Foundations of Computer Science

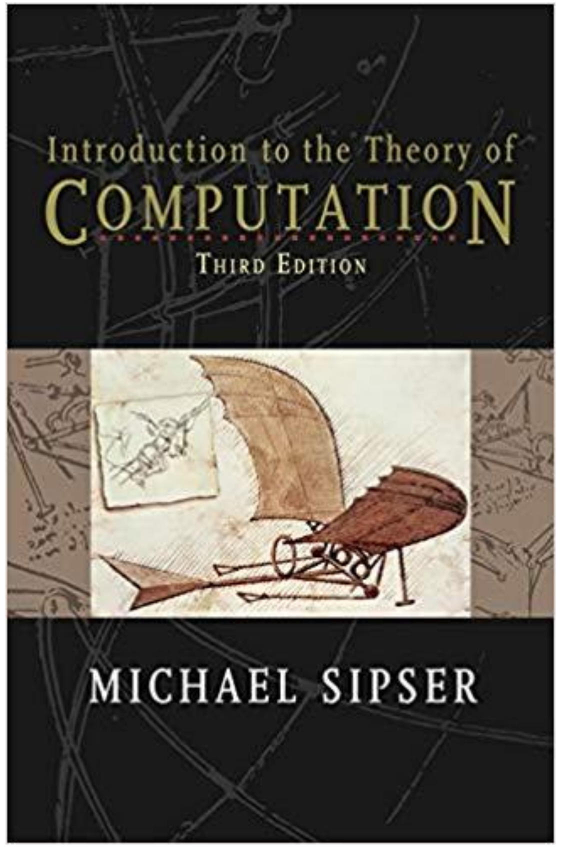
CSC320

Administrative Details

- Instructor: Rich Little
- Office: ECS 516
- E-mail: rlittle@uvic.ca
- Office hours: MR 1:00 – 2:00 p.m.
- Course Website: https://connex.csc.uvic.ca/portal
- Course Outline: https://heat.csc.uvic.ca/coview/course/2019091/CSC320
- Tutorials
    - T01 - T 12:30-13:20 (COR B135),
    - T02 - W 12:30-13:20 (COR B108),
    - T03 - W 13:30-14:20 (ECS 104)
    - Start Tuesday, September 10

Administrative Details

- Textbook – *Introduction to the Theory of Computation*, 3rd Edition, Michael Sipser



- Grading Scheme

Coursework	**Weight (out of 100%)**
Homework Assignments	30%
Midterm Exams	20%
Final Exam	50%

Administrative Details

- Two Midterms:
  - Worth 10% each
  - Midterm 1 – Monday, October 7, 2019
  - Midterm 2 – Monday, November 18, 2019

- Six Assignments (Tentative):

Assignment	Assigned Date	Due Date
1	Sep 13	Sep 20
2	Sep 27	Oct 4
3	Oct 11	Oct 18
4	Oct 25	Nov 1
5	Nov 8	Nov 15
6	Nov 22	Nov 29

What is this course about?

- We will study the *fundamental nature* of *computation*

Can you be a little more specific?

- What *problems* are *solvable* by a *computer*?

OK, but…

1. What is a *problem*?
2. What is a *solution*?
3. How do we model a *computer*?

Let’s start with 3 – modeling computation

- In this course, we address computational models from *three* perspectives
    1. Automata theory
    2. Computability theory
    3. Complexity theory

Complexity theory

- Hard vs easy (= solvable in polynomial time)
  - Can we schedule exams over a two week period so that no student is scheduled to take two finals at the same time? (Scheduling)
  - Can we pair up students with co-op jobs so that there is no (student, employer) who are not paired up but would prefer each other? (Stable marriage)
  - In a network, can we route calls from the origins to their destinations so that no two share a common link? (Disjoint paths)
  - Can we send messages secretly without a prior set-up (Public key cryptography)
  - Can we filter out bots by creating a test that only humans can solve quickly? (CAPTCHA)

- Basic question: what computational problems are *hard to solve*?
  - Flip side of the theory of algorithms (CSC 225,226)

- Why are they hard to solve?
  - We don’t know!

Achievements of Complexity Theory

- Developed an elegant scheme for classifying problems by their difficulty
  - Evidence they are hard without the proof
- If you can pinpoint the root of the difficulty, you may alter the problem to make it easier
- Discovery of approximation algorithms
- Discovery of random algorithms
- Application of complexity theory to cryptography

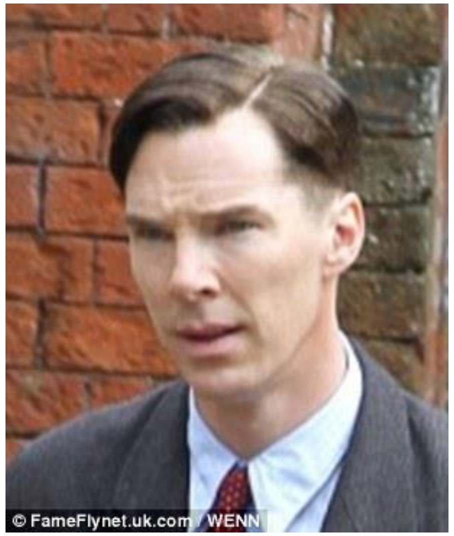
Computability theory

- Basic question: are there problems which are impossible to solve computationally
- Origins: Hilbert’s program (early 20th century) – mechanization of mathematics
- 1930s – Church, Turing and Gödel demolish Hilbert’s program, and in the process propose a mathematical model of computation
    - They prove these models are equivalent
    - Church-Turing thesis – Any problem solvable in principle by a computer can be solved on a Turing Machine (TM)

Computability theory

- Now we can ask: are there problems that cannot be solved by any computer?
  - Halting problem: Given a TM M and an input x, does M halt on x?
  - Turing showed that this problem is *undecidable*

$\neq$



Why is this interesting?

- Real-world problems
  - Universal debuggers?
  - Universal interpreters?
  - Universal malware detectors?
- Philosophical/physical implications
  - Can humans solve problems not solvable by computers?
  - Do assumptions about the physical world (e.g. quantum mechanics) make a difference?
  - What about other extensions of the model (e.g. nondeterminism)?
- A whole new perspective – understanding what *cannot be solved by computer*

Automata theory

- Complexity theory imposes limitations on computational resources of Turing machines (e.g., time, space, randomness, nondeterminism) as a function of input size
- We could also try to put structural limitations on the model
    - More natural to do this with space then with time
    - Finite (fixed) memory – finite automata (FAs)
    - Unbounded memory, organized as a stack – pushdown automata (PDAs)
- Why?
    - Traditional area of study with important applications (e.g. in compilers)
    - Good warm-up for later topics

Finite automata

- Models computation with a finite amount of memory, independent of problem size
- Corresponds to many real-life systems – finite control systems, digital circuits, communication protocols, lexical analyzers, concurrent systems
- Important equivalence – finite automata and regular expressions

Pushdown automata

- Allows unbounded memory – but organized as a stack
- Important model for language processors
- PDAs are equivalent to context-free grammars

Course Outline

1. Preliminaries – alphabets, languages and problems
2. Finite automata
    1. Basic definitions – regular languages
    2. Constructions which preserve regularity
    3. Nondeterministic FAs (NFAs) and their equivalence to deterministic FAs
    4. Regular expressions (REs) and their equivalence to Fas
    5. FA minimization
    6. Proving that languages are not regular – the Pumping Lemma

Course Outline

3. Pushdown automata (PDAs)
    1. Context-free grammars (CFGs) – derivations, ambiguity, context-free languages
    2. Normal forms and algorithms
    3. Equivalence of PDAs and CFGs
4. Turing machines (TMs)
    1. Basic model – single tape TM
    2. Extensions – multitrack, multitape, nondeterministic
    3. Equivalence of all extensions to basic model – Church-Turing thesis
    4. Simulation of a general-purpose computer by a TM
    5. Encoding – allowing TMs to take structured data as input

Course Outline

5. Undecidability
    1. TM acceptance – an undecidable problem
    2. Using the undecidability of TM acceptance to show that other problems are undecidable
    3. Reductions as a tool for proving undecidability
6. Complexity and intractability
    1. Polynomial time TMs and the class P
    2. Nondeterministic poly-time TMs and the class NP
    3. The P vs NP problem
    4. NP-completeness
    5. The Cook-Levin Theorem

The P vs NP Problem

- This is the culmination of our course
- One of the most fundamental problems in Computer Science, and also Mathematics (one of the Clay Foundation’s seven *Millenium Problems*)
- Basic question: Is it harder to *solve* a problem than to *verify* a solution?
- Introduced by Stephen A. Cook in 1971 – very little significant progress has been made (other than ruling out possible approaches)
- Read *Scientific American* article “Machines of the Infinite” – link available in **Resources** folder on course web page