American International University-Bangladesh

Faculty of Science and Technology

AIUB

Department of Computer Science

CSC3113: Theory of Computation

Lecture: # 2 Week: # 1 Semester: Summer 2020-2021

Deterministic Finite Automaton (DFA)

Instructor: Sharfuddin Mahmood, Assistant Professor,

Department of Computer Science, Faculty of Science & Technology.

smahmood@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.

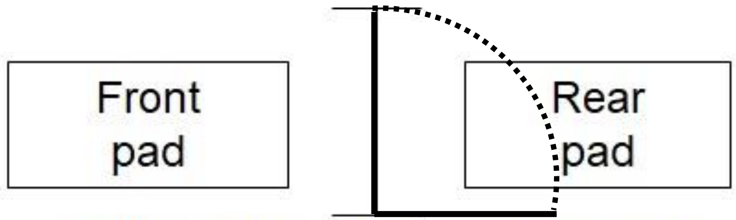


Figure: Top view of an automatic door

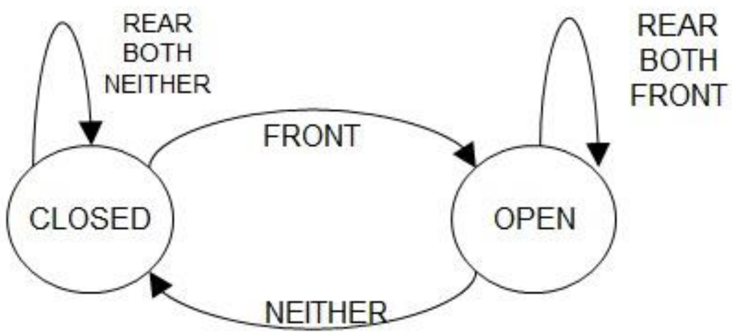


Figure: State diagram for Automatic door controller

		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

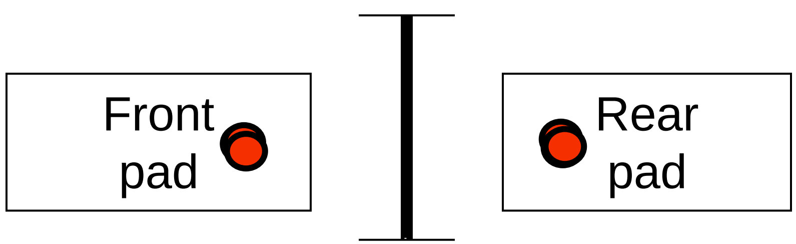


Figure: Top view of an automatic door

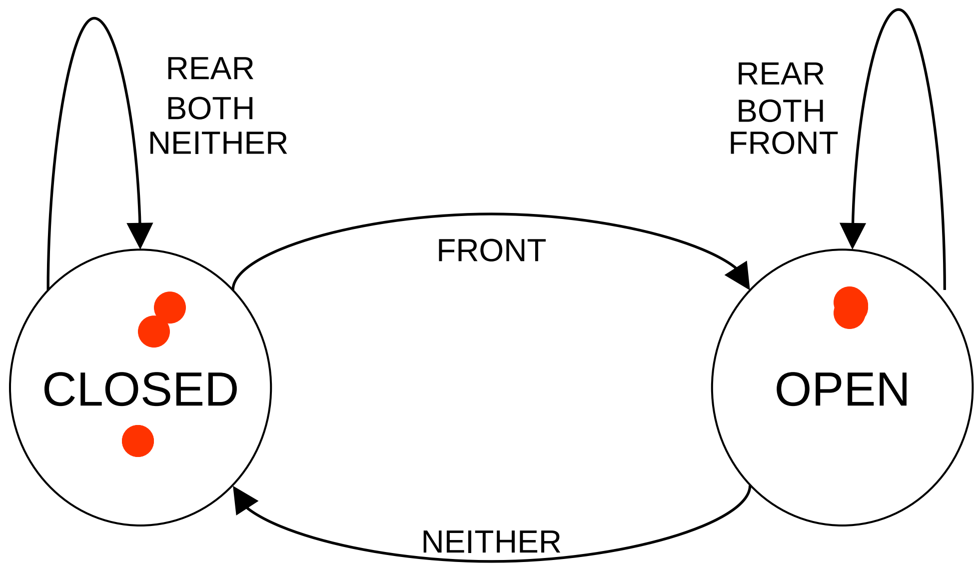


Figure: State diagram for Automatic door controller

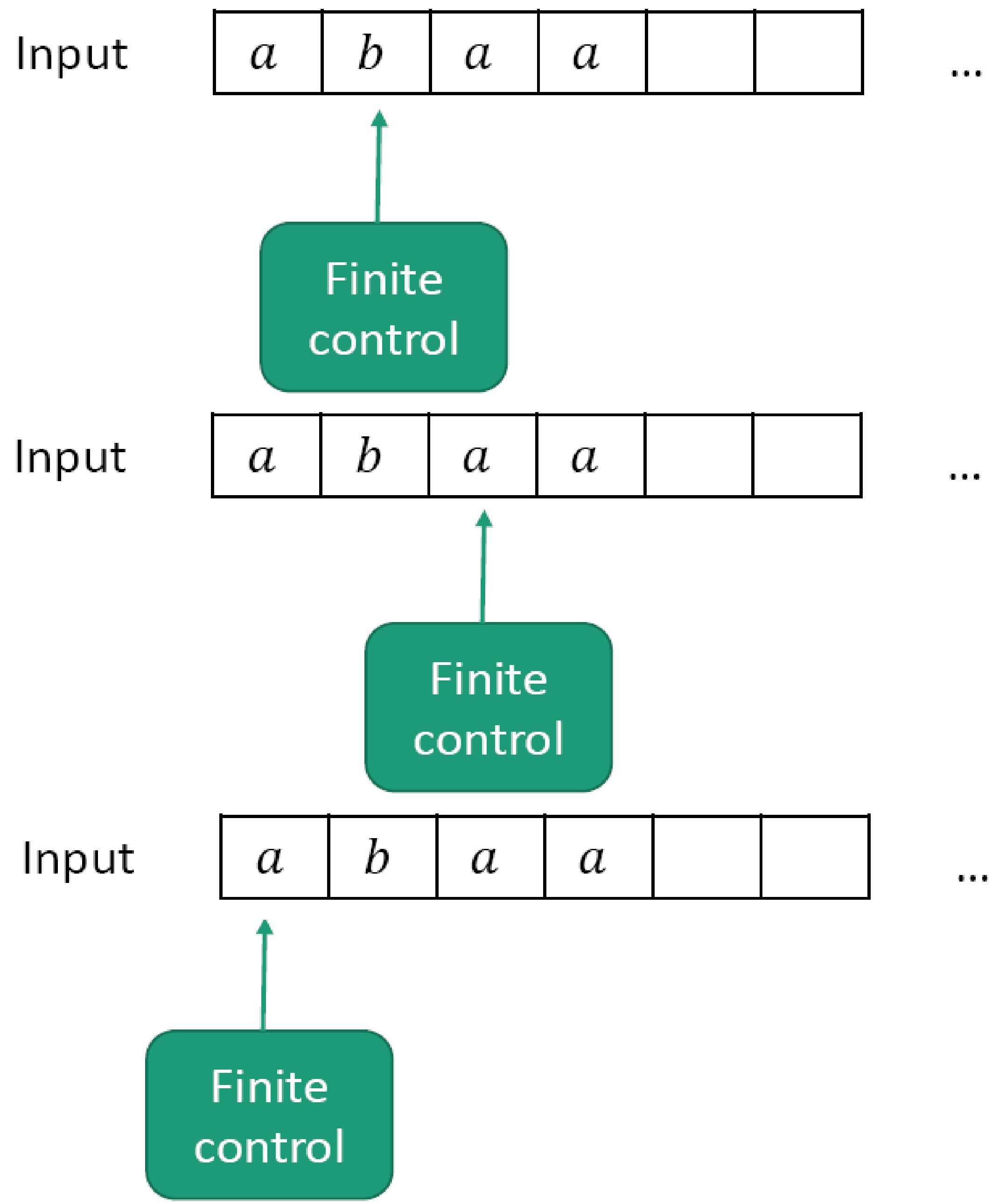
Machine Models

- 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 –

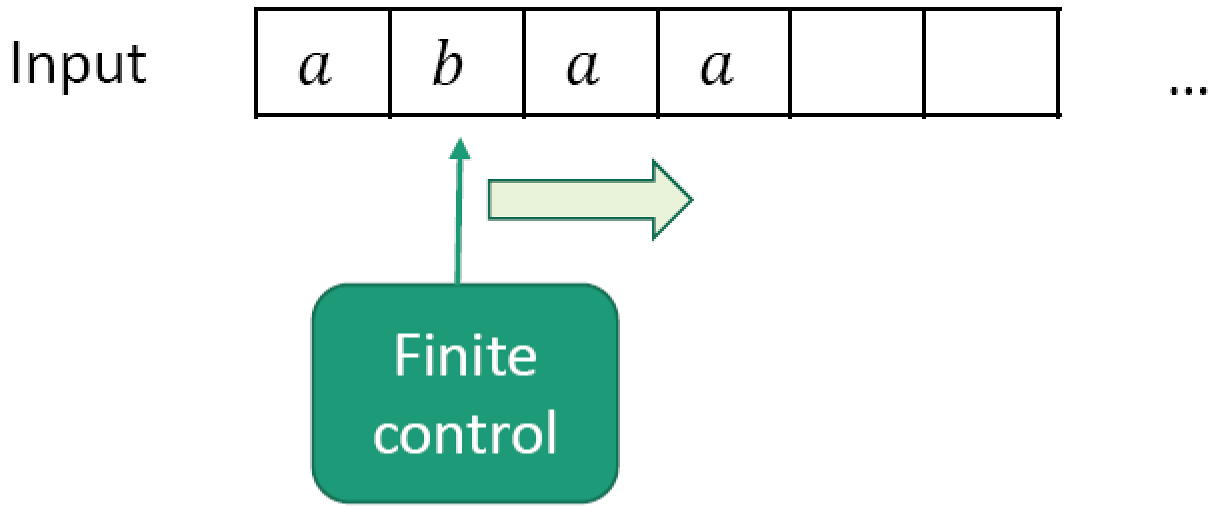
- read a given input string
- have a finite number of states, and
- transition between states using fixed rules.

Finite State Machine



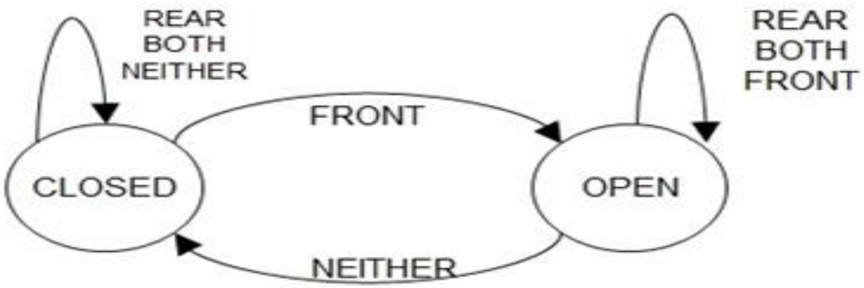
Machine Model

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

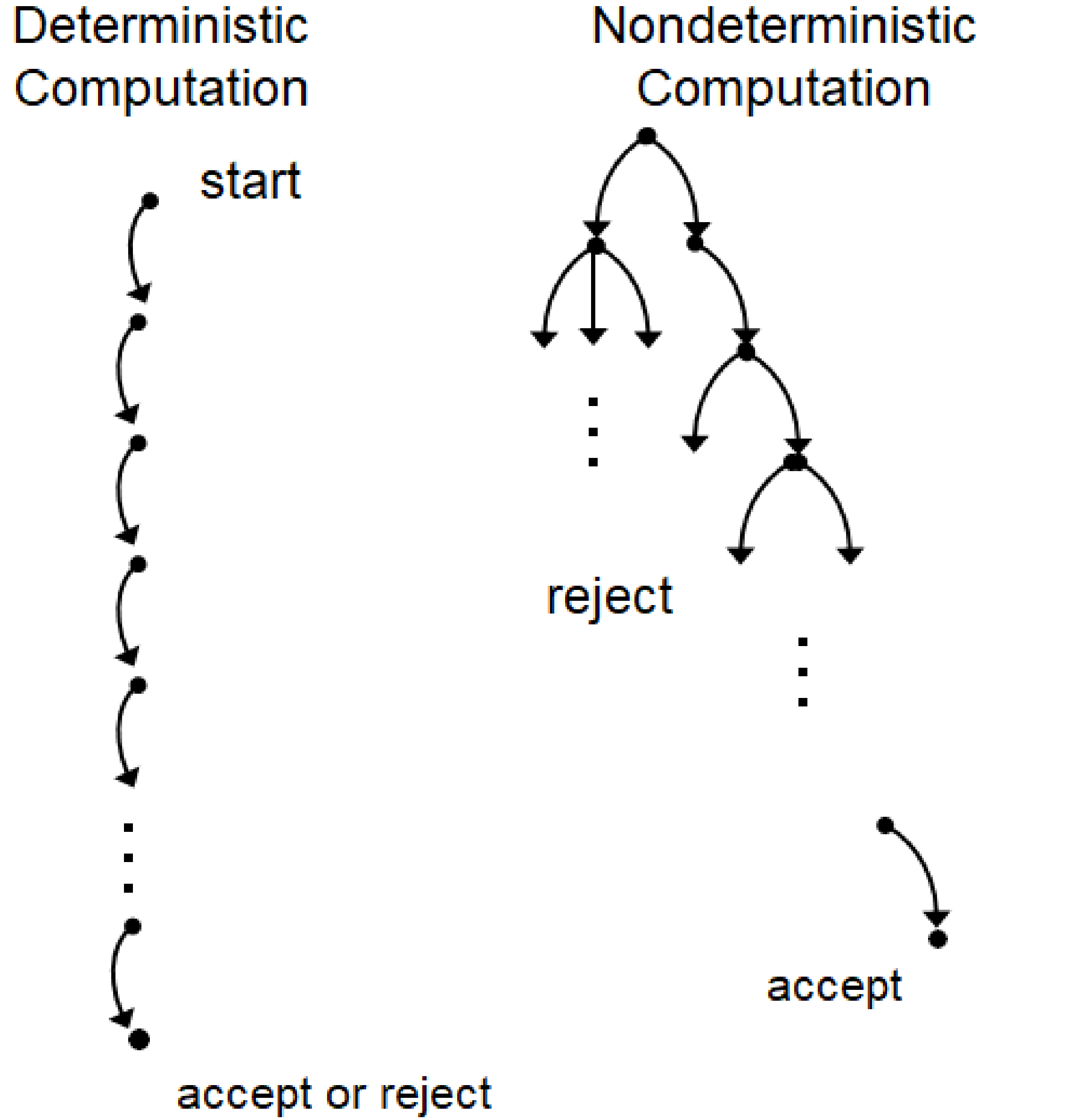


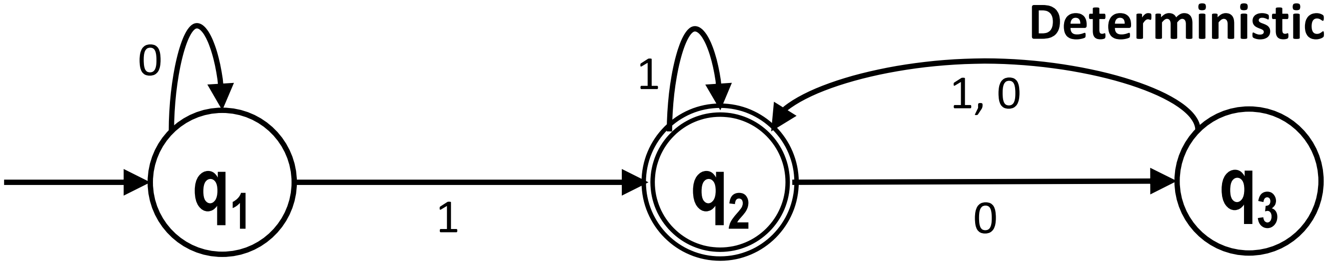
Figure: Deterministic and nondeterministic computations with an accepting branch

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?

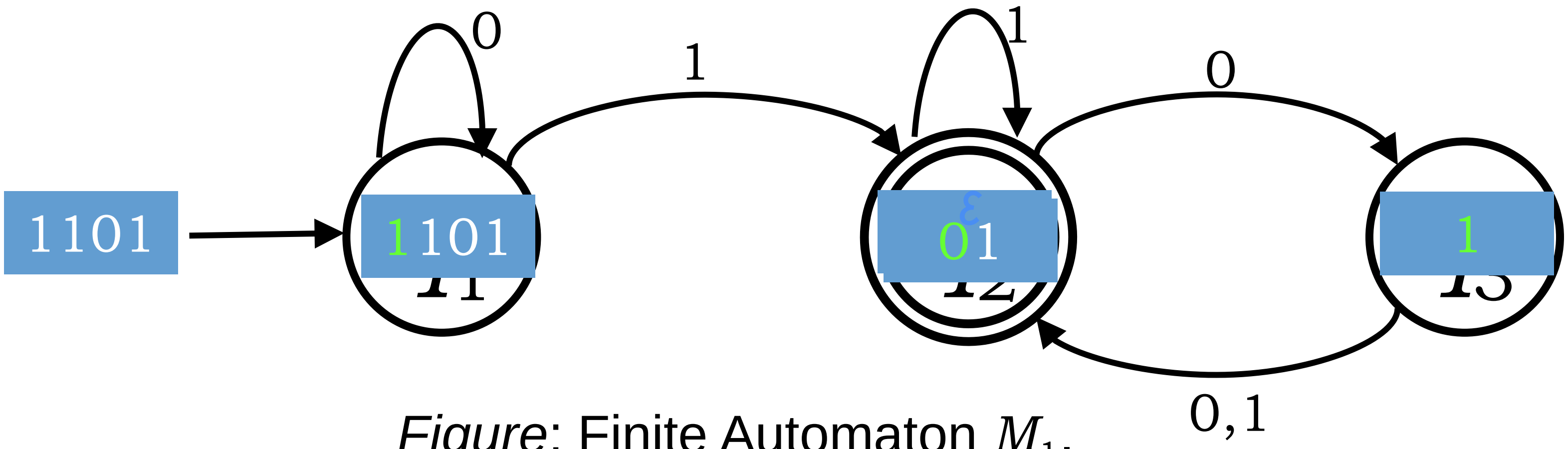


Figure: Finite Automaton M_1.

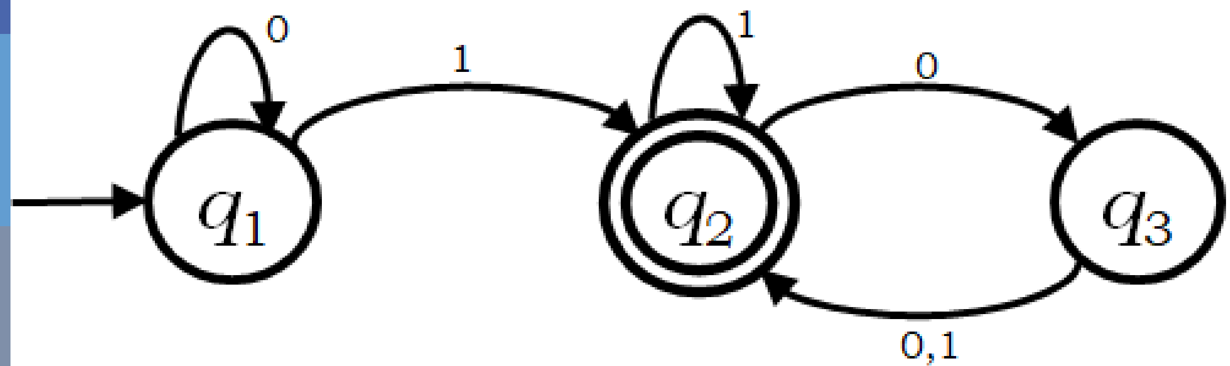
- After feeding the input string **1101** to the above machine, the processing proceeds as follows –
  - Start in state q_1;
  - Read 1, follow transition from q_1 to q_2;
  - Read 1, follow transition from q_2 to q_2;
  - Read 0, follow transition from q_2 to q_3;
  - Read 1, follow transition from 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***,
  - Σ 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 –
  - $Q = \{q_1, q_2, q_3\}$,
  - $\Sigma = \{0, 1\}$,
  - $q_0 = q_1$,
  - $F = \{q_2\}$,
  - δ *is describe as* –

$$\boxed{\begin{array}{l} \delta(q_1,0) = q_1,\ \delta(q_1,1) = q_2, \\ \delta(q_2,0) = q_3,\ \delta(q_2,1) = q_2, \\ \delta(q_3,0) = q_2,\ \delta(q_3,1) = q_2. \end{array}}$$

Transition Function

OR

δ	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 Σ), 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 $L = \{w : M$ ***accepts*** $w\}$.

SIMULATION – DFA COMPUTATION

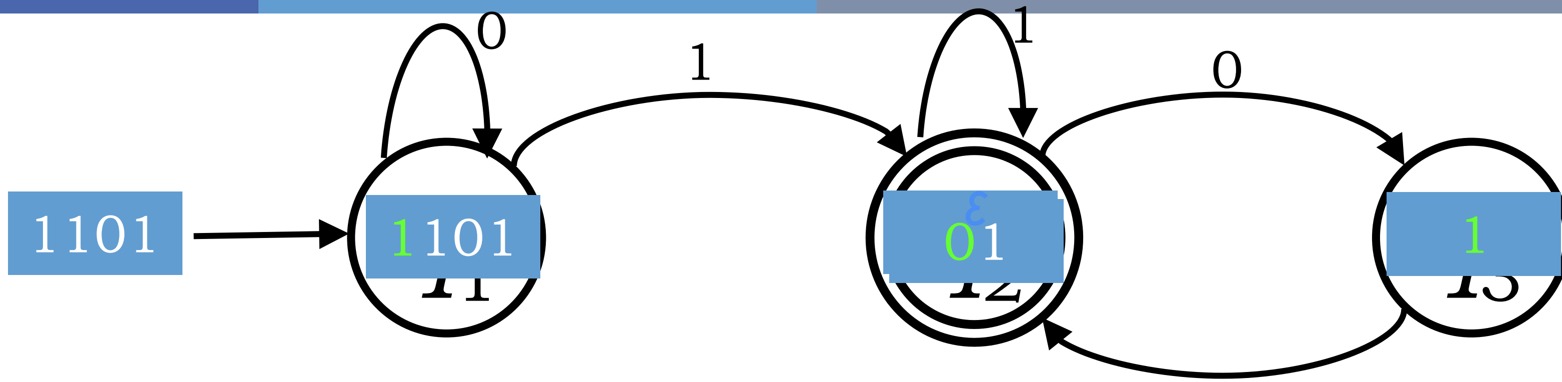


Figure: Finite Automaton M_1.

- M=(Q, Σ, δ, q_0, F)=($\{q_1, q_2, q_3\}$, $\{0, 1\}$, δ, 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, (3rd ed),
  - DFA; All Exercises;
- Elements of the Theory of Computation, Papadimitriou (2nd ed), Chapter 1.

American International University-Bangladesh

Faculty of Science and Technology AIUB

Department of Computer Science

CSC3113: Theory of Computation

Lecture: # 3 Week: # 2 Semester: Summer 2020-2021

Deterministic Finite Automaton (DFA)
Regular Language

Instructor: Sharfuddin Mahmood, Assistant Professor,

Department of Computer Science, Faculty of Science & Technology.

smahmood@aiub.edu

Lecture Outline

- Practice Problem with DFA.
  - Exercise solving from the book.
  - DFA design Issues.
- Regular Language
  - Closure
  - Operations
  - Example: Regular Language closed under Union Operation

LEARNING OBJECTIVE

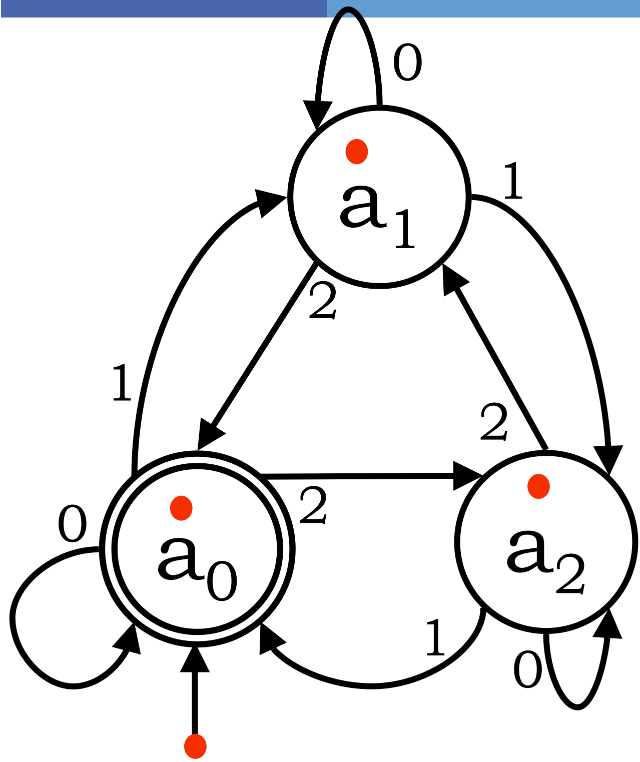
- Understand, learn & practice with example
  - Practice designing DFA.
- Learn Language, Regular Language & Regular Operations
- Analyze Closure under regular Operation
- Build one machine from 2 machine using closure under union

LEARNING OUTCOME

ALL OUTCOME ARE REPRESENTED WITH EXAMPLES

- Understand, learn & practice design of DFA.
- Analyze and Design new machine model from one or more machine model(s) using closure rule of regular operation (example: Union).
- In doing so, understand that, there are certain cases where DFA might not give a desired machine model.

Designing DFA – Example 1



- Input example: 01120101
- Present State:

a_2

- Input symbol:

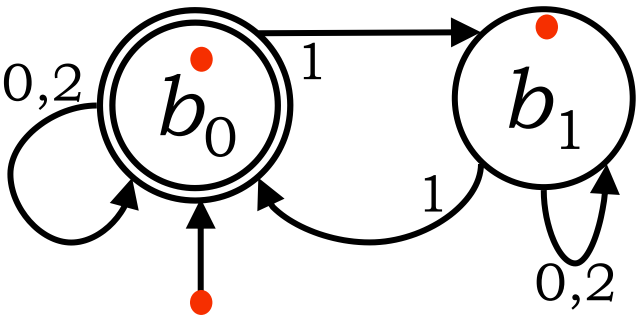
Accepted

ε

- **Alphabet Σ={0,1,2}.**
- **Language A_1 = {*w* : the sum of all the symbols in *w* is multiple of 3 }.**
  - Can be represented as follows –
    - *S*= the sum of all the symbols in *w*.
    - If *S* modulo 3 = 0 then the sum is multiple of 3.
    - So the sum of all the symbols in *w* is 0 modulo 3.
    - We can model a_i as state representing *S* modulo 3 = *i*.
- The finite state machine $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, where –
  - $Q_1 = \{a_0, a_1, a_2\}$,
  - $q_1 = a_0$,
  - $F_1 = \{a_0\}$,
  - δ_1

	0	1	2
a_0	a_0	a_1	a_2
a_1	a_1	a_2	a_0
a_2	a_2	a_0	a_1

Designing DFA – Example 2



- Input example: 01120101
- Present State:

b_1

- Input symbol:

ε

Accepted

- **Alphabet Σ={0,1,2}.**
- **Language A_2 = {*w* : the sum of all the symbols in *w* is an even number }.**
  - Can be represented as follows –
    - *S*= the sum of all the symbols in *w*.
    - If *S* modulo 2 = 0 then the sum is even.
    - Here, b_i is modeled as *S* modulo 2 = *i*.
- The finite state machine $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, where –
  - $Q_2 = \{b_0, b_1\}$,
  - $q_2 = b_0$,
  - $F_2 = \{b_0\}$,
  - δ_2

	0	1	2
b_0	b_0	b_1	b_0
b_1	b_1	b_0	b_1

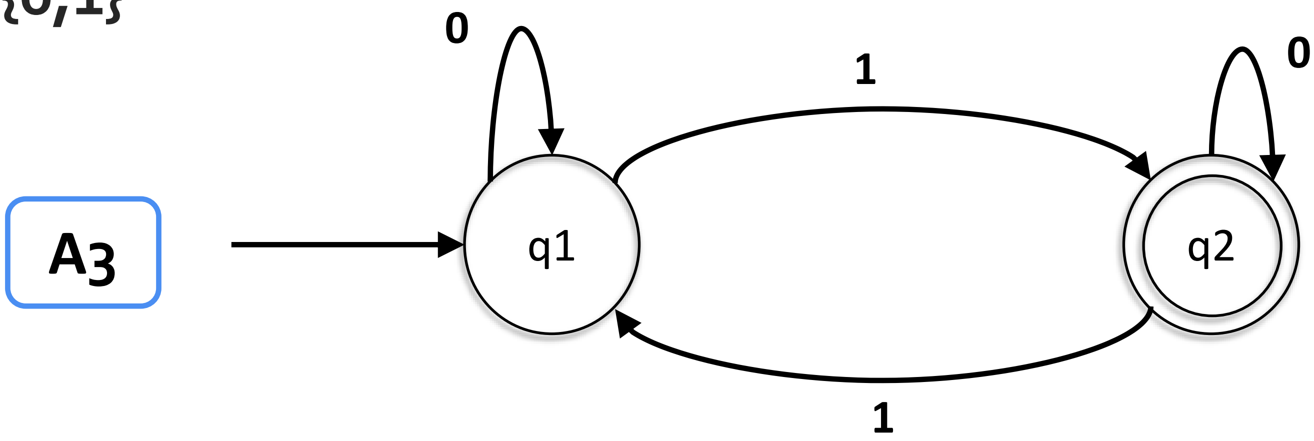
DFA DESIGN ISSUES

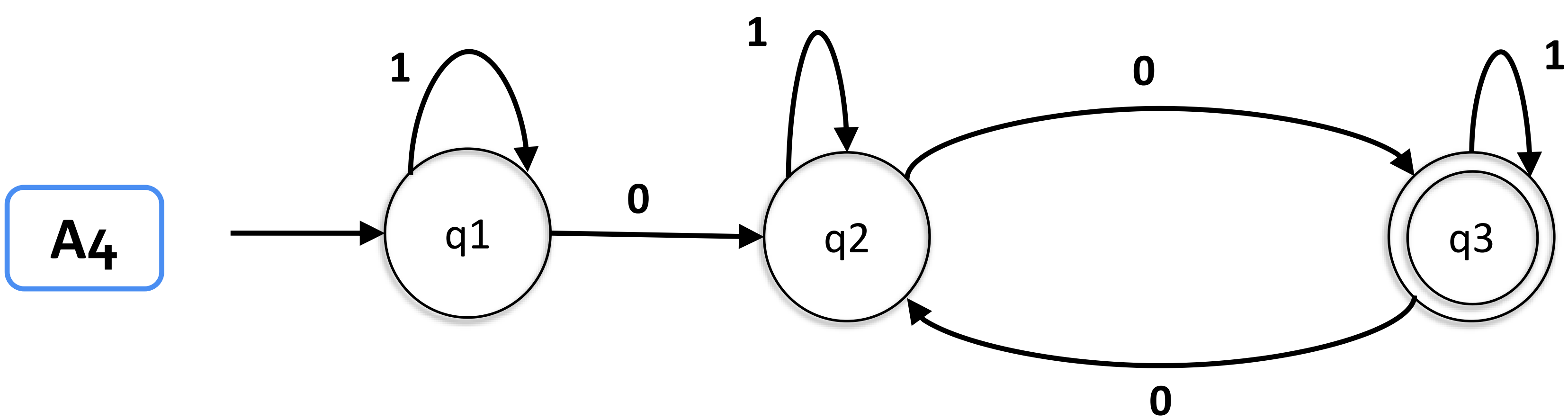
- The key concept here is to understand the given language.
  - Type/pattern of input strings that the language gives.
  - Match the pattern from left to right with the states & transitions of the state diagram, one by one.
  - First match the necessary conditions with the transition then complete the diagram with rest of the transition maintaining the rules.
  - Maintain the rules –
    - From each state there must be exactly one transition for each input symbol.
    - Self-loop represents zero or more number of occurrences of the input symbol.
    - There can be one or more number of final states.
  - Make some of your own input strings based on the given language. Few that should be ACCEPTED and few that should be REJECTED. Test them on your state diagram after completing or while drawing the state diagram and update accordingly.
- Let us practice some DFA in the following slides.

A3 ={w : w is a binary string containing an odd number of 1s}.

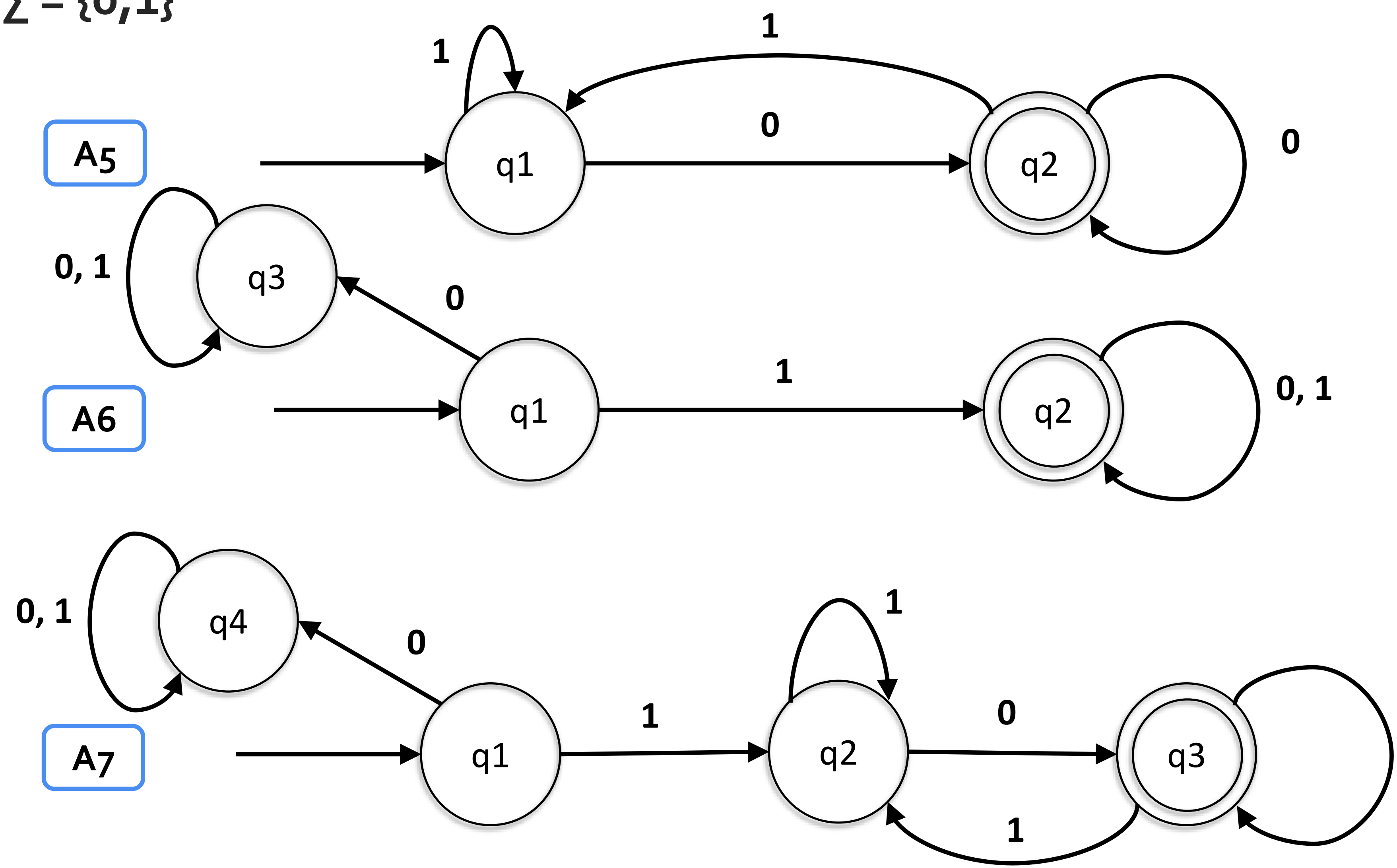
A4 ={w : w is a binary string containing an even number of 0s}.

∑ = {0,1}





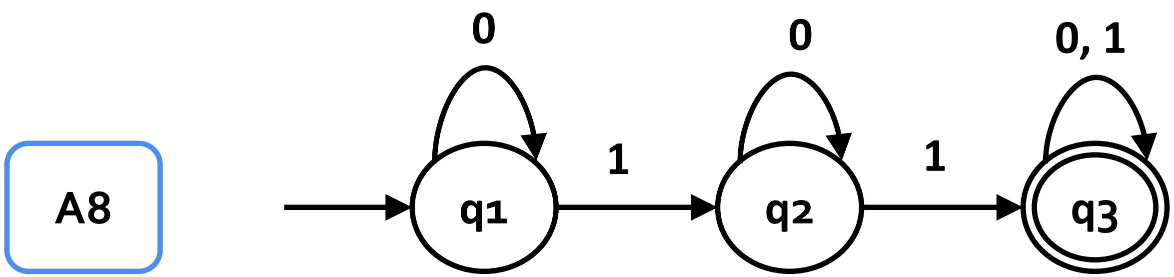
A5 ={w| w ends with a *0*}.
A6 ={w| w begins with a *1*}.
A7 ={w| w begins with a *1* and ends with a *0*}.
$\Sigma = \{0,1\}$

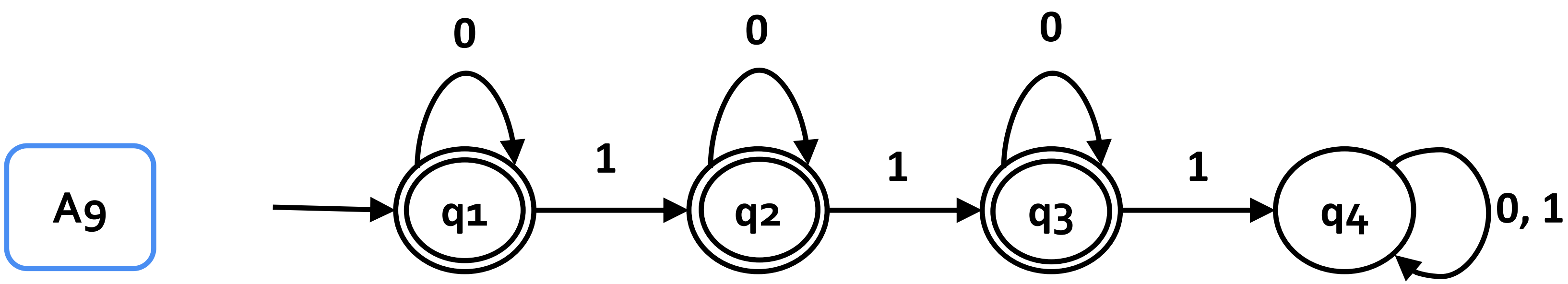


A8 ={w| w has at least two *1s*}.

A9 ={w| w has at most two *1s*}.

∑ = {0,1}

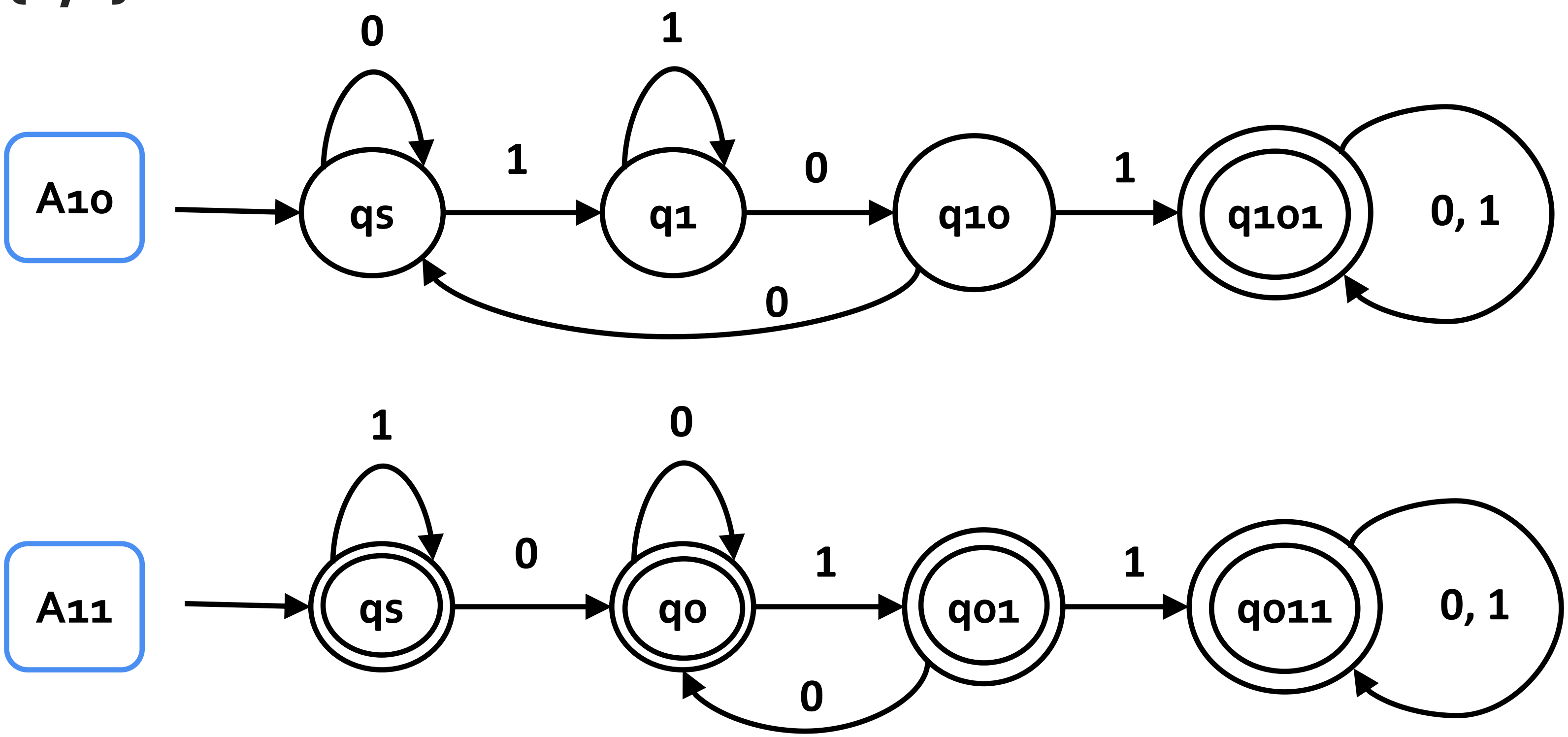




A10 ={w| w has substring 101}.

A11 ={w| w has substring 011}.

$\sum = \{0,1\}$



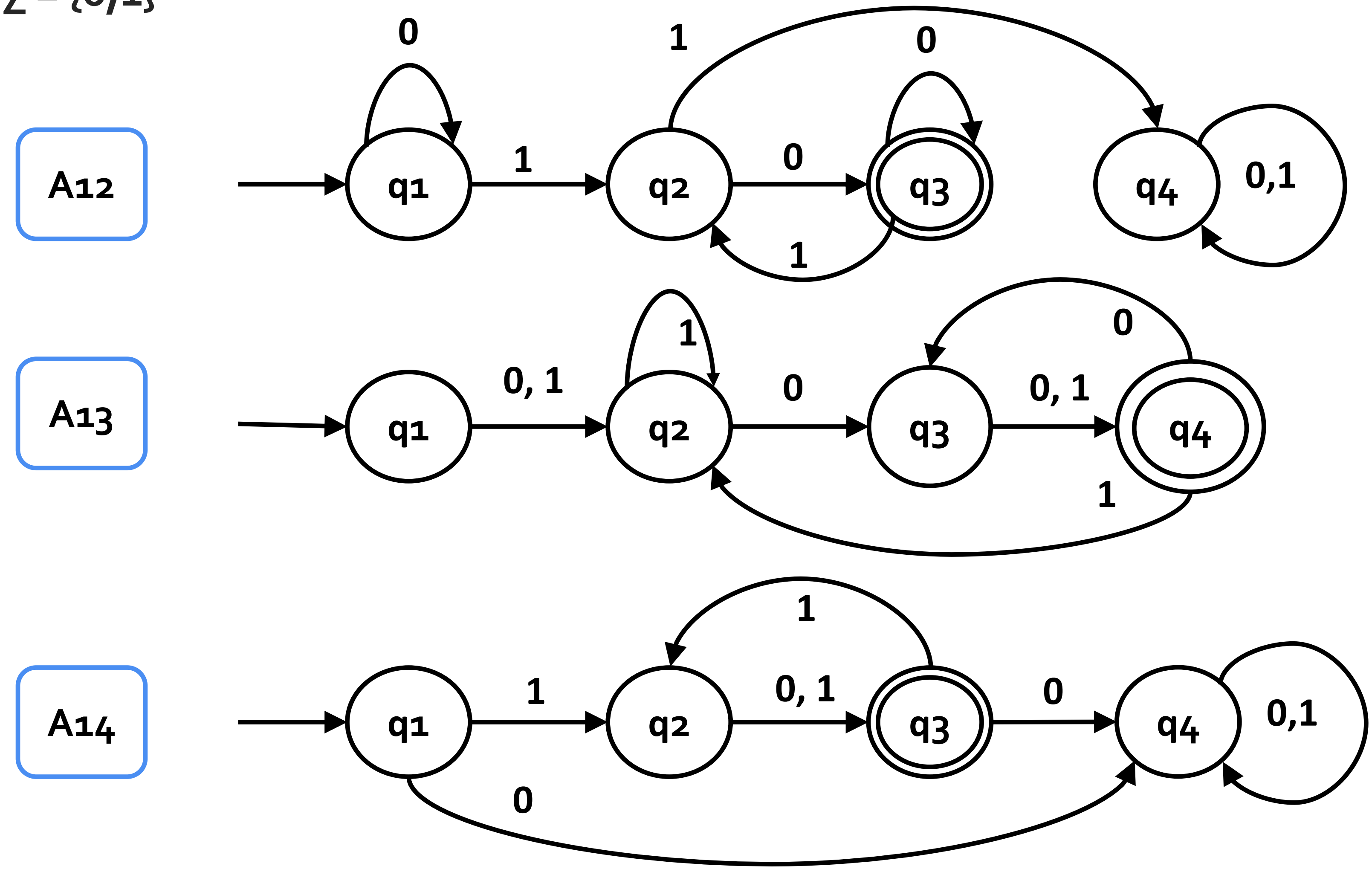
What happens for the language, **A11 ={w| w does not have substring 011}**?

A12={w| each *1* in w is followed by at least one *0*}.

A13 ={w | The length of w is at least three and have *0* in 2nd last position}

A14 ={w : The length of w is even and contains *1* in every odd position}

$\sum = \{0,1\}$



REGULAR OPERATIONS

AN ANALOGY

- In algebra, we try to identify operations which are common to many different mathematical structures.
- Example:
  - The integers $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ are closed under –
    - Addition: $X + Y$
    - Multiplication: $X \times Y$
    - Negation: $-X$
    - ...but NOT Division: X/Y
- We'd like to investigate similar closure properties of the class of regular languages

Regular Operations on Languages

- Let $A, B \subseteq \sum^*$ be languages.
- Union: $A \cup B = \{w \in \sum^* | w \in A \; or \; w \in B\}$.
- Concatenation: $A \circ B = \{wv | w \in A \; and \; v \in B\}$
- Star: $A^* = \{w_1 w_2 w_3 \ldots w_n | n \geq 0 \; and w_i \in A \; for \; i = 1 \ldots n\}$
- Complement: $\bar{A} = \{w \in \sum^* | w \notin A\}$
- Intersection: $A \cap B = \{w \in \sum^* | w \in A \; and \; w \in B\}$
- Reverse: $A^R = \{w_1 w_2 w_3 \ldots w_n | \; w_n w_{n-1} \ldots w_2 w_1 \in A\}$

REGULAR LANGUAGE

- A language is called a ***regular language*** if some finite automaton recognizes it.
- Regular Operations: Let A={good, bad}, B = {boy, girl}.
- Basic 3 operations used to study the properties of the regular languages –
  - ***Union***: $A \cup B = \{x : x \in A \text{ or } x \in B\}$ = {good, bad, boy, girl}.
    - Takes all the strings in both A and B and lumps them together into one language.
  - ***Concatenation***: $A \bullet B = \{xy : x \in A \text{ and } y \in B\}$ = {goodboy, goodgirl, badboy, badgirl}.
    - Attaches a string from A in front of a string B in all possible ways to get the strings in the new language.
  - ***Star***: $A^* = \{x_1x_2 \ldots x_k : k \geq 0 \text{ and each } x_i \in A\}$ = {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, …}.
    - Attaching any number of strings in A together to get a string in the new language. It is a unary operation, where ε is always a member of A^* (as 'any number' also includes 0).

Closure

- A collection of objects is closed under some operation, if applying that operation to the members of the collection returns an object still in the collection.
- Theorem: The class of regular languages is closed under all three regular operations (union, concatenation, star), as well as under complement, intersection, and reverse.
  - i.e., if set *A* and *B* are regular, applying any of these operations on these sets yields a regular language.
- Next, we will prove it for *Union* operation.

Regular Language closed under Union

- We will prove it by construction.
- Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.
- Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$.
  - $Q = \{(r_1, r_2) : r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
    - $Q = Q_1 \times Q_2$. (All combination of states of machine M_1 and M_2).
  - $\Sigma = \Sigma_1 \cup \Sigma_2$.
    - But here, for simplicity, we have considered $\Sigma_1 = \Sigma_2$ to be same.
  - For each $(r_1,r_2) \in Q$ and each $a \in \Sigma$, let $\delta((r_1,r_2), a) = (\delta(r_1, a), \delta(r_2, a))$.
    - Hence δ 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 is the pair (q_1, q_2).
  - $F = \{ (r_1, r_2) : r_1 \in F_1 \text{ or } r_2 \in F_2 \}$
    - $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

DFA UNION EXAMPLE

- Let,
- $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, where –
  - $Q_1 = \{a_0, a_1, a_2\}$,
  - $\Sigma = \{0, 1, 2\}$
  - $q_1 = a_0$,
  - $F_1 = \{a_0\}$,
  - δ_1 **Machine 1**

	0	1	2
a_0	a_0	a_1	a_2
a_1	a_1	a_2	a_0
a_2	a_2	a_0	a_1

- $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, where –
  - $Q_2 = \{b_0, b_1\}$,
  - $\Sigma = \{0, 1, 2\}$
  - $q_2 = b_0$,
  - $F_2 = \{b_0\}$,
  - δ_2 **Machine 2**

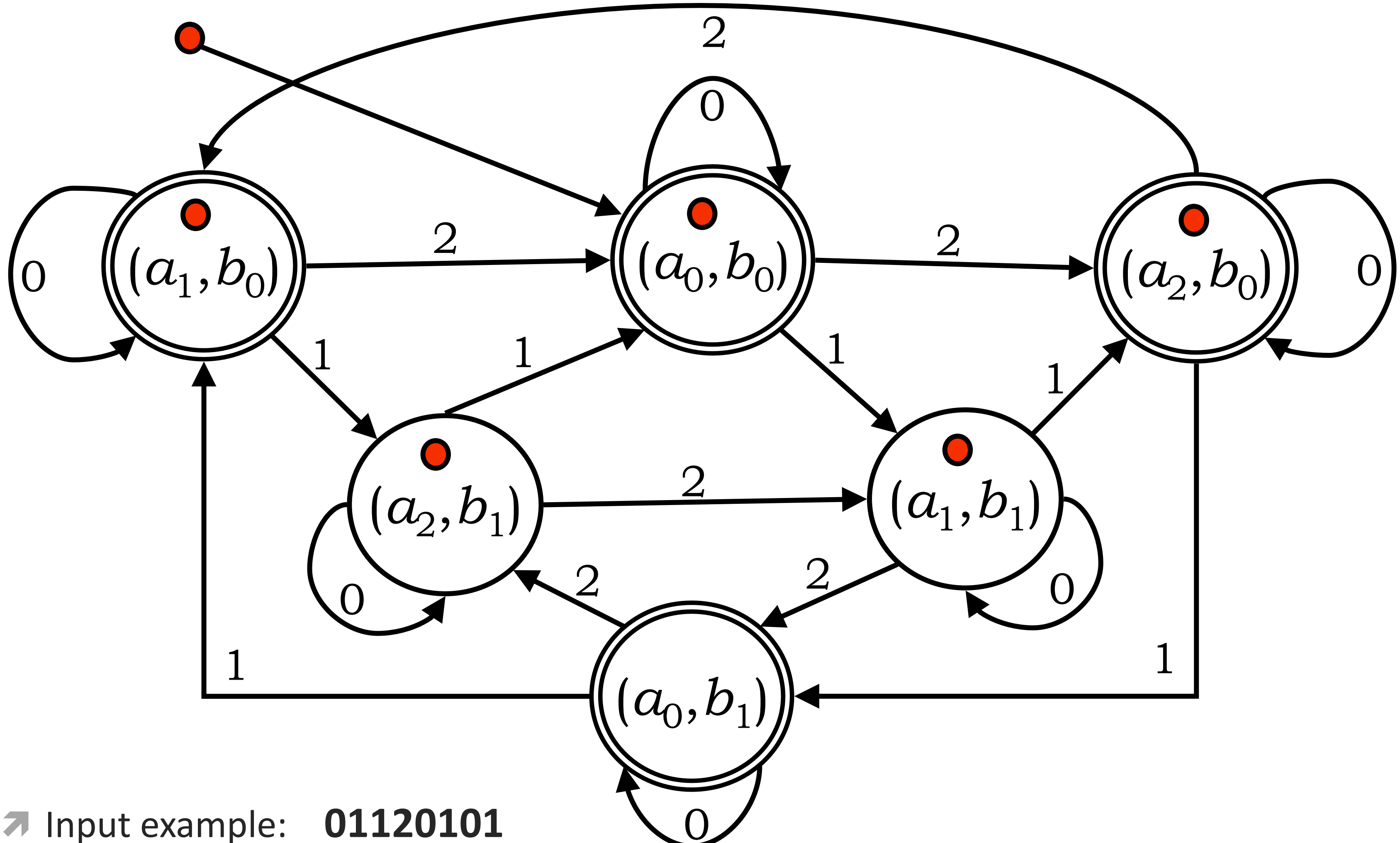
	0	1	2
b_0	b_0	b_1	b_0
b_1	b_1	b_0	b_1

- $M = (Q, \Sigma, \delta, q_0, F)$, where –
  - $Q = \{(r_1, r_2) : r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2$
    $Q = \{(a_0, b_0), (a_0, b_1), (a_1, b_0), (a_1, b_1), (a_2, b_0), (a_2, b_1)\}$,
  - $\Sigma = \Sigma_1 \cup \Sigma_2 = \{0, 1, 2\}$
  - $q_0 = (q_1, q_2) = (a_0, b_0)$
  - $F = \{ (r_1, r_2) : r_1 \in F_1 \text{ or } r_2 \in F_2 \} = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
    $F = \{(a_0, b_0), (a_0, b_1), (a_1, b_0), (a_2, b_0)\}$
  -

δ	0	1	2
(a_0, b_0)	(a_0, b_0)	(a_1, b_1)	(a_2, b_0)
(a_0, b_1)	(a_0, b_1)	(a_1, b_0)	(a_2, b_1)
(a_1, b_0)	(a_1, b_0)	(a_2, b_1)	(a_0, b_0)
(a_1, b_1)	(a_1, b_1)	(a_2, b_0)	(a_0, b_1)
(a_2, b_0)	(a_2, b_0)	(a_0, b_1)	(a_1, b_0)
(a_2, b_1)	(a_2, b_1)	(a_0, b_0)	(a_1, b_1)

- The above finite automata machine should give the same output for any given input to machine M_1 or M_2.
- Here M_1 recognizes A_1 and M_2 recognizes A_2. So M should recognize $A = A_1 \cup A_2$.
- A_1= {w: sum of all the symbols in w is multiple of 3}.
- A_2 = {w: sum of all the symbols in w is even}.

DFA Union Simulation



- Input example: **01120101**
- Input symbol:

Accepted

Closure under Concatenation

- Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.
- Construct M to recognize $A_1 \circ A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$.
- M must accept if its input can be broken into two pieces where M_1 accepts the first piece and M_2 accepts the second piece.

- ***Problem***: M doesn't know where to break its input. i.e., where the first part ends and the second begins.
- To solve the problem we will learn a new technique called *nondeterministic automaton*.

REFERENCES

- Elements of the Theory of Computation, Papadimitriou (2^{nd} ed), DFA + Exercise.
- Introduction to Automata Theory, Languages, and Computation (3^{rd} ed), DFA + Exercise.