In the name of Allah, the most compassionate, the most merciful

Words of Wisdom

If you want to lift yourself up, lift up someone else. (Brooker T. Washington)

NUST
Defining futures

Automata theory

- Why we need Automata theory
- How to make a Finite State Automaton
- Real life examples of Automata

Computation

NUST
Defining futures

- Computation can be defined as finding a solution to a problem from given inputs by means of an algorithm.
- Model of Computation
  - A formal or an abstract computer
- Types of Computation Models
  - Serial Models
  - Parallel Models

Computation Devices

NUST
Defining futures

- In this study we consider a *mathematical model* of computing.
- Such a model strips the computational machinery down to the bare minimum, so that it's easy to manipulate these theoretical machines mathematically to prove things about their capabilities.

Applications

NUST
Defining futures

- Finite state machines are a useful model for many important kinds of hardware and software. e.g.
  - Software for designing digital circuits
  - Lexical analyzer of a compiler
  - Searching for keywords in a file or on the Web
  - Software for verifying finite state systems, such as communication protocols
  - Operating System (UNIX grep)
  - Text Editors
  - Markup Languages (HTML, XML)
  - Natural Language Processing

Central Concepts

NUST
Defining futures

- Symbol
  - A symbol is a single object and is an abstract entity that has no meaning by itself.
  - It can be alphabet, character or special character

Central Concepts

NUST
Defining futures

- Alphabet
  - An ***alphabet*** is a finite, nonempty set of symbols, Σ (sigma) is used for an alphabet.
  - For example

    {0, 1} is an alphabet with two symbols

    {a, b} is another alphabet with two symbols, and

    English alphabet is also an alphabet
    - Σ = {0,1}, or Σ = the binary alphabet
    - Σ = {a,b,…..z}, or Σ=the set of all lower-case letters

Central Concepts

NUST
Defining futures

- String
  - A ***string*** is a finite sequence of symbols chosen from some alphabet. e.g. 01101 is a string from the binary alphabet Σ = {0,1}.
  - The number of symbols in a *string* is called the ***length of a string.*** e.g. 011101 has length 5 and the number of symbols are 2. The standard notation for the length of a string *w* is |*w*|. e.g. |011| = 3 and | ε | = 0

Central Concepts

NUST
Defining futures

- Strings
  - ***Concatenation of Strings***: Let *x* and *y* be strings. Then *xy* denotes the string obtained by concatenating *x* with *y*, that is, *xy* is the string obtained by appending the sequence os symbols of *y* to that of *x*. e.g. if *x* = *aab* and *y* = *bbab*, then *xy* = *aabbbab*. Note that $xy \neq yx$.

Central Concepts

NUST
Defining futures

Languages

- A *language* is a set of strings over an alphabet. Thus *{a, ab, baa}* is a language (over alphabet *{a, b}*).
- A set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet, is called a *language*. If Σ is an alphabet, and L Σ^* , then L is a *language* over Σ.
  - An example is English, where the collection of legal English words is a set of strings over the alphabet that consists of all the letters.

Examples of Languages

NUST
Defining futures

- The language of all strings consisting of *n* 0's followed by *n* 1's, for some n ≥ 0:
  - {ε, 01, 0011, 000111,…..}
- The set of strings of 0's and 1's with an equal number of each:
  - {ε, 01,10,0011,0101,1001,…..}
- The set of binary numbers whose value is a prime:
  - {10, 11, 101,111,1011…..}

Note: The only important constraint on what can be a language is that all alphabets are finite, although they can have an infinite number of strings.

Set Notation

NUST
Defining futures

- It is common to describe a language using a "set-notation":

  {*w* | something about *w* }

  It is read "*the set of strings **w** such that (whatever is said about w to the right of the vertical bar)*"

  Examples:

  - { w | w consists of an equal number of 0's and 1's }
  - { w | w is a binary integer that is prime }
  - { w | w is a syntactically correct C++ program }

Set-Notation as a way to Define Languages

- It is also common to replace *w* by some expression with parameters and describe the strings in the language by stating conditions on the parameters. For example:
  1. $\{ 0^n 1^n \mid n \geq 1 \}$
  2. $\{0^i 1^j \mid 0 \leq i \leq j \}$

Some Special Languages

NUST
Defining futures

- The empty set Ø is a language which has no strings.
- The set { ε } is a language which has one string, namely ε . Though ε has no symbols, this set has an object in it. So it is not empty.
- For any alphabet Σ, the set of all strings over Σ is denoted by Σ*. Thus a language over alphabet Σ is a subset of Σ*.

Finite Automata

NUST
Defining futures

- Definition of an Automaton

  "*An automaton is defined as a system where energy, materials and information are transformed, transmitted and used for performing some functions without direct participation of man.*"

  - Examples are
    - Automatic machine tools, automatic packing machines, and automatic photo printing machines.

Example 1

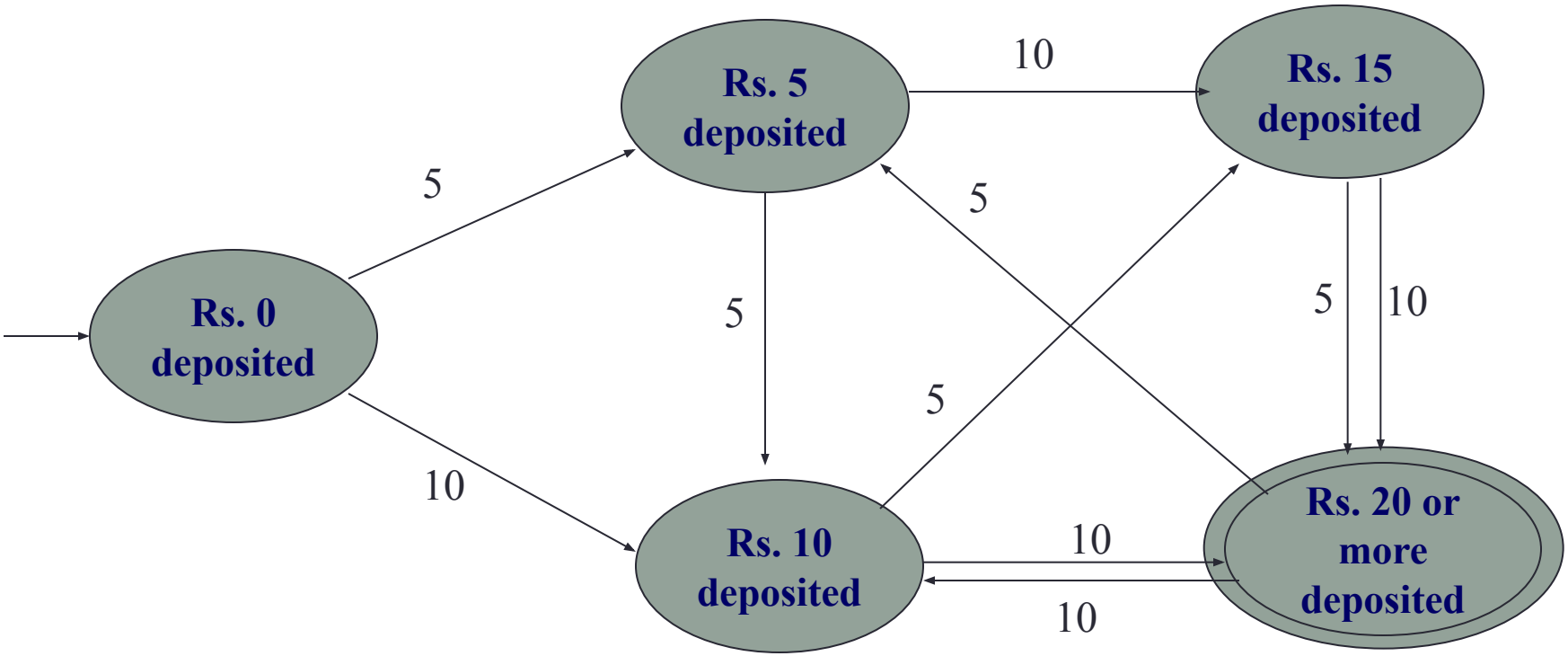
NUST
Defining futures

- A vending machine dispense piece of candy that cost Rs. 20 each. The machine accepts coins of Rs. 5 and Rs. 10 only and does not return change.

  As soon as the amount deposited equal or exceeds Rs. 20, the machine releases a piece of candy.

Example 1 (contd:)

- Each circle represents a state of the machine
- Unlabelled arrow indicates the *initial state* of the machine
- Double circle indicates that candy is released, called *accepting state*.

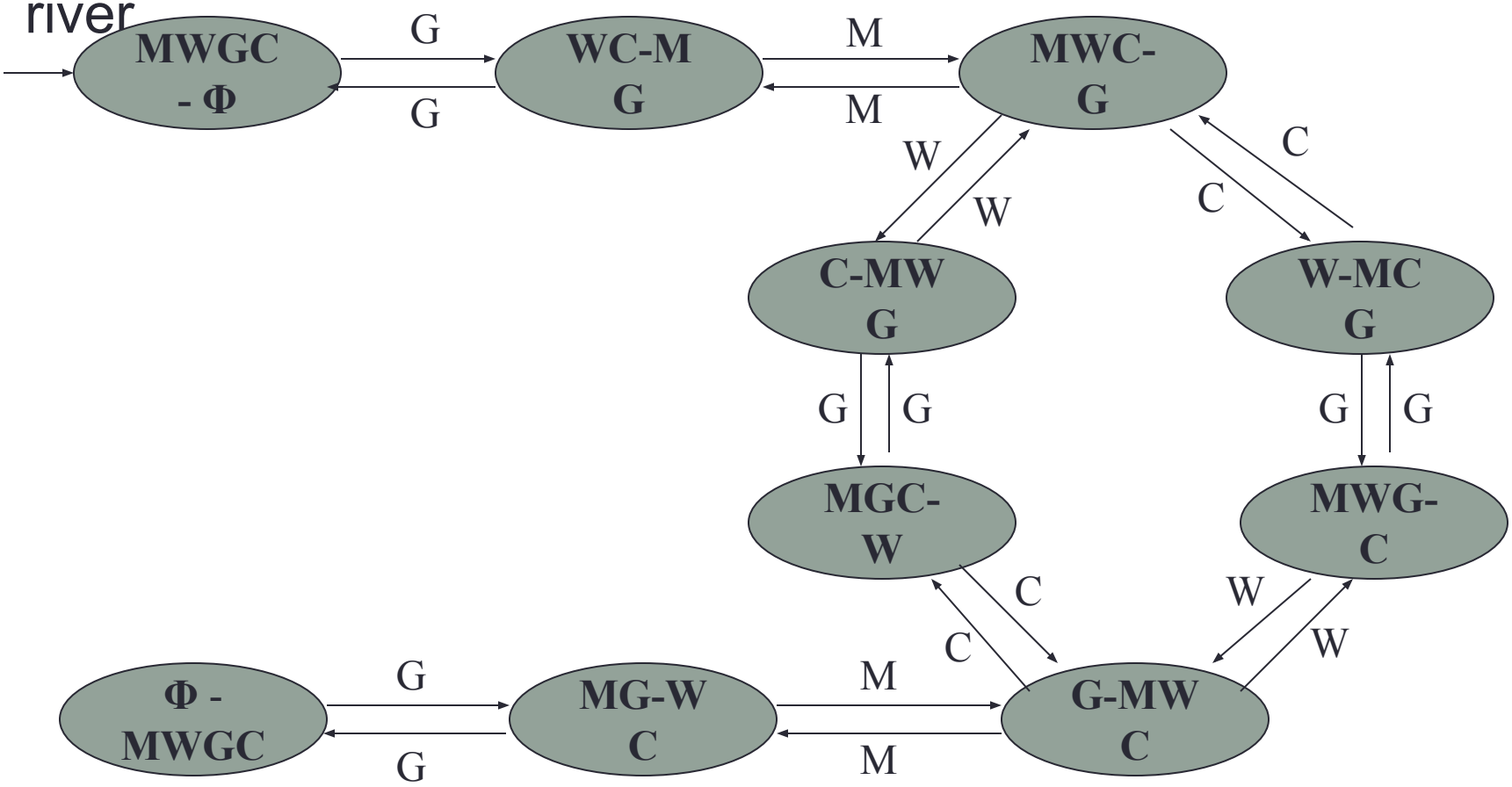


Example 1 (contd:)

Next State Table			
		Inputs	
	State	5	10
→	Rs. 0 deposited	Rs. 5 deposited	Rs. 10 deposited
	Rs. 5 deposited	Rs. 10 deposited	Rs. 15 deposited
	Rs. 10 deposited	Rs. 15 deposited	Rs. 20 or more deposited
	Rs. 15 deposited	Rs. 20 or more deposited	Rs. 20 or more deposited
◎	Rs. 20 deposited	Rs. 5 deposited	Rs. 10 deposited

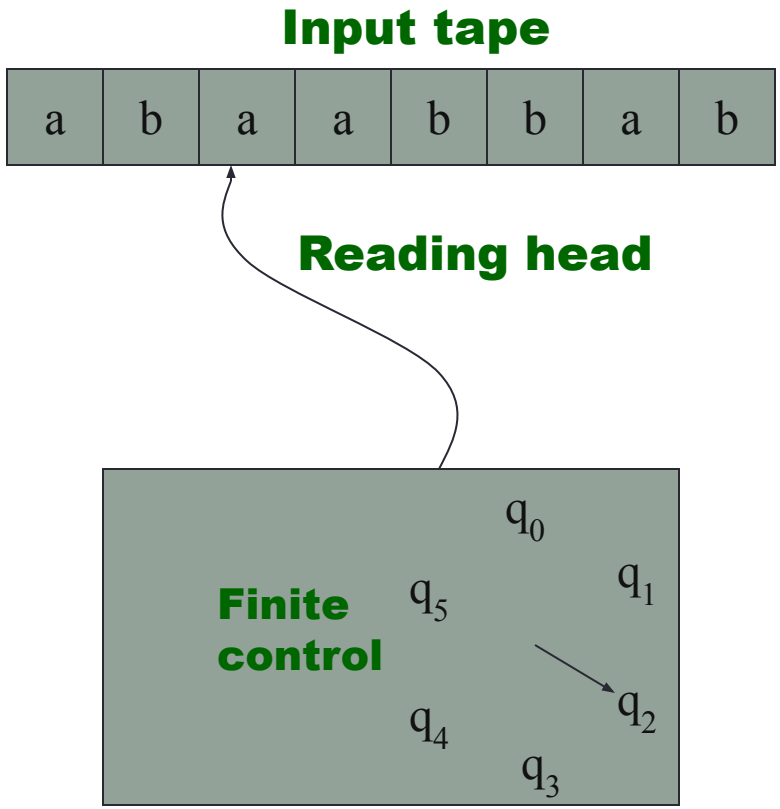
Example 2

- A story of man with wolf, goat and cabbage to cross the river



Deterministic Finite Automata

- A deterministic finite automaton is a simple language recognition device. It is called deterministic because their operation is completely determined by their input.
- Strings are fed into the device by means of an input tape, which is divided into squares, with one symbol inscribed in each tape square.



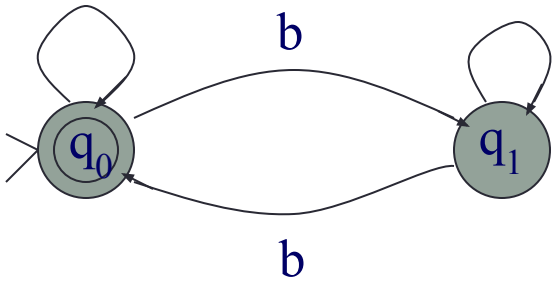
Transition Diagram

- A *transition diagram* for a DFA M = (Q, Σ, δ, s, F) is a graph defined as follows:

1. For each state in Q there is a node (vertex).
2. For each state *q* in Q and each input symbol *a* in Σ, let δ(q, a) = *p*. Then the transition diagram has an arc (edge) from node *q* to node *p*, labeled *a*.
3. There is an arrow into the start state q_0, labeled *start*.
4. Nodes corresponding to accepting states (those in F) are marked by a double circle.

a

a



How a DFA Processes Strings?

Let the input string is aabba.

$\delta(q_0, a) = q_0$

$\delta(q_0, a) = q_0$

$\delta(q_0, b) = q_1$

$\delta(q_1, b) = q_0$

$\delta(q_0, a) = q_0$

Since q0 belong to F therefore the string is accepted.

Example-2

NUST
Defining futures

- **Let us design a DFA *M* that accepts the language *L(M) = { w | w Є {0,1}* and w is of the form x01y for some strings x and y consisting of 0's and 1's only}***

- To decide whether 01 is a substring of the input, M needs to remember:

  1. Has it already seen 01? If so, then it accepts every sequence of further inputs.

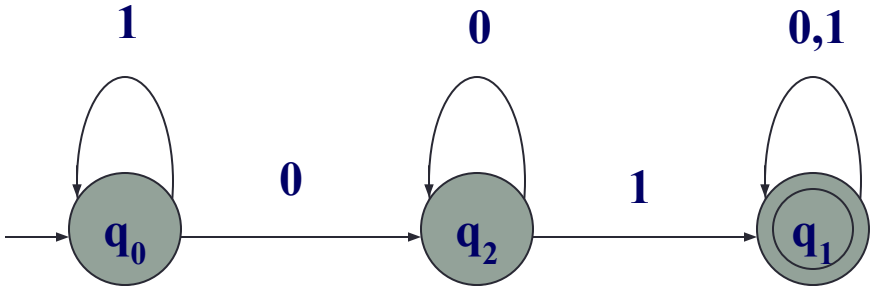
  2. Has it never seen 01, but its most recent input was 0, so if it now sees a 1, it will have seen 01.

  3. Has it never seen 01, but its last input was either nonexistent or it last saw a 1?

Example (continued)

- These three conditions can each be represented by three states q_0, q_1, q_2. The transition functions are:

$$\delta(q_0, 0) = q_2$$
$$\delta(q_0, 1) = q_0$$
$$\delta(q_2, 0) = q_2$$
$$\delta(q_2, 1) = q_1$$
$$\delta(q_1, 0) = q_1$$
$$\delta(q_1, 1) = q_1$$



Transition Tables

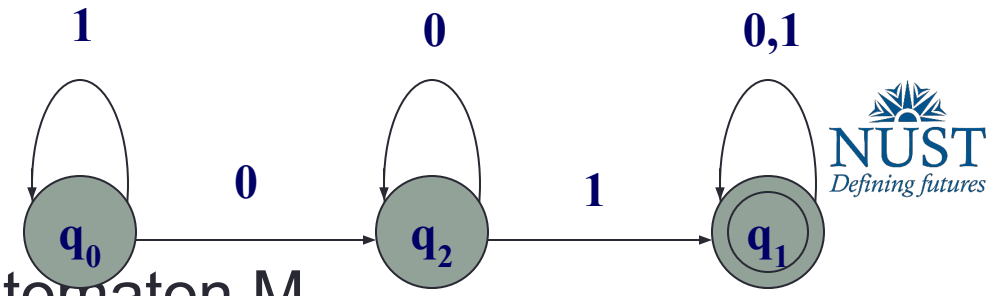
NUST
Defining futures

- A *transition table* is a conventional, tabular representation of a function like δ that takes two arguments and returns a value.
- The rows of the table correspond to the states, and the columns correspond to the inputs.

	0	1
→ q_0	q_2	q_0
*q_1	q_1	q_1
q_2	q_2	q_1

Transition table for the DFA

Exercise # 1



Consider the finite state automaton M.

- What are the states of M?
- What are the input symbols of M?
- What is the initial state of M?
- What are the accepting states of M?
- Find $\delta(q_0, 1100101)$.
- Find the next state table for M.
- Describe informally the language accepted by the DFA.

Exercise # 2

NUST
Defining futures

- Design a DFA to accept the language:
  L = { *w* | *w Є {English language letters}* and w has "**the**" as a substring*}
  - What are the states of M?
  - What are the input symbols of M?
  - What is the initial state of M?
  - What are the accepting states of M?
  - Find $\delta(q_1, \text{brothers})$.
  - Find the transition state table for M.
  - Draw the transition diagram.

Recommended Book(s)

NUST
Defining futures

- Text Book(s):
  - Nasir S.F.B and P.K. Srimani, "**A Textbook on Automata Theory**", Cambridge University Press, India, 2008
  - Sikander H. Khiyal, "***Theory of Automata and Computation***", National Book Foundation, 2004
- Reference Book(s):
  - John E. Hopcroft, Rajeev Motvani, and Jeffrey D. Ullman, "***Introduction to Automata theory, Languages and Computation***", Second Edition, Addison-Wesley, New York, 2001.
  - Daniel I. A. Cohen, "**Introduction to Computer Theory**", Second Edition.
  - K.L.P. Mishra, N. Chandrasekaran, "***Theory of Computer Science (Automata, Languages and Computation)***", Prentice-Hall of India, 2002.

Summary

NUST
Defining futures

- **Safe from bugs.** Grammars and regular expressions are declarative specifications for strings and streams, which can be used directly by libraries and tools. These specifications are often simpler, more direct, and less likely to be buggy then parsing code written by hand.
- **Easy to understand.** A grammar captures the shape of a sequence in a form that is easier to understand than hand-written parsing code. Regular expressions, alas, are often not easy to understand, because they are a one-line reduced form of what might have been a more understandable regular grammar.
- **Ready for change**. A grammar can be easily edited, but regular expressions, unfortunately, are much harder to change, because a complex regular expression is cryptic and hard to understand.