFA.

Step 2: Draw the transition table for all pair of states.

Step 3: Now split the transition table into two tables T1 and T2. T1 contains all final states, and T2 contains non-final states.

Step 4: Find similar rows from T1

find the two states which have the same value of a and b and remove one of them.

Step 5: Repeat step 3 until we find no similar rows available in the transition table T1.

Step 6: Repeat step 3 and step 4 for table T2 also.

Step 7: Now combine the reduced T1 and T2 tables. The combined transition table is the transition table of minimized DFA.

Limitations of Finite Automata:

1. FA can only count finite input.
2. There is no finite auto ma that can find and recognize set of binary string of equal Os & 1s.
3. Set of strings over "(" and ")" & have balanced parenthesis.
4. Input tape is read only and only memory it has is, state to state.

5. It can have only string pattern.
6. Head movement is in only one direction.

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).

The Myhill–Nerode theorem states that **a language is regular if and only if has a finite number of equivalence classes**, and moreover, that this number is equal to the number of states in the minimal deterministic finite automaton (DFA) recognizing.

Regular Expression

A regular expression can also be described as a sequence of pattern that defines a string

- The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.
- The languages accepted by some regular expression are referred to as Regular languages.
- A regular expression can also be described as a sequence of pattern that defines a string.
- Regular expressions are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string.
-

Operations on Regular Language

The various operations on regular language are:

Union: If L and M are two regular languages then their union L U M is also a union.

Intersection: If L and M are two regular languages then their intersection is also an intersection.

Kleen closure: If L is a regular language then its Kleen closure L1* will also be a regular language.

Conversion of RE to FA

To convert the RE to FA, we are going to use a method called the subset method. This method is used to obtain FA from the given regular expression. This method is given below:

Step 1: Design a transition diagram for given regular expression, using NFA with ε moves.

Step 2: Convert this NFA with ε to NFA without ε.

Step 3: Convert the obtained NFA to equivalent DFA.

Arden's Theorem

The Arden's Theorem is useful for checking the equivalence of two regular expressions as well as in the conversion of DFA to a regular expression.

Algebraic laws for regular expressions

- + r = r.
- .r= r. =
- .r = r. =r.
- * = and * =
- r + r = r.
- r*.r* = r*
- r.r* = r*. r = r^{+}.
- (r*)* = r*

Moore Machine

oore machine is a finite state machine in which the ne t state is decided by the current state and current input symbol. The output symbol at a i en

time depends only on the present state of the machine. oore machine can be described by 6 tuples (Q, q0, ∑, , δ,) where,

1. Q: finite set of states
2. q0: initial state of machine
3. ∑: finite set of input symbols
4. O: output alphabet
5. δ: transition function where Q ∑ Q
6. : output function where Q

Mealy Machine

A ealy machine is a machine in which output symbol depends upon the present input symbol and present state of the machine. n the ealy machine, the output is represented with each input symbol for each state separated by /. The ealy machine can be described by 6 tuples (Q, q0, ∑, , δ, ') where

1. Q: finite set of states
2. q0: initial state of machine
3. ∑: finite set of input alphabet
4. O: output alphabet
5. δ: transition function where Q ∑ Q
6. ': output function where Q ∑

Conversion from Mealy machine to Moore Machine

In Moore machine, the output is associated with every state, and in Mealy machine, the output is given along the edge with input symbol. To convert Moore machine to Mealy machine, state output symbols are distributed to input symbol paths. But while converting the Mealy machine to Moore machine, we will create a separate state for every new output symbol and according to incoming and outgoing edges are distributed.

The following steps are used for converting Mealy machine to the Moore machine:

Step 1: For each state(Qi), calculate the number of different outputs that are available in the transition table of the Mealy machine.

Step 2: Copy state Qi, if all the outputs of Qi are the same. Break qi into n states as Qin, if it has n distinct outputs where n = 0, 1, 2....

Step 3: If the output of initial state is 0, insert a new initial state at the starting which gives 1 output.

Conversion from Moore machine to Mealy Machine

In the Moore machine, the output is associated with every state, and in the mealy machine, the output is given along the edge with input symbol. The equivalence of the Moore machine and Mealy machine means both the machines generate the same output string for same input string.

We cannot directly convert Moore machine to its equivalent Mealy machine because the length of the Moore machine is one longer than the Mealy machine for the given input. To convert Moore machine to Mealy machine, state output symbols are distributed into input symbol paths.

leene's Theorem states the equi alence of the followin three statements

- A language accepted by Finite Automata can also be accepted by a Transition graph.
- A language accepted by a Transition graph can also be accepted by Regular Expression.
- A language accepted by Regular Expression can also be accepted by finite Automata.

Basic Theorems in TOC (Myhill nerode theorem)

The **Myhill Nerode** theorem is a fundamental result coming down to the theory of languages. This theory was proven by **John Myhill** and **Anil Nerode** in 1958. It is used to prove whether or not a language L is regular and it is also used for minimization of states in DFA(Deterministic Finite Automata).

The Pigeonhole Principle

If n pigeonholes are occupied by n+1 or more pigeons, then at least one pigeonhole is occupied by greater than one pigeon. Generalized pigeonhole principle is: - If n pigeonholes are occupied by kn+1 or more pigeons, where k is a positive integer, then at least one pigeonhole is occupied by k+1 or more pigeons.

Applications of Pumping Lemma

Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

- If L is regular, it satisfies Pumping Lemma.
- If L does not satisfy Pumping Lemma, it is non-regular

Context-Free Grammar (CFG)

CFG stands for context-free grammar. It is is a formal grammar which is used to generate all possible patterns of strings in a given formal language. Context-free grammar G can be defined by four tuples as:

1. G = (V, T, P, S)

Where,

G is the grammar, which consists of a set of the production rule. It is used to generate the string of a language.

T is the final set of a terminal symbol. It is denoted by lower case letters.

V is the final set of a non-terminal symbol. It is denoted by capital letters.

P is a set of production rules, which is used for replacing non-terminals symbols(on the left side of the production) in a string with other terminal or non-terminal symbols(on the right side of the production).

S is the start symbol which is used to derive the string. We can derive the string by repeatedly replacing a non-terminal by the right-hand side of the production until all non-terminal have been replaced by terminal symbols.

Derivation

Derivation is a sequence of production rules. It is used to get the input string through these production rules. During parsing, we have to take two decisions. These are as follows:

- We have to decide the non-terminal which is to be replaced.
- We have to decide the production rule by which the non-terminal will be replaced.

We have two options to decide which non-terminal to be placed with production rule.

1. Leftmost Derivation:

In the leftmost derivation, the input is scanned and replaced with the production rule from left to right. So in leftmost derivation, we read the input string from left to right.

Production rules:

1. E = E + E
2. E = E - E
3. E = a | b

2. Rightmost Derivation:

In rightmost derivation, the input is scanned and replaced with the production rule from right to left. So in rightmost derivation, we read the input string from right to left.

Example

Production rules:

1. E = E + E
2. E = E - E

3. E = a | b

Derivation Tree

Derivation tree is a graphical representation for the derivation of the given production rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from a given set of production rules. The derivation tree is also called a parse tree.

Parse tree follows the precedence of operators. The deepest sub-tree traversed first. So, the operator in the parent node has less precedence over the operator in the sub-tree.

A parse tree contains the following properties:

1. The root node is always a node indicating start symbols.
2. The derivation is read from left to right.
3. The leaf node is always terminal nodes.
4. The interior nodes are always the non-terminal nodes.

Production rules:34.5M

1. E = E + E
2. E = E * E
3. E = a | b | c

Ambiguity in Grammar

A grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string. If the grammar is not ambiguous, then it is called unambiguous.

If the grammar has ambiguity, then it is not good for compiler construction. No method can automatically detect and remove the ambiguity, but we can remove ambiguity by re-writing the whole grammar without ambiguity.

Unambiguous Grammar

A grammar can be unambiguous if the grammar does not contain ambiguity that means if it does not contain more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string.

To convert ambiguous grammar to unambiguous grammar, we will apply the following rules:

1. If the left associative operators (+, -, *, /) are used in the production rule, then apply left recursion in the production rule. Left recursion means that the leftmost symbol on the right side is the same as the non-terminal on the left side. For example,

1. a

2. If the right associative operates(^) is used in the production rule then apply right recursion in the production rule. Right recursion means that the rightmost symbol on the left side is the same as the non-terminal on the right side. For example,

1. a

Simplification of CFG

As we have seen, various languages can efficiently be represented by a context-free grammar. All the grammar are not always optimized that means the grammar may consist of some extra symbols(non-terminal). Having extra symbols, unnecessary increase the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols. The properties of reduced grammar are given below:

1. Each variable (i.e. non-terminal) and each terminal of G appears in the derivation of some word in L.
2. There should not be any production as where and are non-terminal.
3. f ϵ is not in the lan ua e then there need not to be the production ϵ.

Removal of Useless Symbols

A symbol can be useless if it does not appear on the right-hand side of the production rule and does not take part in the derivation of any string. That symbol is known as a useless symbol. Similarly, a variable can be useless if it does not take part in the derivation of any string. That variable is known as a useless variable.

Elimination of ε Production

The productions of type ϵ are called ϵ productions. These type of productions can only be removed from those grammars that do not generate ϵ.

Step 1: First find out all nullable non-terminal variable which derives ε.

Step 2: For each production A a, construct all production A , where is obtained from a by removing one or more non-terminal from step 1.

Step 3: Now combine the result of step 2 with the original production and remove ε productions.

Removing Unit Productions

The unit productions are the productions in which one non-terminal gives another non-terminal. Use the following steps to remove unit production:

Step 1: To remove , add production a to the grammar rule whenever a occurs in the grammar.

Step 2: ow delete from the grammar.

Step 3: Repeat step 1 and step 2 until all unit productions are removed.

Chomsky's Normal Form (CNF)

CNF stands for Chomsky normal form. A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

- tart symbol generating ϵ. For example, A ϵ.

- A non-terminal generating two non-terminals. For example, A .
- A non-terminal generating a terminal. For example, a.

Greibach Normal Form (GNF)

GNF stands for Greibach normal form. A CFG(context free grammar) is in GNF(Greibach normal form) if all the production rules satisfy one of the following conditions:

- A start symbol generating ϵ. For example, ϵ.
- A non-terminal generating a terminal. For example, A a.
- A non-terminal generating a terminal which is followed by any number of non-terminals. For example, aA .

Applications of Context Free Grammar (CFG) are:

- Context Free Grammars are used in Compilers (like GCC) for parsing. In this step, it takes a program (a set of strings).
- Context Free Grammars are used to define the High Level Structure of a Programming Languages.
- Every Context Free Grammars can be converted to a Parser which is a component of a Compiler that identifies the structure of a Program and converts the Program into a Tree.
- Document Type Definition in XML is a Context Free Grammars which describes the HTML tags and the rules to use the tags in a nested fashion.

Pushdown Automata(PDA)

- Pushdown automata is a way to implement a CFG in the same way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.
- Pushdown automata is simply an NFA augmented with an "external stack memory". The addition of stack is used to provide a last-in-first-out memory management capability to Pushdown automata. Pushdown automata can store an unbounded amount of information on the stack. It can access a limited amount of information on the stack. A PDA can push an element onto the top of the stack and pop off an element from the top of the stack. To read an element into the stack, the top elements must be popped off and are lost.
- A PDA is more powerful than FA. Any language which can be acceptable by FA can also be acceptable by PDA. PDA also accepts a class of language which PDA Components:
- **Input tape:** The input tape is divided in many cells or symbols. The input head is read-only and may only move from left to right, one symbol at a time.
- **Finite control:** The finite control has some pointer which points the current symbol which is to be read.
- **Stack:** The stack is a structure in which we can push and remove the items from one end only. It has an infinite size. In PDA, the stack is used to store the items temporarily.
- Formal definition of PDA:
- The PDA can be defined as a collection of 7 components:
- **Q:** the finite set of states
- **∑:** the input set
- **Γ:** a stack symbol which can be pushed and popped from the stack
- **q0:** the initial state
- **Z:** a start symbol which is in Γ.
- **F:** a set of final states
- **δ:** mapping function which is used for moving from current state to next state.
- even cannot be accepted by FA. Thus PDA is much more superior to FA.

Non-deterministic Pushdown Automata

The non-deterministic pushdown automata is very much similar to NFA. We will discuss some CFGs which accepts NPDA.

The CFG which accepts deterministic PDA accepts non-deterministic PDAs as well. Similarly, there are some CFGs which can be accepted only by NPDA and not by DPDA. Thus NPDA is more powerful than DPDA.

Turing Machine

Turing machine was invented in 1936 by **Alan Turing**. It is an accepting device which accepts Recursive Enumerable Language generated by type 0 grammar.

There are various features of the Turing machine:

1. It has an external memory which remembers arbitrary long sequence of input.
2. It has unlimited memory capability.
3. The model has a facility by which the input at left or right on the tape can be read easily.
4. The machine can produce a certain output based on its input. Sometimes it may be required that the same input has to be used to generate the output. So in this machine, the distinction between input and output has been removed. Thus a common set of alphabets can be used for the Turing machine.

Formal definition of Turing machine

A Turing machine can be defined as a collection of 7 components:

Q: the finite set of states

∑: the finite set of input symbols

T: the tape symbol

q0: the initial state

F: a set of final states

B: a blank symbol used as a end marker for input

δ: a transition or mapping function.

Language accepted by Turing machine

The turing machine accepts all the language even though they are recursively enumerable. Recursive means repeating the same set of rules for any number of times and enumerable means a list of elements. The TM also accepts the computable functions, such as addition, multiplication, subtraction, division, power function, and many more.

Introduction to Undecidability

In the theory of computation, we often come across such problems that are answered either 'yes' or 'no'. The class of problems which can be answered as 'yes' are called solvable or decidable. Otherwise, the class of problems is said to be unsolvable or undecidable.

Undecidability of Universal Languages:

The universal language L_u is a recursively enumerable language and we have to prove that it is undecidable (non-recursive).