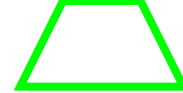


# Introduction to Algorithm Theory

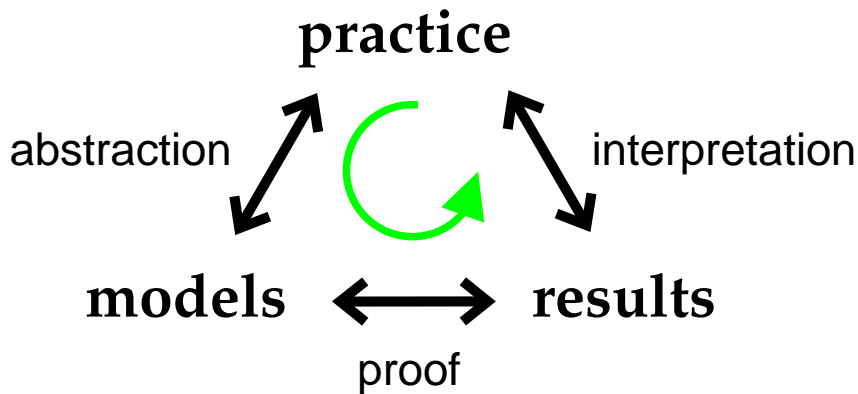
## Overview

- Our approach - modeling
- The subject matter - what is this all about
- Historical introduction
- How to model problems
- How to model solutions

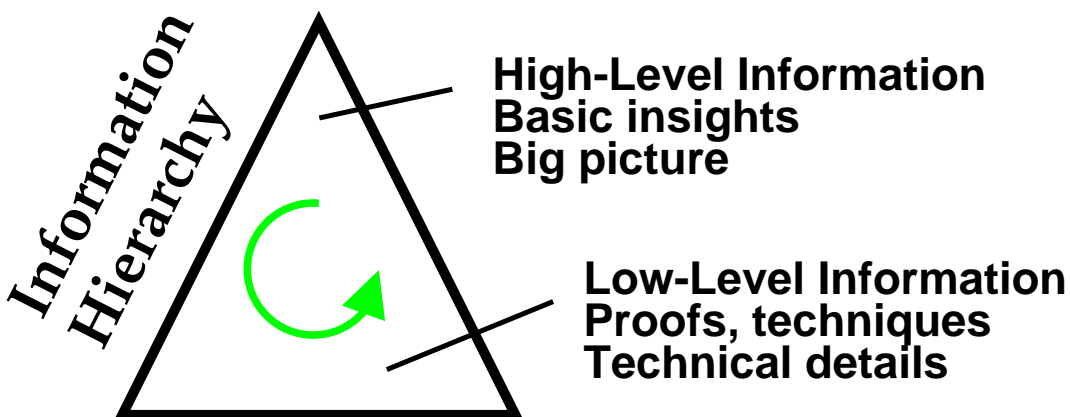


# Our approach

## Modeling



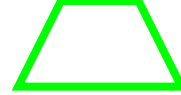
## Perspective



Lectures → Mainly high-level understanding

Group sessions → Practice skills: proofs, problems

Studying strategy: Don't memorize penum – try to understand the whole!



# Subject matter

How to **solve** information-processing **problems efficiently**.



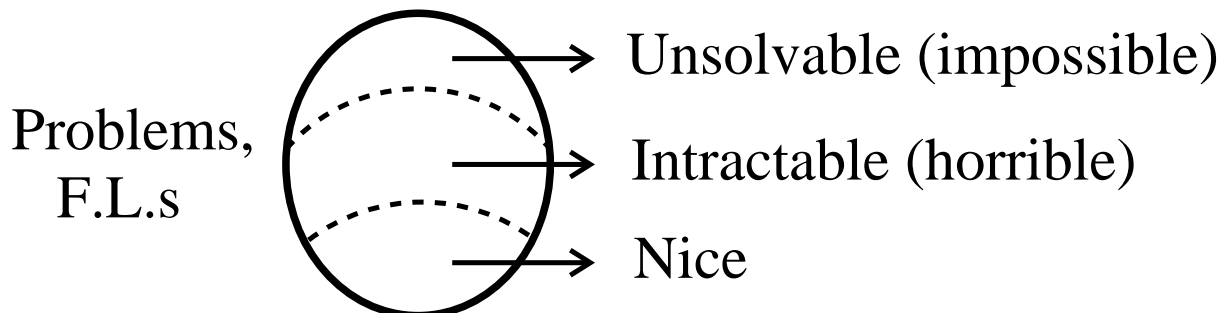
abstraction  
formalisation  
modeling

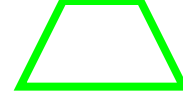
Problems  $\rightsquigarrow$  interesting,  $\rightsquigarrow$  formal  
natural languages  
problems (F.L.s)

(Ex. MATCHING, SORTING, T.S.P.)

Solutions  $\rightsquigarrow$  algorithms  $\rightsquigarrow$  Turing  
machines

Efficiency  $\rightsquigarrow$  complexity  $\rightsquigarrow$  complexity  
classes



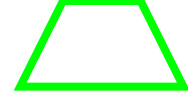


## Historical introduction

In mathematics (cooking, engineering, life)  
solution = algorithm

Examples:

- $\sqrt{253} =$
- $ax^2 + bx + c = 0$
- Euclid's g.c.d. algorithm — the earliest non-trivial algorithm?



# Applications of Euclid's algorithm

From Wikipedia:

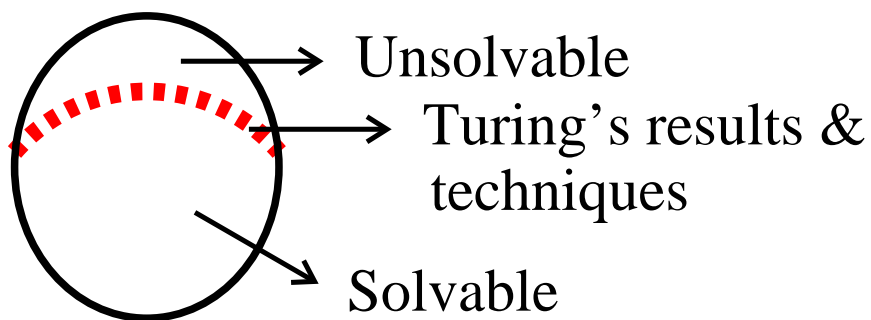
“The Euclidean algorithm has many theoretical and practical applications. It may be used to generate almost all the most important traditional musical rhythms used in different cultures throughout the world. It is a key element of the RSA algorithm, a public-key encryption method widely used in electronic commerce. It is used to solve Diophantine equations, such as finding numbers that satisfy multiple congruences (Chinese remainder theorem) or multiplicative inverses of a finite field. The Euclidean algorithm can also be used in constructing continued fraction, in the Sturm chain method for finding real roots of polynomials, and in several modern integer factorization algorithms. Finally, it is a basic tool for proving theorems in modern number theory, such as Lagrange's four-square theorem, and the fundamental theorem of arithmetic (unique factorization).”



# Origins of undecidability theory

$\exists$  algorithm?  $\rightarrow$  metamathematics

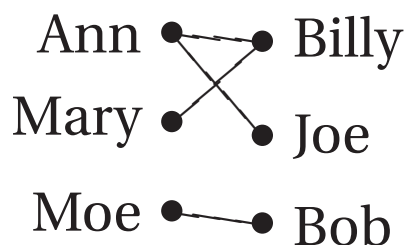
- K. Gödel (1931): nonexistent theories
- A. Turing (1936): nonexistent algorithms (article: “On computable Numbers ...”)





# Origins of complexity theory

- Von Neumann (ca. 1948): first computer
- Edmonds (ca. 1965): an algorithm for  
MAXIMUM MATCHING



Edmonds' article rejected based on existence of trivial algorithm: Try all possibilities!

## Rough complexity analysis of trivial algorithm

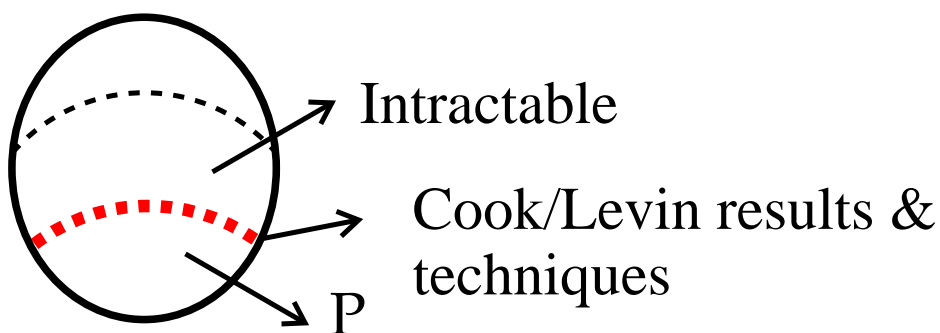
- $n = 100$  boys
- $n! = 100 \times 99 \times \dots \times 1 \geq 10^{90}$  possibilities
- assume  $\leq 10^{12}$  possibilities tested per second
- $\leq 10^{12+4+2+3+2} \leq 10^{23}$  tested per century
- running time of trivial algorithm for  $n = 100$  is  $\geq 10^{90-23} = 10^{67}$  centuries!

Compare: “only” ca.  $10^{13}$  years since Big Bang!

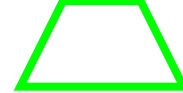


Edmonds: My algorithm is a **polynomial-time** algorithm, the trivial algorithm is **exponential-time**!

- $\exists$  polynomial-time algorithm for a given problem?
- Cook / Levin (1972):  **$\mathcal{NP}$ -completeness**








# Problems, formal languages

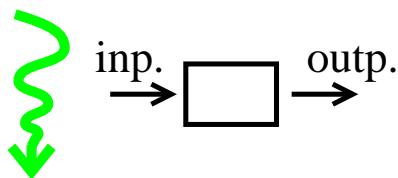
All the world's  
information-processing  
problems

*Ex. compute salaries,  
control Lunar  
module landing*

 graphs,  
numbers ...


“Interesting”,  
“natural”  
problems

MATCHING  
TSP  
SORTING



Functions

*(sets of I/O pairs)*

 output=  
YES/NO

Formal languages

*(sets of 'YES-strings')*

Problem = set of strings (over an alphabet).  
Each string is (the encoding of) a  
YES-instance.



**Def. 1** *Alphabet* = finite set of symbols

Ex.  $\Sigma = \{0, 1\}$ ;  $\Sigma = \{A, \dots, Z\}$

Coding: binary  $\leftrightarrow$  ASCII

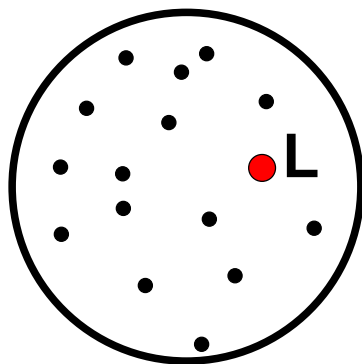
**Def. 2**  $\Sigma^*$  = all finite strings over  $\Sigma$

$\Sigma^* = \{\epsilon, 0, 1, 00, 01, \dots\}$  — in **lexicographic order**

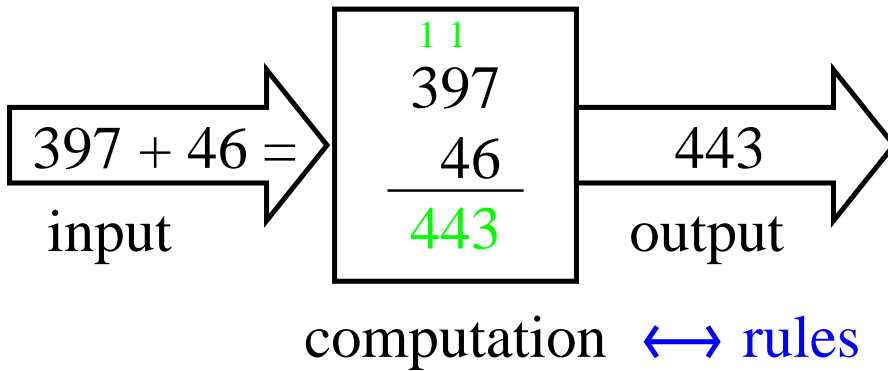
**Def. 3** A *formal language*  $L$  over  $\Sigma$  is a subset of  $\Sigma^*$

$L$  is the set of all “YES-instances”.

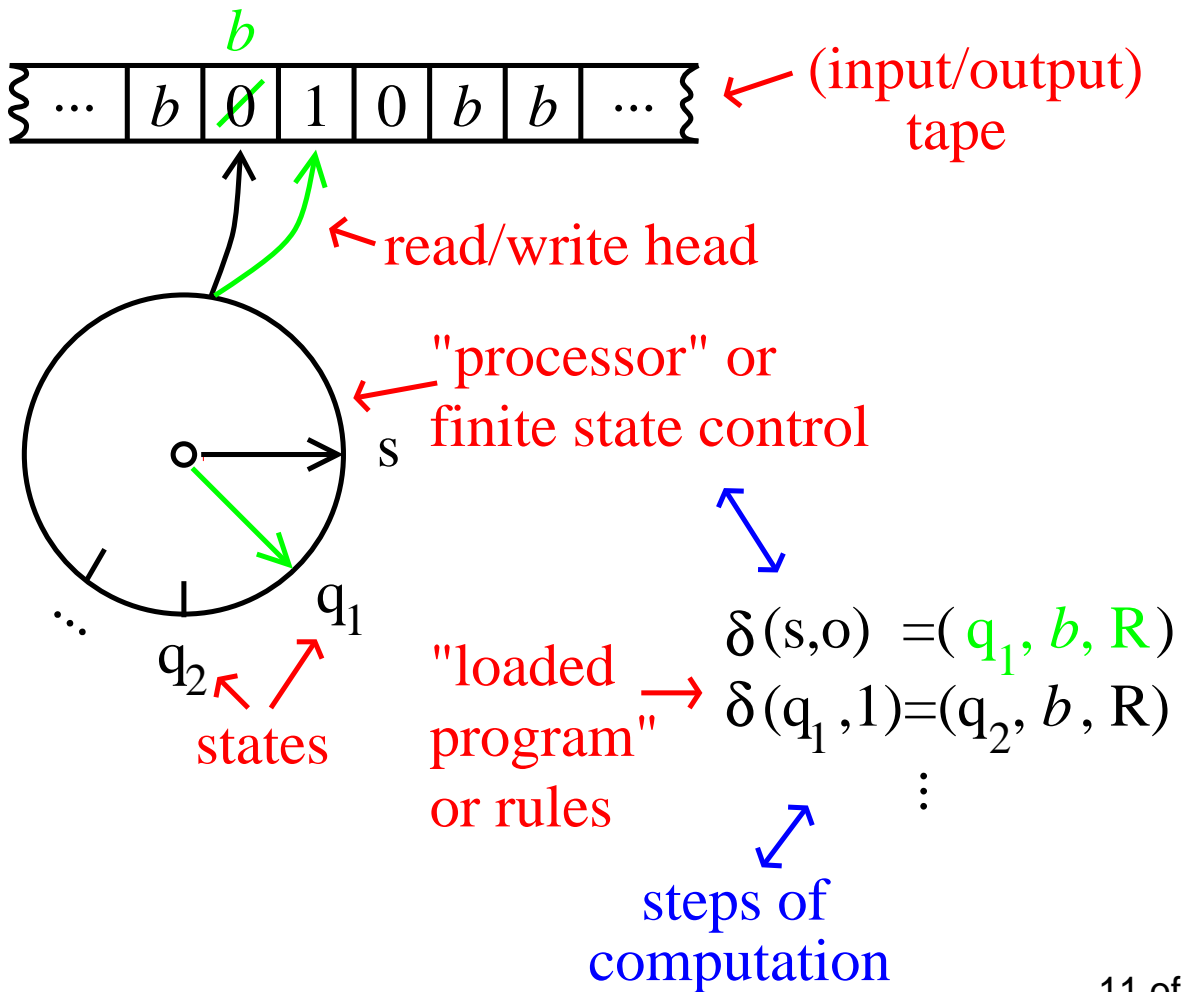
Set of all problems



# Algorithm



# Turing machine – intuitive description





We say that Turing machine  $M$  **decides language L** if (and only if)  $M$  computes the function

$$f : \Sigma^* \rightarrow \{Y, N\} \text{ and for each } x \in L : f(x) = Y \\ \text{for each } x \notin L : f(x) = N$$

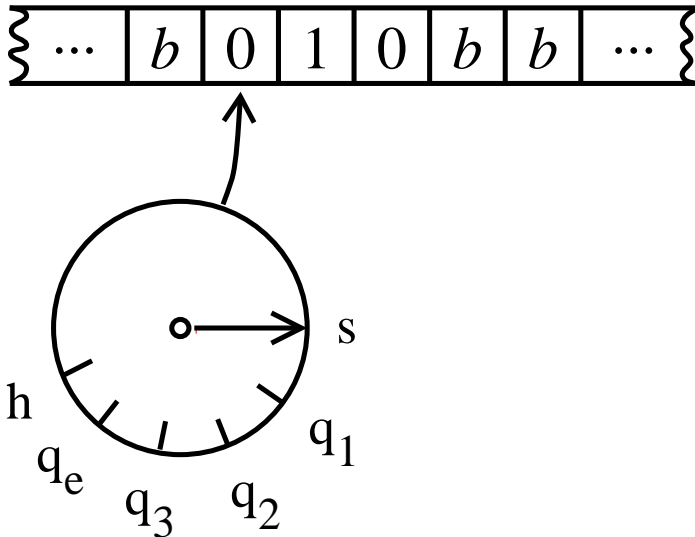
Language  $L$  is **(Turing) decidable** if (and only if) there is a Turing machine which decides it.

We say that Turing machine  $M$  **accepts language L** if  $M$  halts if and only if its input is an string in  $L$ .

Language  $L$  is **(Turing) acceptable** if (and only if) there is a Turing machine which accepts it.

## Example

A Turing machine  $M$  which decides  
 $L = \{010\}$ .



$$M = (\Sigma, \Gamma, Q, \delta) \quad \Sigma = \{0, 1\}$$

$$\Gamma = \{0, 1, b, Y, N\} \quad Q = \{s, h, q_1, q_2, q_3, q_e\}$$

$\delta$  :

	0	1	$b$
$s$	$(q_1, b, R)$	$(q_e, b, R)$	$(h, N, -)$
$q_1$	$(q_e, b, R)$	$(q_2, b, R)$	$(h, N, -)$
$q_2$	$(q_3, b, R)$	$(q_e, b, R)$	$(h, N, -)$
$q_3$	$(q_e, b, R)$	$(q_e, b, R)$	$(h, Y, -)$
$q_e$	$(q_e, b, R)$	$(q_e, b, R)$	$(h, N, -)$

('-' means "don't move the read/write head")



## Church's thesis

### 'Turing machine' $\cong$ 'algorithm'

Turing machines can compute every function that can be computed by some algorithm or program or computer.

### 'Expressive power' of PL's

**Turing complete** programming languages.

### 'Universality' of computer models

Neural networks are Turing complete (McCulloch-Pitts neuron).

### Uncomputability

If a Turing machine cannot compute  $f$ , no computer can!