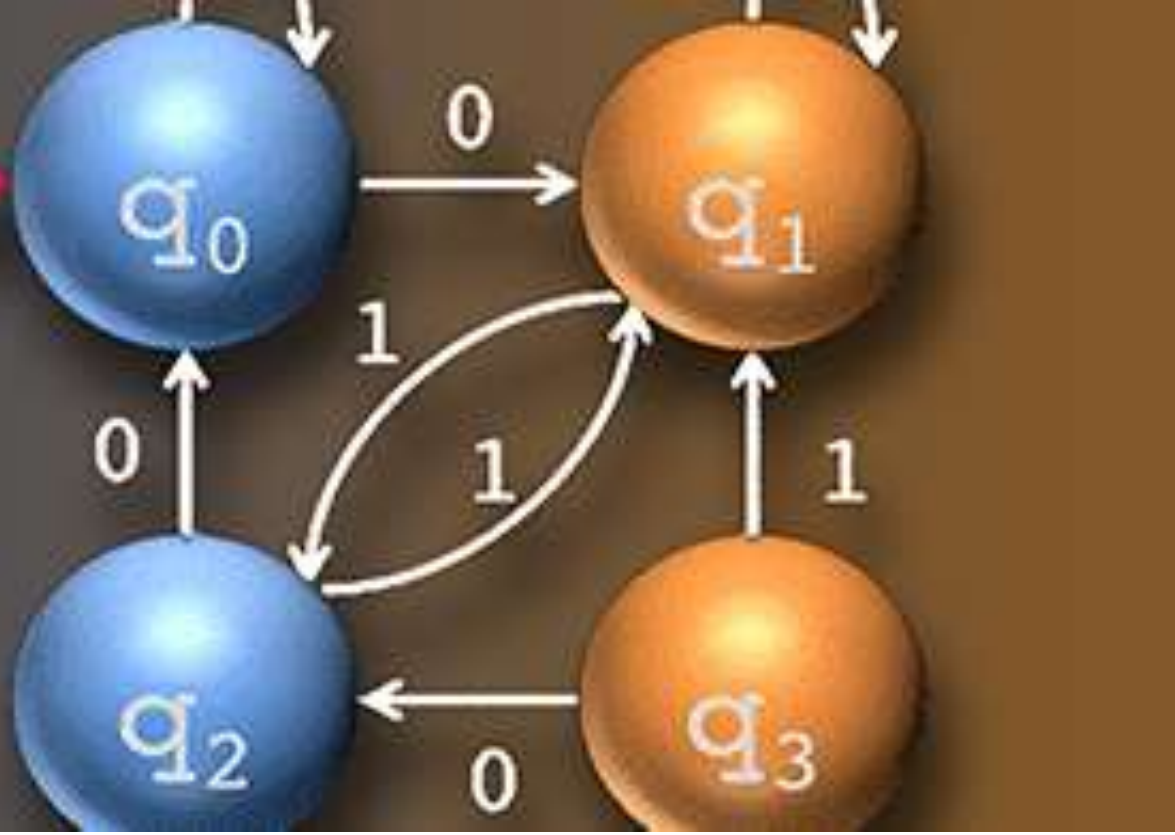
CSE 305

Theory of COMPUTATION



Lecture 4

Introduction (3)

Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering,
Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.
www.mijanrahman.com

Contents

Introduction

- What is the “Theory of Computation”?
- History: Theory of Computation
- Branches of the Theory of Computation
- Automata Theory, and Formal Language
- Overview of Finite Automata, Context-free Grammars, Pushdown Automata, and Turing Machines
- **Computability Theory, Complexity Theory, and Models of Computation**
- **Applications of Theory of computation**

Computability theory

- Computability theory deals with **what can and cannot be computed by the model respectively**. The theoretical models are proposed in order to understand the solvable and unsolvable problems which lead to the development of the real computers.
- **The Computability theory defines whether a problem is "solvable" by any abstract machine.** Some problems are computable while others are not.
- Computation is done by various computation models depending on the nature of the problems. Examples of these machines are: the Turing machine, Finite state machines, and many others.
- **Central Question in Computability Theory:** Classify problems as being solvable or unsolvable.

Computability theory

- In the 1930's, G¨odel, Turing, and Church discovered that some of the fundamental mathematical problems cannot be solved by a "computer". An example of such a problem is "Is an arbitrary mathematical statement true or false?"
- To attack such a problem, we need formal definitions of the notions of ***computer***, ***algorithm***, and ***computation***.
- The theoretical models that were proposed in order to understand solvable and unsolvable problems led to the development of real computers.

- The theories of computability and complexity are closely related. **In complexity theory, the objective is to classify problems as easy ones and hard ones; whereas in computability theory, the classification of problems is by those that are solvable and those that are not.** Computability theory introduces several of the concepts used in complexity theory.

Computability theory

- Therefore, **Computability** deals with the following kinds of questions:
  - Given a specific problem, can its solution be computed?
  - Can it be computed on a particular kind of machine?
  - If it can be computed on one kind of machine, can another kind of machine be used instead?
  - If it can be computed, what amounts of resources (e.g., time or memory) are consumed?
  - What are the minimum amounts of resources needed to solve a problem?
  - Are there trade-offs between resources (e.g., does more speed require more memory)?
  - Are the resource required to solve a problem so great that it is impossible in practice to solve?
  - What are the practical amounts of resources that we can apply to solving problems?
  - Can feasible and intractable classes of problems be defined? If so, is there a clean breakdown between these two categories?

Computability theory

- Before finding the answers of these question, we need a more precise idea of what we mean by "***problem***". For example, a typical problem in an introductory computing course is to write a program that determines an average of its inputs.
  - **Problem:** Computer the average value, $\frac{1}{n}\sum_{i=1}^{n} x_i$.
- Computability plays an important role in a number of different fields, including biology, operations research, physics, and mathematics.
- The theory is applicable to subjects, such as data structure and algorithms, compilers, data communication and networks, operating systems, and software engineering.
- You cannot do computer science without understanding the basics of computability.

Complexity theory

- This theoretical computer science branch is all about studying the cost of solving problems while focusing on resources (time & space) needed as the metric. The running time of an algorithm varies with the inputs and usually grows with the size of the inputs.
- **The main question asked in this area is "What makes some problems computationally hard and other problems easy?" Thus, Complexity theory groups the computable problems based on their hardness.**
- Two major aspects are considered: ***time complexity*** **and** ***space complexity***, which are respectively how many steps does it take to perform a computation, and how much memory is required to perform that computation.
- In order to analyze how much time and space a given algorithm requires, computer scientists express the time or space required to solve the problem as a function of the size of the input problem.

Complexity theory

- For example,
  - Any problem is easy, if it is solved efficiently.
  - Any problem is hard, if it cannot be solved efficiently.
- **Informally, a problem is called “easy”, if it is efficiently solvable. Examples of “easy” problems are (i) sorting a sequence of numbers, (ii) searching for a name in a telephone directory, and (iii) computing the fastest way to drive from Mymensingh to Coxsbazar.**
- **On the other hand, a problem is called “hard”, if it cannot be solved efficiently, or if we don’t know whether it can be solved efficiently. Examples of “hard” problems are (i) time table scheduling for all courses at JKKNIU, (ii) factoring a 300-digit integer into its prime factors, and (iii) computing a layout for chips in VLSI.**

Complexity theory

- **How to measure computational complexity?**
- **Measuring complexity involves an algorithm analysis to determine how much time it takes while solving a problem (time complexity). To evaluate an algorithm, a focus is made on relative rates of growth as the size of the input grows.**
- Since the exact running time of an algorithm often is a complex expression, we usually just estimate it. We measure an algorithm's time requirement as a function of the input size (n) when determining the time complexity of an algorithm.
- As **T(n)**, the time complexity is expressed using the **Big O** notation where only the highest order term in the algebraic expressions are considered while ignoring constant values.

Complexity theory

- The common running times when analyzing algorithms are:
  - O(1) - Constant time or constant space regardless of the input size.
  - O(n) - Linear time or linear space, where the requirement increases uniformly with the size of the input.
  - O(log n) - Logarithmic time, where the requirement increases in a logarithmic nature.
  - O(n^2) - Quadratic time, where the requirement increases in a quadratic nature.
- This analysis is based on 2 bounds that can be used to define the cost of each algorithm.
- They are:
  - Upper (*Worst Case Scenario*)
  - Lower (*Best Case Scenario*)

Complexity theory

- The major classifications of complexities include:
  - *Class P:* The class P consists of those problems that are solvable in **polynomial time**. These are problems that can be solved in time O(n^k) for some constant k where n is the input size to the problem. It is devised to capture the notion of efficient computation.
  - *Class NP:* It forms the class of all problems whose solution can be achieved in **polynomial time by non-deterministic Turing machine**. NP is a complexity class used to classify decision problems.
- A major contributor to the complexity theory is the complexity of the algorithm used to solve the problem. Among several algorithms used in solving computational problems are those whose complexity can range from fairly complex to very complex.

Complexity theory

- The more complex an algorithm, the more computational complexity will be in a given problem.
- Factors that influence program efficiency:
  - The problem being solved.
  - The algorithm used to build the program.
  - Computer hardware.
  - Programming language used.
- **Central Question in Complexity Theory:** Classify problems according to their degree of "difficulty". Thus, the problems that seem to be "hard" are really "hard".

Models of Computation

- In computer science, and more specifically in computability theory and computational complexity theory, a **model of computation** is a model which describes how an output of a mathematical function is computed given an input.
- **A model describes how units of computations, memories, and communications are organized. The computational complexity of an algorithm can be measured given a model of computation.**
- Using a model allows studying the performance of algorithms independently of the variations that are specific to particular implementations and specific technology.
- Models of computation can be classified into three categories: **sequential models, functional models, and concurrent models.**
- Some are **deterministic**, and some are **nondeterministic**. Some of these models have both **deterministic and nondeterministic variants**. Nondeterministic models are not useful for practical computation; they are used in the study of computational complexity of algorithms.

Models of Computation

- **Sequential Models:**
  - Sequence models are those models that input or output sequences of data. Sequential models include:
    - Finite state machines
    - Pushdown automata
    - Register machines (e.g., Random-access machines)
    - Turing machines

- **In this course, we will focus these models.**

Models of Computation

- **Functional Models:**
  - In computer science and engineering, a **functional model** is a structured representation of the functions (activities, actions, processes, operations) within the modeled system or subject area.
  - Functional models include:
    - Abstract rewriting systems
    - Combinatory logic
    - General recursive functions
    - Lambda calculus

Models of Computation

- **Concurrent Models:**
  - Concurrent models are those models within which the various activities happen at the same time, for faster development and a better outcome. The concurrent model is also referred to as a **parallel working model.**
  - Concurrent models include:
    - Actor model
    - Cellular automaton
    - Interaction nets
    - Kahn process networks
    - Logic gates and digital circuits
    - Petri nets
    - Synchronous Data Flow

What Is the Theory of Computation For?

- In the real world, the theory has helped with several projects. For one, a group of computer scientists used the theory to test a book vending machine design. Register machine models are also based on the Theory of Computation, among other things.

- **The theory is applicable to other fields besides computer science, such as engineering and life and social sciences. It is also a fundamental concept in artificial intelligence (AI), natural language processing (NLP), neural networks, and the like.**

- **The Theory of Computation is a basic concept that computer scientists should understand. After all, it reflects the reality of life—no single solution can solve all problems. As such, it is often included in the computer science curriculum of universities.**

- Given a set of problems, it would be a waste of time and effort for computer scientists to create one algorithm to solve all of them. **The Theory of Computation dictates that developers first determine which problems can be solved algorithmically and which ones can't. After that, they would also need to find out how efficient the algorithm would be in terms of time and memory space spent solving the problem.**

What Is the Theory of Computation For?

- **Applications of Theory of computation:** The theory of computation is applied in the following fields−
  - Traffic lights.
  - Lifts and elevators.
  - Marketing.
  - Compilers.
  - Operating Systems
  - Networks
  - Software Engineering
  - Cloud computing.

?

THE END