INTRODUCTION TO FORMAL LANGUAGES AND AUTOMATA THEORY

Lecture Notes_1

BASICS OF AUTOMATA THEORY

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. An automaton with a finite number of states is called a Finite automaton.

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.

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 analyse the dynamic behaviour of discrete systems.

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

1. ∑ = {a, b}
2.
3. ∑ = {A, B, C, D}
4.
5. ∑ = {0, 1, 2}

6.
7. ∑ = {0, 1, ….., 5]
8.
9. ∑ = {#, β, Δ}

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 2:

1. w = 010
2.
3. 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:

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

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:

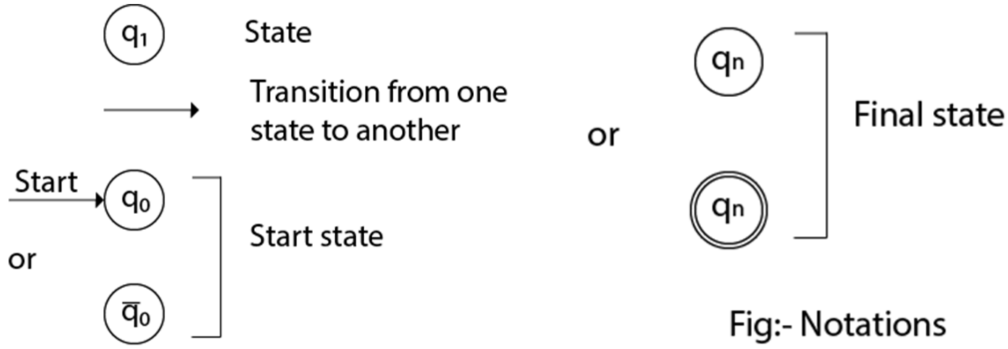


Fig:- Notations

There is a description of how a DFA operates:

1. In DFA, the input to the automata can be any string. Now, put a pointer to the start state q and read the input string w from left to right and move the pointer according to the transition function, δ. We can read one symbol at a time. If the next symbol of string w is a and the pointer is on state p, move the pointer to δ(p, a). When the end of the input string w is encountered, then the pointer is on some state F.

2. The string w is said to be accepted by the DFA if r ∈ F that means the input string w is processed successfully and the automata reached its final state. The string is said to be rejected by DFA if r ∉ F.

Example 1:

DFA with ∑ = {0, 1} accepts all strings starting with 1.

Solution:

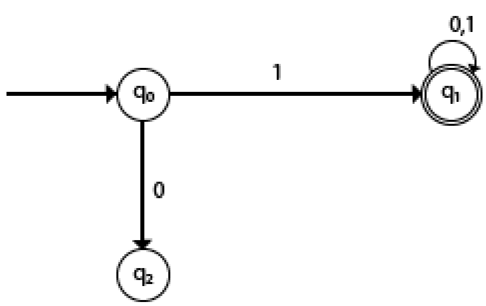


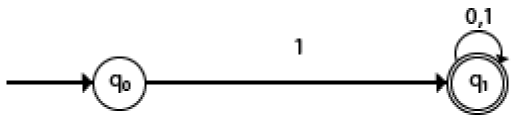
Fig: Transition diagram

The finite automata can be represented using a transition graph. In the above diagram, the machine initially is in start state q0 then on receiving input 1 the machine changes its state to q1. From q0 on receiving 0, the machine changes its state to q2, which is the dead state. From q1 on receiving input 0, 1 the machine changes its state to q1, which is the final state. The possible input strings that can be generated are 10, 11, 110, 101, 111......., that means all string starts with 1.

Example 2:

NFA with ∑ = {0, 1} accepts all strings starting with 1.

Solution:



The NFA can be represented using a transition graph. In the above diagram, the machine initially is in start state q0 then on receiving input 1 the machine changes its state to q1. From q1 on receiving input 0, 1 the machine changes its state to q1. The possible input string that can be generated is 10, 11, 110, 101, 111......, that means all string starts with 1.

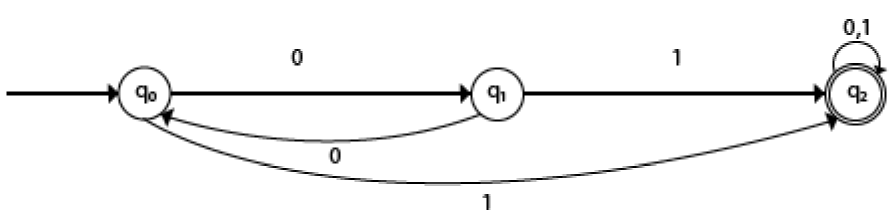
Transition Table

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:

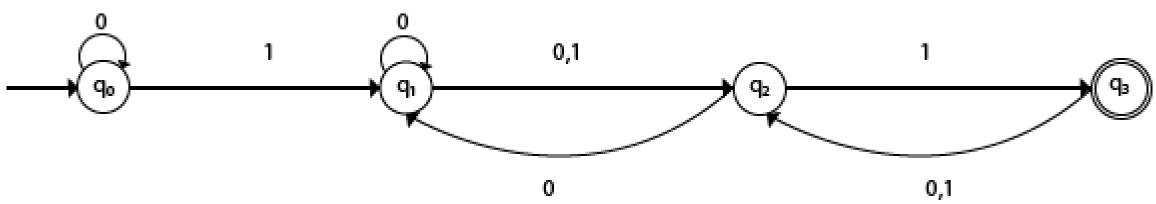
Present State	Next state for Input 0	Next State of Input 1
→q0	q1	q2
q1	q0	q2

*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.

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.

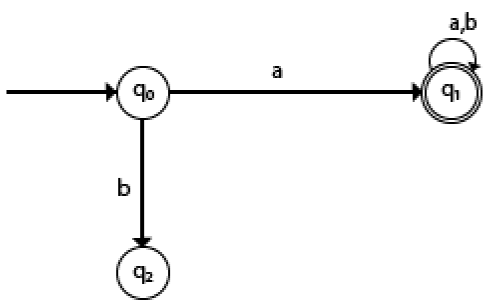


Fig:- DFA

Formal Definition of DFA

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

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

Transition function can be defined as:

1. δ: Q x ∑→Q

Graphical Representation of DFA

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

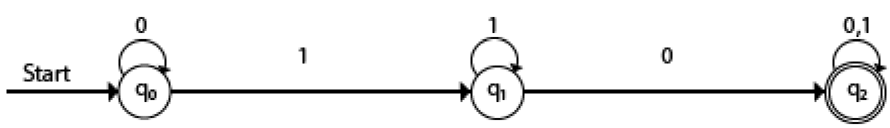
1. The state is represented by vertices.
2. The arc labeled with an input character show the transitions.
3. The initial state is marked with an arrow.
4. The final state is denoted by a double circle.

Example 1:

1. Q = {q0, q1, q2}
2. ∑ = {0, 1}
3. q0 = {q0}
4. F = {q2}

Solution:

Transition Diagram:



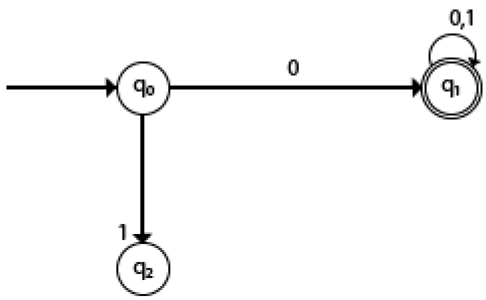
Transition Table:

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

Example 2:

DFA with ∑ = {0, 1} accepts all starting with 0.

Solution:



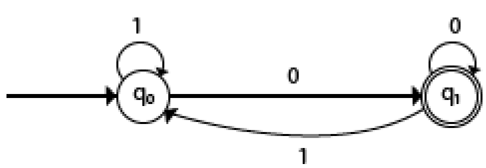
Explanation:

- In the above diagram, we can see that on given 0 as input to DFA in state q0 the DFA changes state to q1 and always go to final state q1 on starting input 0. It can accept 00, 01, 000, 001....etc. It can't accept any string which starts with 1, because it will never go to final state on a string starting with 1.

Example 3:

DFA with ∑ = {0, 1} accepts all ending with 0.

Solution:



Explanation:

In the above diagram, we can see that on given 0 as input to DFA in state q0, the DFA changes state to q1. It can accept any string which ends with 0 like 00, 10, 110, 100....etc. It can't accept any string which ends with 1, because it will never go to the final state q1 on 1 input, so the string ending with 1, will not be accepted or will be rejected.

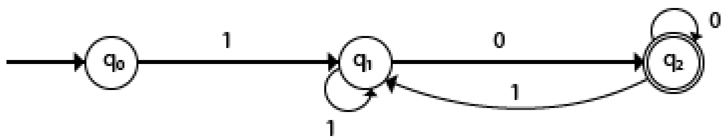
Examples of DFA

Example 1:

Design a FA with ∑ = {0, 1} accepts those string which starts with 1 and ends with 0.

Solution:

The FA will have a start state q0 from which only the edge with input 1 will go to the next state.



In state q1, if we read 1, we will be in state q1, but if we read 0 at state q1, we will reach to state q2 which is the final state. In state q2, if we read either 0 or 1, we will go to q2 state or q1 state respectively. Note that if the input ends with 0, it will be in the final state.

Example 2:

Design a FA with ∑ = {0, 1} accepts the only input 101.

Solution:

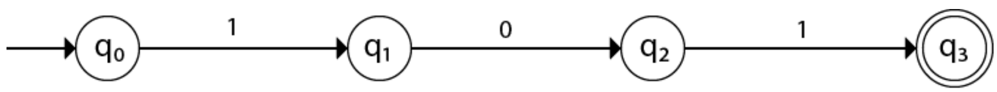


Fig: FA

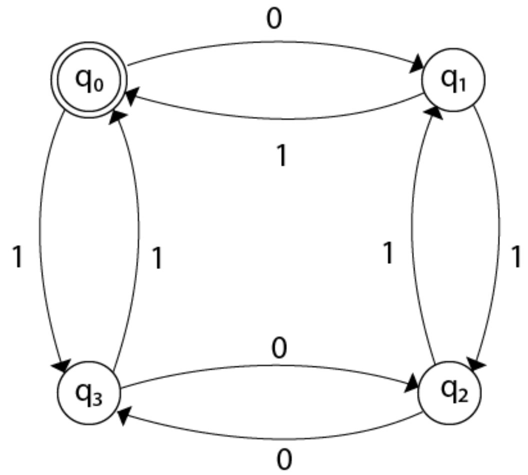
In the given solution, we can see that only input 101 will be accepted. Hence, for input 101, there is no other path shown for other input.

Example 3:

Design FA with ∑ = {0, 1} accepts even number of 0's and even number of 1's.

Solution:

This FA will consider four different stages for input 0 and input 1. The stages could be:



Here q0 is a start state and the final state also. Note carefully that a symmetry of 0's and 1's is maintained. We can associate meanings to each state as:

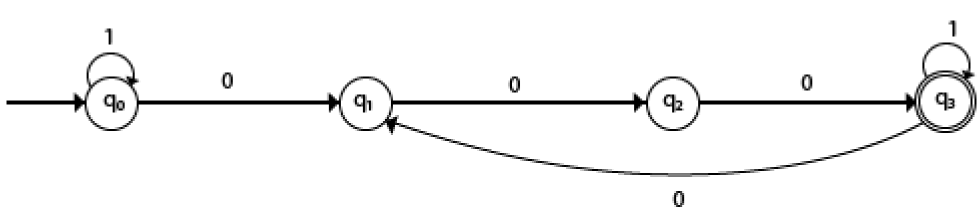
q0: state of even number of 0's and even number of 1's.
q1: state of odd number of 0's and even number of 1's.
q2: state of odd number of 0's and odd number of 1's.
q3: state of even number of 0's and odd number of 1's.

Example 4:

Design FA with ∑ = {0, 1} accepts the set of all strings with three consecutive 0's.

Solution:

The strings that will be generated for this particular languages are 000, 0001, 1000, 10001, ... in which 0 always appears in a clump of 3. The transition graph is as follows:



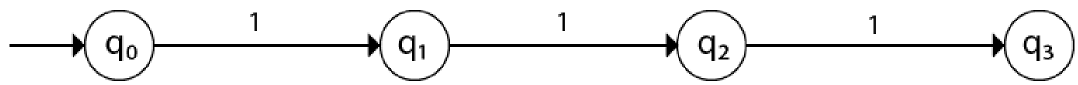
Note that the sequence of triple zeros is maintained to reach the final state.

Example 5:

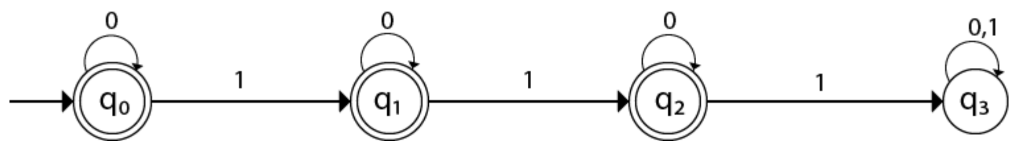
Design a DFA L(M) = {w | w ε {0, 1}*} and W is a string that does not contain consecutive 1's.

Solution:

When three consecutive 1's occur the DFA will be:



Here two consecutive 1's or single 1 is acceptable, hence



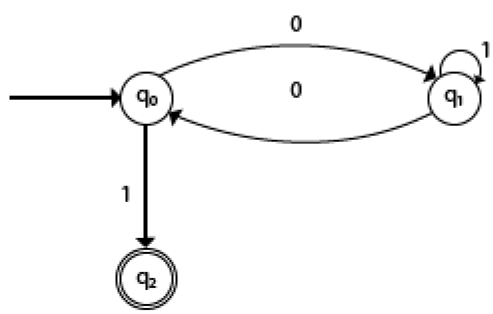
The stages q0, q1, q2 are the final states. The DFA will generate the strings that do not contain consecutive 1's like 10, 110, 101,..... etc.

Example 6:

Design a FA with ∑ = {0, 1} accepts the strings with an even number of 0's followed by single 1.

Solution:

The DFA can be shown by a transition diagram as:



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.

In the following image, we can see that from state q0 for input a, there are two next states q1 and q2, similarly, from q0 for input b, the next states are q0 and q1. Thus it is not fixed or determined that with a particular input where to go next. Hence this FA is called non-deterministic finite automata.