
Introduction to TOC

Introduction

Mathematics:
What can be computed?
And what can't be computed?

Computer Science

Electrical Engineering:
How can we build computers?

TOC - Comprises the fundamental mathematical properties of computer HW, SW, and other applications.

Theory of Computation – TOC

Theory of Computation **tries to answer the following questions:**

- **What are the *mathematical properties* of computer hardware and software?**
- **What is a *computation* and what is an *algorithm*? Can we give rigorous mathematical definitions of these notions?**
- **What are the *limitations of computers*? Can *everything* be computed?**

Purpose of TOC >> Develop formal mathematical models of computation that reflect real-world computers.

Our main purpose is to determine:

- what can and can't be computed?
- how quickly?
- with how much memory?
- **on which type of computational model?**

Central Areas - TOC

Theory of Computation has *3* central areas:

- Complexity Theory
- Computability Theory
- Automata Theory

They are linked by the question:

What are the fundamental *capabilities and limitations* of computers?

In each of the three areas

— Automata, Computability, and Complexity —

this question is interpreted differently, and the answers vary according to the interpretation.

1 – Complexity Theory

What makes some problems computationally hard and other problems easy?

- Though it has been intensively researched for more than 40 years but *we don't know the answer* to it.
- One important achievement by exploring this question and some of its ramifications:

Researchers have discovered an elegant scheme for classifying problems according to their computational difficulty(easy or hard) even if there exists no proof.

- Informally, a problem is called "easy" if it is efficiently solvable.
  Ex - sorting a sequence of , say, 1000000 numbers
- On the other hand, a problem is called "hard", if it can't be solved efficiently, or if we don't know whether it can be solved efficiently.
  Ex - factoring a 300 digit integer into its prime factors

Motivation of Complexity Theory >> Classify problems according to their degree of "computational difficulty" by giving a rigorous proof that certain problems that seem to be "hard" are really "hard".

2 – Computability Theory

Which problems can be solved by computers and which ones cannot?

In the 1930's Gödel, Turing, and Church discovered that some of the fundamental mathematical problems can't be solved by a "computer".

Ex - Determining whether a mathematical statement is true or false!!

It seems like a natural for solution by computer but no computer algorithm *can perform this task*.

The theoretical models that were proposed in order to understand solvable and unsolvable problems led to the development of real computers.

Motivation of Computability Theory >> Classify problems as being solvable or unsolvable.

3 - Automata Theory

Automata Theory deals with the *definitions and properties* of different types of "mathematical models of computation"

For example,

- Finite Automata - These are used in text processing, compilers, and hardware design.
- Context-Free Grammars - These are used to define programming languages and in Artificial Intelligence.
- Turing Machines - These form a simple abstract model of a "real" computer, such as your PC at home.

Motivation of Automata Theory >> Determine the power of different computational models or, which model can solve more problems than the other.

TOC - History

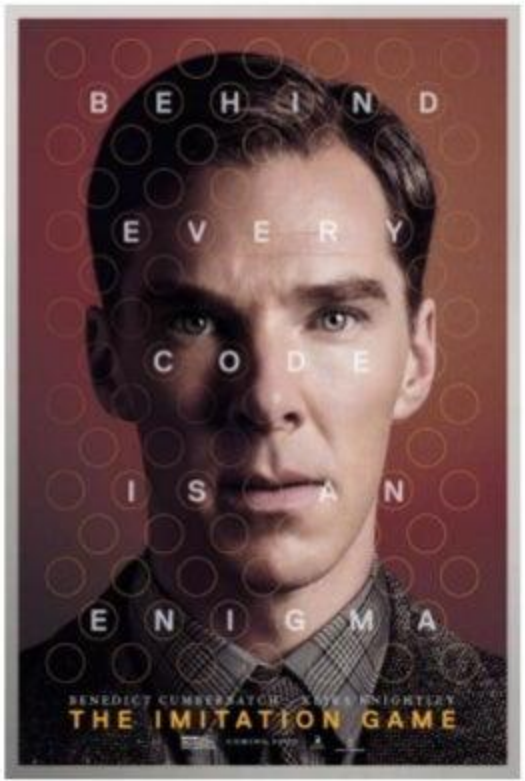
1930s	• Alan Turing studies Turing Machines • Decidability • Halting problem
1940-1950s	• Finite automata machines studied • Noam Chomsky proposes the “Chomsky Hierarchy” for formal languages
1969	• Cook introduces “intractable” problems or “NP-Hard” problems
1970 -	• Modern computer science: compilers, computation and complexity theory evolve

Alan Turing - Facts

Dr. Alan Turing(1912-1954) is one of the founders of Computer Science (he was an English Mathematician).

Important facts:

- "Invented" Turing machines.
- "Invented" the Turing Test.
- Broke the German submarine transmission coding machine "Enigma".
- Was arraigned for being gay and committed suicide soon after.



Why this course?

- This course is about the fundamental capabilities and limitations of computers. These topics form the *core of computer science*.
- It is about *mathematical properties* of computer hardware and software.
- To *design* new programming languages, compilers, string searching, pattern matching, computer security, artificial intelligence etc. this course will provide you the basics.
- This course helps you to *learn problem solving skills*. Theory teaches you how to think, prove, argue, solve problems, express, and abstract.
- This theory *simplifies the complex computers* to an abstract and simple mathematical model, and helps you to understand them better.
- This course is about rigorously *analyzing capabilities and limitations* of systems.

Terminology – Mathematical

Sets - Set, element (a.k.a member), subset, proper subset, power set, multiset, infinite set, natural numbers, integers, empty set, union of sets, intersection of sets, complement of a set, Venn diagrams.

Sequences and Tuples - Sequence, k-tuple, ordered pair, Cartesian product (a.k.a. cross product).

Functions and Relations - Function (a.k.a. mapping), domain, range, onto function, arguments of a function, k-ary function, unary function, binary function, infix notation, prefix notation, predicate (a.k.a. property), relation, k-ary relation, binary relation, equivalence relation (reflexive, symmetric and transitive).

Terminology – Graphs , Boolean

Graphs - Undirected graph, nodes (a.k.a. vertices), edges, degree of a node, labeled graph, subgraph, path, simple path, connected graph, cycle, simple cycle, tree, root of a tree, leaves of a tree, directed graph, outdegree, indegree, directed path, strongly connected graph.

Boolean Logic - Boolean logic, boolean values (True and False), boolean operations, negation (a.k.a. NOT), conjunction (a.k.a. AND), disjunction (a.k.a. OR), operands of an operation, exclusive OR (a.k.a XOR), equality, implication, distributive law.

Terminology – Strings

- An *alphabet* is a finite, nonempty set of symbols.

Conventionally, we use the symbol Σ for an alphabet. For example,

i. $\Sigma = \{ 0, 1 \}$, the binary alphabet.

ii. $\Sigma = \{ a, b, c, \ldots \ldots \ldots, z \}$, the set of all lower-case letters.

- A *string over an alphabet* Σ is a finite sequence of symbols, where each symbol is an element of Σ

- The string w where $w = w_1 w_2 w_3 \ldots \ldots w_n$ represents a string of length n where each $w_i \epsilon \sum$
- The reverse of w is represented as w^R

- The length of w denoted by $|w|$, is the number of symbols contained in w.
- The empty string, denoted by ε, is the string having length zero.

Ex: if the alphabet Σ is equal to {0, 1}, then 10, 1000, 0, 101, and ε are strings over Σ, having lengths 2, 4, 1, 3, and 0 respectively.

Terminology – Strings

Concatenation of Strings:

Let x and y be strings. Then xy denotes the concatenation of x and y, that is, the string formed by making a copy of x and following it by a copy of y.

More precisely, if x is the string composed of i symbols $x = a_1a_2 \ldots a_i$ and y is the string composed of j symbols $y = b_1b_2 \ldots b_j$, then $xy = a_1a_2 \ldots a_ib_1b_2 \ldots b_j$

For example,

if $x = 01101$ and $y = 110$, then $xy = 01101110$

if $x = 01101$, then $\varepsilon x = x\varepsilon = x$ holds.

Terminology – Strings

Powers of an Alphabet: Σ^k is the set of strings of length exactly *k*, each of whose symbols is in Σ

For example,

$\Sigma^0 = \{\varepsilon\}$ regardless of Σ

if $\Sigma = \{ 0, 1 \}$, then

$\Sigma^1 = \{ 0, 1 \}$
$\Sigma^2 = \{ 00, 01, 10, 11 \}$
$\Sigma^3 = \{ 000, 001, 010, 011, 100, 101, 110, 111 \}$

The set of all strings over an alphabet Σ is conveniently denoted as Σ^*, where

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

For example, $\{ 0, 1 \}^* = \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots \}$

The set of all strings except the empty string is denoted as Σ^+, where

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

Terminology – Languages

Language:

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 $L \subseteq \Sigma^*$, then L is a language over Σ

For example,

- Language, L
  = set of strings of 0's and 1's with an equal number of each
  = {ε, 01, 10, 0011, 0101, 1001, 0110, … … }
- Σ^* is a language for any alphabet Σ
- $\emptyset$, the empty language, is a language over any alphabet
- {ε}, the language consisting of only the empty string. Notice that, $\emptyset \neq \{\varepsilon\}$

Terminology – Definition, Theorems, and Proofs

Definition - It describes the objects and notions that we use.

Mathematical Statement - It expresses that some object has a certain property.

Proof - It is a convincing logical argument that a statement is true.

Theorem - It is a mathematical statement proved true.

Lemma - Proved mathematical statements that assist in the proof of another more significant statement.

Corollaries - The statements easily concluded from a theorem or its proof.

Terminology – Types of Proofs

Proof by Construction:

Many theorem state that a particular type of object exists. One way to prove such a theorem is by demonstrating how to construct the object. This technique is a *proof by construction*.

Proof by Contradiction:

First we will assume that the theorem is false and then show that this assumption leads to an obviously false consequence, called a contradiction.

Proof by Induction:

Useful for proving statements for infinite sets. It consists of two parts: the basis and the induction step. Each part is an individual proof on its own.

- The *basis* proves that P(1) is true.
- The *induction step* proves that for each i>=1, if P(i) is true then so is P(i+1).

THANKS!

Any questions?

References:

Chapter 1, Introduction to the Theory of Computation, 3rd Edition by Michael Sipser