# Theory of Computation

The theory of computation begins with a question: What is a computer?

Real computers are quite complicated — too much so to allow us to set up a manageable mathematical theory of them directly.

In early 1900's, the notion of computers as we know now was not developed. Every simple concept about computation had be defined in a precise model.

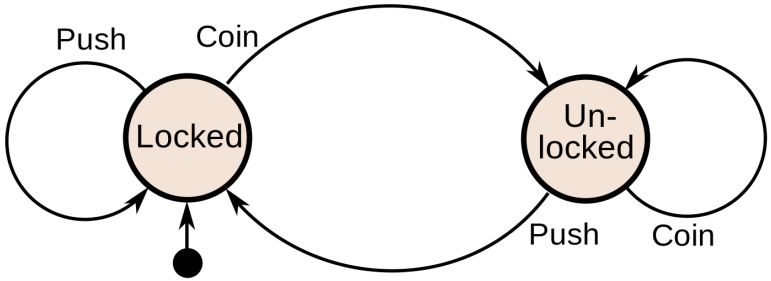
Instead of using an actual physical computer, we shall be using an idealised theoretical computer called a computational model.

We begin with the simplest model, called the finite state machine(FSM) or finite automaton(FA).

# Example:- Turnstile

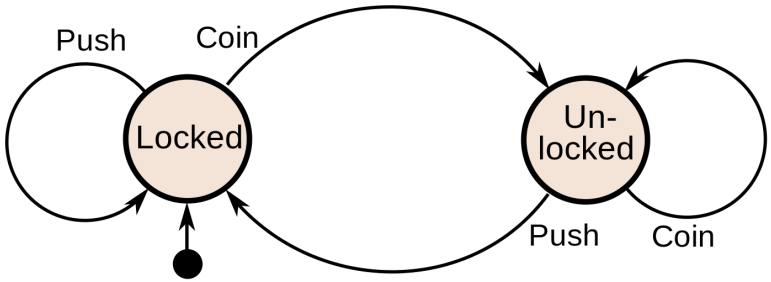
A turnstile, used to control access to metro-railway is a gate with three rotating arms at waist height, one across the entryway. Initially the arms are locked, blocking the entry, preventing patrons from passing through.

Depositing a token in a slot on the turnstile unlocks the arms, allowing a single customer to push through. After the customer passes through, the arms are locked again until another coin is inserted.




The turnstile state machine can also be represented by a directed graph called a state diagram.

- Each state is represented by a node (circle).
- Edges (arrows) show the transitions from one state to another.
- Each arrow is labeled with the input that triggers the transition.
- An input that doesn't cause a change of state (such as a coin input in the Unlocked state) is represented by a circular arrow returning to the original state.
- The arrow into the Locked node from the black dot indicates it is the initial state.



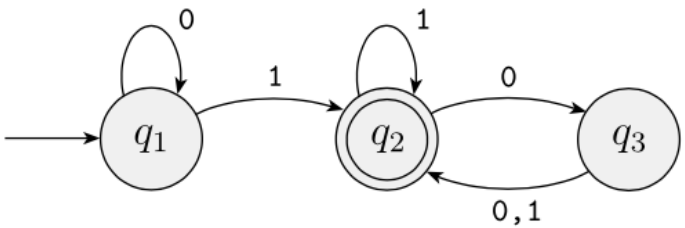

| Current State | Input | Next State | Output |
|---|---|---|---|
| **Locked** | coin | Unlocked | Unlocks the turnstile so that the customer can push through. |
| | push | Locked | None |
| **Unlocked** | coin | Unlocked | None |
| | push | Locked | When the customer has pushed through, locks the turnstile. |

# Definition of a Finite Automata(FA)

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

1. $Q$ is a finite set called the states,
2. $\Sigma$ is a finite set called the alphabet,
3. $\delta : Qx\Sigma -> Q$ is the transition function
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states

# Example




The finite automaton $M_1$

We can describe $M_1$ formally by writing $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. $\delta$ is described as

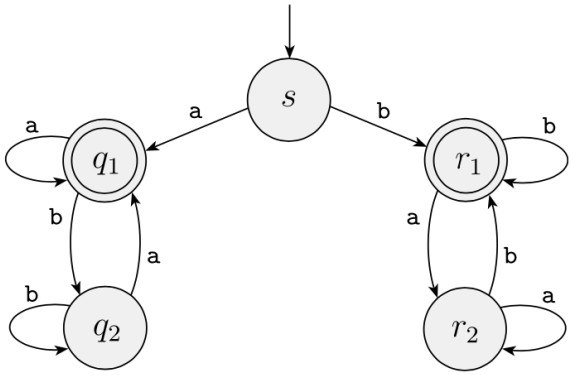
| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$, |

4. $q_1$ is the start state, and
5. $F = \{q_2\}$.

All the accept states are denoted by a double circle.
If $A$ is the set of all strings that machine $M$ accepts, we say that $A$ is the language of machine $M$ and write $L(M) = A$. We say that $M$ recognizes $A$ or that $M$ accepts $A$.

A machine may accept several strings, but it always recognizes only one language. If the machine accepts no strings, it still recognizes one language— namely, the empty language.

In our example, the language $L(M_1) = \{w | w$ contains at least one 1, and the last one is followed by an even number of zeroes, possibly 0 zeroes$\}$




In this FA($M_2$), there are two accepted states, namely $q_1$ and $r_1$. Although the entry state is unique.

This FA accepts all strings that start and end with *a* or that start and end with *b*. This type of simple set theoretic definition of each machine sometimes requires a rigorous proof and sometimes it is not so obvious to come up with.

# Definition of a Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \dots w_n$ be a string where each $w_i$ is a member of the alphabet $\Sigma$.

Then $M$ accepts $w$ if a sequence of states $r_0, r_1, \dots, r_n$ in $Q$ exists with three conditions:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n-1$ and
3. $r_n \in F$

Condition 1 says that the machine starts in the start state.
Condition 2 says that the machine goes from state to state according to the transition function.
Condition 3 says that the machine accepts its input if it ends up in an accept state.

In the above mentioned FA ($M_2$) with 5 states, suppose we are given a string say $w = abba$.

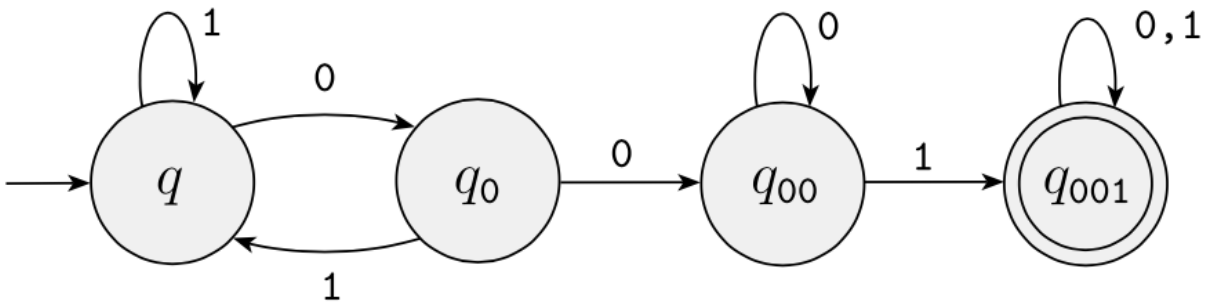
$M_2$ will start its execution from state $s$, then go to state $q_1$, then $q_2$ and again stay at $q_2$ and return to $q_1$. This path along the nodes in that prescribed order denotes a single computation. In this instance the sequence of states is $s, q_1, q_2, q_2, q_1$.

Suppose we have the description of a regular language (say $L'$) which consists of all strings that contain the string 001 as a substring over [0,1] alphabet. After playing around with few sample words we can realise that there are four possibilities:

1. haven't just seen any symbols of the pattern [001],
2. have just seen a 0,
3. have just seen 00, or
4. have seen the entire pattern 001.

These four possibilities give us an idea to build an FA with 4 states.

If we encounter the fourth possibility we enter an accept state, after which point the rest of the input string does not matter.




This FA is one such Automaton which accepts all the strings over binary alphabet which contain 001 as a substring.

A language is called a **regular language** if some finite automaton(FA) recognizes it.

In order to understand regular languages we need to understand regular operations. Let $A$ and $B$ be languages. We define the regular operations **union**, **concatenation**, and **star** as follows:

1. Union: $A \cup B = \{x | x \in A \text{ or } x \in B\}$.
2. Concatenation: $A \cdot B = \{xy | x \in A \text{ and } y \in B\}$.
3. Star: $A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \text{ and each } x_i \in A\}$.

For Star operation, since $k$ can be zero, the empty string denoted by $\epsilon$ is always a member of set $A^*$. Star operation is akin to concatenation operation but unlike concatenation it operates only one only language.

A collection of objects is closed under some operation if applying that operation to members of the collection returns an object still in the collection.
For example $N$ (the set of natural numbers) is closed under addition but not under division.

The regular languages are known to be closed under the aforementioned regular operations.

## Theorem

*The class of regular languages is closed under the union operation. Alternatively, if $A_1$ and $A_2$ are two regular languages, then $A_1 \cup A_2$ is also a regular language.*

**Outline of the Proof:-** If $A_1$ and $A_2$ are regular, we know that some finite automaton $M_1$ recognizes $A_1$ and some finite automaton $M_2$ recognizes $A_2$. To prove that $A_1 \cup A_2$ is regular, we demonstrate a finite automaton say $M$ , that recognizes $A_1 \cup A_2$. This is a proof by construction. We shall construct $M$ from $M_1$ and $M_2$, by simulating them simultaneously.

Let $M_1$ recognize $A_1$, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and let $M_2$ recognize $A_2$, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.
We shall construct $M$ to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$.

- $Q = \{(r_1, r_2) | r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
- $\Sigma$, the alphabet, is the same as in $M_1$ and $M_2$.
- $\delta$, the transition function, is defined as follows. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

Hence $\delta$ gets a state of $M$ (which actually is a pair of states from $M_1$ and $M_2$), together with an input symbol, and returns $M$'s next state.

- $q_0 = (q_1, q_2)$
- F is the set of pairs in which either member is an accept state of M1 or M2. We can write it as

$$F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$$

Since the star operation is similar to the concatenation operation, we shall try to prove that regular languages are closed under concatenation operation.

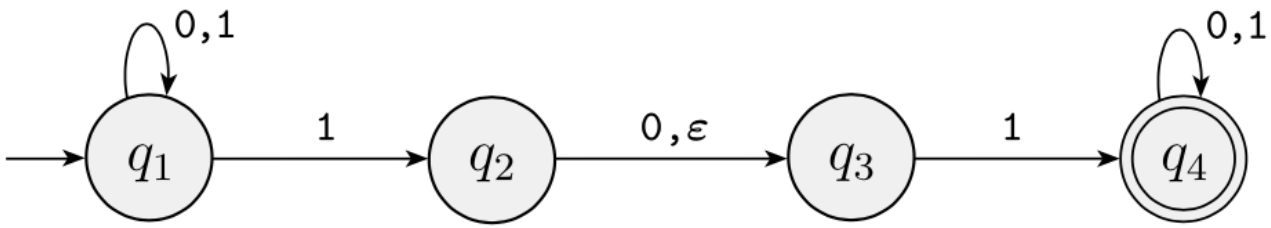
Using a strategy similar to the one used in case of union operation does not work, because in case of concatenation operation the machine does not know when the word of first machine ends.

This requires us to resort to Non-Determinism.

When the machine is in a given state and reads the next input symbol, we know what the next state will be —it is determined. We call this deterministic computation. Till now we have encountered only deterministic FA.

In a nondeterministic machine, several choices may exist for the next state at any point.

Nondeterminism is a generalization of determinism, so every deterministic finite automaton is automatically a nondeterministic finite automaton.




In this example, at state $q_1$, the NFA(Non-Deterministic Finite Automata) after encountering symbol 1, has the choice to either stay at $q_1$ or move to state $q_2$.

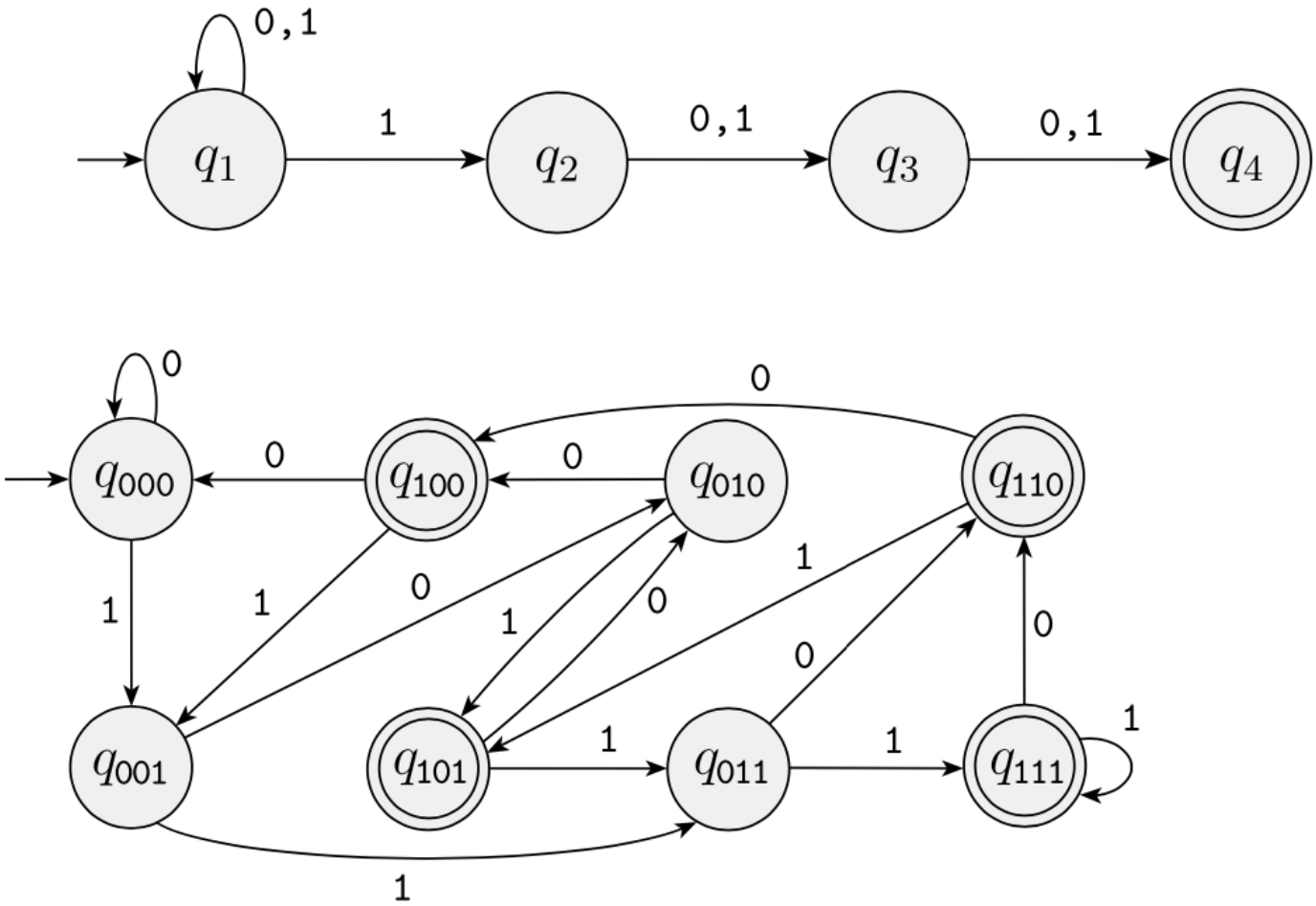
Another feature of NFA's is that they may have states which transition without reading an input symbol, which is alternatively seen has reading an empty symbol denoted by $\epsilon$.

Nondeterminism may be viewed as a kind of parallel computation wherein multiple independent "processes" or "threads" can be running concurrently.

When the NFA splits to follow several choices, that corresponds to a process "forking" into several children, each proceeding separately. If at least one of these processes accepts, then the entire computation accepts.

Every NFA can be converted into an equivalent DFA (somewhat counter-intuitive), and constructing NFAs is sometimes easier than directly constructing DFAs.

An NFA may be much smaller than its deterministic counterpart, or its functioning may be easier to understand.




An NFA and its corresponding DFA. Both accept only those strings which have a 1 at the third last position.