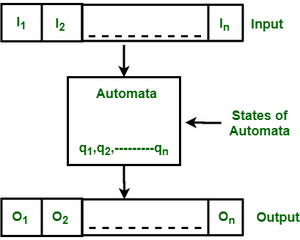
Finite Automata

Finite Automata (FA) is the simplest machine to recognize patterns. The finite automata or finite state machine is an abstract machine which have five elements or tuple. It has a set of states and rules for moving from one state to another but it depends upon the applied input symbol. Basically, it is an abstract model of digital computer. Following figure shows some essential features of a general automation.



***Figure:****Features of Finite Automata*

The above figure shows following features of automata:

1. Input
2. Output
3. States of automata
4. State relation
5. Output relation

A Finite Automata consists of the following: 

Q: Finite set of states.

Σ: set of Input Symbols.

q: Initial state.

F: set of Final States.

δ: Transition Function.

Formal specification of machine is   
{ Q, Σ, q, F, δ}.

FA is characterized into two types:

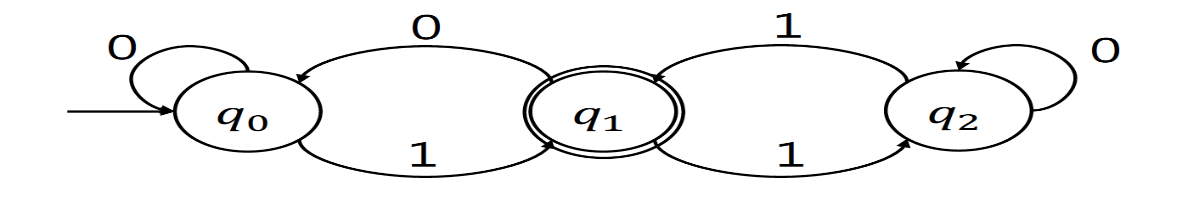


Fig: 1

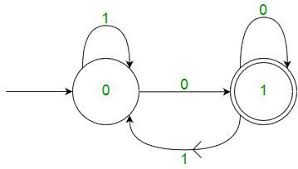


Fig: 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Finite Automata (Alt. Definition)

* 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

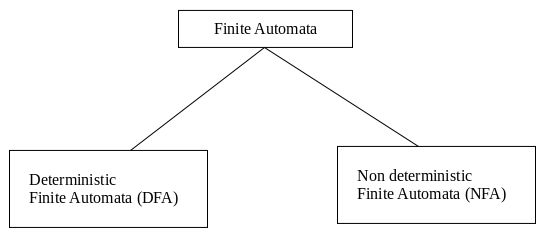
A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

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

Types of Automata:

There are two types of finite automata:

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



**Deterministic Finite Automata (DFA)**

DFA consists of 5 tuples {Q, Σ, q, F, δ}.

Q: set of all states.

Σ: set of input symbols. (Symbols which machine takes as input)

q: Initial state. (Starting state of a machine)

F: set of final state.

δ: Transition Function, defined as δ: Q X Σ --> Q.

{

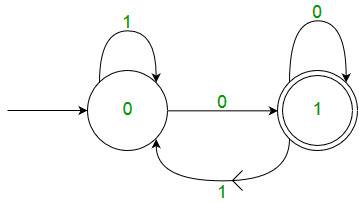
A deterministic finite automaton *M* is a 5-[tuple](https://en.wikipedia.org/wiki/N-tuple), (*Q*, Σ, *δ*, *q*0, *F*), consisting of

* a finite set of [states](https://en.wikipedia.org/wiki/State_(computer_science)) *Q*
* a finite set of input symbols called the [alphabet](https://en.wikipedia.org/wiki/Alphabet_(computer_science)) Σ
* a transition [function](https://en.wikipedia.org/wiki/Function_(mathematics)) *δ* : *Q* × Σ → *Q*
* an initial or [start state](https://en.wikipedia.org/wiki/Finite-state_machine#Start_state) {\displaystyle q\_{0}\in Q}
* a set of [accept states](https://en.wikipedia.org/wiki/Finite-state_machine#Accept_.28or_final.29_states) {\displaystyle F\subseteq Q}

}

In a DFA, for a particular input character, the machine goes to one state only. A transition function is defined on every state for every input symbol. Also, in DFA null (or ε) move is not allowed, i.e., DFA cannot change state without any input character.

1. For example, below DFA with Σ = {0, 1} accepts all strings ending with 0.



***Figure:****DFA with  Σ = {0, 1}*

One important thing to note is, ***there can be many possible DFAs for a pattern***. A DFA with minimum number of states is generally preferred.

**Nondeterministic Finite Automata (NFA)**

NFA is similar to DFA except following additional features:   
1. Null (or ε) move is allowed i.e., it can move forward without reading symbols.

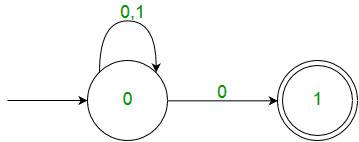
2. Ability to transmit to any number of states for a particular input.   
However, these above features don’t add any power to NFA. If we compare both in terms of power, both are equivalent.

Due to above additional features, NFA has a different transition function, rest is same as DFA.

δ: Transition Function

δ:  Q X (Σ U ε) --> 2 ^ Q.

For example, below is a NFA for above problem 



***NFA***

One important thing to note is, ***in NFA, if any path for an input string leads to a final state, then the input string accepted***. For example, in above NFA, there are multiple paths for input string “00”. Since, one of the paths leads to a final state, “00” is accepted by above NFA.

**Justification:**

Since all the tuples in DFA and NFA are the same except for one of the tuples, which is Transition Function (δ)

In case of DFA

δ: Q X Σ --> Q

In case of NFA

δ: Q X Σ --> 2Q

Now if you observe you’ll find out Q X Σ –> Q is part of Q X Σ –> 2Q.

In the RHS side, Q is the subset of 2Q which indicates Q is contained in 2Q or Q is a part of 2Q, however, the reverse isn’t true. So mathematically, we can conclude that **every DFA is NFA but not vice-versa**. Yet there is a way to convert an NFA to DFA, so **there exists an equivalent DFA for every NFA**. 

1. Both NFA and DFA have same power and each NFA can be translated into a DFA.   
2. There can be multiple final states in both DFA and NFA.   
3. NFA is more of a theoretical concept.   
4. DFA is used in Lexical Analysis in Compiler.

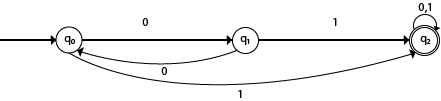
# Transition Table

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 1:**



**Solution:**

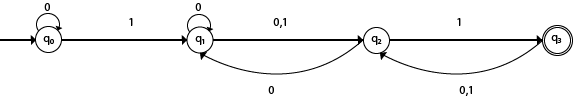
Transition table of given DFA is as follows:

|  |  |  |
| --- | --- | --- |
| **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.

### **Example 2:**



**Solution:**

Transition table of given NFA is as follows:

|  |  |  |
| --- | --- | --- |
| **Present State** | **Next state for Input 0** | **Next State of Input 1** |
| →q0 | q0 | q1 |
| q1 | q1, q2 | q2 |
| q2 | q1 | q3 |
| \*q3 | q2 | q2 |

**Explanation:**

* 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 q0 and on input 1 the next state will be q1.
* In the second row, when the current state is q1, on input 0 the next state will be either q1 or q2, 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 q1, and on 1 input the next state will be q3.
* In the fourth row, when the current state is q3 on input 0, the next state will be q2, and on 1 input the next state will be q2.