Theory of Computations

Lecturer: Dr. Manar Elkady

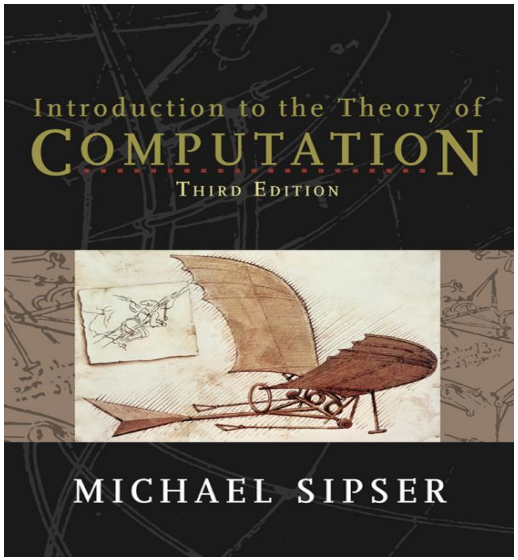
Topics covered

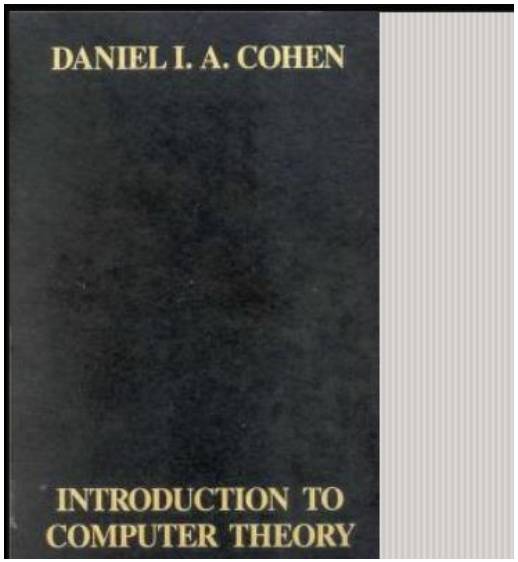
- Formal Modeling
- Basic Concepts of Languages and Grammar
- Automata
- Regular languages
- Context free
- Push down Automata
- Context Sensitive Grammar
- Turing Machines

Course Assessment

- Assignments (10)
- project (10)
- Midterm (20)
- Final (60)

Course Material





- Theory of Computations in **Google Classroom**. Code is sharing in the lecture

- **Books** used for lecture:
  - Introduction to the Theory of Computation, Third edition, Michael Sipser.
  - Introduction To Computer Theory By Daniel I. A Cohen 2nd Edition

Why to study Theory of Computation?

- It is the most abstract course in the whole curriculum
- But a fundamental course for all CSians
- This course is not about how to develop a program or build a computer
- It helps you know a lot more about how people looked to CS as a sciencein the past 70 years

Why to study Theory of Computation?

- It is about:
  - What kind of things can we really compute?
  - How fast we can do it? and
  - How much space/memory it needs to be computed?

Why to study Theory of Computation?

- This course is about different models of computations you can use to compute stuff
- But ***abstracted***from the hardware/machine specifications
- We will deal with kinds of problems that takes ***inputs***, and ***process it***, then ***decide***whether these inputs are ***acceptable***or not
- Inputs are set of symbols formed by an alphabet (binary, English alphabet, ...)

Why to study Theory of Computation?

- There are a lot of applications that connect to this course:
- Theory behind writing *compilers/translators*for programming languages
- *Computer Architecture*→ model particular processes/states using finite-state-machines (FSMs)
- *String search*algorithms, *word processing*algorithms, any kind of editors → that is a RE/FSM
- When you do a representation of a language like XML, it is just a *grammar-writing*

Turing Machine

- Turing Machine invented by *Alan-Turing* in 1940's
- It is an abstraction of how programming languages and computers work nowadays
- He invented a *mathematical abstraction*of a complete representation of how might we can do a computation
- Any problem that can be computed by a program on a computer, it can be *computed by*(*simulated on*) a Turing Machine

Turing Machine

- Turing Machine is like a *human with organs*, cutting one organ (going down into lower machine) will make it handicap, or taking higher time and resources
- The lower level is the FSM, and then add up pieces to form more powerful machines (till getting a Turing Machine)
- Then you will get into the *twilight zone* that has problems can't be computed