AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Science and Technology

AIUB

DEPARTMENT OF COMPUTER SCIENCE

## CSC3113: THEORY OF COMPUTATION

Lecture: # 2 | Week: # 1 | Semester: Spring 2022-2023

# DETERMINISTIC FINITE AUTOMATON (DFA)

Instructor: Shakila Rahman, Lecturer,

Department of Computer Science, Faculty of Science & Technology.

Shakila.Rahman@aiub.edu

# Lecture Outline

- Finite Automata (FA).
  - Example and Simulation of FA.
  - Finite state machine models.
  - Definition
- Deterministic Finite Automata (all with examples)
  - Terminologies & State Diagram
  - Formal Mathematical Definition
  - Formal Computational Definition

# LEARNING OBJECTIVE

- Understand, learn & practice with example
  - Finite Automata (FA)
  - FA Machine Models
  - Finite State Machine
  - Deterministic Finite Automata (DFA)
  - Formal Definition of DFA
  - Computational Definition of DFA

# LEARNING OUTCOME

## ALL OUTCOME ARE REPRESENTED WITH EXAMPLES

- Know all the components of a finite state machine.
- Learn the terminologies, conditions, and representation of the machine models.
- How to define a machine model, along with its characteristics, using mathematical structure.
- How to define the computation perform by the machine model using mathematical structure.
- Understand the mathematical model for DFA
- Students will be able to
  - Formally define a given DFA machine model
  - Run the machine for given input and determine if it is accepted or rejected.

# AUTOMATA THEORY

- Automata comes from the Greek word (Αυτόματα) which means that something is doing something by itself.
- Automata deals with the study of abstract (**mathematical model**) machines or systems (**definition and properties**) and the computational problems (**defined in terms of formal languages**) that can be solved (**recognized**) using these machines.
  - Automata are used as theoretical models for computing machines (input, process, output),
  - An automaton can be a *finite representation of a formal language* that may be an infinite set (*language theory*). Formal languages are the preferred mode of specification (**input**) for any problem that must be computed (**processed**).
  - These abstract computing machines are used for proofs about computability (**solvability**).
- Such models include –
  - *finite automaton*, used in text processing, compilers, and hardware design
  - *Context-free grammar*, used in programming languages and artificial intelligence

# FINITE AUTOMATA

- We will use several different models, depending on the features we want to focus on. Begin with the simplest model, called the **finite automaton**.
- Good models for computing device with an extremely *limited amount of memory*.
  - For example, various household appliances such as dishwashers and electronic thermostats, as well as parts of digital watches and calculators.
- The design of such devices requires keeping the *methodology* and *terminology* of finite automata in mind.
- Next, we will analyze an example to get an idea of the *methodology* and *terminology* of finite automata and then we go for a *formal definition*.

# Automatic Door

## An Example

- Automatic doors swing open when sensing that a person is approaching.
- An automatic door has a pad in front to detect the presence of a person about to walk through the doorway.
- Another pad is located to the rear of the doorway so that –
  - The controller can hold the door long enough for the person to pass all the way through.
  - The door does not strike someone standing behind it as it opens.

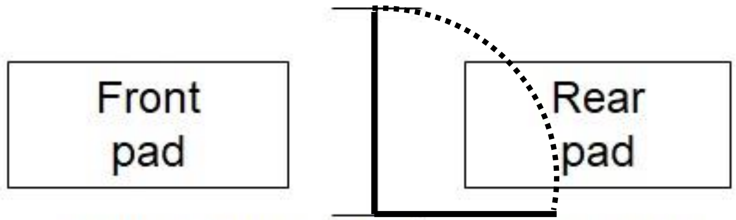



*Figure: Top view of an automatic door*

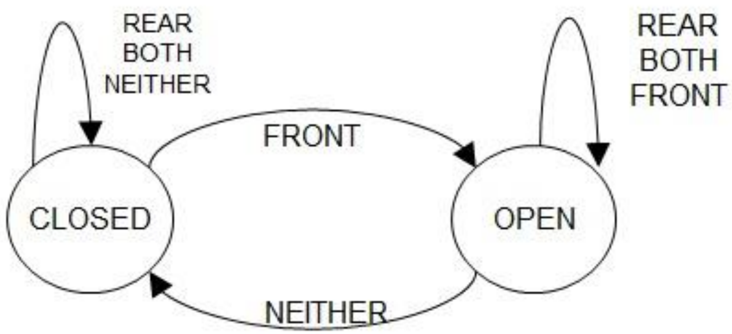



*Figure: State diagram for Automatic door controller*

| | | Input Signal | | | |
|---|---|---|---|---|---|
| | | NEITHER | FRONT | REAR | BOTH |
| State | CLOSED | CLOSED | OPEN | CLOSED | CLOSED |
| | OPEN | CLOSED | OPEN | OPEN | OPEN |

Figure: State Transition table for automatic door controller

# Simulation – Automatic Door

***Input Example***:

***Initial State***: CLOSED

***Input Signal Sequence***: FRONT, REAR, NEITHER, FRONT, BOTH, NEITHER, REAR, NEITHER.

- Present State: OPEN
- Input Signal: BOTH

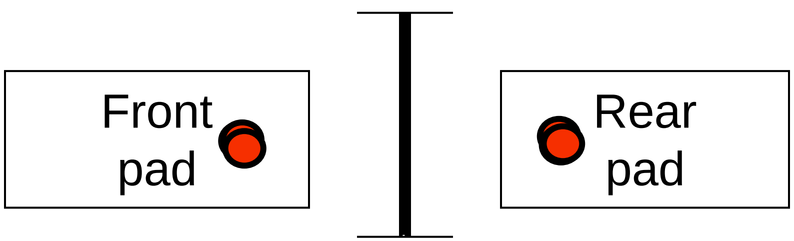



Figure: Top view of an automatic door

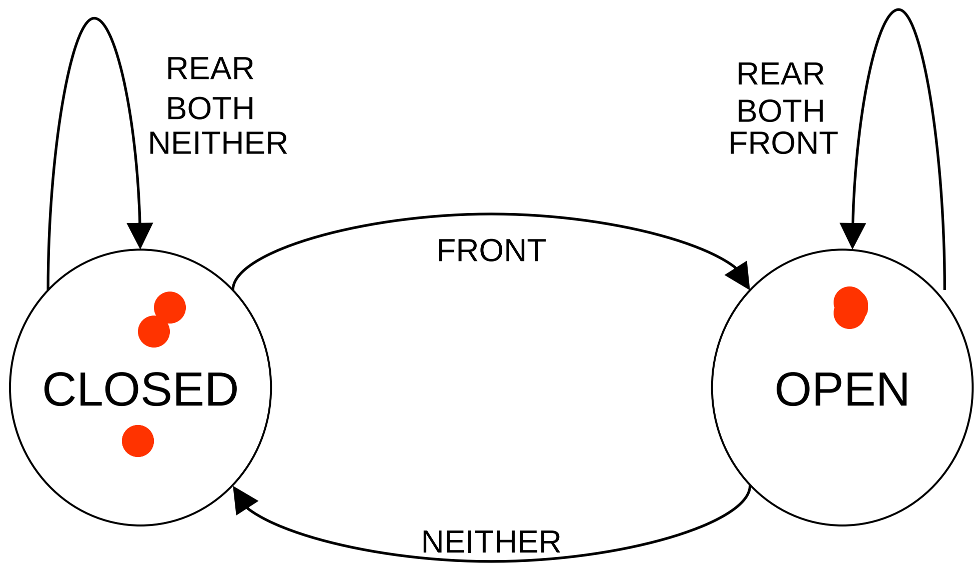


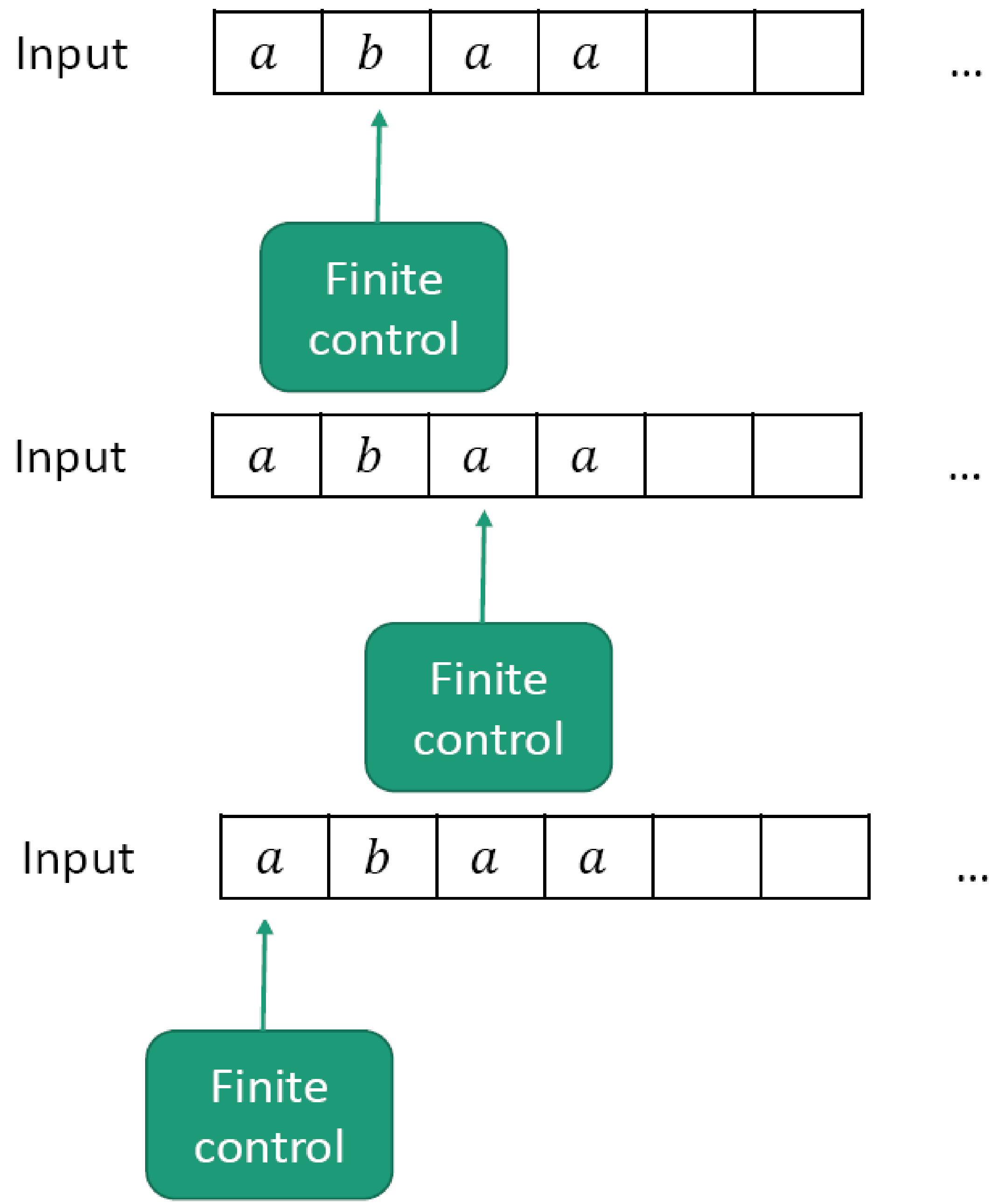

Figure: State diagram for Automatic door controller

# MACHINE MODELS

- Computation is the processing of information by the **unlimited application** of a **finite set** of operations or rules.
- **Abstraction:** We don't care how the control is implemented.

We just require it to –

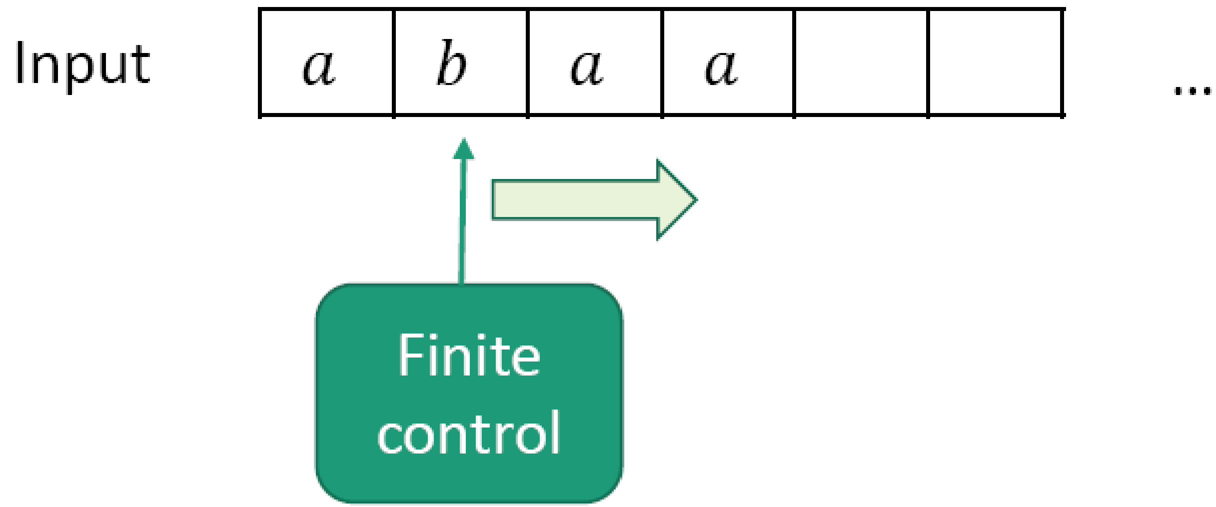
- read a given input string
- have a finite number of states, and
- transition between states using fixed rules.




## FINITE STATE MACHINE

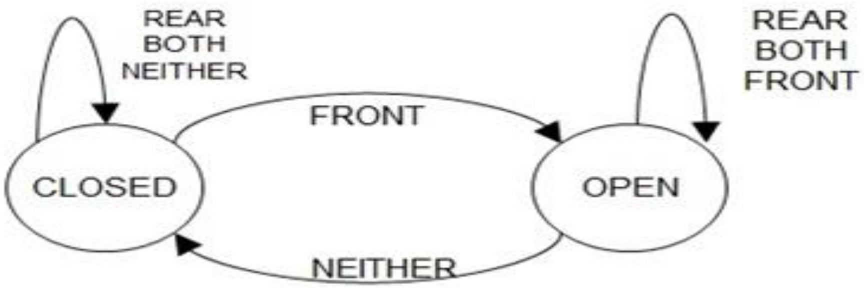
## Machine Model

### Finite Automata (FA)




- Machine with a *finite* amount of unstructured *memory*.
  - *Control* scans a given input string only once (*from some language*) left-to-right, one by one.
  - Can check/match simple patterns (by *transiting from state to state based on some given rules*)
  - Can't perform unlimited counting
- Useful for modeling chips, simple control systems, adventure games…

# FINITE AUTOMATA – DEFINITION




| | | Input Signal | | | |
|---|---|---|---|---|---|
| | | NEITHER | FRONT | REAR | BOTH |
| State | CLOSED | CLOSED | OPEN | CLOSED | CLOSED |
| | OPEN | CLOSED | OPEN | OPEN | OPEN |

Figure: State Transition table for automatic door controller

- A finite automata has several parts –
  - It has a precise **set** of **inputs** (*Language*)
    - Example: FRONT, REAR, BOTH, NEITHER.
  - **Set** of **states**
    - Example: the auto door has CLOSED and OPEN states.
  - Initial (**start**) state must be defined
    - Example: CLOSED in the door example.
  - **Rules** for going from one state to another based on input Also known as transition rules.
    - Example: if door is in CLOSED state and [*rule*] someone is only on the front pad, then the door will go to OPEN state based on input, FRONT.
  - May have one or more state(s) as goal to reach from start state. Also known as **set** of **final/accept** states
    - Example: CLOSED as the last input signal is NIETHER in the door example.

# Types of Finite Automata

- Based on the type of computation finite automata can be of two types -
  - **Deterministic Finite Automata (DFA):** Where every next step is pre-determined by some deterministic rules/computation.
  - **Nondeterministic Finite Automata (NFA):** Where every next step may have zero or more number of choices to move on.

## Tree representation of DFA and NFA

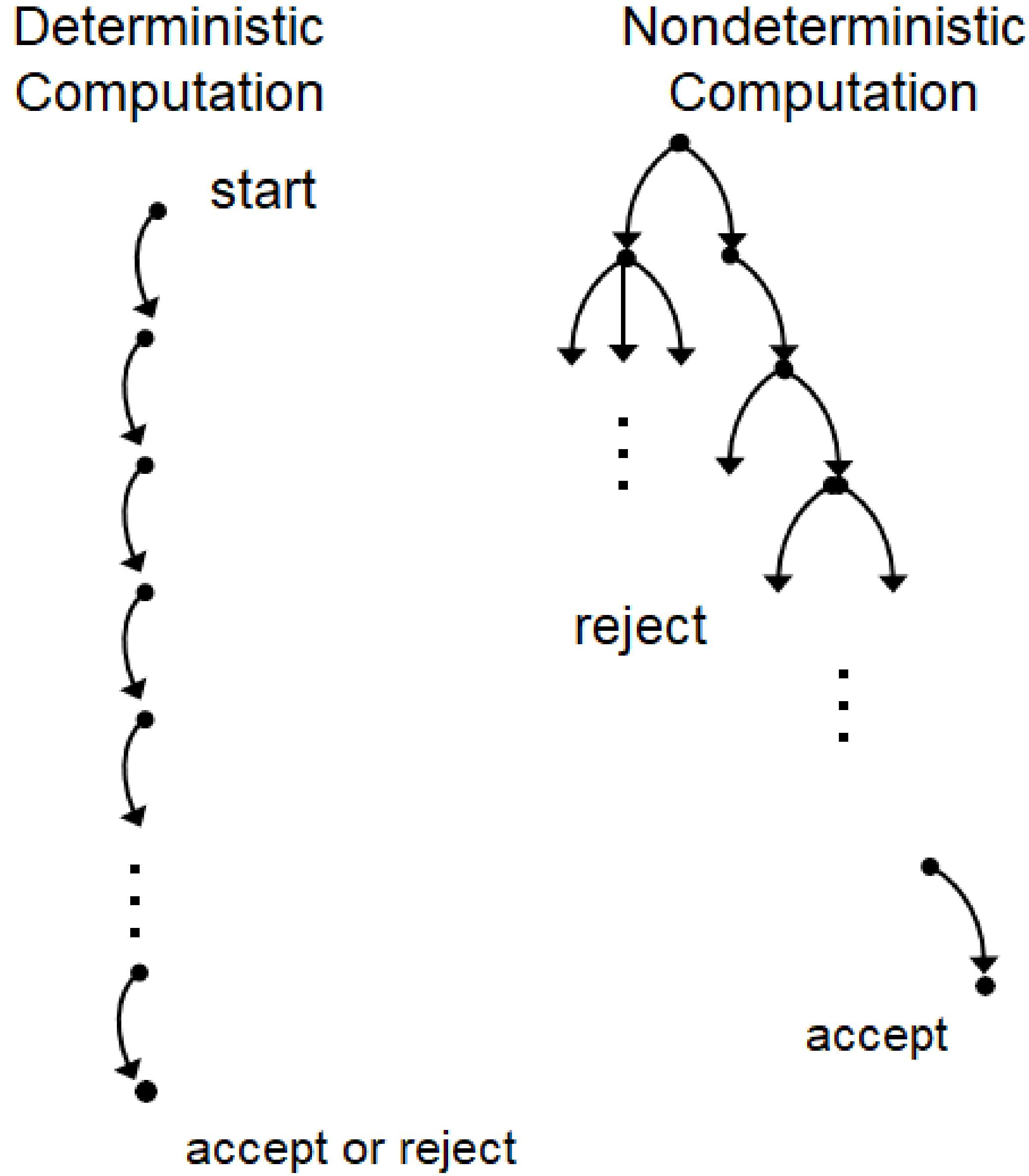


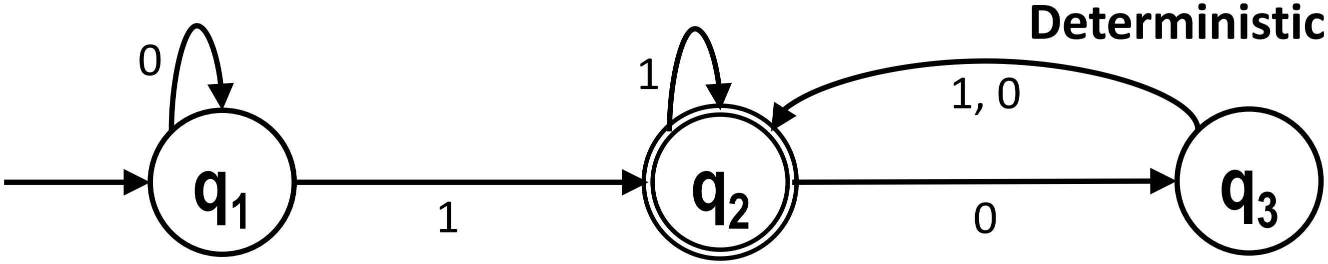

*Figure*: Deterministic and nondeterministic computations with an accepting branch

# Deterministic Finite Automata (DFA)

- Every step of a computation follows in a unique way from the preceding step (deterministic computation).
- When the machine is in a given state and reads the next input symbol, we know the next state will be – it is determined.
- The transition rules are of the form –
  - Each state must have exactly one transition for each input from the input set to any individual state (including itself).
  - If there are ***n*** number of inputs, then each state must have exactly ***n*** transitions to any states (including itself).
  - There must be exactly one start state to start the transition and one or more final states to finish the transition.
- Let us go through –
  - a precise definition of a deterministic finite automaton,
  - terminologies for describing and manipulating DFA,
  - theoretical results that describe their powers and limitations.
- Let us now investigate the terminologies through an example.

# TERMINOLOGIES

**Deterministic Finite Automata (DFA)**
**State Diagram**




- 3 *states*, labeled $q_1$, $q_2$, and $q_3$.
- The *start state* is $q_1$, indicated by the arrow pointing at it from no where.
- The *accept state*, $q_2$, is the one with a double circle.
- The arrow going from one state to another (or to itself [loop]) are called *transitions*.
- The symbol(s) along the transition is called *label*.
- Each label is from input set {0, 1}.
- From each state there are exactly one transition for each input 0 and 1.

- $M_1$ works as follows –
  - The automaton receives the symbols from the input string one by one from left to right.
  - After reading each symbol, M1 moves from one state to another along the transition that has the symbol as its label.
  - When it reads the last symbol, M1 produces the output.
  - The output is ACCEPT if M1 is now in an accept state and REJECT if it is not.

# SIMULATION – HOW IT WORKS?

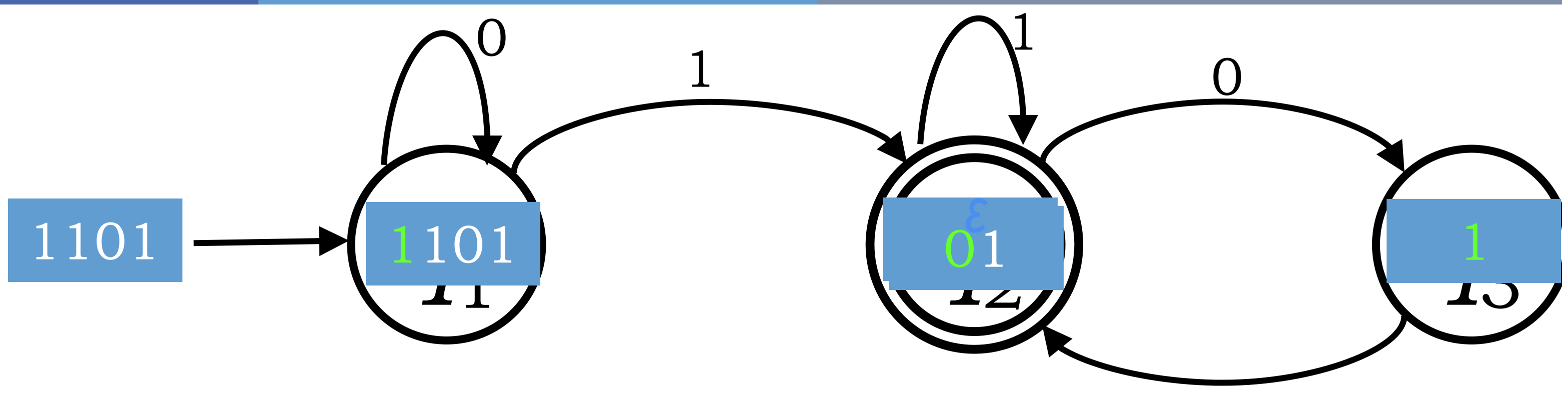



*Figure*: Finite Automaton $M_1$.

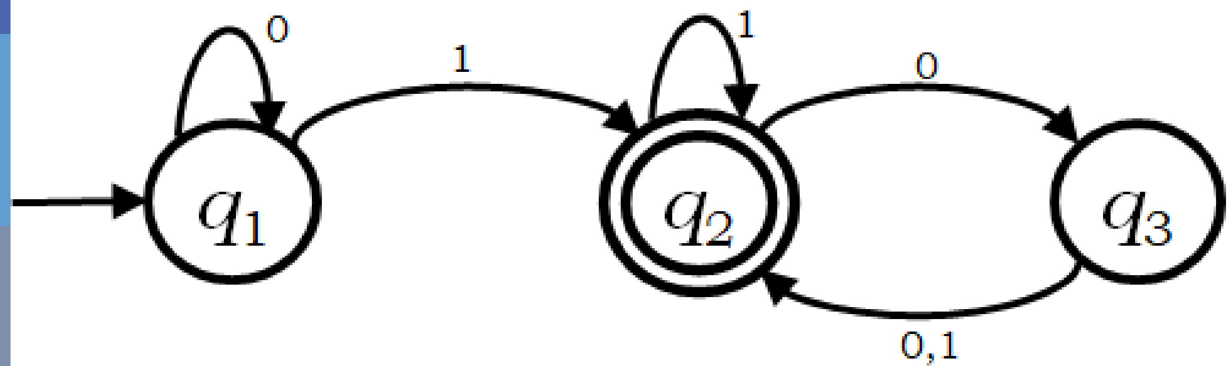
- After feeding the input string **1101** to the above machine, the processing proceeds as follows –
  - Start in state $q_1$;
  - Read 1, follow transition from $q_1$ to $q_2$;
  - Read 1, follow transition from $q_2$ to $q_2$;
  - Read 0, follow transition from $q_2$ to $q_3$;
  - Read 1, follow transition from $q_3$ to $q_2$;
  - ACCEPT, as the machine $M_1$ is in an accept state $q_2$ at the end of the input string.

# Formal Definition - DFA

- A Deterministic Finite Automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  - $Q$ is a finite set called the ***states***,
  - $\Sigma$ is a finite set called the ***alphabet***,
  - $\delta : Q \times \Sigma \rightarrow Q$ is the ***transition function***,
  - $q_0 \in Q$ is the ***start state***,
  - $F \subseteq Q$ is the set of ***accept*** (*final*) ***states***.
- If A is the set of all strings that a machine *M* accepts, we say that *A* is the ***language of machine M*** and write $L(M)=A$, ***M recognizes A*** or ***M accepts A***.

# Formal Definition for Machine $M_1$

## Example




*state diagram* of Machine $M_1$

- $M_1 = (Q, \Sigma, \delta, q_0, F)$, where –
  - $Q = \{q_1, q_2, q_3\}$ ,
  - $\Sigma = \{0, 1\}$,
  - $q_0 = q_1$,
  - $F = \{q_2\}$,
  - *$\delta$ is describe as* –

$$\begin{aligned}
\delta(q_1,0) &= q_1, \ \delta(q_1,1) = q_2, \\
\delta(q_2,0) &= q_3, \ \delta(q_2,1) = q_2, \\
\delta(q_3,0) &= q_2, \ \delta(q_3,1) = q_2.
\end{aligned}$$

**Transition Function**

OR

| $\delta$ | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$ |

**Transition Table**

# Formal Definition of DFA Computation

- Now we formalize the Deterministic Finite Automaton's computation, mathematically.
- Let,
  - $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA,
  - $w = w_1w_2 \ldots w_n \in \Sigma^*$ (a string over the alphabet $\Sigma$), where each $w_i \in \Sigma$.
- Then $M$ ***accepts*** $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists in $Q$ with the following three conditions –
  - $r_0 = q_0$,
  - $\delta(r_i, w_{i+1}) = r_{i+1}$, for i = 0, 1, 2, … , $n$-1, and
  - $r_n \in F$.
- $M$ **recognizes language** $L$ if $L = \{w : M \textbf{\textit{ accepts }} w\}$.

# SIMULATION – DFA COMPUTATION

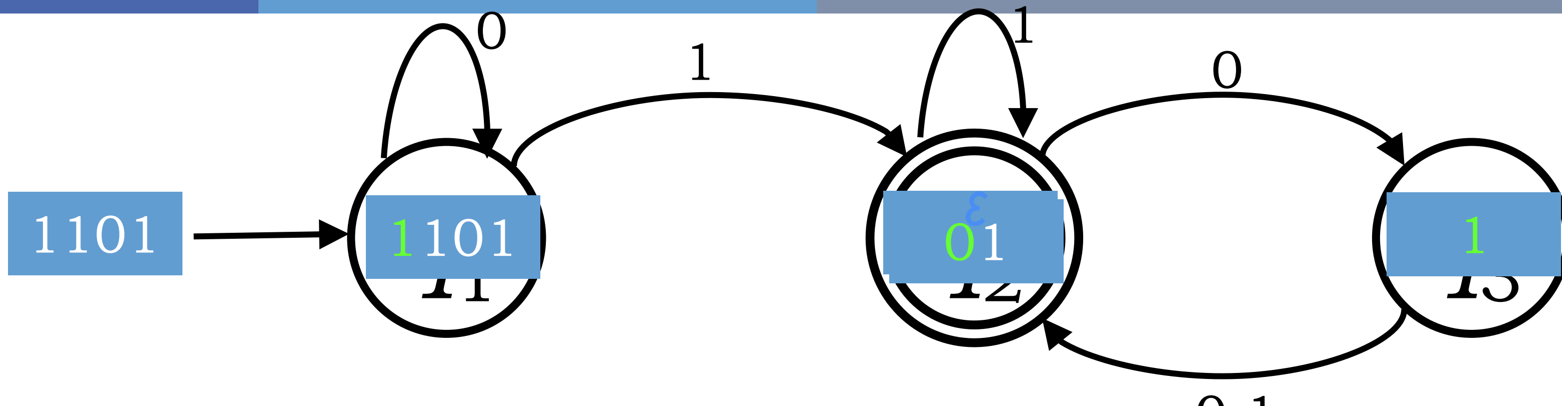



*Figure*: Finite Automaton $M_1$.

- M=($Q$, $\Sigma$, $\delta$, $q_0$, $F$)=($\{q_1, q_2, q_3\}$, $\{0, 1\}$, $\delta$, $q_1$, $\{q_2\}$) and $\delta = \{\delta(q_1,0)=q_1, \delta(q_1,1)=q_2, \delta(q_2,0)=q_3, \delta(q_2,1)=q_2, \delta(q_3,0)=q_2, \delta(q_3,1)=q_2\}$
- Input string $w = w_1w_2w_3w_4$ = **1101** to $M_1$ gives a sequence of states $r_0,r_1,r_2,r_3,r_4$ in the following computation (here $n$ = 4) –
  - Start in state $r_0 = q_1$; → $r_0=q_0$,
  - $\delta(r_0,w_1) = \delta(q_1,1) = q_2 = r_1$; → $\delta(r_i, w_{i+1}) = r_{i+1}$, for i = 0, 1, 2, … , $n$-1,
  - $\delta(r_1,w_2) = \delta(q_2,1) = q_2 = r_2$; → $\delta(r_i, w_{i+1}) = r_{i+1}$, for i = 0, 1, 2, … , $n$-1,
  - $\delta(r_2,w_3) = \delta(q_2,0) = q_3 = r_3$; → $\delta(r_i, w_{i+1}) = r_{i+1}$, for i = 0, 1, 2, … , $n$-1,
  - $\delta(r_3,w_4) = \delta(q_3,1) = q_2 = r_4$; → $r_n \in F$.
  - Accept, as the machine $M_1$ is in an accept state $q_2$ at the end of the input string.
  - $M_1$ **recognizes language** $L$ if $L = \{w : M_1$ ***accepts*** $w\}$.

# References

## Practice the exercises

- Introduction to Theory of Computation, Sipser, ($3^{rd}$ ed),
  - DFA; All Exercises;
- Elements of the Theory of Computation, Papadimitriou ($2^{nd}$ ed), Chapter 1.