Theory of Automata

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 analyze the dynamic behavior of discrete systems.

This automaton consists of states and transitions. The **State** is represented by **circles**, and the **Transitions** are represented by **arrows**.

Automata is the kind of machine which takes some string as input and this input goes through a finite number of states and may enter in the final state.

There are the basic terminologies that are important and frequently used in automata:

### Symbols:

Symbols are an entity or individual objects, which can be any letter, alphabet or any picture.

### Example:

1, a, b, #

### Alphabets:

Alphabets are a finite set of symbols. It is denoted by ∑.

### Examples:

∑ = {a, b}

∑ = {A, B, C, D}

∑ = {0, 1, 2}

**String:**

It is a finite collection of symbols from the alphabet. The string is denoted by w.

### Example 1:

If ∑ = {a, b}, various string that can be generated from ∑ are {ab, aa, aaa, bb, bbb, ba, aba.....}.

* A string with zero occurrences of symbols is known as an empty string. It is represented by ε.
* The number of symbols in a string w is called the length of a string. It is denoted by |w|.

### Example:

w = 010

Number of Sting |w| = 3

### Language:

A language is a collection of appropriate string. A language which is formed over Σ can be **Finite** or **Infinite**.

### Example: 1

L1 = {Set of string of length 2}

= {aa, bb, ba, bb} **Finite Language**

### Example: 2

L2 = {Set of all strings starts with 'a'}

= {a, aa, aaa, abb, abbb, ababb} **Infinite Language**

**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.

## Formal Definition of FA

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

Q: finite set of states

∑: finite set of the input symbol

q0: initial state

F: **final** state

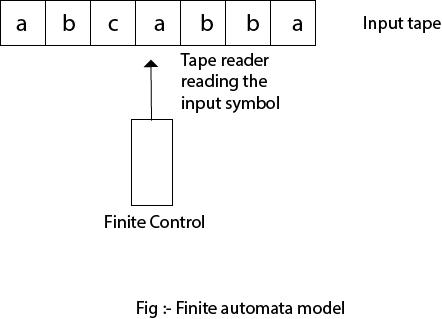
δ: Transition function

## Finite Automata Model:

Finite automata can be represented by input tape and finite control.

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

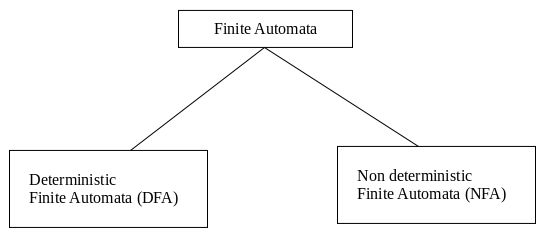
**Finite control:** The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right, and at a time only one input symbol is read.



## Types of Automata:

There are two types of finite automata:

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



**1. 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:**

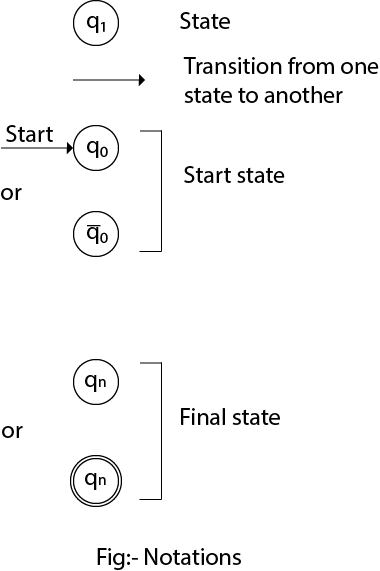
1. Every DFA is NFA, but NFA is not DFA.
2. There can be multiple final states in both NFA and DFA.
3. DFA is used in Lexical Analysis in Compiler.
4. NFA is more of a theoretical concept.

**Transition Diagram**

A transition diagram or state transition diagram is a directed graph which can be constructed as follows:

* There is a node for each state in Q, which is represented by the circle.
* There is a directed edge from node q to node p labeled a if δ(q, a) = p.
* In the start state, there is an arrow with no source.
* Accepting states or final states are indicating by a double circle.

Some Notations that are used in the transition diagram:



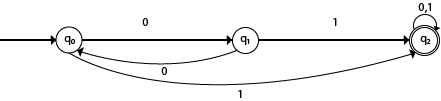
**Transition Table**

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.

Example



|  |  |  |
| --- | --- | --- |
| **Present State** | **Next state for Input 0** | **Next State of Input 1** |
| →q0 | q1 | q2 |
| q1 | q0 | q2 |
| \*q2 | q2 | q2 |

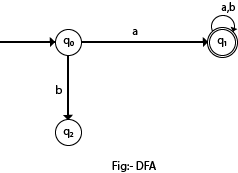
**Explanation:**

* In the above table, the first column indicates all the current states. Under column 0 and 1, the next states are shown.
* The first row of the transition table can be read as, when the current state is q0, on input 0 the next state will be q1 and on input 1 the next state will be q2.
* In the second row, when the current state is q1, on input 0, the next state will be q0, and on 1 input the next state will be q2.
* In the third row, when the current state is q2 on input 0, the next state will be q2, and on 1 input the next state will be q2.
* The arrow marked to q0 indicates that it is a start state and circle marked to q2 indicates that it is a final state.

**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.

In the following diagram, we can see that from state q0 for input a, there is only one path which is going to q1. Similarly, from q0, there is only one path for input b going to q2.



## Formal Definition of DFA

A DFA is a collection of 5-tuples same as we described in the definition of FA.

Q: finite set of states

∑: finite set of the input symbol

q0: initial state

F: **final** state

δ: Transition function

Transition function can be defined as:

δ: Q x ∑→Q

### Example 1:

Q = {q0, q1, q2}

∑ = {0, 1}

q0 = {q0}

F = {q2}

**Solution:**

Transition Diagram:

Deterministic finite automata

**Transition Table:**

|  |  |  |
| --- | --- | --- |
| **Present State** | **Next state for Input 0** | **Next State of Input 1** |
| →q0 | q0 | q1 |
| q1 | q2 | q1 |
| \*q2 | q2 | q2 |

**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.

## Formal definition of NFA:

NFA also has five states same as DFA, but with different transition function, as shown follows:

δ: Q x ∑ →2Q

where,

Q: finite set of states

∑: finite set of the input symbol

q0: initial state

F: **final** state

δ: Transition function

### Example 1:

Q = {q0, q1, q2}

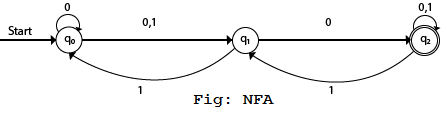
∑ = {0, 1}

q0 = {q0}

F = {q2}

**Solution:**

Transition diagram:

  
Transition Table:

|  |  |  |
| --- | --- | --- |
| **Present State** | **Next state for Input 0** | **Next State of Input 1** |
| →q0 | q0, q1 | q1 |
| q1 | q2 | q0 |
| \*q2 | q2 | q1, q2 |

**Eliminating 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.

*Example: Discussed in the class*

**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 such that:

δ (q, a) = p

δ (r, a) = p

That means, 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.

*Example: Discussed in the class*

# Mealy Machine

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

Q: finite set of states

q0: initial state of machine

∑: finite set of input alphabet

O: output alphabet

δ: transition function where Q × ∑ → Q

λ': output function where Q × ∑ →O

*Example: Discussed in the class*

**Moore Machine**

Moore machine is a finite state machine in which the next state is decided by the current state and current input symbol. The output symbol at a given time depends only on the present state of the machine. Moore machine can be described by 6 tuples (Q, q0, ∑, O, δ, λ) where,

Q: finite set of states

q0: initial state of machine

∑: finite set of input symbols

O: output alphabet

δ: transition function where Q × ∑ → Q

λ: output function where Q → O

*Example: Discussed in the class*

**Regular Expression**

* 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.

**For instance:**

In a regular expression, x\* means zero or more occurrence of x. It can generate {e, x, xx, xxx, xxxx, .....}

In a regular expression, x+ means one or more occurrence of x. It can generate {x, xx, xxx, xxxx, .....}

## 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.

 L U M = {s | s is in L or s is in M}

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

 L ⋂ M = {st | s is in L and t is in M}

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

 L\* = Zero or more occurrence of language L.

### Example 1:

Write the regular expression for the language accepting all combinations of a's, over the set ∑ = {a}

**Solution:**

All combinations of a's means a may be zero, single, double and so on. If a is appearing zero times, that means a null string. That is we expect the set of {ε, a, aa, aaa, ....}. So we give a regular expression for this as:

R = a\*

That is Kleen closure of a.

### Example 2:

Write the regular expression for the language accepting all combinations of a's except the null string, over the set ∑ = {a}

**Solution:**

The regular expression has to be built for the language

L = {a, aa, aaa, ....}

This set indicates that there is no null string. So we can denote regular expression as:

R = a+

### Example 3:

Write the regular expression for the language accepting all the string containing any number of a's and b's.

**Solution:**

The regular expression will be:

r.e. = (a + b)\*

This will give the set as L = {ε, a, aa, b, bb, ab, ba, aba, bab, .....}, any combination of a and b.

The (a + b)\* shows any combination with a and b even a null string.

***More Examples were discussed in the classroom***

**Regular Sets and Grammar**

Any set that represents the value of the Regular Expression is called a **Regular Set.**

### Closure Properties of Regular Sets

**Property 1**.  *The union of two regular set is regular.*

**Proof** −

Let us take two regular expressions

RE1 = a(aa)\* and RE2 = (aa)\*

So, L1 = {a, aaa, aaaaa,.....} (Strings of odd length excluding Null)

and L2 ={ ε, aa, aaaa, aaaaaa,.......} (Strings of even length including Null)

L1 ∪ L2 = { ε, a, aa, aaa, aaaa, aaaaa, aaaaaa,.......}

(Strings of all possible lengths including Null)

RE (L1 ∪ L2) = a\* (which is a regular expression itself)

**Hence, proved.**

**Property 2.** *The intersection of two regular set is regular.*

**Proof** −

Let us take two regular expressions

RE1 = a(a\*) and RE2 = (aa)\*

So, L1 = { a,aa, aaa, aaaa, ....} (Strings of all possible lengths excluding Null)

L2 = { ε, aa, aaaa, aaaaaa,.......} (Strings of even length including Null)

L1 ∩ L2 = { aa, aaaa, aaaaaa,.......} (Strings of even length excluding Null)

RE (L1 ∩ L2) = aa(aa)\* which is a regular expression itself.

**Hence, proved.**

**Property 3.** *The complement of a regular set is regular.*

**Proof** −

Let us take a regular expression −

RE = (aa)\*

So, L = {ε, aa, aaaa, aaaaaa, .......} (Strings of even length including Null)

Complement of **L** is all the strings that is not in **L**.

So, L’ = {a, aaa, aaaaa, .....} (Strings of odd length excluding Null)

RE (L’) = a(aa)\* which is a regular expression itself.

**Hence, proved.**

**Construction of an FA from an RE**

We can use Thompson's Construction to find out a Finite Automaton from a Regular Expression. We will reduce the regular expression into smallest regular expressions and converting these to NFA and finally to DFA.

Some basic RA expressions are the following −

**Thompson’s Method**

* For ɛ

q0

ɛ

q

1

* For a ∈ ∑

q0

a

q1

* For a+b



q1

a

q2

ɛ

q0

ɛ

ɛ

q3

b

q5

q4

ɛ

* For ab

q0

a

q1

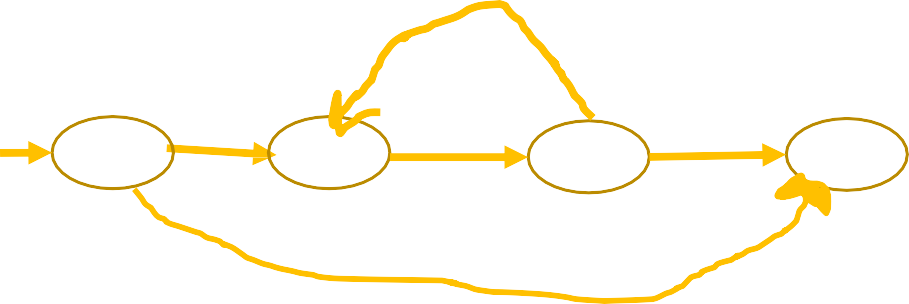
b

q

2

* For a\*

ɛ



q0

ɛ

q1

a

q

ɛ

2

q

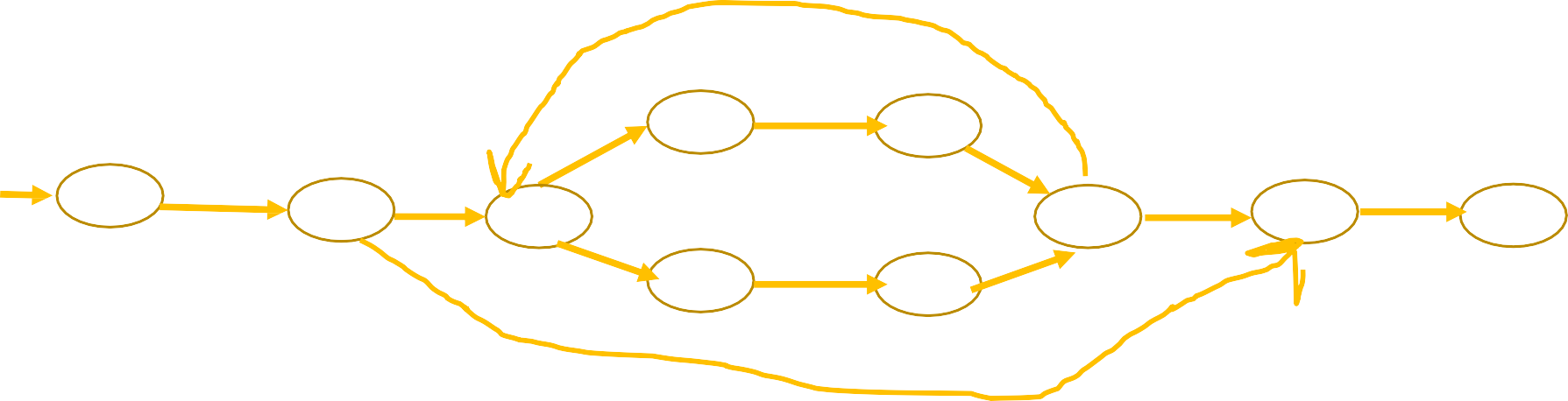
3

ɛ

Example1

Construct a FA equivalent to RE a(a+b)\*bb

ɛ



q

a

q1

a

q2

s

q6

ɛ

q0

ɛ

ɛ

q5

ɛ

q7

b

q7

b

q7

ɛ

q3

b

q4

ɛ

ɛ

**NFA Without ɛ move**

q0

a

q1

b

b

q

2

q3

a,b

\*\* More examples discussed in the class

**Algebraic properties of regular expressions :**

**1. Commutative Law for Union**

L+M=M+L

2. Associative Law for Union

(L+M)+N=L+(M+N)

3. Associative Law for Concatenation

L.(M.N)=(L.M).N

.

4. Distributive Law

L.(M+N)=L.M+L.N (left distributive)

(M+N).L=M.L+N.L (right distributive)

5. Identity and Annihilator

* ∅ **+ L = L +** ∅ **= L**  ∅ Is the identity for **union**
* **ε.L=L. ε =L**  **ε** is the identity for **concatenation**
* ∅**.L = L .**∅ **=** ∅  ∅ Is the annihilator for **concatenation**

6. Idempotent Law

L+L=L

7. DeMorgan Type Law for Regular Expressions

(L + B)\* = (L\*B\*)\*

8. Laws involving Closure

**( L\*)\* = L\***

∅ **\*= ε**

**ε\*=ε**

L+ = L .L\* = L\*.L

**L\* = L + +ε**

**9. Identities for regular expression –**  
There are many identities for the regular expression. Let p, q and r are regular expressions.

* **∅ + r = r**
* **∅.r= r.∅ = ∅**
* **∈.r = r.∈ =r**
* **∈\* = ∈ and ∅\* = ∈**
* **r + r = r**
* **r\*.r\* = r\***
* **r.r\* = r\*.r = r+.**
* **(r\*)\*  =  r\***
* **∈ +r.r\* = r\* = ∈ + r.r\***
* **(p.q)\*.p = p.(q.p)\***
* **(p + q)\* = (p\*.q\*)\* = (p\* + q\*)\***
* **(p+ q).r= p.r+ q.r and r.(p+q) = r.p + r.q**