Automata Theory - Lecture 7

Models of Computation – Finite State Machines

Lecturer: Martha Gichuki

Lecture learning outcomes

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

(i) Describe Finite State Machines.

(ii) Describe Finite Automata using State Diagrams

(iii) Formally describe various NFA and DFA machines

Finite Automata

A finite-state machine (finite automaton) is a simple, limited computer model with a number of states and transitions.

Finite Automata are limited in memory but they can achieve many things and we interact with such computers all the time since they are at the heart of various electromechanical devices.

As discussed in lecture 6, an automaton is a mathematical model for a Finite State Machine (FSM), 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.

FORMAL DEFINITION OF A FINITE AUTOMATON

Two ways are used to describe machines: **state diagrams**, which are easier to grasp and the **formal definition** which provides a good notation that helps us think and express our thoughts clearly in a more precise way meaning it resolves any uncertainties about what is allowed in a finite automaton.

A finite automaton has five parts and in mathematical language a list of five elements is often called a **5-tuple**. Hence, we define a finite automaton to be a 5-tuple consisting of five parts: -

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

We use the transition function, denoted as δ to define the rules for moving. 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.

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 five-tuple M= (Q, $\sum$, δ, q_0, F); where: -

Q = Finite set of Internal States

$\sum$= Finite Set of Symbols "Input Alphabet"

δ: Q X $\sum \rightarrow$Q is the Transition function

$q_0 \in Q$ is the Initial State

$F \subseteq Q$ is the Set of Final States

The input mechanism can move from left to right and reads exactly one symbol on each step.

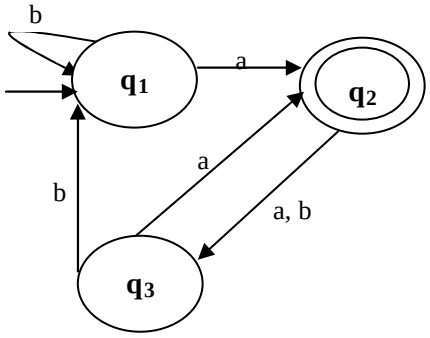
The transition from one internal state to another is governed by the transition function (δ).

For 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.

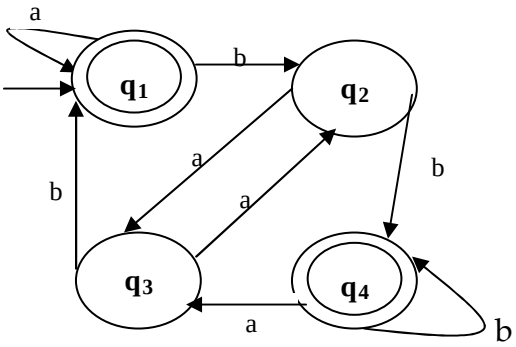
The start state is normally represented using an Oval shape that has an arrow pointing into it from nowhere. The Final state is represented using a double circle/oval.

Example One:

The following are state diagrams of two Deterministic Finite Automata (DFAs). Answer the following questions about these machines: -



M3



M4

i. What are the start states of M_3 and M_4? ***q1 and q1 respectively***

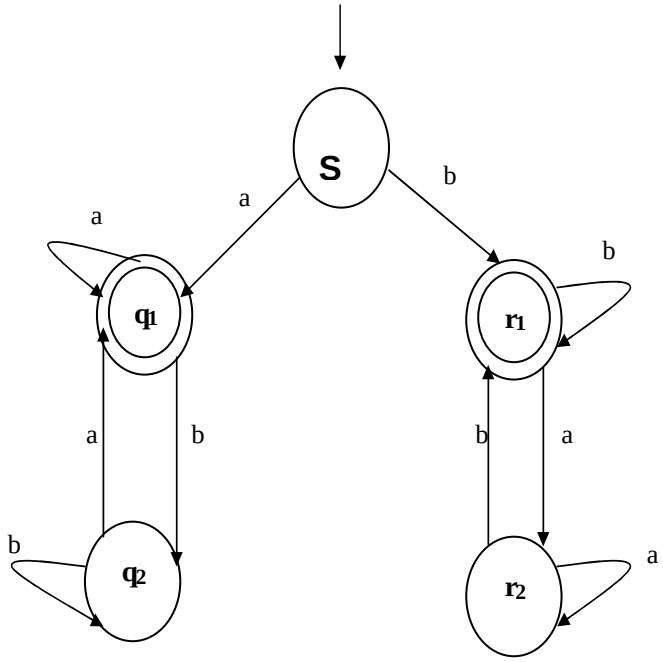
ii. What the sets are of accept /Final states of M_3 and M_4? ***{q2} and {q1, q4} respectively.***

iii. What sequence of states does M_3 go through on input aabb? ***q1, q1, q2, q4***

iv. Does M_3 accept the string aabb? Give a reason for your answer ***Yes, it does because the final state q4 is an accept state***

v. Does M_4 accept the string ∈? Give a reason for your answer ***Yes, it does, given that even without any input, we are already in an accept state***

2. The figure below shows the state diagram of a finite automaton M_1. Give the formal description of this automaton. (5 Marks)



M_1

Q = {S, q1, q2, r1, r2}

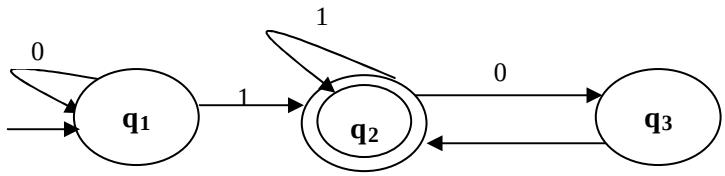
∑= {a, b}

δ = Transition Symbol

	Input Alphabet	
States	a	b
S	q1	r1
q1	q1	q2
q2	q1	q2
r1	r2	r1
r2	r2	r1

q_0= {S} , F = {q1, r1}

3.) Given the state diagram below for a finite automaton M_5, describe the machine formally:-



0,1

Solution:

M= (Q, ∑, δ , q_0, F) Where ,

R = { q1, q2, ,q3}

∑ = {0,1}

δ is the transition function given by the table

	Input Alphabet	
States	0	1
q1	q1	q2
q2	q3	q2
q3	q2	q2

q_0 = {q1}

F = {q2}

It is important to note that for a DFA, exactly one transition arrow exits every state for each possible input alphabet.

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.

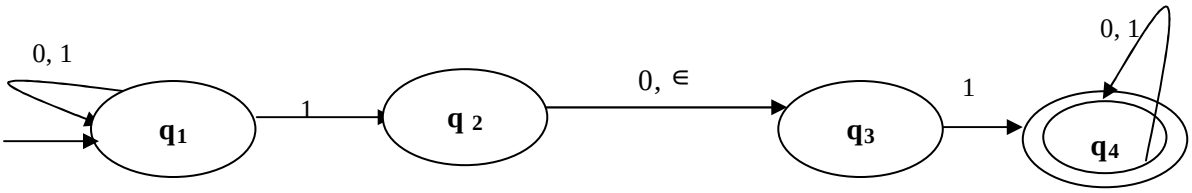
Example: Machine M_5 above recognizes language A where: -
A= {W ∈ w contains at least one 1 and an even number of 0s follow the last 1}. Therefore L (M_5) = A or equivalently, M_5 recognizes A.

Note: A language is called regular if some finite automaton recognizes it. A finite Automaton has only a finite number of states, which means a finite memory and this translates into limited storage capacity.

2. Non-deterministic Finite Automata

Non-determinism is a useful concept that has had great impact on the theory of computation. 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 call this deterministic 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.

Example: - Consider the NFA machine N1 below: -



From this state diagram we can come up with a table showing the 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.

How does an NFA compute?

Suppose we consider the above NFA (N_1). We realize when the NFA, reads an input string, there are multiple ways to proceed. E.g. in state q1, if the next symbol is a 1, the machine splits into multiple copies of it and follows all the possibilities in parallel.

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.

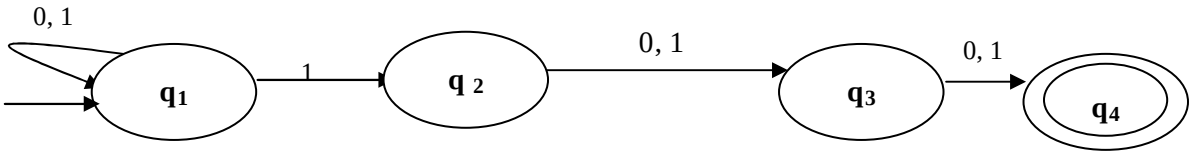
Non- determinism may be viewed as a kind of parallel computation wherein several processes can be running concurrently. The NFA splits to follow several choices that correspond to a process "forking" into several children each proceeding separately and if at least one of these processes accepts then 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 if at least one of the computation branches ends in an accept state.

Non-deterministic finite automata are useful in several respects. Every NFA can be converted into an equivalent DFA 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. 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.

Example of NFA:

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 it stays in the start state q1 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 q2 and uses q3 and q4 to "check" on whether its guess was correct.

Formal Definition of a Nondeterministic Finite Automaton (NFA)

The formal definition of a nondeterministic finite automaton is similar to that of a deterministic finite automaton. 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. For the NFA, the function produces a set of possible next states. (As mentioned earlier in the table). For any set Q we write P(Q) to be the collection of all subsets of Q. Here P(Q) is called the power set of Q. For any alphabet $\sum$ we write $\sum_{\in}$ to be $\sum \cup \{\in\}$. Now we can easily write the formal description of the type of the transition function in an NFA. It is $\delta: Q \times \sum_{\in} \rightarrow P(Q)$.

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

Q is a Finite set of Internal States

$\sum$ is a Finite Set of Symbols "Input Alphabet"

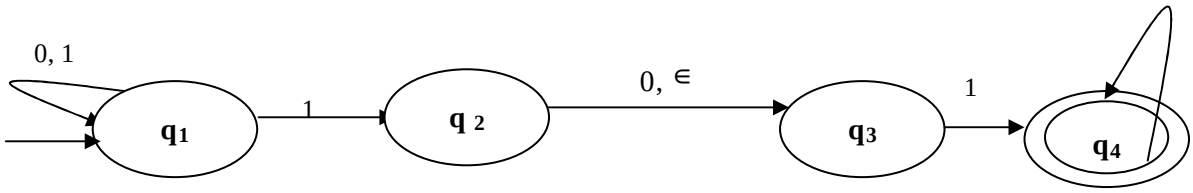
$\delta: Q \times \sum \in \rightarrow P(Q)$ is the transition Symbol $q_0 \in Q$ is the initial State

$F \subseteq Q$ is the Set of Final/accept States

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

Review Question One

Given the state diagram for the NFA machine N_1 below, provide the formal description of the machine.



Solution:

Q = {q1, q2, q3, q4}

∑ = {0,1} δ is the transition function given by the table

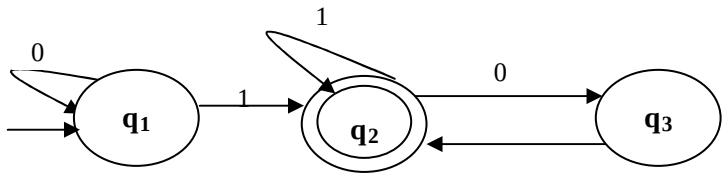
	Input Alphabet		
States	0	1	∈
q1	{q1}	{q1, q2}	Ø
q2	{q3}	Ø	{q3}
q3	Ø	{q4}	Ø
Q4	{q4}	{q4}	Ø

q_0 the start state = {q1}

F = {q4}

Question Two

Given the state diagram below for a finite automaton M5, describe the machine formally:



0, 1

Solution:

M= (Q, ∑, δ, q_0, F)

Where

Q = {q1, q2, q3}

∑ = {0,1} δ is the transition function given by the table

	Input Alphabet	
States	0	1
q1	q1	q2
q2	q3	q2
Q3	q2	q2

q_0 = {q1}

F = {q2}

References

1 Rowan G. & John T., (2009), *Discrete Mathematics: Proofs, Structures and Applications*, CRC Press, ISBN: 9781439812808.

2 W. D. Wallis (2003), *A Beginners Guide to Discrete Mathematics*, Springer Science & Business Media, ISBN: 978-0817642693.

3 Introduction to the theory of computation (3rd ed.), Michael, S. Boston, Cengage Learning. ISBN-13: 978-1133187790, (2012).

4 Introduction to languages and the theory of computation (3rd ed.), Martin, J., New York: McGraw-Hill. ISBN-13: 978-0072322002, (2002)