Course: Automata Theory

Lecture 6: Deterministic and Non-Deterministic Finite Automata

Lecturer: Martha Gichuki

Course description

- The course begins with an introduction to logic and formal grammar where learners will do a recap on sets, logic and truth tables, sequences, relations and functions
- A coverage of finite state machines, Push Down automata and Turing Machines (The Church's thesis) will culminate the study of various models of computation.
- Formal language and grammar will then follow to enable learners differentiate regular and context free languages.
- An evaluation of the computability and complexity of practical computational problems which are the foundations of automata theory will then be done and the outcome will be problem description.

Learning outcomes:

Lecture 6: Deterministic and Non-Deterministic Finite Automata

At the end of the lecture the learner will be able to:

- Define Finite Automata.
- Describe Finite Automata formally.
- Differentiate Deterministic and Non-Deterministic Finite Automata

6.1 Finite Automata

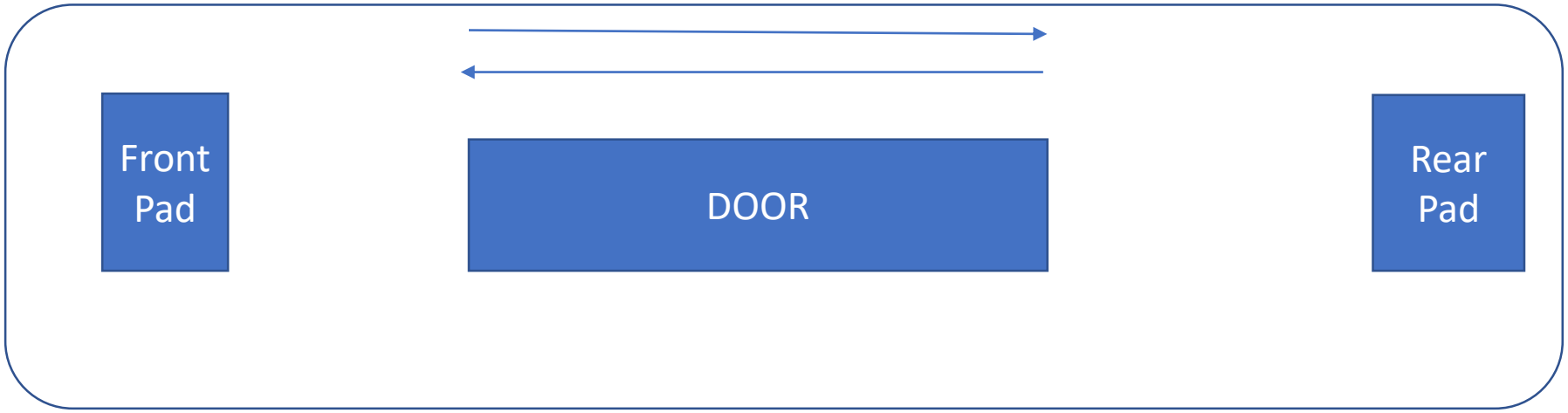
Introduction:

- A finite-state machine (finite automaton) is a simple, limited model of computation with a number of **states** and **transitions**.
- Finite Automata are good models for computers with an extremely limited amount of memory.
- Though the memory is limited, we interact with such computers all the time as they lie at the heart of various electromechanical devices.

- An automaton is a mathematical model for a Finite State Machine (FSM).
- A finite State Machine is a machine that, given an **input of symbols**, "**jumps**" or transitions through a series of states according to a transition function (which can be expressed as a table).
- This **transition function** tells the automaton **which state to go to next** given a **current state** and a **current symbol**.
- The input is read **symbol by symbol**, until it is consumed completely, after which, the automaton is said to have stopped.
- Depending on the **state in which the automaton stops**, it's said that the automaton either **accepts or rejects the input**.
- The **set of all the words accepted by an automaton** is called the **language accepted by the automaton.**

Example one: Automatic Door

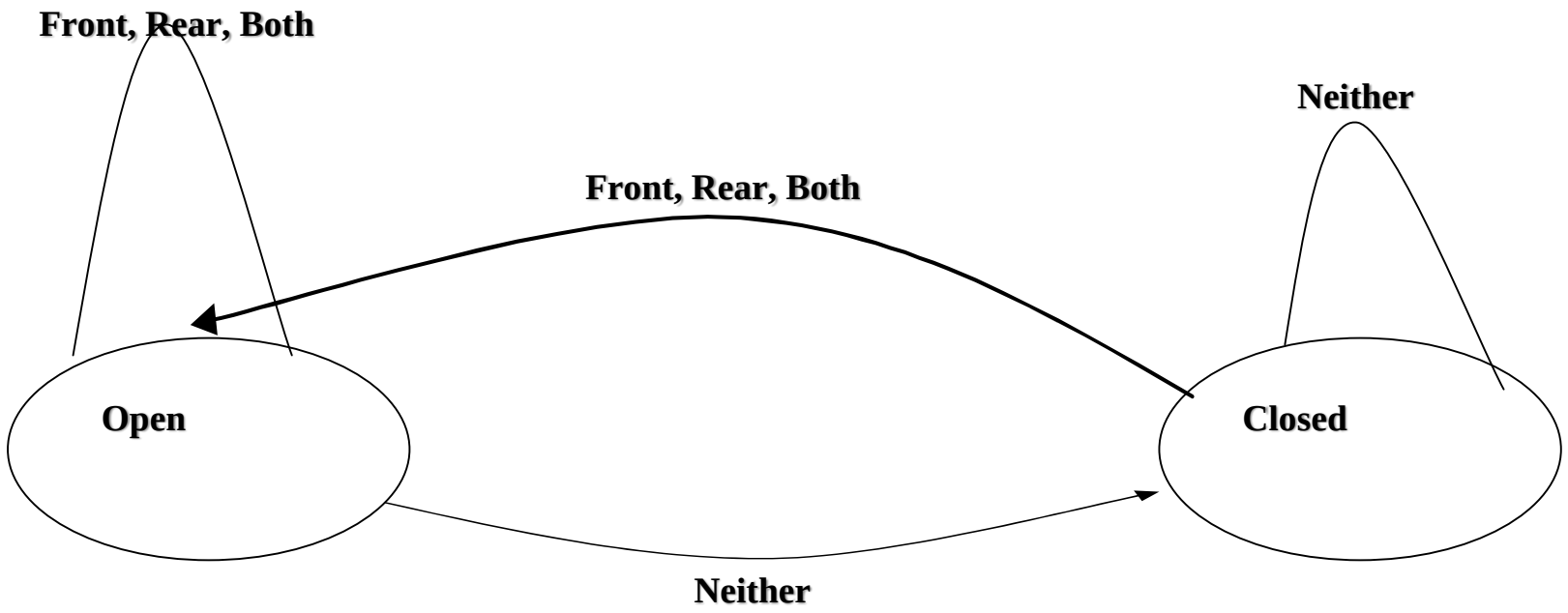
- **States**: - Door is either **open** or **closed**
- **Transitions**: - Changes the door from one state to another, based on certain controls like a **foot pad** that once **stepped on**, senses that someone is **standing** nearby.



This behaviour of the door can be represented in a **transition table** as shown below:-

Actions (Transitions)	Both	Front	Rear	Neither
States				
Closed	Open	Open	Open	Closed
Open	Open	Open	Open	Closed

The same machine can be illustrated using a state diagram as shown below: -



Example Two: Elevator Controller

- In an elevator controller a **state** may represent the **floor the elevator is on** and the **inputs** might be the **signals received from the buttons**.
- This computer might need **several bits** to keep track of this information.

Other examples:

- Controllers for various household appliances such as **dish washers and electronic thermostats, as well as parts of digital watches** and **calculators** are additional examples of **computers with limited memories**. The design of such devices requires keeping the methodology and terminology of **finite automata** in mind.

FORMAL DEFINITION OF A FINITE AUTOMATON

- Other than using state diagrams, we now define Finite automata formally.
- Though state diagrams are easier to grasp we need the formal definition for two main reasons: -

i. A formal definition is **precise** – It resolves any uncertainties about what is allowed in a finite automaton.

ii. Good notation provides a **clear way** to think and express thoughts

- From lecture 1 we indicated that a list of **five elements is called a 5-tuple**.
- The formal definition of a **finite automaton** is a **5-tuple** consisting of five parts.

i. A set of states
ii. Input alphabet showing the allowed input symbols
iii. Rules for going from one state to the other depending on the input symbol
iv. The start state and
v. A set of accept states– sometimes called final states .

The Transition Function

- We denote the transition function as **δ** *(delta sign)* to define the **rules for m**oving.
- If the finite automaton has an arrow **from a state X to a state Y labeled with the input symbol 1**, that means that, if the automaton is in **state X when it reads a 1, it then moves to state Y.**
- We can indicate the same thing with the transition function by saying **that δ(X, 1) = Y.**
- This notation is a kind of mathematical shorthand.

Types of Automatons

1. Deterministic Finite Automata (DFA)

- This is an automaton in which **each move (transition from one state to another) is uniquely determined by the current configuration**.
- If the **internal state, input and contents of the storage** are known, **it is possible to predict the future behaviour of the automaton**.
- This is said to be Deterministic Finite Automaton (DFA), **otherwise** it is Non-Deterministic Finite Automaton (NFA).
- An automaton whose output response is “Yes” or “No” is called an **acceptor.**

The **formal description of a DFA is a <u>five</u>-tuple**

- **M= (Q,∑** *(sigma sign)* **, δ** *(delta sign)* **, q_0, F);**

  where: -

Q = Finite set of Internal States,

∑= Finite Set of Symbols "Input Alphabet" ,

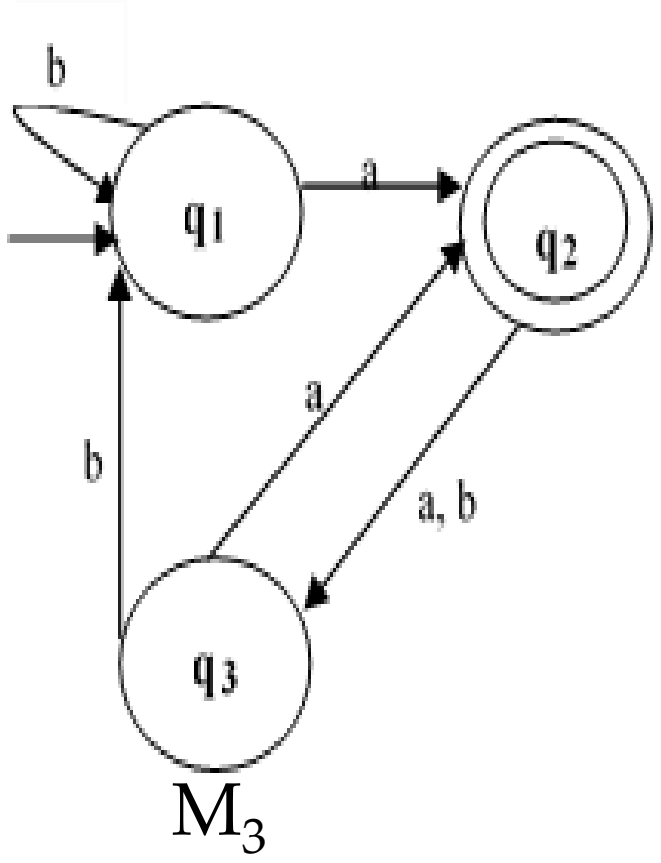
δ : Q X ∑→Q is the Transition Function

q_0∈Q is the Initial State

F ⊆ Q is the **<u>Set</u>** of Final States

- The input mechanism can move from left to right and reads exactly one symbol on each step.
- The transition from one internal state to another is governed by the transition function (δ).
- **Example:** if $\delta(q_0, a) = q_1$, this means that if the DFA is in state "q_0" and the current input symbol is "a", then the DFA will go into state q_1.
- The **start state** is normally represented using an **oval shape** that has an ***arrow pointing into it from nowhere***.
- The **final state** is represented using a **double circle/oval**.
- With a DFA, **exactly one transition arrow exits every state for each possible input symbol.**

Example: Consider the following state diagram of a DFA M_3, answer the questions presented about this machine: -



M_3

a)What is the start state of M_3? **q_1**

b)What are the sets are of accept /Final states of M_3? **{q_2}**

c) What sequence of states does M_3 go through on input aabba? **q_1, q_2, q_3, q_1, q_1,q_2**

d) Does M_3 accept the string aabb? Give a reason for your answer. **q_1, q_2, q_3, q_1, q_1, - No because it does not take us to the final state q_2.**

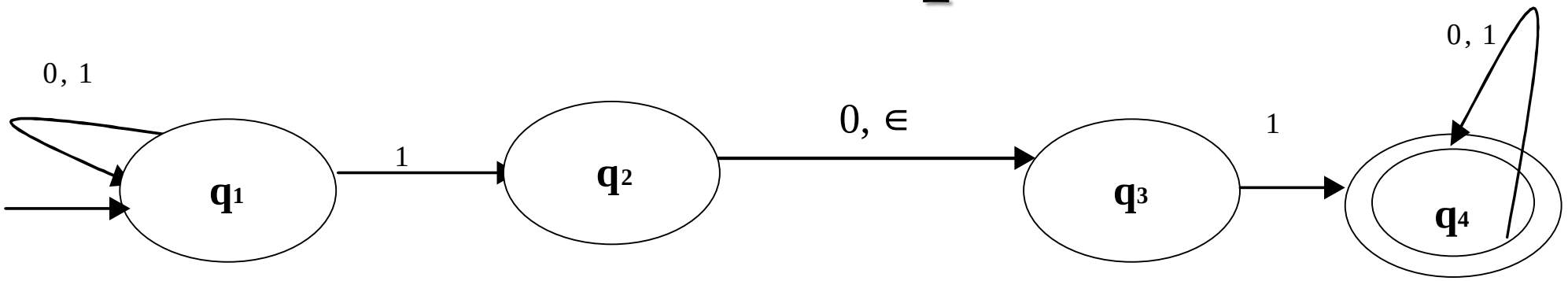
e) Does M_3 accept the string string { } ? Give a reason for your answer. **No it does not, given that we don't have empty input into q_2, which is the accept state**

- So far in our discussion, every step of a computation follows in a **unique way from the preceding step**.
- 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 called this **deterministic computation**.

2. Non-deterministic Finite Automata (NFA)

- Non-determinism is a useful concept with a great impact on the theory of computation.
- In a non-deterministic machine, **several choices may exist for the next state at any point** and the machine therefore may have **additional features**.

How does an NFA compute?



Example: - Consider the state diagram for an NFA machine N_1 above.

- We realize when the NFA, reads an input string, there are **multiple ways to proceed**.
- Example: While in state q_1, if the next symbol is a 1, the machine splits into multiple copies of it and follows all the possibilities in parallel.

How does an NFA compute?

- If **a state with a ∈ symbol** on an exiting arrow is encountered, something similar happens.
- Without reading any input, the machine splits into multiple copies, one following each of the exiting ∈-labeled arrows and one staying at the current state.
- Then the machine **splits non-deterministically** as before.

From this state diagram we can come up with a table showing **three differences between DFA and NFA** as follows: -

DFA	NFA
Every state of a DFA always has exactly one exiting transition arrow for each symbol in the alphabet.	A state may have zero, one or many exiting arrows for each alphabet symbol.
Labels on the transition arrows are symbols from the alphabet.	May have arrows labeled with members of the alphabet or ∈. Zero, one, or many arrows may exit from each state with the label ∈.
The transition function takes a state and an input symbol to produce the next state.	The transition function takes a state and input symbol or the empty string ∈, to produce the set of possible next states.

- Non- determinism may be viewed as a kind of **parallel computation** wherein **several processes can be running concurrently**.
- When the NFA splits to follow several choices; this corresponds to **a process** known as "**forking**" into **several children each proceeding separately**.
- If **at least one of these processes leads to an accept state** then the **entire computation accepts**.

- Another way to think of a nondeterministic computation is as a **tree of possibilities**.
- The **root** of the tree corresponds to the **start of the computation**.
- Every **branching point** in the tree corresponds to a point in the computation at which the machine **has multiple choices**.
- The **machine accepts the computation if at least one of the computation branches ends in an accept state.**

Importance of NFA

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

ii. Non-determinism in finite automata is also a good introduction to **non-determinism in more powerful computational models** because finite automata are especially easy to understand.

iii. Every NFA can be converted into an equivalent DFA and constructing NFAs is sometimes easier than directly constructing DFAs.

Formal Definition of a Nondeterministic Finite Automaton (NFA)

- The formal definition of an NFA is similar to that of a DFA.
- Both have **states, an input alphabet, a transition function, a start state and a collection of accept states**.
- However, they differ in one essential way: in the **type of transition function**.

- Recall from Lecture 1 that a set E with no elements is called an **empty set** and it is denoted by { } or ∅
- For the NFA, the transition function produces **a set of possible next states.**
- For any **set Q** we write the power set of Q as **P(Q)** to be the collection of all subsets of Q.
- Here **P(Q)** is called the **power set of Q**.
- For any **input alphabet** $\sum$ we write $\sum_\epsilon$ to be $\sum \cup \{\epsilon\}$.
- Now we can easily write the formal description of the type of the transition function in an NFA. It is $\delta: Q \times \sum_\epsilon \rightarrow P(Q)$.

NFA Formal Definition

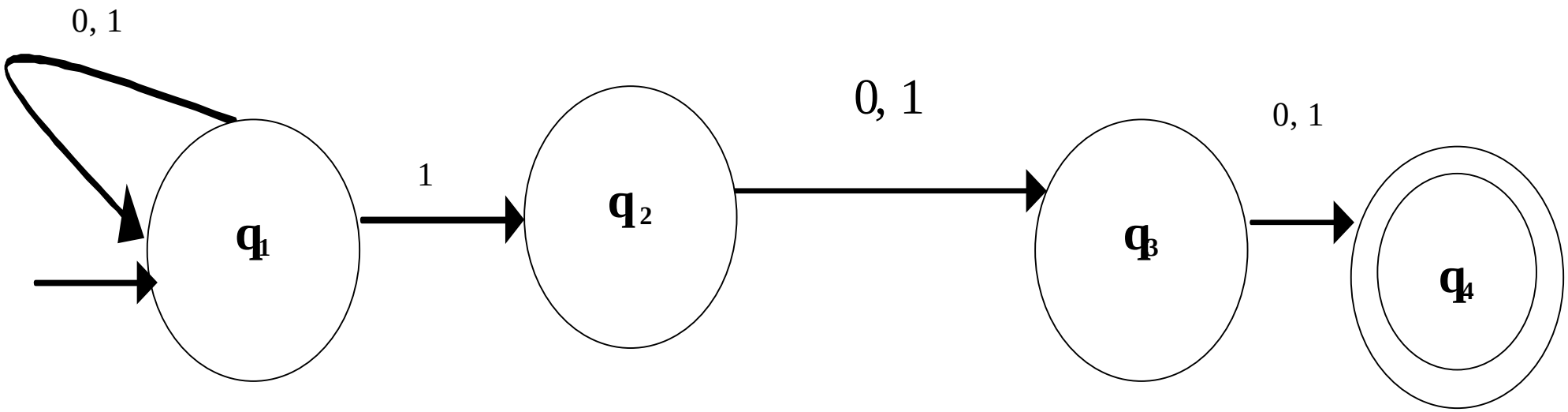
A Nondeterministic Finite Automaton is a 5-tuple $M= (Q,\sum, \delta , q_0, F)$; where: -

1. Q is a Finite set of Internal States
2. $\sum$ is a Finite Set of Symbols "Input Alphabet"
3. δ: $Q \times \sum_{\epsilon} \rightarrow P(Q)$ is the transition symbol
4. $q_0 \in Q$ is the initial state
5. $F \subseteq Q$ is the Set of Final/accept States

- *Recall that the formal definition precisely describes what we mean by a finite automaton.*

Example 1:

- Let A be the language consisting of all strings over {0, 1} containing a 1 in the third position from the end (e.g. 000100 is in A but 0011 is not). The following four-state NFA N2 recognizes A.



One good way to view the computation of this NFA is to say that:-

- The machine stays in the **start state q_1** until it "**guesses**" that it is **three places from the end**.
- At that point, **if the input symbol is a 1**, it **branches to state q_2 and uses q_3 and q_4** to "**check**" on whether **its guess was correct**.

References

- Rowan G. & John T., (2009), *Discrete Mathematics: Proofs, Structures and Applications*, CRC Press, ISBN: 9781439812808.
- W. D. Wallis (2003), *A Beginners Guide to Discrete Mathematics*, Springer Science & Business Media, ISBN: 978-0817642693.
- Introduction to the theory of computation (3rd ed.), Michael, S. Boston, Cengage Learning. ISBN-13: 978-1133187790, (2012).
- Introduction to languages and the theory of computation (3rd ed.), Martin, J., New York: McGraw-Hill. ISBN-13: 978-0072322002, (2002)

Course: Automata Theory

Lecture 6: Deterministic and Non-Deterministic Finite Automata

Lecturer: Martha Gichuki