計算理論 Theory of Computation

顏嗣鈞

Dept. of Electrical Engineering
National Taiwan University

- E-mail: hcyen@ntu.edu.tw
- Web: http://www.ee.ntu.edu.tw/ ~ yen
- Time: 2:20-5:20 PM, Tuesday
- Place: BL 112
- Office hours: by appointment
- Class web page:
  http://ccf.ee.ntu.edu.tw/ ~ yen/courses/TOC-2024.html

Prerequisites and Grading

- Prerequisites:

Familiar with basic materials in discrete mathematics, such as sets, relations, functions, graphs, propositional logic, induction principle, ...

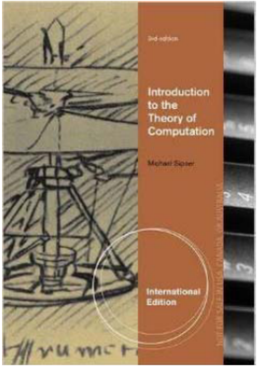
- Grading:
  - Homework : 20 %
  - Midterm exam.: 40 %
  - Final exam.: 40 %

This is not a programming course; there will be NO programming assignments.

Reference Book – Michael Sipser, 3rd Edition

Introduction to the Theory of Computation

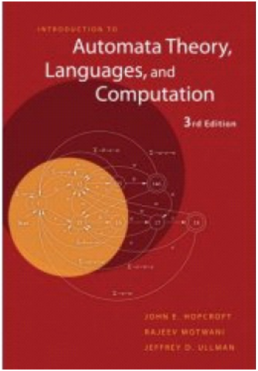
Michael Sipser
(Thomson, 2012)



Reference Book – HMU 3rd Edition

Introduction to Automata Theory, Languages, and Computation

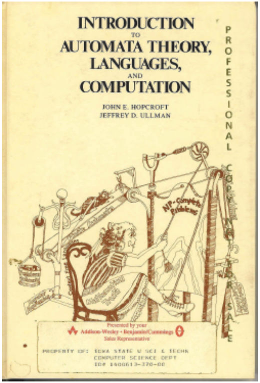
John E. Hopcroft, Rajeev Motwani,
Jeffrey D. Ullman
(Addison-Wesley, 2006)



Reference Book – HU 1st Edition

Introduction to Automata Theory, Languages, and Computation

John E. Hopcroft, Jeffrey D. Ullman
(Addison-Wesley, 1979)


INTRODUCTION TO AUTOMATA THEORY, LANGUAGES, AND COMPUTATION
JOHN E. HOPCROFT
JEFFREY D. ULLMAN
PROFESSIONAL COPY NOT FOR SALE
NP-Complete Problems
Presented by your Addison-Wesley • Benjamin/Cummings Sales Representative
PROPERTY OF: IOWA STATE U SCI & TECHN
COMPUTER SCIENCE DEPT

Aims of the Course

- To familiarize you with key Computer Science concepts in central areas like
  - Automata Theory
  - Formal Languages
  - Models of Computation
  - Complexity Theory
  - ...
- To equip you with tools with wide applicability in the fields of CS and EE, e.g. for
  - Software/Hardware Verification
  - Cryptography
  - Discrete Event Dynamic System
  - Quantum Computing
  - ...

Fundamental Theme

- What are the capabilities and limitations of computers and computer programs?
  - What can we do with computers/programs?
  - Are there things we cannot do with computers/programs?

Studying the Theme

- How do we prove something **CAN** be done by **SOME** program?
- How do we prove something **CANNOT** be done by **ANY** program?

Example: The Halting Problem

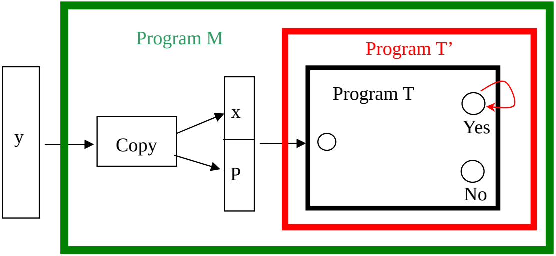
Consider the following problem:

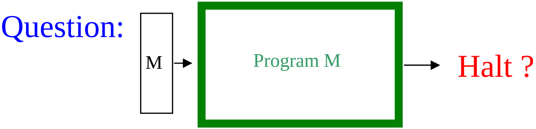
- **Input:** A program P with input x
- **Goal:** Decide whether P halts on x eventually.

- It turns out that the above problem is undecidable, meaning that it is impossible to write a program that gives the correct answer.

- What might be surprising is that it is possible to prove such a result formally. This was first done by the British mathematician **Alan Turing.**

Proof of the Halting Problem





- Halt: T enters "Yes" $\Rightarrow$ Not Halt
- Not Halt: T enters "No" $\Rightarrow$ Halt

A Related "Halting Problem" (The $3n + 1$ Problem)

Consider the following program. Does it terminate for all values of $n \geq 1$?

```
while $(n > 1)$
        if even$(n)$
          $n = \frac{n}{2}$;
        else
          $n = n * 3 + 1$;
```

The $3n + 1$ Problem (cont'd)

Not as easy to answer as it might first seem.

Say we start with $n = 7$, for example:

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

In fact, for all numbers that have been tried (a lot!), it does terminate ...

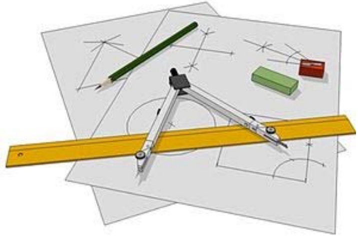
... but in general?

The problem remains **open**!

What is ”Computation”?

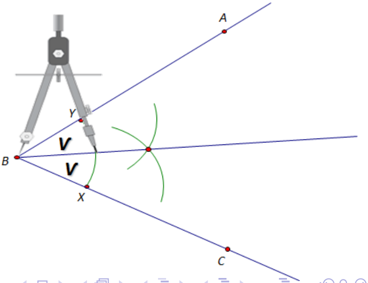
Ruler and Compass Construction

Plato (5th century B.C.) believed that the only ”perfect” geometric figures were the straight line and the circle.



In Ancient Greek geometry, there were only two instruments available to perform geometric constructions (computations):

- **Ruler:** It can only be used to draw a line segment between two points, or to extend an existing line segment.
- **Compass:** Circles and circular arcs can be drawn starting from two given points: the center and a point on the circle.



Ruler and Compass Construction (cont'd)

The ancient Greeks were unable to solve the following problems:

Squaring the circle

Draw a square with the same area as a given circle.

Doubling the cube

Draw a cube with twice the volume of a given cube.

Trisecting an angle

Divide an angle (such as 60^o) into three smaller angles of the same size.

In 1837, Pierre Wantzel used *Field Theory* to prove that the above three constructions were impossible.

Finding Roots of Polynomials

Roots of polynomials

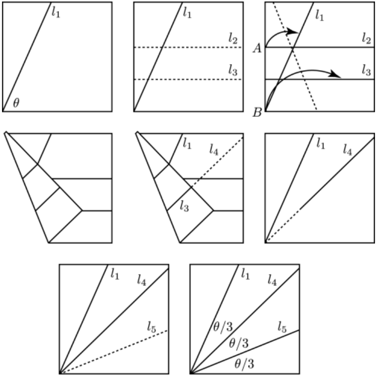
Given a polynomial $a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 = 0$, find its roots (in $\mathbb{C}$) in terms of a finite number of additions ($+$), subtractions ($-$), multiplications ($\times$), divisions ($\div$), and root extractions ($\sqrt[n]{\cdot}$).

- For $n = 1, 2, 3, 4$, they are solvable; however, the general quintic (of degree 5) cannot be solved algebraically. Recall that the solutions of $ax^2 + bx + c = 0$ are $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, if $a \neq 0$.
- The problem was shown by Abel (1824) (also Ruffini 1813) to be impossible using a tool in abstract algebra now known as **Galois Theory** .

Trisecting an Angle

It is impossible to trisect an arbitrary angle via ruler-and-compass. However, if we use origami instead, an arbitrary angle can be trisected easily.

Paper folding is more powerful than ruler and compass. $\sqrt[3]{2}$ can also be computed using origami.



Subclasses of Real Numbers

Goal: Classify interesting subclasses of real numbers.

1. Natural numbers ($\mathbb{N}$): 0, 1, 2, ...
2. Integers $\mathbb{Z}$: ... -2, -1, 0, 1, 2, ...
3. Rational numbers ($\mathbb{Q}$): $\{\frac{q}{p} \mid p, q \in \mathbb{Z}, p \neq 0\}$
4. Constructable numbers: numbers that can be constructed using ruler and compass. E.g., $\sqrt[2]{2}$
5. Algebraic numbers: numbers that are solutions of a polynomial with integer coefficients. E.g., $\sqrt[3]{2}$ (which is not a constructable number).
6. Transcendental numbers: numbers that are not roots of any integer polynomials. E.g., e, π. Recall that $e = lim_{n \to \infty}(1 + \frac{1}{n})^n$
7. Computable numbers: numbers that can be computed using a Turing machine
8. Real numbers ($\mathbb{R}$): $\sqrt{2}, e, \pi$, ...

Subclasses of Real Numbers

- (1) Natural Number $\subset$ (2) Integer $\subset$
  (3) Rational $\subset$ (4) Constructable $\subset$
  (5) Algebraic $\subset$ (7) Computable $\subset$
  (8) Real
- (1)-(7) are countable; (8) is not countable
- Algebraic + Transcendental = Real
- Proving a number being transcendental is usually DIFFICULT.
- Is $e + \pi$ an irrational number? (Open problem)
- How to prove that e is transcendental?
  One way is to use the idea similar to proving e to be irrational.
  (Proof sketch) $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{n!} + \ldots$
  Suppose e were rational, $e = \frac{q}{p}$.
  $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{p!} + \frac{1}{(p+1)!} + \ldots =$
  $\frac{integer}{p!} + \frac{1}{p!}(\frac{1}{p+1} + \frac{1}{(p+1)(p+2)}\ldots) = \frac{integer}{p!} + \frac{r}{p!}$, $0 < r < 1$ – a contradiction.

What is ”Computation”?

- What the ruler-and-compass construction and finding the roots of polynomials have in common?
  - They both involve solving a problem by repeatedly performing operations from a finite set of possible actions.
  - For the former, the operations are ”draw a straight line” and ”draw a circle”; while for the latter, the operations are $+, -, \times, \div, \sqrt[n]{.}$.
- (**Question:**) Are those operations powerful enough to capture the notion of a **computation**?
- **Church-Turing Thesis**: A function can be calculated by an effective method if and only if it is computable by a Turing machine.
- The above notion of computability is quite robust, as

$$\text{Turing computable (Turing)} \equiv \lambda\text{-computable (Church)} \equiv \\ \text{Recursive function definable (Gödel)}$$

Axiomatic Set Theory

- The **Russell's Paradox** (1901): A barber shaves anyone who does not shave himself, and none else. The question is, does the barber shave himself?
- The Russell's paradox exposed a huge problem for the ”naïve” set theory, and changed the entire direction of twentieth century mathematics.
- **Naïve set theory**: a set is just a collection of objects that satisfy some conditions. What happens if we define the set $X = \{a : a \notin a\}$. Is $X \in X$?
- **Modern Set Theory**: The so-called Zermelo-Fraenkel axiomatisation of set theory came to the rescue, using axioms and inference.

Why Study Theory of Computation?

Computation Theory is essential for the study of the limits of computation. Two issues:

- What can a computer do at all?
  (Decidability vs. Undecidability)
- What can a computer do efficiently?
  (Tractability vs. Intractability)

How "Hard" is a Set?

Consider the following sets, can you list their complexities (i.e., difficulties) in ascending order?

1. $A = \{n | n$ is a student enrolled in National Taiwan University$\}$,
2. $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$, i.e., the sets of natural numbers, integers, rational numbers, real numbers, and complex numbers, respectively,
3. $P = \{p | p$ is a prime$\}$,
4. $S = \{G | G$ is a graph with a Hamiltonian cycle$\}$,
5. $H = \{P | P$ is a program that halts$\}$,
6. $\widehat{H} = \{P | P$ is a program that prints a specific symbol infinitely many times$\}$.

Well, it depends on the complexity metrics.

Another Question: Consider the following two sequences $01010101 \cdots$ and $011010010111001 \cdots$, which one is more "complex"?

How to Compare the Difficulty Between Two Sets (or Sequences)?

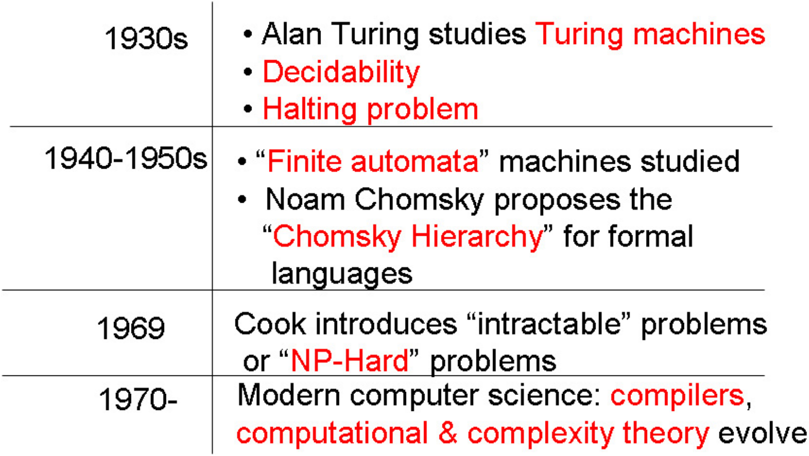
A Possible Attempt: For sets A and B, we write a program P_A (resp., P_B) to answer the following question: Given an x, is $x \in A$ (resp., B)?

- If P_A takes more "resource" than P_B, then A is harder than B.
- Now the question is, what kind of a resource we care most? Time, memory, program size ...?
- What kind of a programming language suitable for the above comparison?

The above idea makes sense, except that the kind of "devices" used for the comparison have to be precise enough so that time, memory, program size ... can be characterized in an accurate manner.

Automata

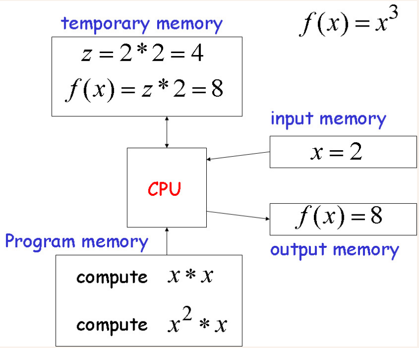
Theory of Computation: A Historical Perspective



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, computational & complexity theory evolve

Computer vs. Automaton

Computer



temporary memory

$$f(x) = x^3$$

$z = 2 * 2 = 4$

$f(x) = z * 2 = 8$

input memory

$x = 2$

CPU

$f(x) = 8$

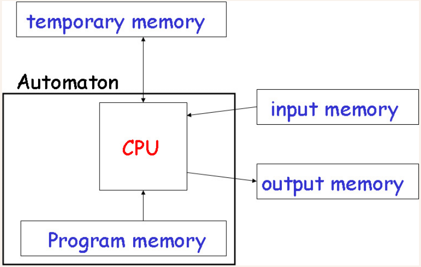
output memory

Program memory

compute $x * x$

compute $x^2 * x$

Automaton



temporary memory

Automaton

CPU

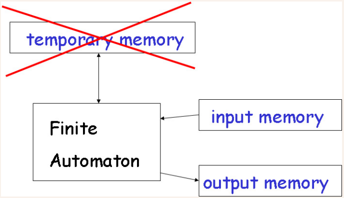
input memory

output memory

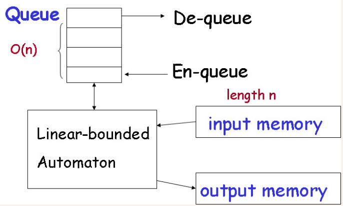
Program memory

Various Automata

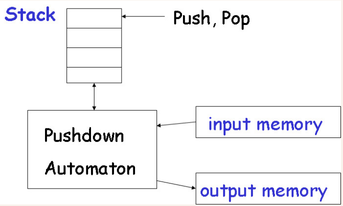
Finite Automaton



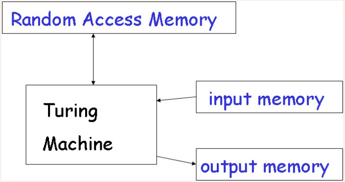
Linear-bounded Automaton



Pushdown Automaton

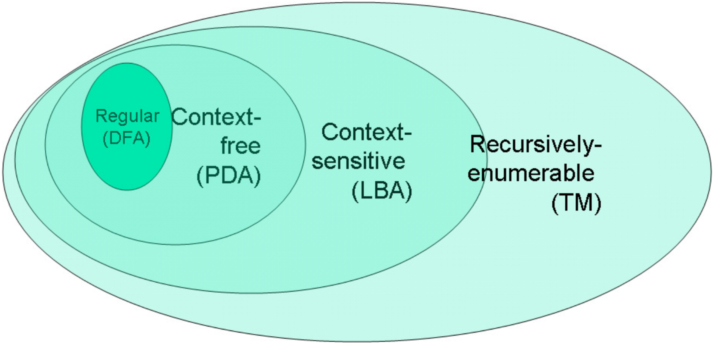


Turing Machine

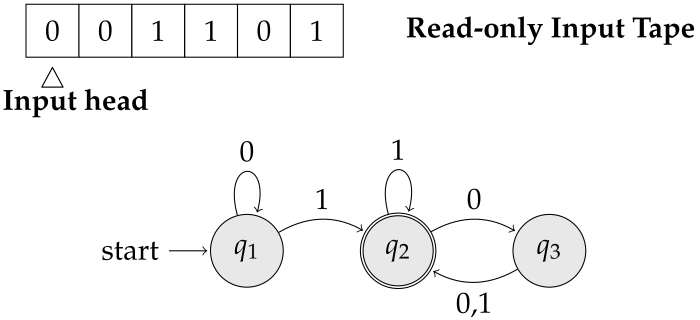


Chomsky Hierarchy

Classifying automata, grammars and languages and their descriptive power.



Finite Automata



0	0	1	1	0	1

Read-only Input Tape

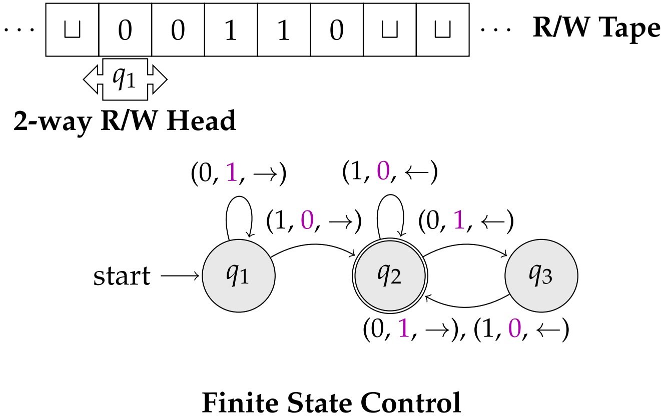
Input head

Finite State Control

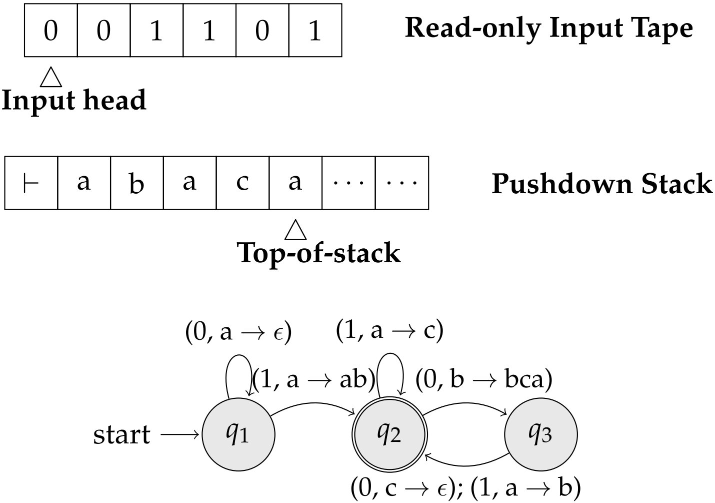
Basic Components:

- Input tape: containing symbols from an alphabet ($\{0, 1\}$).
- Finite state control: containing states and transitions.
- Input head: pointing to the current input symbol.

Turing Machine

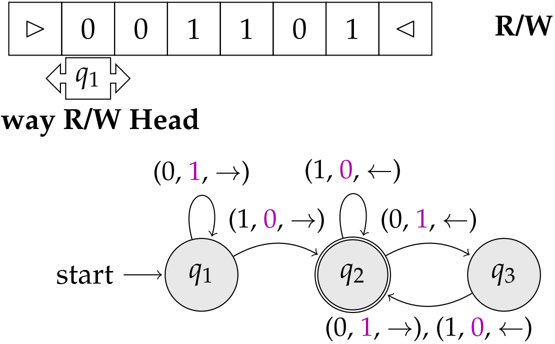


Pushdown Automata


0 0 1 1 0 1
Read-only Input Tape
Input head
⊢ a b a c a ··· ···
Pushdown Stack
Top-of-stack
$(0, \mathrm{a} \rightarrow \epsilon)$
$(1, \mathrm{a} \rightarrow \mathrm{c})$
$(1, \mathrm{a} \rightarrow \mathrm{ab})$
$(0, \mathrm{b} \rightarrow \mathrm{bca})$
start
q_1
q_2
q_3
$(0, \mathrm{c} \rightarrow \epsilon); (1, \mathrm{a} \rightarrow \mathrm{b})$

Finite State Control

Linear Bounded Automata



R/W Tape

2-way R/W Head

Finite State Control

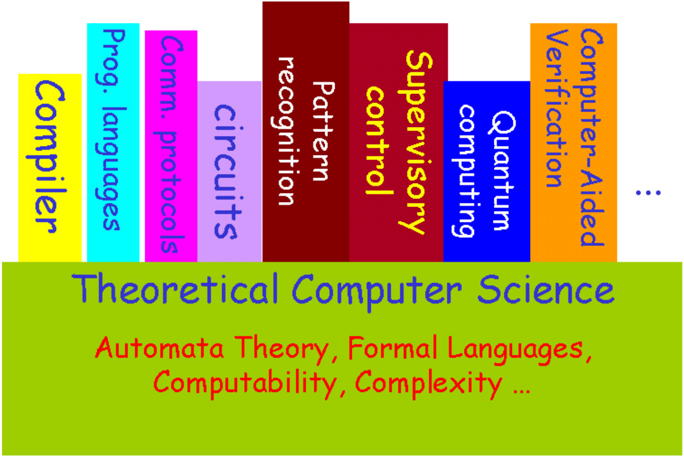
Topics

- **Automata and Formal Languages**:
  - Finite automata, pushdown automata, linear bounded automata, Turing machines, and their variants;
  - Regular, context-free, context sensitive, and unrestricted grammars;
  - Closure and decision properties of various language classes;
  - Transducers (i.e., automata with outputs), weighted automata, probabilistic automata, quantum automata, tree automata, ... etc.

- **Computability Theory**: Turing-recognizable languages, Turing-decidable languages, Halting problem, reducibility, Post correspondence problem, ... (If time permits, also Primitive recursive function, μ-recursive function, partial recursive function, total recursive function.)

- **Complexity Theory**:
  - Various resource bounded complexity classes, including NLOGSPACE, P, NP, PSPACE, EXPTIME, and many more;
  - Randomized complexity classes, including BPP, RP, ZPP, ... etc. Interactive proof system. Zero-knowledge proof.
  - Oracle and alternating computations.

Applications



Compiler

Prog. languages

Comm. protocols

circuits

Pattern recognition

Supervisory control

Quantum computing

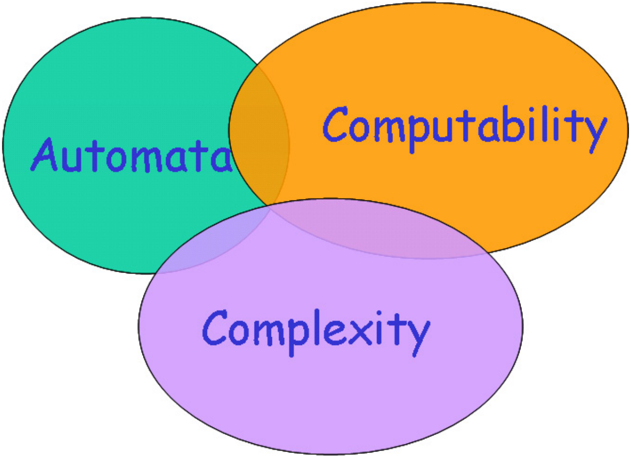
Computer-Aided Verification

...

Theoretical Computer Science

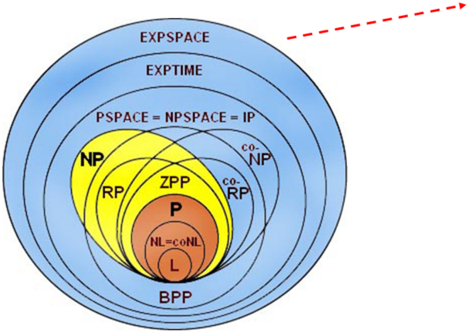
Automata Theory, Formal Languages,
Computability, Complexity ...

Our Focus



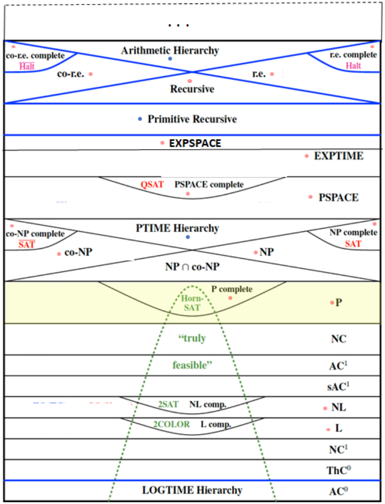
The Big Picture

Major complexity classes discussed in this class



Randomized complexity classes : RP, ZPP, BPP

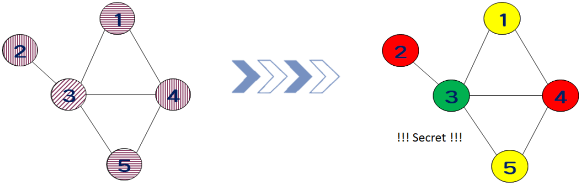
● Covered in this class; ● Supplementary



The Power of ”Randomization” (Zero-Knowledge Proof)

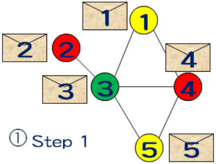
Example: **3-colorability of graphs**.
Given a graph $G(V, E)$, deciding whether nodes in V can be colored with three colors so that adjacent nodes have distinct colors.
Given a graph G, consider the following scenario:

- Bob claims that he has a ”major” discovery of a 3-coloring of G. Bob wants to convince Alice that G is indeed 3-colorable, but does not want to reveal the exact ”proof” (how nodes are colored) to Alice.

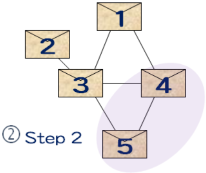


Zero-Knowledge Proof of 3-Colorability

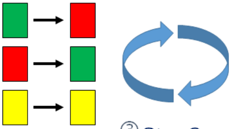
  - Repeat Steps (1) - (3) n times.
    1. Bob puts the color of each node in an envelope, and gives the set of envelopes (each associated with a node of G) to Alice.
    2. Alice opens a pair of envelopes associated with adjacent nodes. If same color, ”reject”.
    3. Bob randomly permutes the coloring. Repeat Step (1).
  - ”accept”.
- Claim:
  - 3-colorable $\Rightarrow$ prob. of acc. is ONE.
  - Not 3-colorable $\Rightarrow$ prob. of rej. is ”HIGH” for suff. large n.
- Using such scheme, Alice does not know the exact coloring.
- If one-way functions exists, $NP \subseteq ZK$.



① Step 1



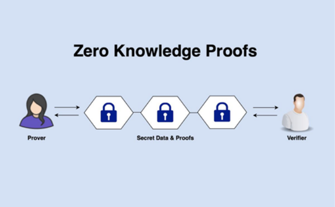
② Step 2

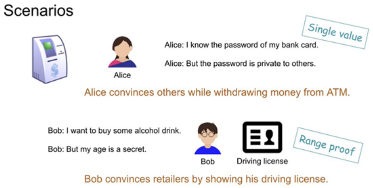


③ Step 3

Zero-Knowledge Proof

- A ZKP allows one party to prove the knowledge of certain information to the other party without revealing the data in question.
  - Completeness - a verifier will be confident that the given information is true
  - Soundness - if the provers' info is false, it cannot convince the verifier otherwise
  - Zero-knowledge - no other information will be revealed other then the given information





Alice: I know the password of my bank card.

Alice: But the password is private to others.

Alice convinces others while withdrawing money from ATM.

Bob: I want to buy some alcohol drink.

Bob: But my age is a secret.

Bob convinces retailers by showing his driving license.

(Fig. from https://en.rattibha.com/thread/1511742345053900800)

- How about a student convincing the instructor that he/she deserves a full credit without taking an exam?