
Uke 14,
NP og NP-komplett





Complexity of algorithms



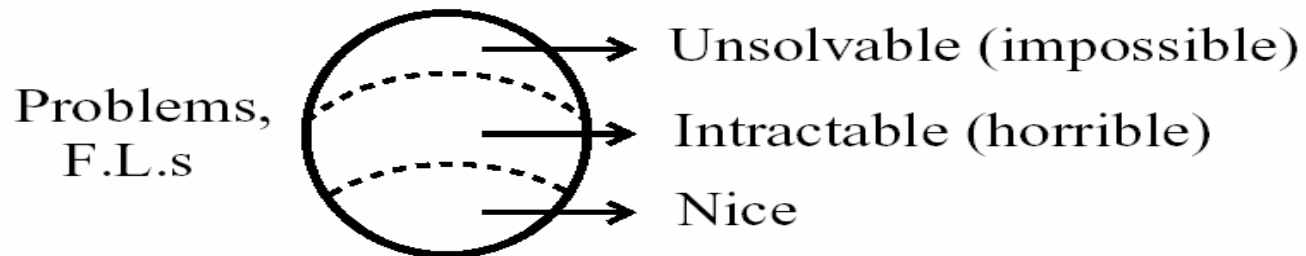
TEMA: NP og NP-kompletthet

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

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

Solutions \rightsquigarrow algorithms \rightsquigarrow Turing machines

Efficiency \rightsquigarrow complexity \rightsquigarrow complexity classes



Note: This is from in210, first 2 lectures

Historical introduction

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

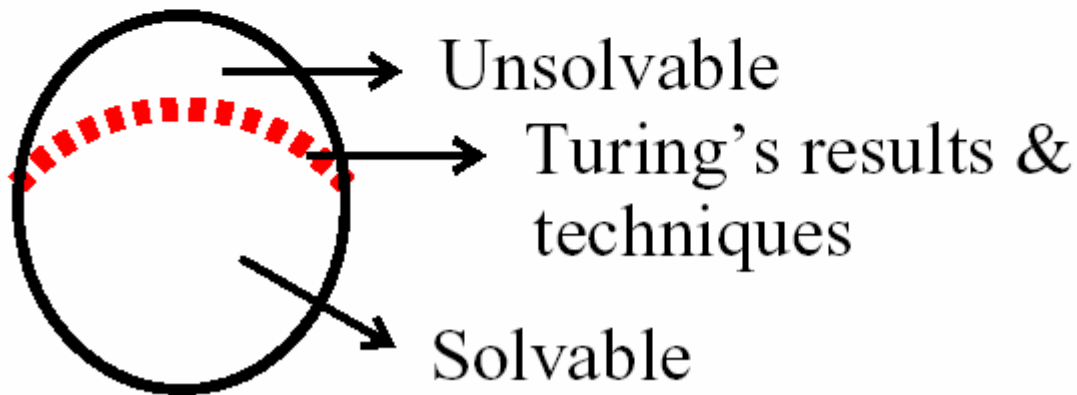
Examples:

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

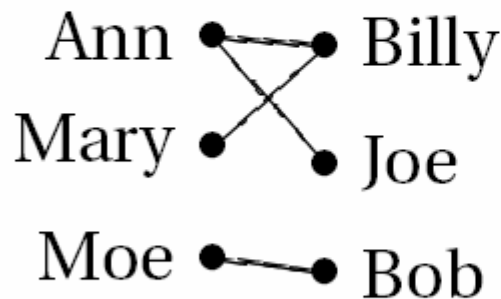


\exists algorithm? \rightarrow metamathematics

- 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)

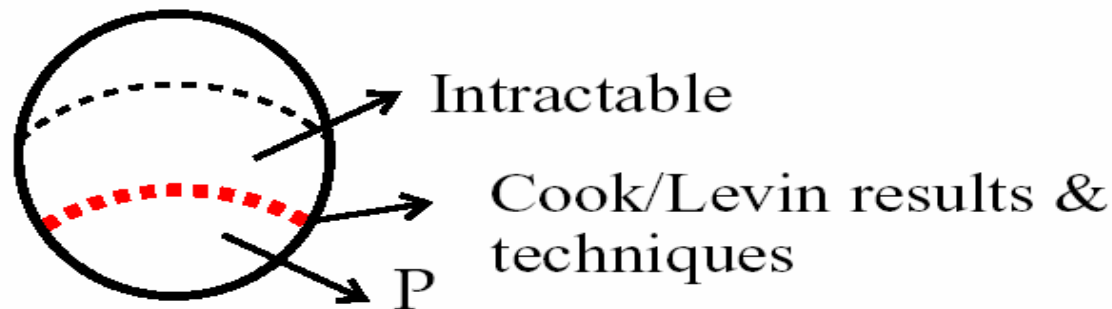
- $n = 100$ boys
- $n! = 100 \times 99 \times \dots \times 1 \geq 10^{90}$ possibilities
- assume $\leq 10^{12}$ possibilities tested per second
- $\leq 10^{12+4+2+3+2} \leq 10^{23}$ tested per century
- running time of trivial algorithm for $n = 100$ is $\geq 10^{90-23} = 10^{67}$ centuries!

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



Edmonds: Mine 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**



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

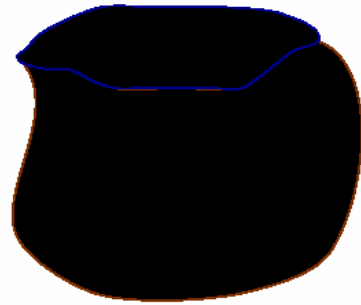
↪ : abstraction, formalisation

Problems ↪ I/O pairs, functions, “interesting problems” ↪ formal languages

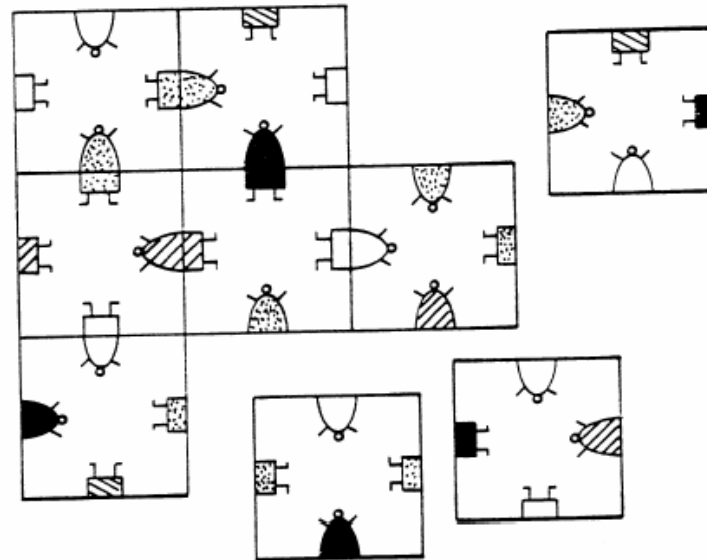
solutions ↪ algorithms ↪ Turing machines

efficiency ↪ resources, upper/lower bounds ↪ complexity classes



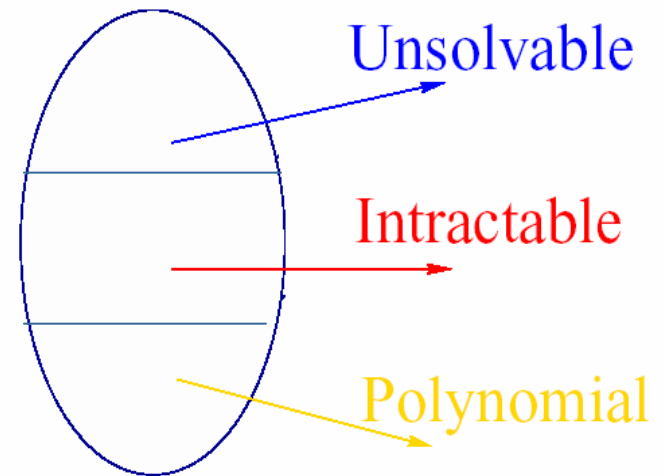
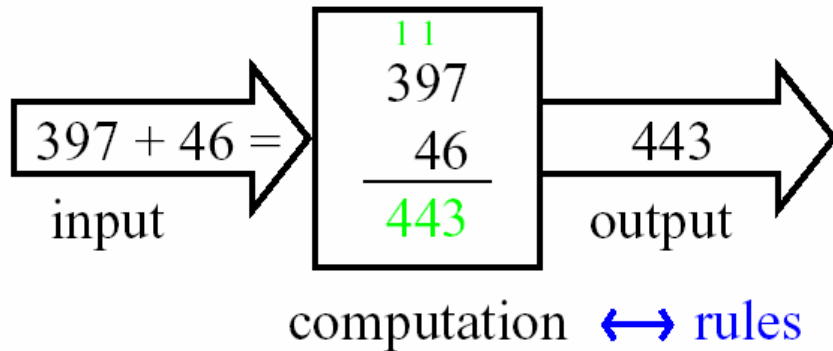


- All algorithms in the world live in the basket
- Infinitely many of them - most of them are unknown to us
- Meaning of unsolvability: no algorithm in the basket solves the problem
- Meaning of solvability: there is an algorithm in the basket that solves the problem (but we do not necessarily know what the algorithm looks like)

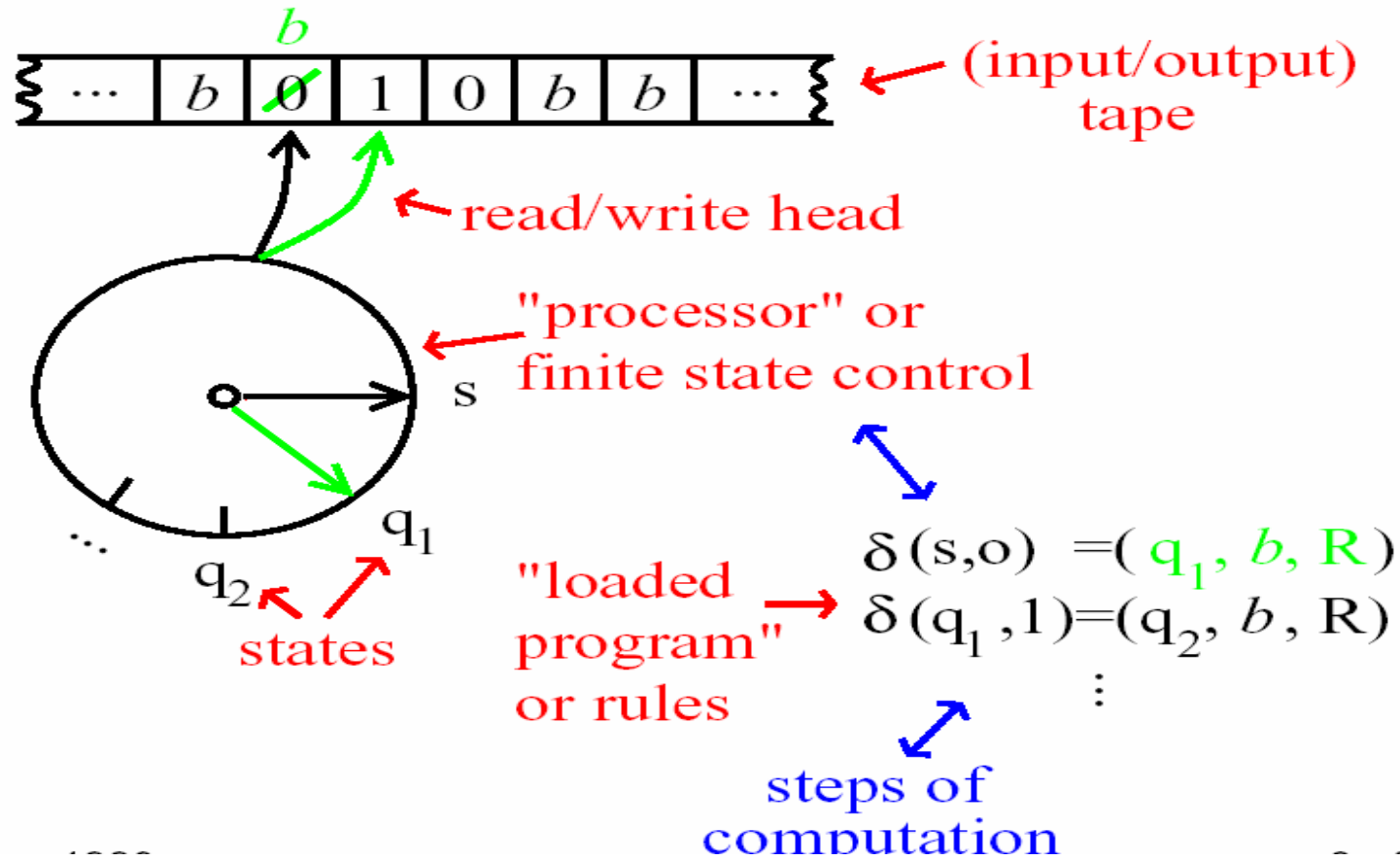


Monkey puzzle is an example of a problem that does not have a reasonable solution (or polynomial time). Such problems are called **intractable**

Algorithm



Turing machine – intuitive description



Turing machine – formal description

A **Turing machine (TM)** is $M = (\Sigma, \Gamma, Q, \delta)$
where

Σ , the **input alphabet** is a finite set of input symbols

Γ , the **tape alphabet** is a finite set of tape symbols which includes Σ , a special **blank symbol** $b \in \Gamma \setminus \Sigma$, and possibly other symbols

Q is a finite set of **states** which includes a **start state** s and a **halt state** h

δ , the **transition function** is

$$\delta : (Q \setminus \{h\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$



Computation – formal definition

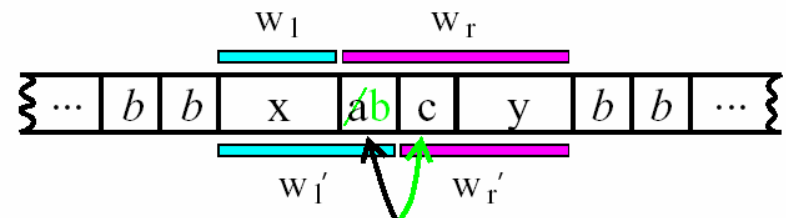
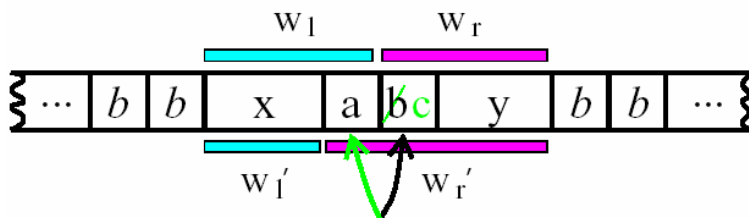
A **configuration** of a Turing machine M is a triple $C = (q, w_l, w_r)$ where $q \in Q$ is a state and w_l and w_r are strings over the tape alphabet.

We say that a configuration (q, w_l, w_r) **yields in one step** configuration (q', w_l', w_r') and write $(q, w_l, w_r) \xrightarrow{M} (q', w_l', w_r')$ if (and only if) for some $a, b, c \in \Gamma$ and $x, y \in \Gamma^*$ either

$$\begin{array}{lll} w_l = xa & w_r = by & \text{and} \\ w_l' = x & w_r' = acy & \delta(q, b) = (q', c, L) \end{array}$$

or

$$\begin{array}{lll} w_l = x & w_r = acy & \text{and} \\ w_l' = xb & w_r' = cy & \delta(q, a) = (q', b, R) \end{array}$$



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

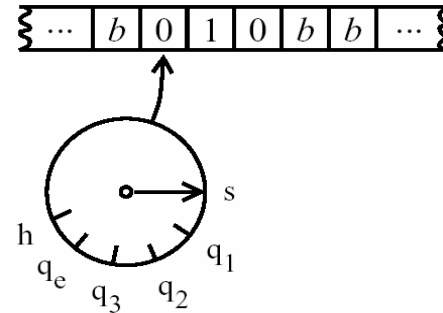
Neural networks are Turing complete (Mc Cullok, Pitts).

Uncomputability

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

Example

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



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

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

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")

NP vs P

NP stands for nondeterministic polynomial time.

A deterministic machine, given an instruction, executes it and goes to the next instruction, which is unique.

A nondeterministic machine, after each instruction, has a choice of the next instruction and it always, magically, makes the right choice.

Nondeterministic machine seems like a funny concept and too powerfull. It is not so. For example, undecided problems remain undecided. A problem is in NP if, in polynomial time, we can prove that any “yes” instance of the problem (a certificate) is correct. NP includes all problems that have polynomial time solutions.

Is $P = NP$???



Class NPC

Among all the problems known to be in NP, there is a subset known as **NP-complete** problems, which contains the hardest problems in NP (intractable, with polynomial certificates). These have also one more property that is extremely interesting: they all have a common fate: i.e. there exist **a polynomial time reduction** from any one problem in NPC to any other problem in NPC. Reduction can be quite simple, or it can actually involve several intermediate reductions.

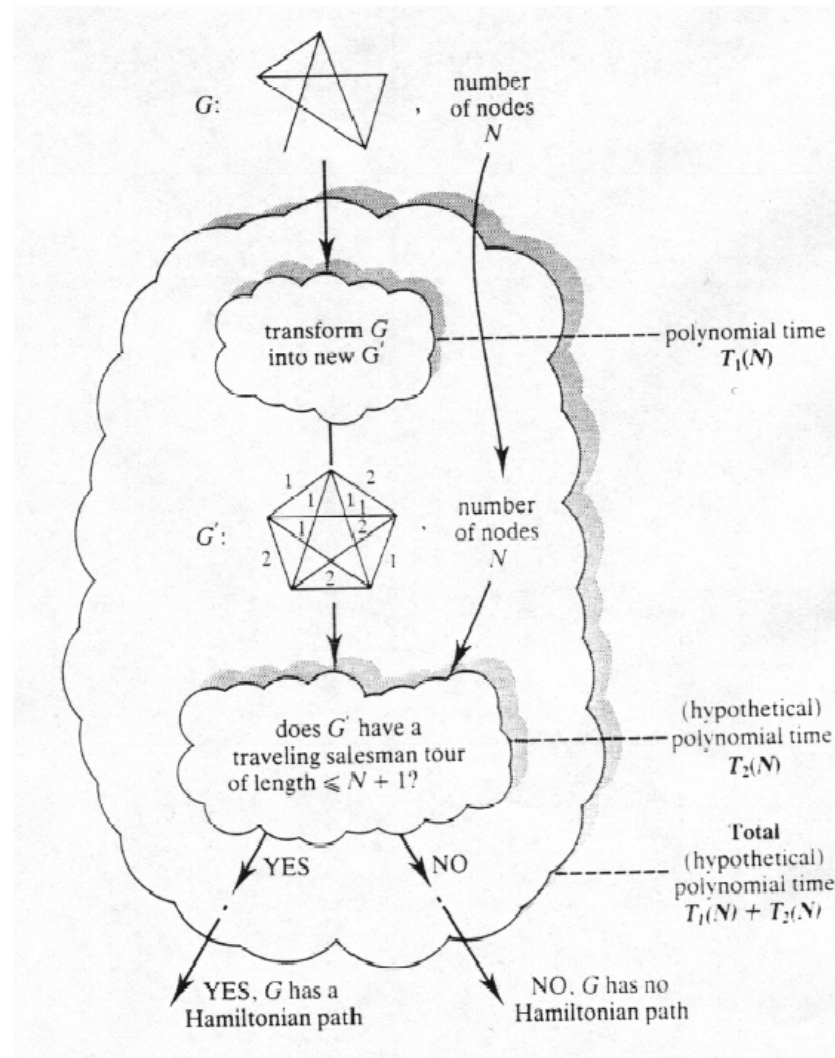


Reducing Hamiltonian paths to traveling salesman

Hamiltonian path is a simple path containing all the vertices of the graph G .

Traveling salesman problem is a problem of finding a simple cycle in the weighted graph G of minimum weight.

TEMA: NP og NP-kompletthet

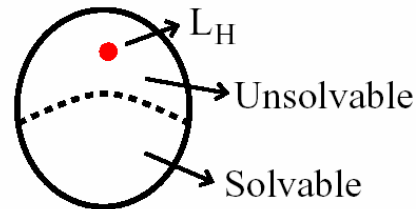


Uncomputability

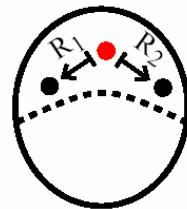
What algorithmic can and cannot do.

Strategy

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



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



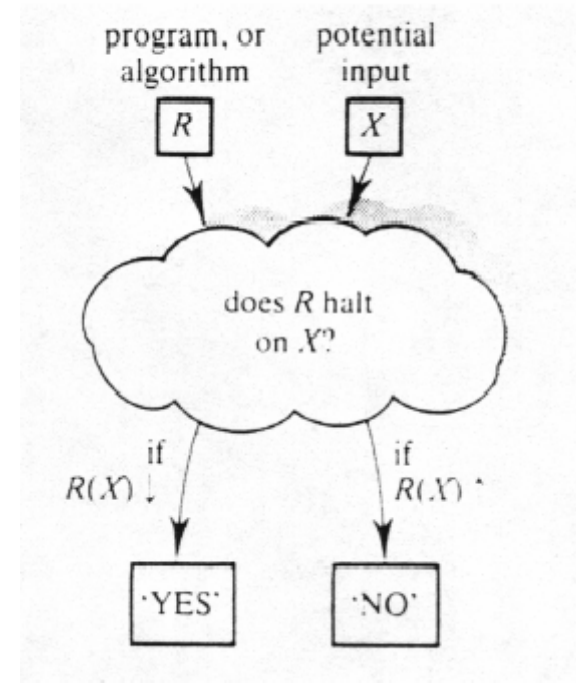
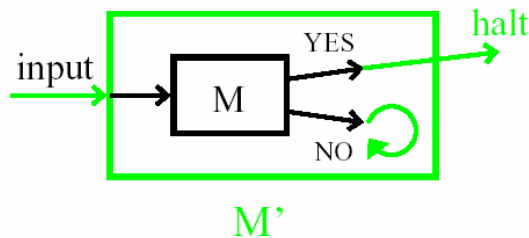
Step 1: HALTING is unsolvable

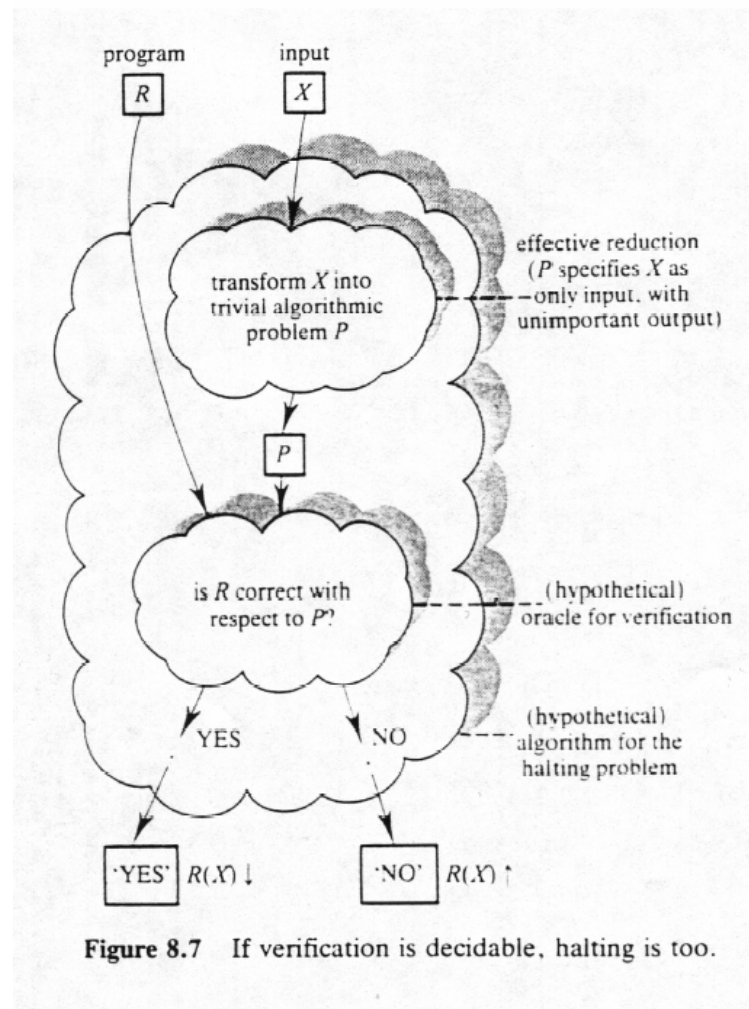
Def. 1 (HALTING)

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

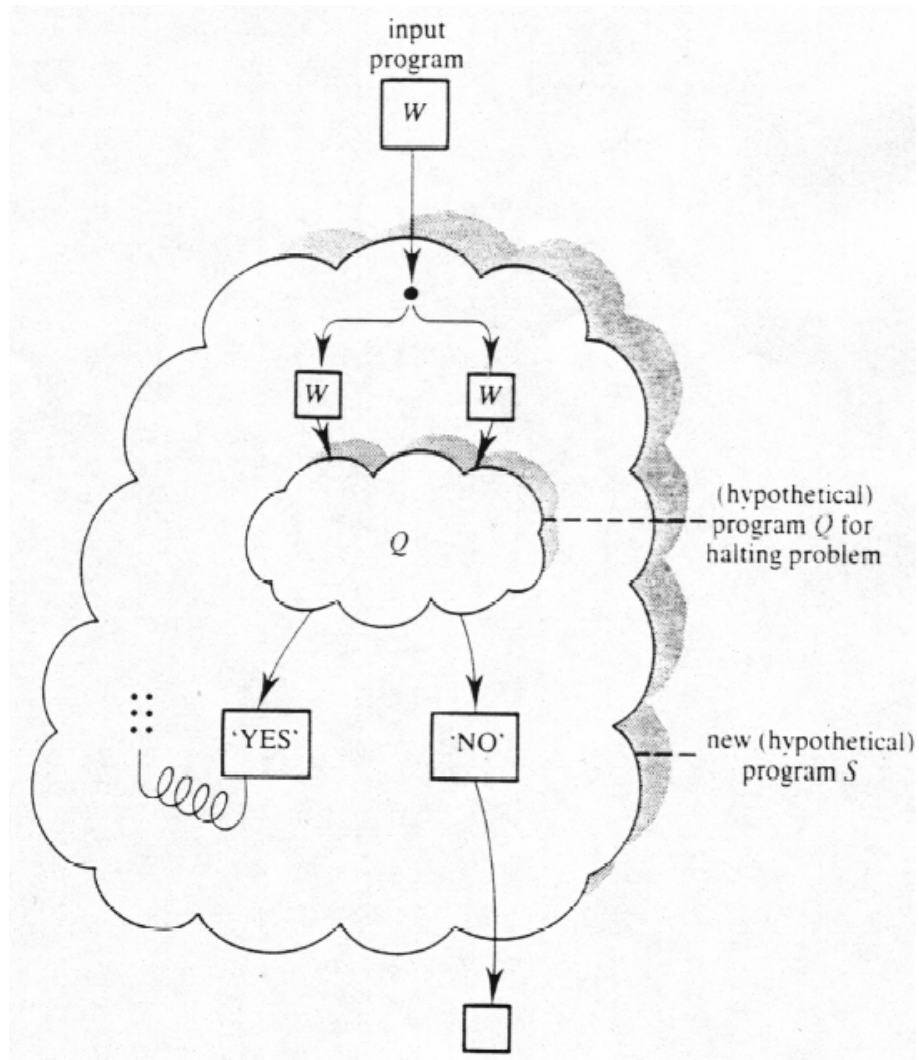
Lemma 1 Every Turing decidable language is Turing acceptable.

Proof (by **reduction**): Given a Turing machine M that decides L we can construct a Turing machine M' that accepts L as follows:

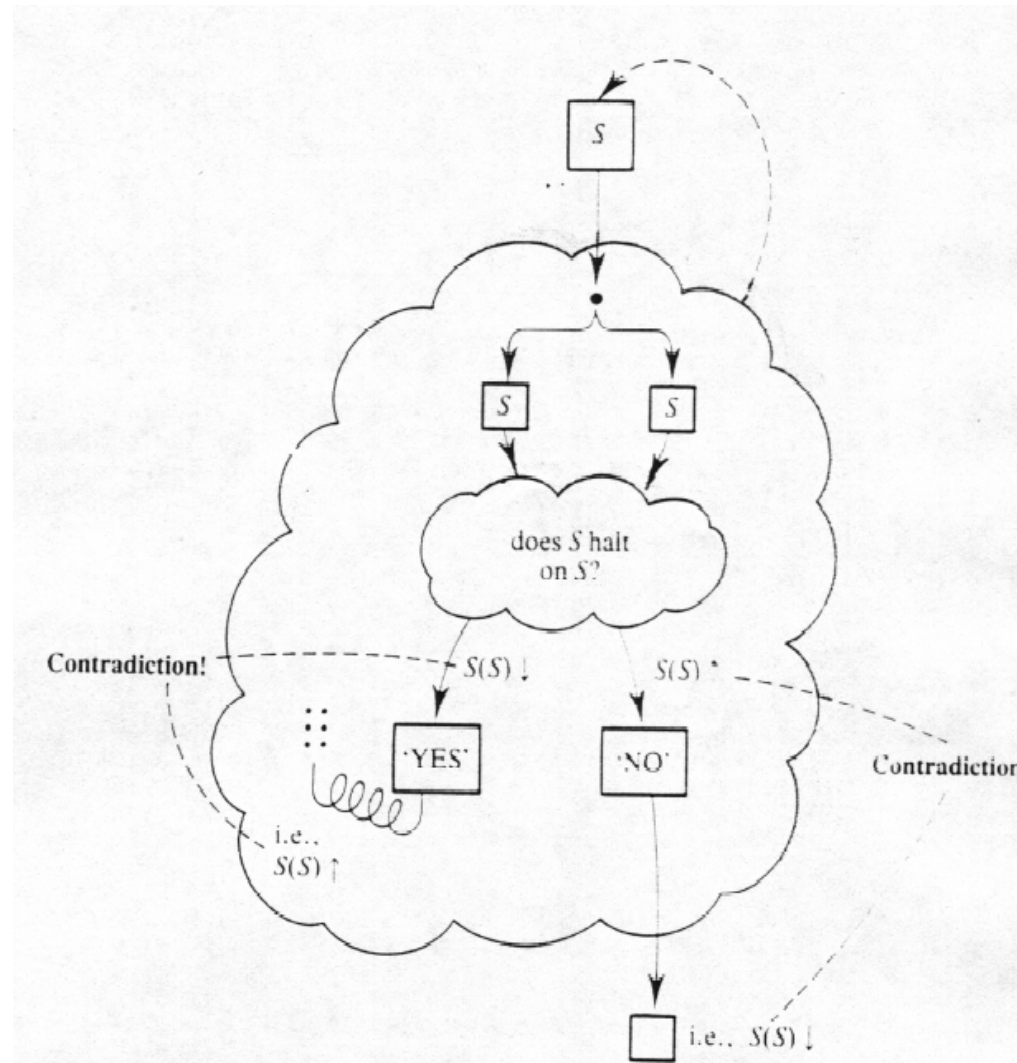




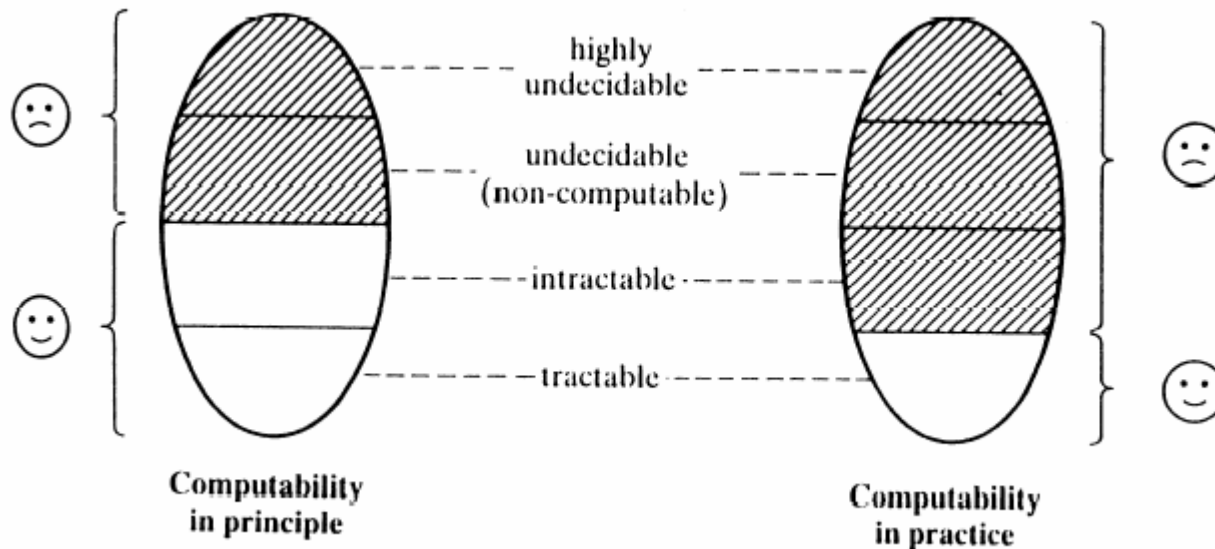
TEMA: NP og NP-kompletthet



TEMA: NP og NP-kompletthet



TEMA: NP og NP-kompletthet



TEMA: NP og NP-kompletthet

