DETERMINISTIC FINITE AUTOMATA

iACADEMY
SCHOOL OF COMPUTING • SCHOOL OF BUSINESS • SCHOOL OF DESIGN

SCHOOL OF COMPUTING

MITCH M. ANDAYA

BASIC INFORMATION

- Professor's Name: Mitch M. Andaya
- Title: Vice President for Academics and Dean of the School of Computing
- Office: 3rd Floor
- Email: mitch.andaya@iacademy.edu.ph

BASIC INFORMATION

- **Cell phones, tablets, and other gadgets must be kept hidden.**
- **100% attention is required from you. Direct questions to the instructor, not the seatmate.**
- Attendance: Maximum of 3 absences (1 late is equal to ½ absence)

BASIC INFORMATION

- Class Requirements : 4 or 5 Major Exams
- Cancel the lowest major exam
- If you missed an exam, you have one (1) week to take it. There will be no difficulty factor for missed exams.
- Passing is 70%. Absolutely no extra work will be given for those who will fail.

THEORY OF COMPUTATION

- ***Theory of computation*** is the branch of theoretical computer science and mathematics, that deals with how efficiently problems can be solved on a model of computation, using an algorithm.

- The field is divided into three major branches:

    1. Automata Theory and Language
    2. Computability Theory
    3. Computational Complexity Theory

- All three areas are linked by the question: "*What are the fundamental capabilities and limitations of computers?*"

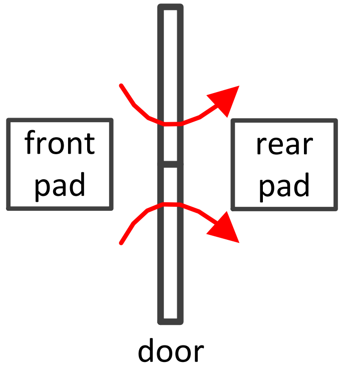
iACADEMY
SCHOOL OF COMPUTING • SCHOOL OF BUSINESS • SCHOOL OF DESIGN

AUTOMATA THEORY

- ***Automata theory*** is the study of abstract machines.
- Abstract machines are theoretical representations of real-world machines (such as a computer) but without the unnecessary details associated with a particular instance of that machine.
- By removing these details, it is easier to analyze the operation of the machine being studied.

AUTOMATA THEORY

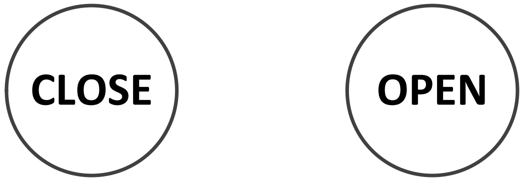
- The controller for an automatic door is one example of such a machine. Automatic doors swing open when the controller senses that a person is approaching.
- An automatic door has a pad in front to detect the presence of a person about to walk through the doorway.
- Another pad is located to the rear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



AUTOMATA THEORY

- The controller is in either of two states: "open" or "close," representing the corresponding condition of the door.

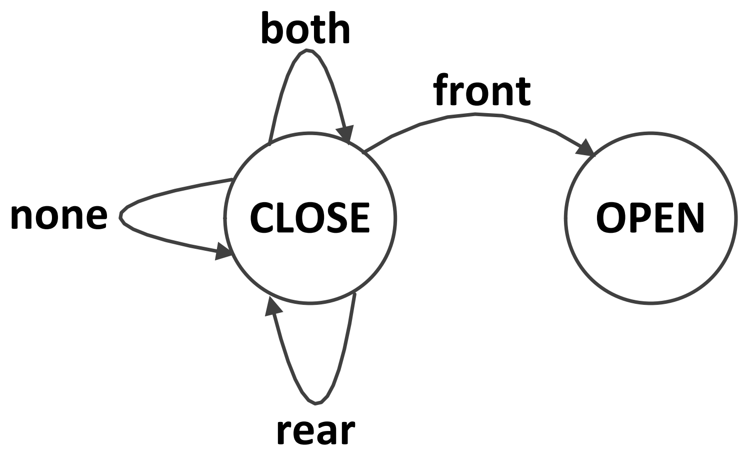
- There are four possible input conditions: "front" (meaning that a person is standing on the pad in front of the doorway), "rear" (meaning that a person is standing on the pad to the rear of the doorway), "none" (meaning that no one is standing on either pad), and "both" (meaning that people are standing on both pads).



AUTOMATA THEORY

- The controller moves from state to state, depending on the input it receives. Assume the machine is in the *close* state.

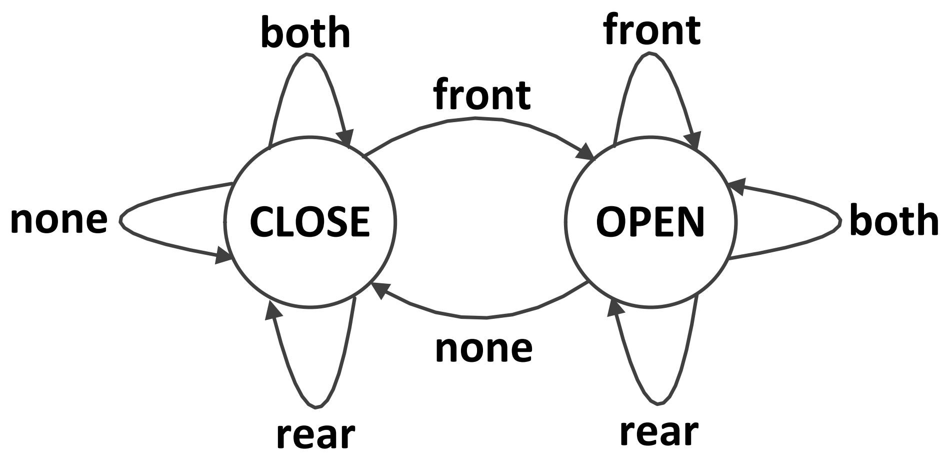
INPUT	STATE
front	open
rear	close
none	close
both	close



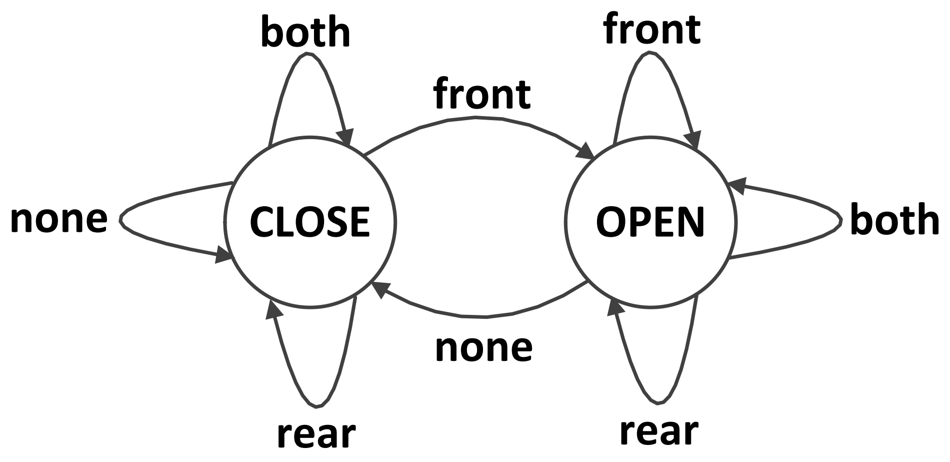
AUTOMATA THEORY

- Assume now the machine is in the *open* state.

INPUT	STATE
front	open
rear	open
none	close
both	open



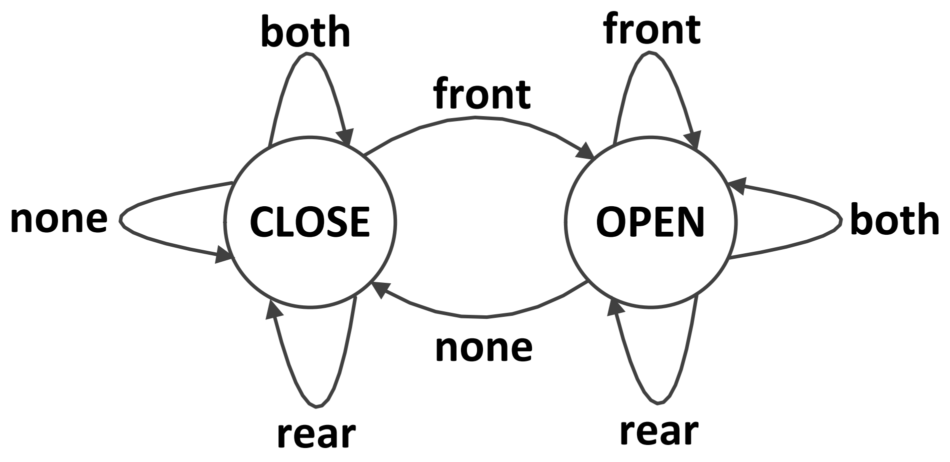
AUTOMATA THEORY



- This abstract model of the automatic door machine is called an ***automaton*** and the graph that represents it is called its ***state diagram***.

- Its operation can now be analyzed without having the actual physical machine present.

AUTOMATA THEORY



- For example, a controller might start in state *close* and receive the series of input signals *front*, *rear*, *none*, *front*, *both*, *none*, *rear*, and *none*.

  It then would go through the series of states *close* (starting), *open*, *open*, *close*, *open*, *open*, *close*, *close*, and *close*.

AUTOMATA THEORY

- Automata theory therefore uses abstract or mathematical models to represent machines and computers.
- Like the automatic door machine, a computer can also be in one state or another depending upon the contents of its main memory, processor registers, etc.
- When input data arrives or an event occurs (like the clicking of the mouse or the typing of a character on the keyboard), the computer moves from one state to another.

AUTOMATA THEORY

- The following are just some of the things that can be done once models have been established:

1. Determine the capabilities and limitations of each machine.

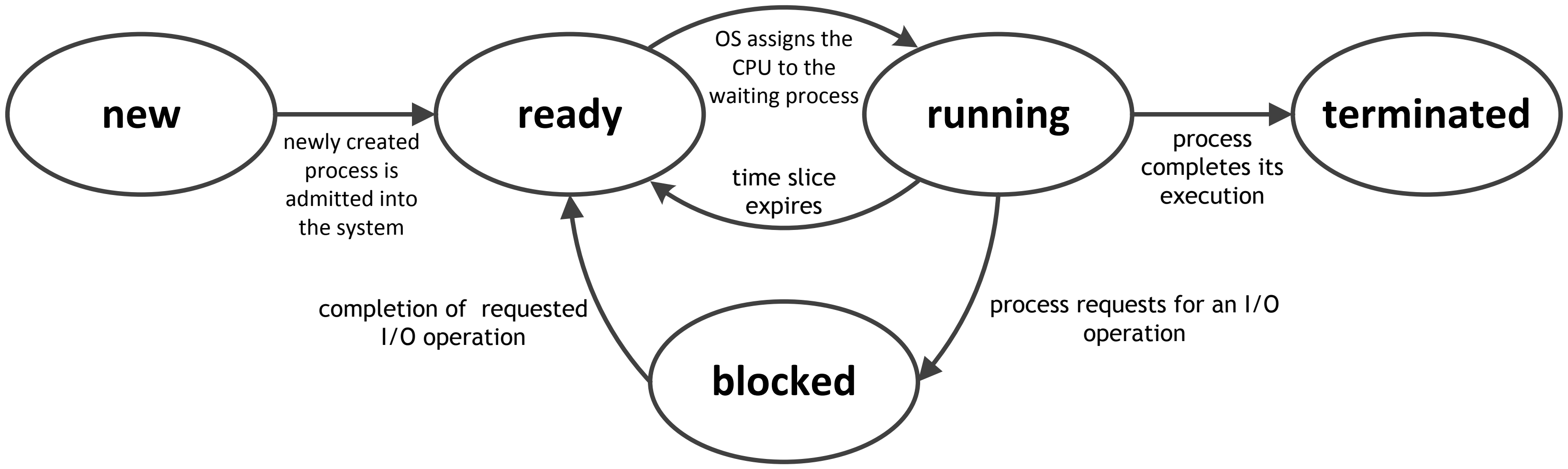
2. Determine which machine is more powerful.

3. Determine what each machine can compute.

AUTOMATA THEORY

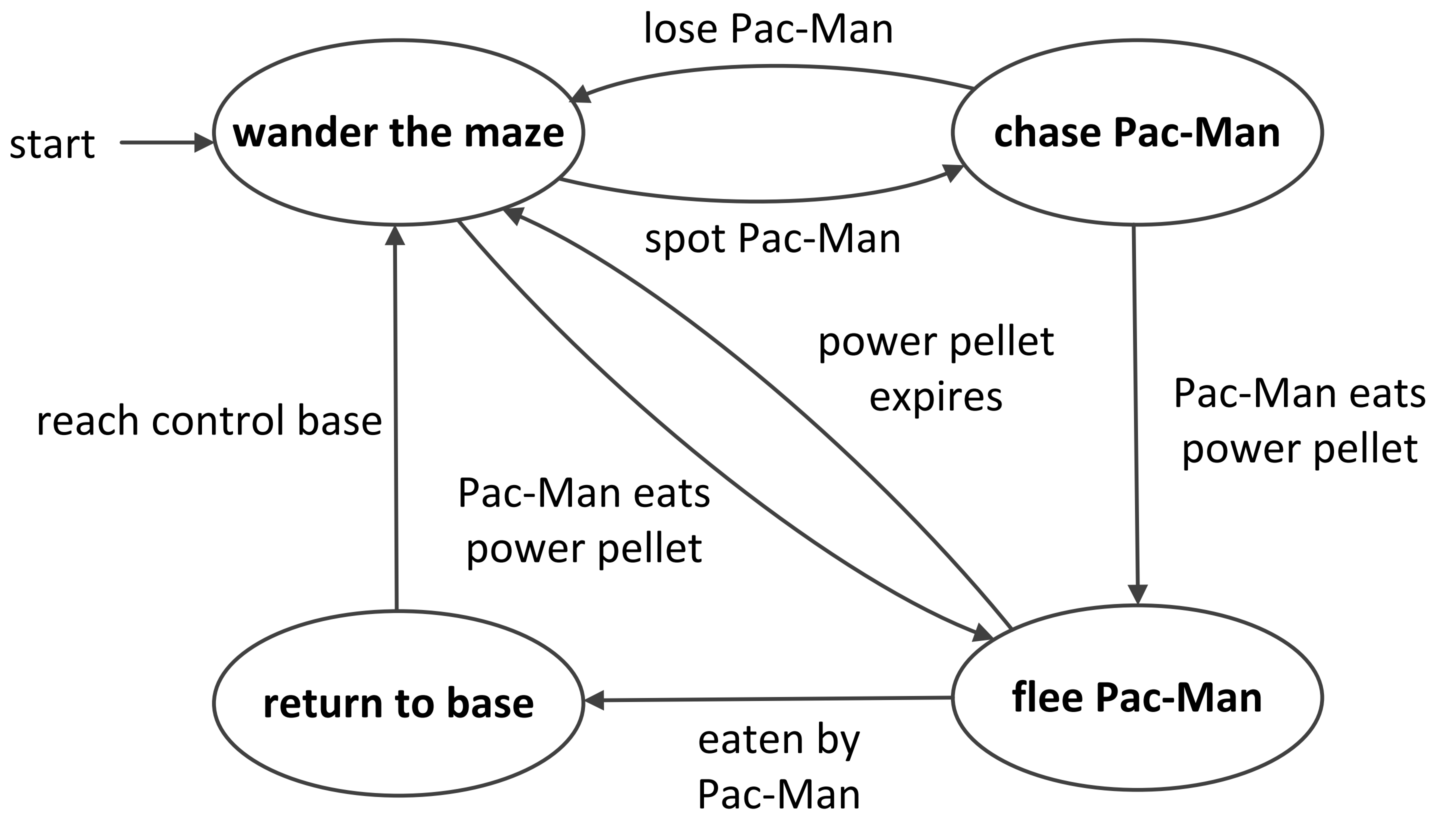
- An automaton is not only used to represent machines (hardware) but they can also be used to represent software (such as operating systems and compilers) and other processes.

- State Diagram of a Process (Operating Systems)


new
newly created process is admitted into the system
ready
OS assigns the CPU to the waiting process
time slice expires
running
process completes its execution
terminated
completion of requested I/O operation
process requests for an I/O operation
blocked

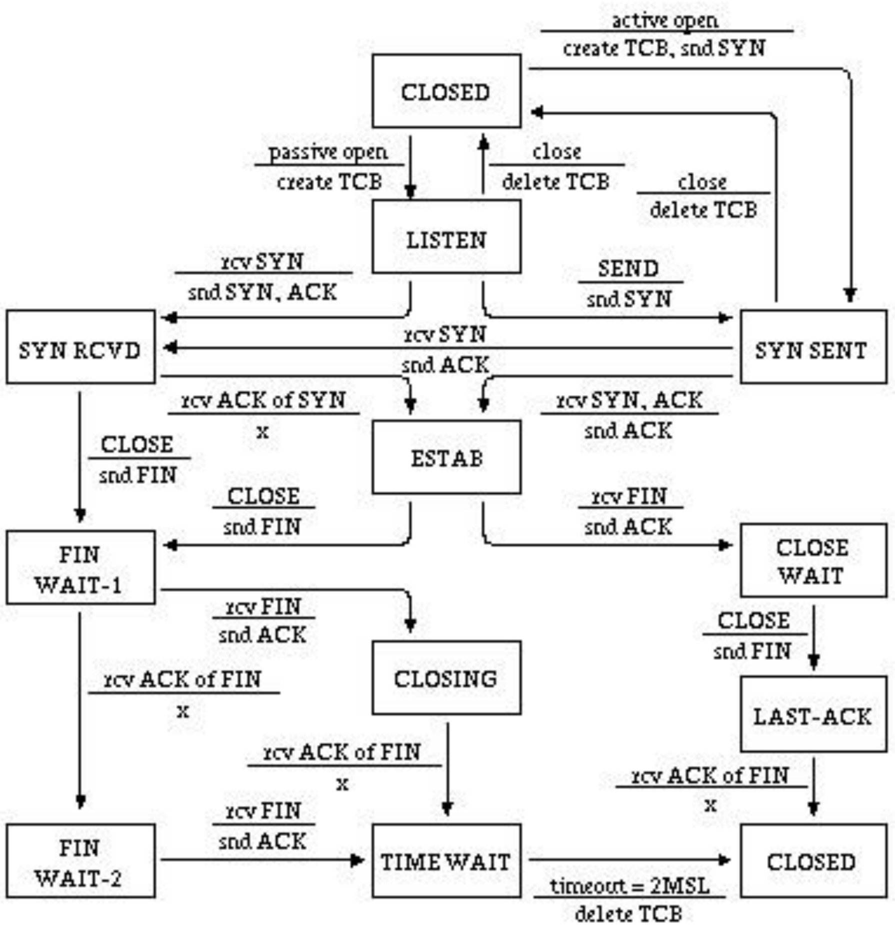
AUTOMATA THEORY

- State Diagram of the Behavior of a Ghost in Pac-Man



AUTOMATA THEORY

- State Diagram of a TCP Connection



AUTOMATA THEORY

- The different types of automata that will be encountered in this course are:
  - Deterministic Finite Automata
  - Nondeterministic Finite Automata
  - Pushdown Automata
  - Turing Machines

ALPHABET

- An ***alphabet*** is defined to be a non-empty finite set whose members are called the ***symbols*** of the alphabet.

- The symbols can be letters, numbers, special characters, musical notes, emoticons, etc.

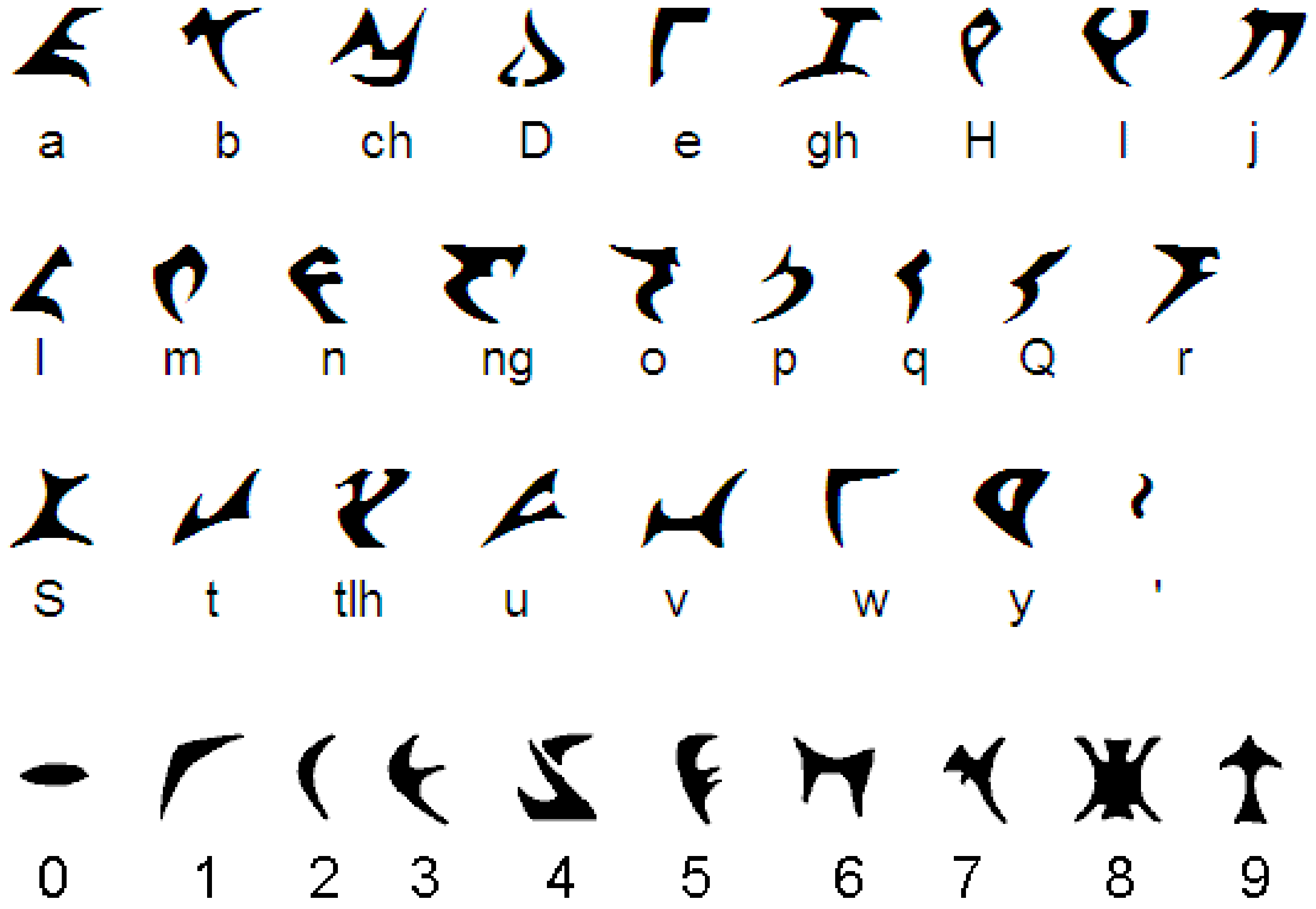
Examples:

Σ_1 = {0, 1}

Σ_2 = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}

Σ_3 = {♠, ♣, ♥, ♦}

ALPHABET


a b ch D e gh H I j
l m n ng o p q Q r
S t tlh u v w y '
0 1 2 3 4 5 6 7 8 9

STRINGS

- A ***string*** over an alphabet is a finite sequence of 0 or more symbols from that alphabet juxtaposed or placed side by side (not separated by commas).

  Examples:

  For the binary alphabet Σ_1, 000, 101010, 11, and 11010 are strings that can be formed over Σ_1.

  For the alphabet Σ_2, *abba*, *xyz*, *iacademy*, and *abracadabra* are strings that can be formed over Σ_2.

- A string may contain no symbols at all, and is called the ***empty string***, designated as ε.

STRINGS

- A fundamental operation that can be performed on strings is ***concatenation***.

  Given string *x* of length *m* and string *y* of length *n*, the concatenation of *x* and *y*, written as *xy*, is the string obtained by writing the string *x* followed by the string *y* with no intervening space between them, as in

$$xy = x_1 \ldots x_m y_1 \ldots y_n$$

For example:

If string *x* = iacad and string *y* = emy, then the concatenation of *x* and *y* is:

xy = iacademy

LANGUAGES

- A ***language*** is a set of strings over a given alphabet.

  For example:

  Given the alphabet $\Sigma = \{0, 1\}$. If the language L_1 is defined as the set of all 3-bit binary numbers, then:

  $$L_1 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

  Given the alphabet $\Sigma = \{0, 1\}$. If the language L_2 is defined as the set of all palindromes, then:

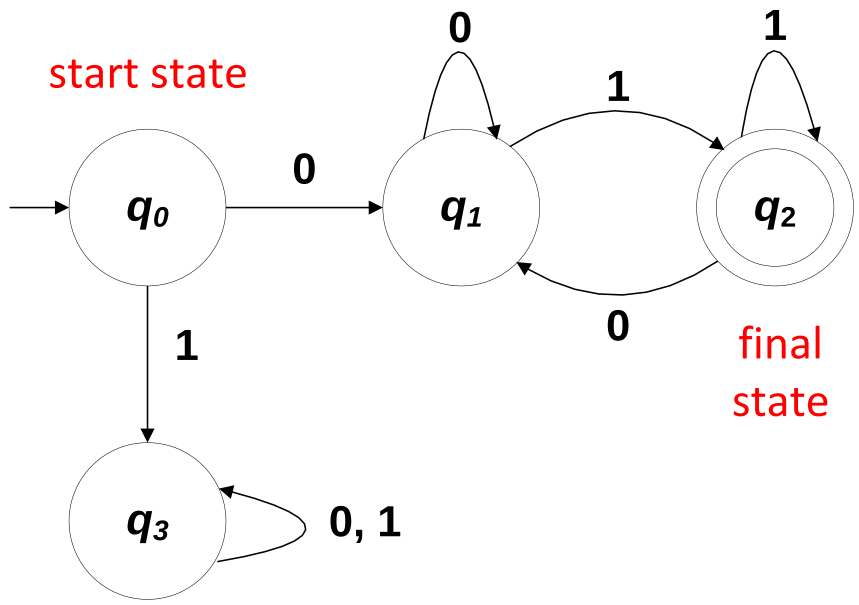
  $$L_2 = \{\varepsilon, 0, 1, 00, 11, 010, 101, 110011, \ldots\}$$

DETERMINISTIC FINITE AUTOMATA

- A ***finite automaton*** is a machine that has a limited (finite) number of states.
- Because of this limitation, these are normally used to model computers with a limited amount of memory or programs that process only a small amount of data.
- Discussions here will focus on hypothetical machines whose only task is to determine whether an input string or a series of input symbols is acceptable or not based on some predefined rule.

DETERMINISTIC FINITE AUTOMATA

- Given the following state diagram of a finite automaton M_1.



$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, "0"] \arrow[d, "1"] & q_1 \arrow[loop above, "0"] \arrow[r, bend left, "1"] & q_2 \arrow[loop above, "1"] \arrow[l, bend left, "0"] \\
& q_3 \arrow[loop right, "{0, 1}"] & &
\end{tikzcd}$$

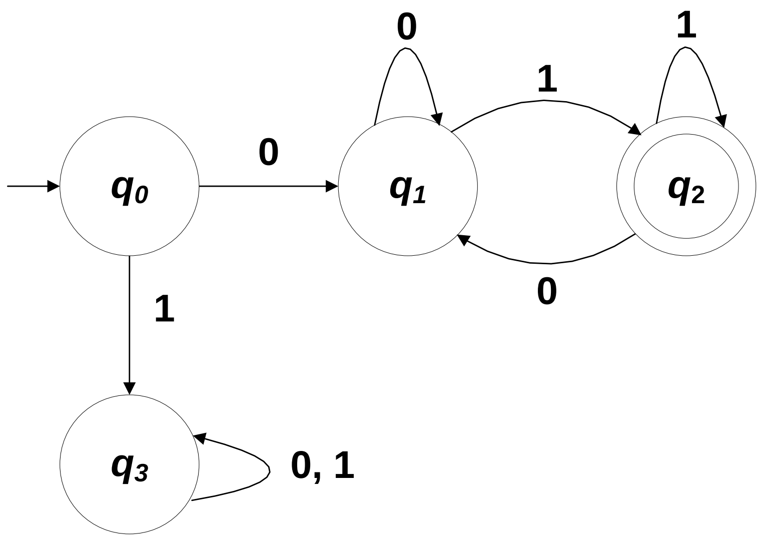
Take note of the following facts about M_1:

1. It has four states labeled q_0, q_1, q_2, and q_3.
2. It has an start or initial start state which is q_0.
3. It has one final or accept state which is q_2.
4. It has edges connecting the states. These edges represent transitions.

DETERMINISTIC FINITE AUTOMATA

Consider the following process flow for M_1 for input 00101:

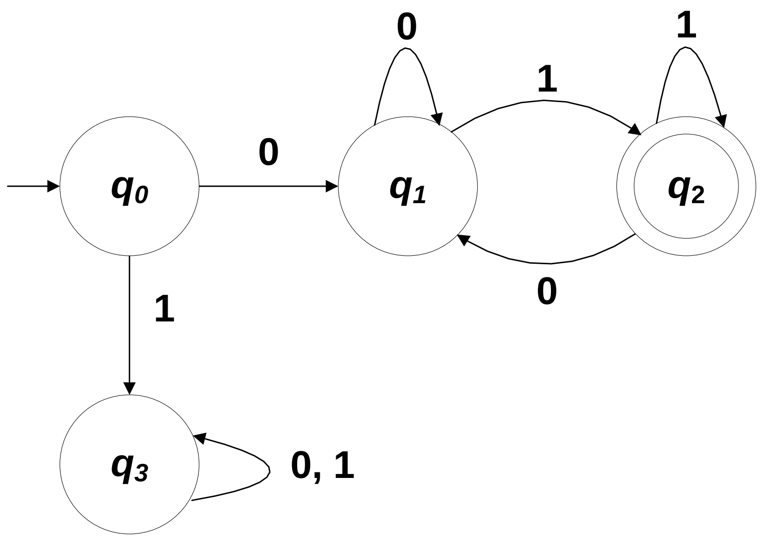
1. The machine is currently at the initial state, q_0.
2. The system receives a 0 as its input. So the system moves from q_0 to q_1.
3. The system receives another 0 as its second input. So the system stays at state q_1.
4. The system receives a 1 as its third input. So the system moves from state q_1 to q_2.



DETERMINISTIC FINITE AUTOMATA

Consider the following process flow for M_1 for input 00101:

5. The system receives a 0 as its fourth input. So the system moves from state q_2 to q_1.
6. The system receives a 1 as its fifth and last input. So the system moves from state q_1 to q_2.



Since the system is now at state q_2 and it is a final state, then it is said that the system accepts the input string 00101.

To generalize, M_1 accepts any string that starts with a 0, followed by any number of 0s and 1s, as long as the last input is a 1.

FORMAL DEFINITION OF DFA

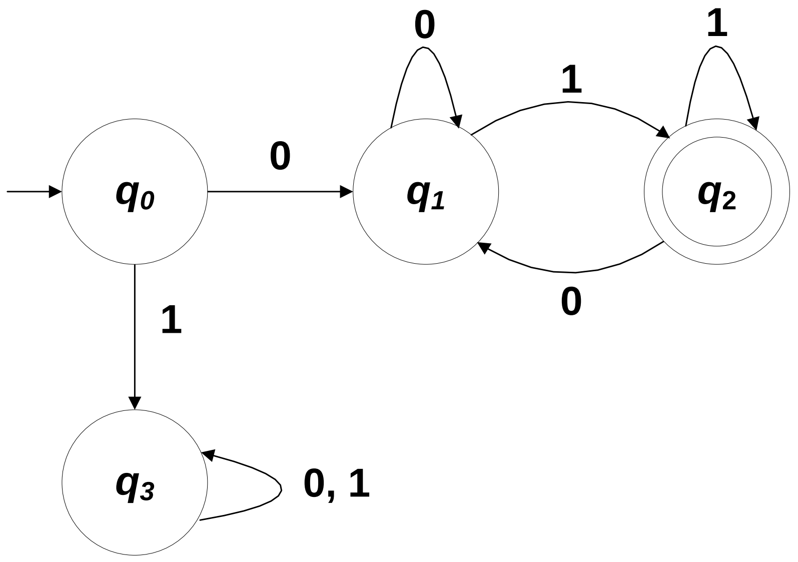
- A ***deterministic finite automaton*** (DFA) is composed of
  - A finite ***set of states*** designated as *Q*.
  - A finite ***set of symbols*** (***alphabet***) used as inputs designated as Σ.
  - A ***transition function*** designated as δ.
  - A ***start state*** designated as q_o.
  - A set of ***final states*** or ***accept states*** designated as *F*.

TRANSITION FUNCTION (δ)

- The ***transition function*** is a table that specifies when state transitions (movement from one state to another) are carried out.

- For the DFA M_1, its transition function δ is:

	0	1
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_1	q_2
q_3	q_3	q_3



For each input symbol, the automaton can move to only one state from any given current state. This is what is meant by the term *"deterministic."*

iACADEMY
SCHOOL OF COMPUTING • SCHOOL OF BUSINESS • SCHOOL OF DESIGN

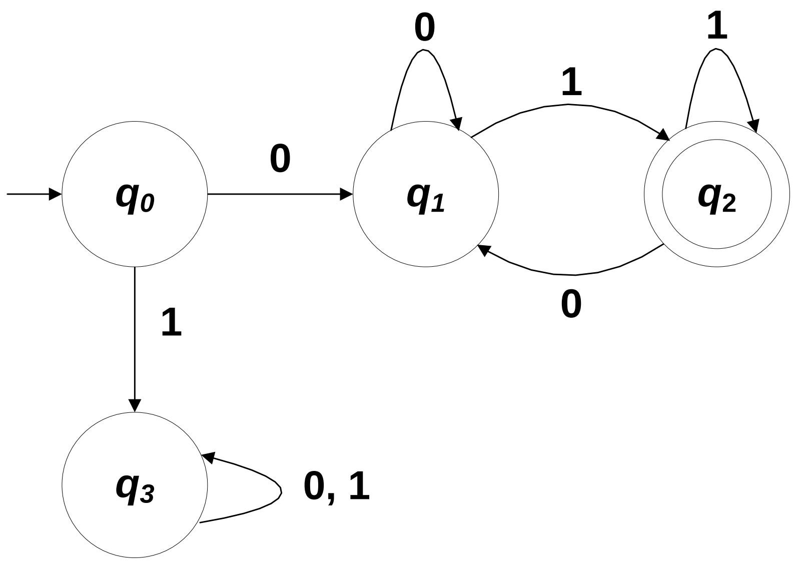
FORMAL DEFINITION OF DFA

- Using the formal definition of a DFA, M_1 can now be described as a 5-tuple $M_1 = \{Q, \Sigma, \delta, q_o, F\}$ where:

1. $Q = \{q_o, q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. δ:

	0	1
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_1	q_2
q_3	q_3	q_3

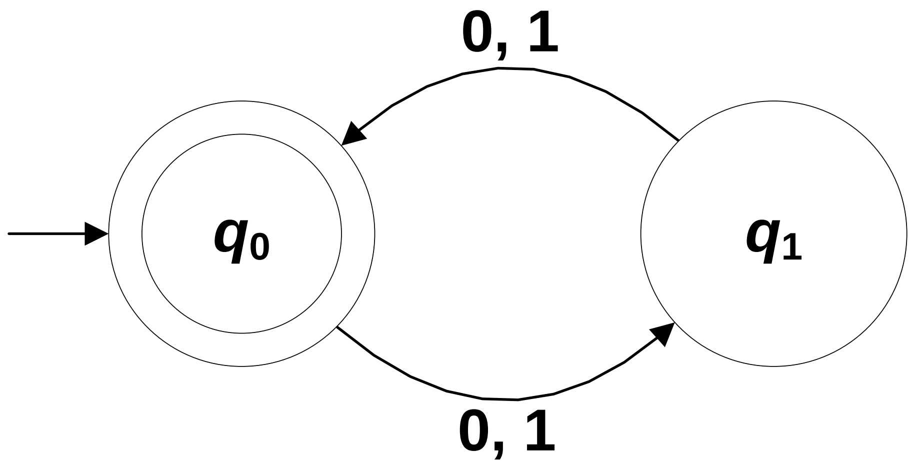
4. Start State = q_o
5. $F = \{q_2\}$



FORMAL DEFINITION OF DFA

- Another example of a DFA:

DFA M_2


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, bend right, "{0, 1}"'] & q_1 \arrow[l, bend right, "{0, 1}"']
\end{tikzcd}$$

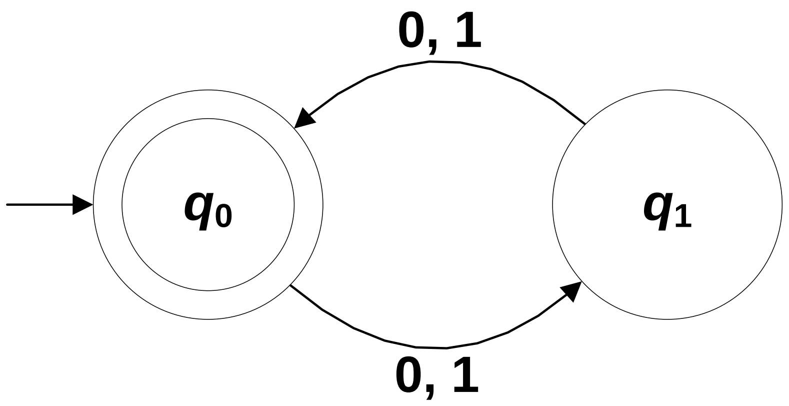
FORMAL DEFINITION OF DFA

- For DFA M_2:

1. $Q = \{q_0, q_1\}$
2. $\Sigma = \{0, 1\}$
3. δ:

	0	1
q_0	q_1	q_1
q_1	q_0	q_0

4. Start State = q_0
5. $F = \{q_0\}$



M_2 accepts all strings (including the empty string ε) whose length is even.

LANGUAGE OF THE AUTOMATON

- The set of strings that an automaton accepts is called the ***language of the automaton***.
- For example, the language of DFA M_2 is the set of all strings over the alphabet Σ = {0, 1} whose length is even.
- If L is the language recognized by a DFA M, then it is designated as $L(M)$.
- So for the language of M_2, it can be mathematically described as:

$$L(M_2) = \{w \mid \text{the length of } w \text{ is even}\}$$

REGULAR LANGUAGE

- A language that is recognized by a DFA is called a ***regular language***.
- For example, the language composed of all strings over the alphabet $\Sigma = \{0, 1\}$ whose length is even is a regular language because there is a DFA (specifically, M_2) that recognizes it.
- Therefore, $L(M_2)$ is a regular language.

DESIGNING DFAs

- Case Study 1: Design a DFA that accepts all strings over the alphabet $\Sigma = \{0, 1\}$ with an odd number of 1s.
  - The states of a DFA represent something the machine must "remember" about the input string.
  - Assume that several input symbols have already arrived. Upon the arrival of the next symbol, the machine does not have to remember the exact number of 1s it has received.

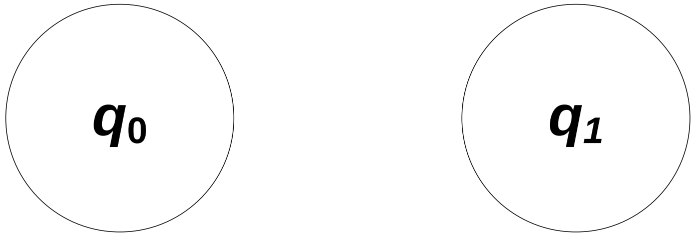
DESIGNING DFAs

- Instead, it only has to remember two things:
  1. Whether the number of 1s in the input string at any point is odd.
  2. Whether the number of 1s in the input string at any point is even.

- This implies that only two states are needed. One to represent the fact that the current number of 1s is odd, and the other to represent even.

- Let these two states be:

  q_0 = current number of 1s is even.
  q_1 = current number of 1s is odd.

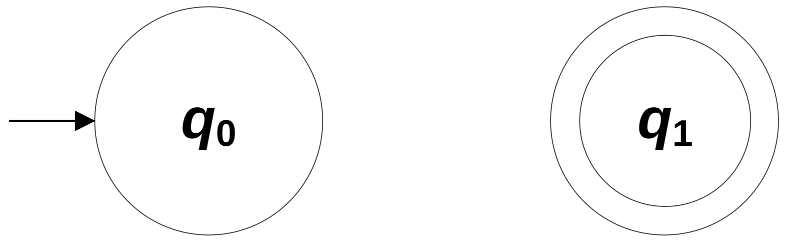


DESIGNING DFAs

- The state in which the number of 1s is even (q_o) will be designated as the start state.

  This is because while there is no input symbol yet, the number of 1s is zero (even).

- Obviously, the state in which the number of 1s is odd (q_1) will be designated as the final state.

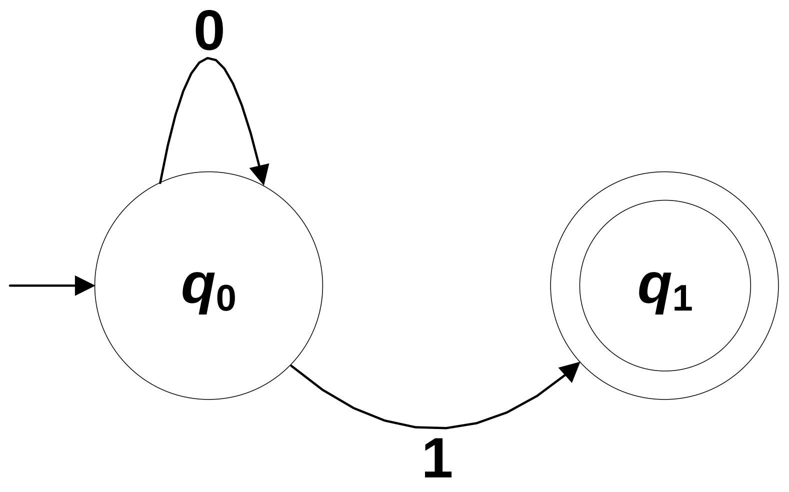


DESIGNING DFAs

– Assume that the DFA is in state q_o (number of 1s is even):

  If it receives a 0, it remains in that state since the number of 1s is still even.

  If it receives a 1, it moves to state q_1 since the number of 1s is now odd

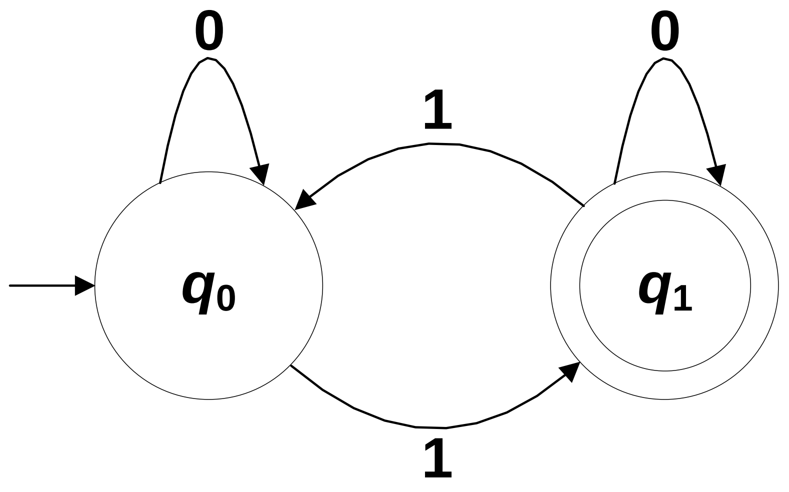

$$\begin{tikzcd} \arrow[r] & q_0 \arrow[loop above, "0"] \arrow[r, bend right, "1"'] & \boxed{q_1} \end{tikzcd}$$

DESIGNING DFAs

– Assume that the DFA is in state q_1 (number of 1s is odd):

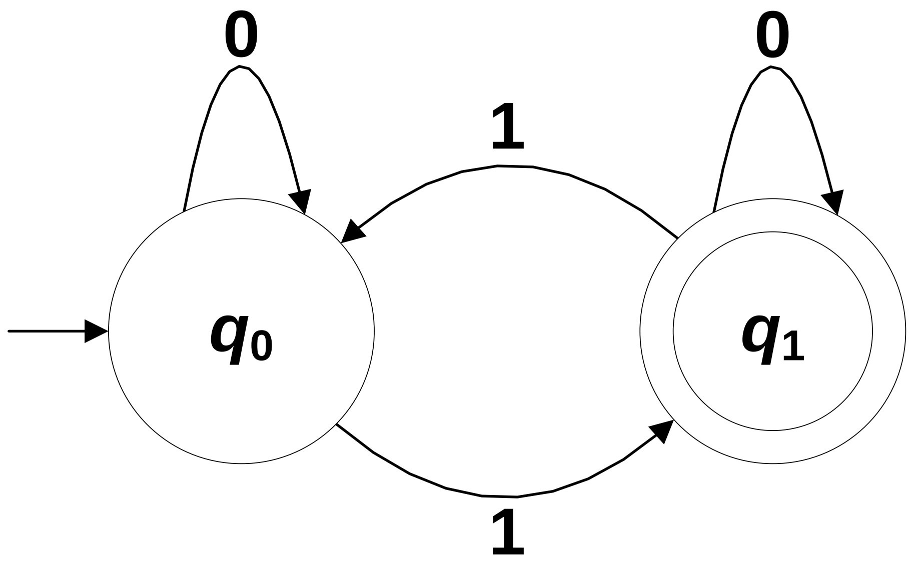
  If it receives a 0, it remains in that state since the number of 1s is still odd.

  If it receives a 1, it moves to state q_0 since the number of 1s is now even.



DESIGNING DFAs

– The state diagram for the required DFA:


$$\begin{tikzcd} \arrow[r] & q_0 \arrow[loop above, "0"] \arrow[r, bend right, "1"'] & \boxed{q_1} \arrow[loop above, "0"] \arrow[l, bend right, "1"'] \end{tikzcd}$$