CS112: Theory of Computation (LFA)

Lecture 1: Introduction

Dumitru Bogdan

Faculty of Computer Science
University of Bucharest

February 28, 2025

Table of contents

Section 1

What is this course about

Theory of Computation

You are about to embark on the study of a fascinating and important subject **the theory of computation** and **complexity theory** (third year)

It comprises the fundamental mathematical properties of computer hardware, software, and certain applications thereof.

In studying this subject, we seek to determine **what can and cannot be computed**, how quickly, with how much memory, and on which type of computational model.

Theory of Computation

This course remains fundamentally important because:

Foundational Knowledge: This course provides the theoretical backbone of computer science. Concepts like finite automata, regular expressions, context-free grammars, and Turing machines explain how computation works at its core. Understanding these ideas is essential for any student aiming to grasp the principles behind modern software, compilers, or even artificial intelligence systems.

Critical Thinking: The course trains students to approach problems methodically. For example, proving whether a language is regular or designing an automaton to recognize it requires logical reasoning and step-by-step analysis—skills that enhance a student's ability to tackle complex challenges in any domain.

Theory of Computation

Formalization: Students learn to take vague, real-world problems and express them in precise, mathematical terms (e.g., defining a language or constructing a grammar). This ability to formalize is invaluable in fields like software design, where clear specifications are critical.

Abstraction: By working with models like automata, students practice abstracting away irrelevant details to focus on a problem's essence. This skill is transferable to areas like algorithm design, system architecture, and even interdisciplinary fields like data science or bioinformatics.

Understanding Computational Limits: Topics like undecidability and the halting problem teach students what computers cannot do. In an era where AI is often overhyped, this perspective is crucial for developing realistic expectations and innovative solutions.

Complexity Theory

Computer problems come in different varieties; some are easy, and some are hard.

For example, the sorting problem is an easy one. Say that you need to arrange a list of numbers in ascending order. Even a small computer can sort a million numbers rather quickly.

Schedule of classes for the entire university to satisfy some reasonable constraints, such as that no two classes take place in the same room at the same time. **finding best schedule may require centuries, even with a supercomputer**.

Complexity Theory

What makes some problems computationally hard and others easy?

This is the central question of complexity theory. Remarkably, we don't know the answer to it, though it has been intensively researched for over 70 years.

In one important achievement of complexity theory thus far, researchers have discovered an elegant scheme for classifying problems according to their computational difficulty.

It is analogous to the periodic table for classifying elements according to their chemical properties.

Computability Theory

In the beginning of the twentieth century, Kurt Gödel, Alan Turing, and Alonzo Church discovered that certain basic problems cannot be solved by computers.

One example of this phenomenon is the problem of determining whether a mathematical statement is true or false.

```
Program(n)
    Input integer n>1
    while (n!=1)
        if (n is even)
            return Program(n/2)
        else
            return Program(3n+1)
```

Computability Theory

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.

Automata Theory

The theories of computability and complexity require a precise definition of a computer.

Automata theory deals with the definitions and properties of mathematical models of computation.

Section 2

What's in it for me

Benefits&Gains

For you:

1. **Level 1**: Nothing, zero, $\emptyset$
2. **Level 2**: Pass the exam and move on
3. **Level 3**: Little more than ∞
   - easy way into Complexity Theory, Machine Learning, Reinforcement Learning
   - differentiate yourself from the pack
   - industry is (still) heavily using automata and regexp
   - create new languages, compilers
   - create new AI models
   - become famous (separating words problem)

Benefits&Gains

For me:

- **Level 1**: Less than 10% of you to get lost between level 1 and level 2
- **Level 2**: More than 50% of you to land between (your) level 2 and level 3
- **Level 2**: Cure or at least diminish abstractophobia and theoriphobia for some of you
- **Level 4**: Someone to solve a really hard problem

Section 3

How to do well

Few tips

- You must understand the concepts well!
  –If you do not , there is almost no chance of success =(
- Topics get hard quickly
- Attending lectures/seminars/labs is highly advised!
- Participate in class!
- Don't postpone learning; you will not be able to "make up" later.
  –Start the homework early on!
- You must do problems. There's no replacement for this.
  -Discuss problems and solutions in group but always write your own solutions

Section 4

Administrative details

Team members

- Dumitru Bogdan - lectures and labs
  bogdan.dumitru1@unibuc.ro
  bogdan.dumitru@fmi.unibuc.ro

- Cheval Horațiu - seminars
  andrei-horatiu.cheval@unibuc.ro
  horatiu.cheval@fmi.unibuc.ro

Textbook and course materials

- Textbook: Michael Sipser - Introduction to Theory of Computation. (3^{nd} edition recommended).
- Course materials: teams (chanel code is: ugevl2z)
- Seminar and labs materials: only for class attendees
- Interaction: teams, email

Grading

The following components will contribute with indicated points to your grade:

1. Lab homework - 30 points
2. Seminar activity - 30 points
3. Exam - 90 points

To get a 10 you need 100 points, for a 5 you need 50 points.

There is more:

1. Ad-hoc - 10+ points
2. Working on hard problems - 100+ points

Section 5

Math review

- Mathematical Notation
  - Sets
  - Sequences and Tuples
  - Functions and Relations
  - Graphs
  - Strings and Languages
  - Boolean Logic

- Proofs and Types of Proofs
  - Construction
  - Contradiction
  - Induction

Sets

A **set** is a group of objects represented as a unit. Sets may contain any type of object, including numbers, symbols, and even other sets. The objects in a set are called its elements or members Formally may be described as listing elements inside braces:

$$S = \{7, 21, 57\}$$

Order is not important.

$$S = \{1, 2, 3, \dots\}$$

One way of writing an infinite set.

Sets

When we want to describe a set containing elements according to some rule we write:

$$\{n \mid \textit{rule about } n\}$$

So, the set of perfect squares is:

$$\{n \mid n = m^2 \textit{ for some } m \in \mathbb{N}\}$$

Sets

Most used operations on sets are:

- **union**: $A \cup B$ - is the set we get by combining all the elements in A and B into a single set
- **intersection**: $A \cap B$ - is the set of elements that are in both A and B
- **complement**: $\overline{A}$ - is the set of all elements under consideration that are not in A
- **Cartesian product**: $A \times B$ is the set of all ordered pairs wherein the first element is a member of A and the second element is a member of B
- **Cardinality**: $|A|$ - number of elements in set A

Sets

Power Set: All possible subsets of a set

- If $A = \{0, 1\}$ then $P(A) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$
- cardinality of $P(B) = 2^{|B|}$

Sequences and Tuples

- A sequence is a list of objects, order matters
  - Example: $(1, 3, 5)$ or $(5, 3, 1)$
- In this course we will use term tuple instead
  - $(1, 3, 5)$ is a 3-tuple and a k-tuple has k elements

Functions and Relations

- A function maps an input to a (single) output
  - $f(a) = b$ - f maps a to b
- The set of possible inputs is the **domain** and the set of possible outputs is the **range**
  - $f : D \to R$
  - Example 1: for the *abs* function, if $D = \mathbb{Z}$, then $R = \mathbb{N}$
  - Example 2: *sum* function $sum : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$
- Functions can be described using tables
  - Example: Describe $f(x) = 2x$ for $D = \{1, 2, 3, 4\}$

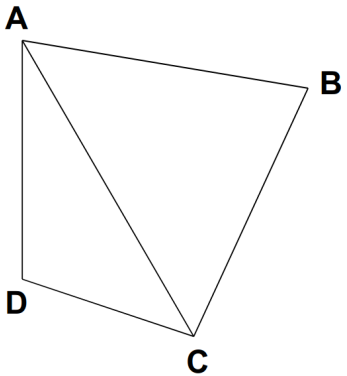
x	f(x)
1	2
2	4
3	6
4	8

Relations

- A predicate is a function with range $\{True, False\}$
  - Example: $even(4) = True$
- A (k-ary) relation is a predicate whose domain is a set of k-tuples $A \times A \times A \cdots \times A$
  - If $k = 2$ then binary relation (e.g., $=, <$)
- Relations may have 3 key properties:
  - reflexive, symmetric, transitive
  - A binary relation is an equivalence relation if it has all 3
  - Example: $=, <$, *friend*

Graphs

- A graph is a set of vertices V and edges E
  - $G = (V, E)$ and can use this to describe a graph
  - $V = \{A, B, C, D\}$
  - $E = \{(A, B), (A, C), (C, D), (A, D), (B, C)\}$



Graphs

Few important definitions:

- The **degree** of a vertex is the number of edges touching it
- A **path** is a sequence of nodes connected by edges
- A **simple path** does not repeat nodes
- A path is a **cycle** if it starts and ends at same node
- A **simple cycle** repeats only first and last node
- A graph is a **tree** if it is connected and has no simple cycles

Strings and Languages

- This is essential for this course
- An **alphabet** is any non-empty finite set
  - Members of the alphabet are alphabet symbols
  - $\Sigma_1 = \{0, 1\}$
  - $\Sigma_2 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$
  - $\Sigma_3 = \{good, god, bad\}$
- A **string** over an alphabet is a finite sequence of symbols from the alphabet
  - 0100 is a string from Σ_1 and *cat* is a string from Σ_2

Strings and Languages

- The length of a string w, $|w|$ is its number of symbols
- The empty string, ϵ, has length 0
- If w has length n then it can be written as $w = w_1 w_2 w_3 \dots w_n$ where $w_i \in \Sigma$
- Strings can be concatenated
  - ab is string a concatenated with string b
  - a string x can be concatenated with itself k times. It is written x^k
- A **language** is a set of strings

Boolean Logic

- Boolean logic is a mathematical system built around True (or 1) and False (or 0)
- Below are the boolean operators, which can be defined by a truth table

$$0 \wedge 0 = 0$$
$$0 \wedge 1 = 0$$
$$1 \wedge 0 = 0$$
$$1 \wedge 1 = 1$$

$$0 \vee 0 = 0$$
$$0 \vee 1 = 1$$
$$1 \vee 0 = 1$$
$$1 \vee 1 = 1$$

$$\neg 0 = 1$$
$$\neg 1 = 0$$

Boolean Logic

- More boolean operators

$$0 \oplus 0 = 0 \qquad 0 \leftrightarrow 0 = 1 \qquad 0 \rightarrow 0 = 1$$
$$0 \oplus 1 = 1 \qquad 0 \leftrightarrow 1 = 0 \qquad 0 \rightarrow 1 = 1$$
$$1 \oplus 0 = 1 \qquad 1 \leftrightarrow 0 = 0 \qquad 1 \rightarrow 0 = 0$$
$$1 \oplus 1 = 0 \qquad 1 \leftrightarrow 1 = 1 \qquad 1 \rightarrow 1 = 1$$

Proofs

Proofs are a big part of this class.

What is a proof?

A proof is a convincing logical argument

Proofs

Types of Proofs:

- $A \Leftrightarrow B$ means A if and only if B
- Proof by counterexample (prove false via an example)
- Proof by construction (main proof technique we will use)
- Proof by contradiction
- Proof by induction

Proofs

Prove for every graph G sum of degrees of all nodes is even
This is a proof by induction. Informal reasoning: every edge you add touches two vertices and increases the degree of both of these by 1
A proof by induction means showing

1. it is true for some base case and then
2. if true for any n then it is true for $n + 1$

1. **Base case**: 0 edges in G means $sum - degrees = 0$, is even.
2. **Induction step**: if sum-degrees even with n edges then show even with $n + 1$ edges
   - When you add an edge, it is by definition between two vertices (but can be the same). Each vertex then has its degree increase by 1, or 2 overall
   - even number + 2 = even (we will accept that for now)

Proofs

For any two sets A and B we have $\overline{A \cup B} = \overline{A} \cap \overline{B}$

We prove sets are equal by showing that they have the same elements. Prove in each direction:

$\Longrightarrow$

- assume $x \in \overline{A \cup B}$
- then $x \notin A \cup B$
- then $x \notin A$ and $x \notin B$
- so $x \in \overline{A}$ and $x \in \overline{B}$

$\Longleftarrow$

- assume $x \in \overline{A} \cap \overline{B}$
- then $x \in \overline{A}$ and $x \in \overline{B}$
- then $x \notin A$ and $x \notin B$
- so $x \notin A \cup B$
- so $x \in \overline{A \cup B}$

Proofs

For every even number $n > 3$, there is a $3 -$ *regular* graph with n nodes. A graph is k-regular if every node has degree k.
We will use a proof by construction.

- Many theorems say that a specific type of object exists. One way to prove it exists is by constructing it
- May sound weird, but this is by far the most common proof technique we will use in this course
- We may be asked to show that some property is true. We may need to construct a model which makes it clear that this property is true

Proofs

- Place the nodes in a circle and then connect each node to the ones next to it, which gives us a 2-regular graph
- Then connect each node to the one opposite it and you are done
- This is guaranteed to work because if the number of nodes is even, the opposite node will always get hit exactly once

Proofs

Prove $\sqrt{2}$ is irrational
Proof by contradiction, assume it is rational

- Rational numbers can be written as m/n for integer m, n
- Assume with no loss of generality we reduce the fraction $\sqrt{2} = m/n$
- This means that m and n cannot both be even
- Let's do some math:

$$n\sqrt{2} = m$$
$$2n^2 = m^2$$

- This mean that m^2 is even so m must be even