Undecidability and Complexity in Four Lectures

Overview

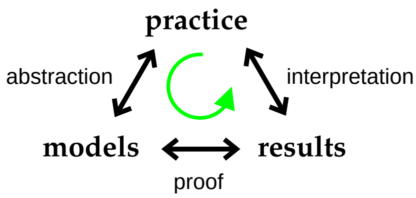
- Lecture 1: Introduction. Uncomputability.
- Lecture 2: Intractability.
- Lecture 3: Proving Intractability.
- Lecture 4: Coping With Intractability.

Lecture 1 overview

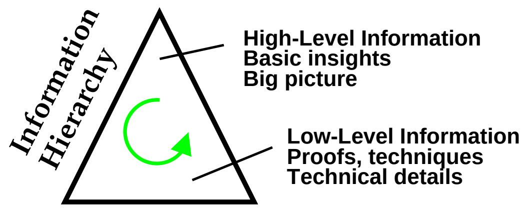
- Our approach - modeling
- The subject matter - what is this all about
- Historical introduction
- How to model problems
- How to model solutions
- How to prove that some problems have no solutions

Our approach

Modeling



Perspective



Lectures	→	Mainly high-level understanding
Group sessions	→	Practice skills: proofs, problems

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

Subject matter

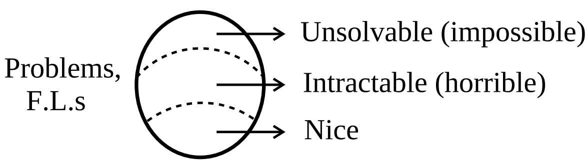
How to **solve** information-processing **problems efficiently**.

abstraction
formalisation
modeling

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

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

Solutions	$\leadsto$	algorithms	$\leadsto$	Turing machines
Efficiency	$\leadsto$	complexity	$\leadsto$	complexity classes



Historical introduction

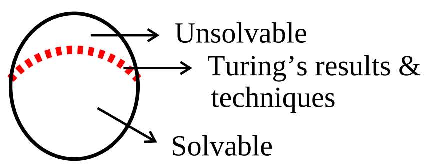
In mathematics (cooking, engineering, life)
solution = algorithm

Examples:

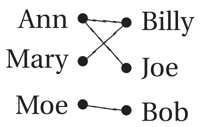
- $\sqrt{253} =$
- $ax^2 + bx + c = 0$
- Euclid's g.c.d. algorithm — the earliest non-trivial algorithm?

$\exists$ algorithm? → metamathematics

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



- 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!

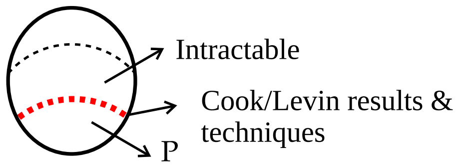
Complexity analysis of trivial algorithm (using approximation)

- $n = 100$ boys
- $n! = 100 \times 99 \times \cdots \times 1 \geq 10^{90}$ possibilities
- assume $\leq 10^{12}$ possibilites 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!

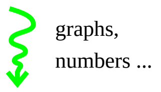
Edmonds: My algorithm is a **polynomial-time** algorithm, the trivial algorithm is **exponential-time**!

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

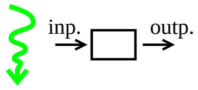


Problems, formal languages

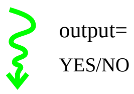
All the world's information-processing problems — *Ex. compute salaries, control Lunar module landing*



"Interesting", "natural" problems — MATCHING, TSP, SORTING



Functions — *(sets of I/O pairs)*



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. $\sum = \{0, 1\}$; $\Sigma = \{A, \ldots, Z\}$

Coding: binary $\leftrightarrow$ ASCII

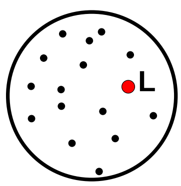
Def. 2 $\sum^*$ = *all finite strings over* $\sum$

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

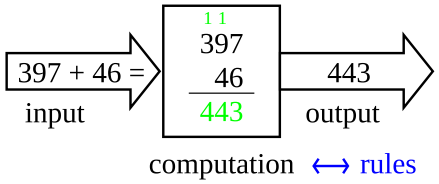
Def. 3 *A* ***formal language*** *L over $\sum$ is a subset of $\sum^*$*

L is the set of all "YES-instances".

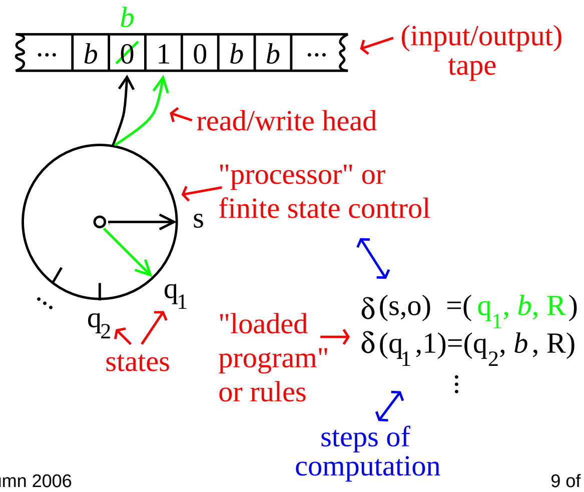
Set of all problems



Algorithm



Turing machine – intuitive description



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

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

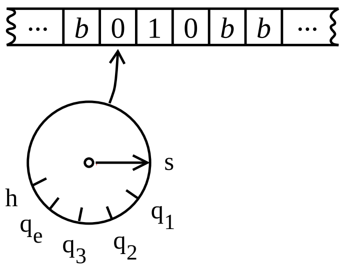
Language L is **(Turing) decidable** if (and only if) there is a Turing machine which decides it.

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 $L = \{010\}$.



$M = (\Sigma, \Gamma, Q, \delta)$ $\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, b, Y, N\}$ $Q = \{s, h, q_1, q_2, q_3, q_e\}$

δ :

	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' ≅ '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 (Mc Cullok, Pitts).

Uncomputability

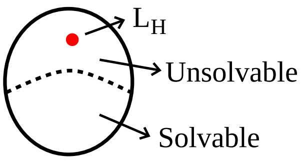
If a Turing machine cannot compute f, no computer can!

Uncomputability

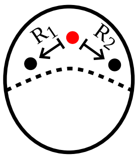
What algorithms can and cannot do.

Strategy

1. Show that HALTING (the Halting problem) is unsolvable



2. Use **reductions** $\overset{R}{\longmapsto}$ to show that other problems are unsolvable



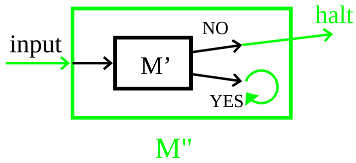
Step 1: HALTING is unsolvable

Def. 4 (HALTING)

$$L_H = \{(M, x) | M \text{ halts on input } x\}$$

Theorem 1 *The Halting Problem is undecidable.*

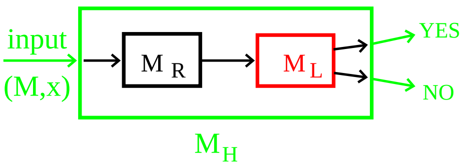
Proof (by **diagonalization**): Given a Turing machine M' that decides L_H we can construct a Turing machine M'' as follows:



QUESTION: What does M'' do when given M'', M'' as input?

CONCLUSION: Since the assumption that M' exists leads to a contradiction (i.e. an impossible machine), it must be false.

Reductions



Meaning of a reduction

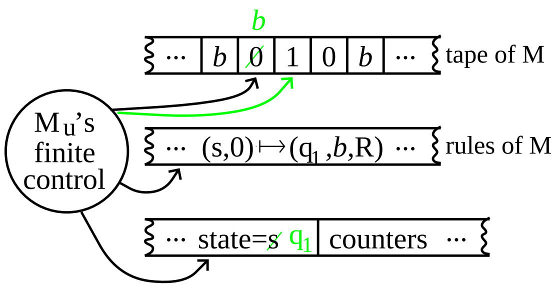
Image: You meet an old friend with a brand new M_L-machine under his shoulder. Without even looking at the machine you say: "It is fake!"

How the reduction works

Image (an old riddle): You are standing at a crossroad deep in the forest. One way leads to the hungry crocodiles, the other way to the castle with the huge piles of gold. In front of you stands one of the two twin brothers. One of them always lies, the other always tells the truth. You can ask one question. What do you say?

The universal Turing machine M_u

- M_u works like an ordinary computer: It takes a code (program) M and a string x as input and simulates (runs) M on input x.
- M_u exists by Church's thesis.
- To **prove** existence of M_u we must construct it. Here is a 3-tape M_u:

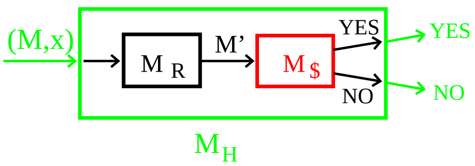


A typical reduction

$$L_\$ = \{M | M \text{(eventually) writes a } \$ \text{ when started with a blank tape}\}$$

Claim: $L_\$$ is undecidable

Proof:



M':

```
Simulate M on input x;
IF M halts THEN write a $;
```

Important points:

- M' must not write a \$ during the simulation of M!
- 'Write a \$' is an arbitrarly chosen action!

M_R:
Output the M_u code modified as follows: Instead of reading its input M and x, the modified M_u has them stored in its finite control and it **writes them** on its tape. After that the modified M_u proceeds as the ordinary M_u untill the simulation is finished. Then it writes a \$.

Reduction as mathematical function

Given a reduction from L_1 to L_2. Then M_R computes a function

$$f_R : \sum^* \to \sum^*$$

which is such that

$$x \in L_1 \Rightarrow f_R(x) \in L_2$$
$$x \notin L_1 \Rightarrow f_R(x) \notin L_2$$