Presentazione

Gli argomenti trattati

- Ricordiamo alcune nozioni di base
- Introduzione alla calcolabità
  - macchine di Turing
  - decidibilità
  - linguaggi
  - esempi di problemi decidibili
- Argomenti avanzati
  - problemi indecidibili
  - riduzione
  - teorema di Rice
  - minimun description length
  - indecidibilità della logica del primo ordine

Gli argomenti trattati

- Introduzione alla complessità
  - definizione di P, NP, coP, coNP
  - esempi di problemi in queste classi
  - riduzione e NP-completezza
- Argomenti avanzati
  - teorema di Cook-Levin
  - relazioni fra P, NP, NP-completi e i loro complementi
  - isomorfismo, linguaggi sparsi
  - complessità spaziale
  - algoritmi probabilistici
  - quantum computing

Obbiettivo

L'obbiettivo del corso è

- rivedere in maniera formale i concetti su calcolabilità e complessità già visti durante il corso di laurea
- introdurre alcuni argomenti avanzati

Si tratta di argomenti matematici (definizioni e teoremi)...

per renderli meno noiosi e per ricordarli meglio

- cercheremo di dimostrare insieme alcuni risultati
- vedremo numerosi esempi

Esame

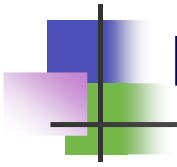
L'esame consiste in

- scegliere un argomento o un problema connesso con gli argomenti del corso
- approfondirlo cercando e studiando la letteratura come se si dovesse scrivere una review
- scrivere una piccolo documento (5-10 pagine) che riassume quanto appresso
  - lo stile dovrebbe essere quello di una review che non si limiti ad elencare solo i riferimenti ma cerchi di riassumerli e spiegarli (ad esempio, come se stesse preparando delle dispense per gli studenti del dottorato ….)

Esame

Esempi di argomenti per l'esame

- minimum description length e machine learning
- quantum computing
- algoritmi di fattorizzazione intera
- algoritmi per il graph matching
- …



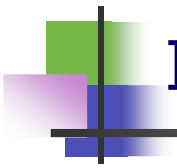
Libri

Libro principale

- M. Sipser, "Introduction to theory of computation", 2° edition, Thompson

Altri riferimenti

- C. Papadimitriou, "Computational complexity", Addison Wesley
- C. Toffalori, F. Corradini, S. Leonesi, S. Mancini, "Teoria della computabilità e complessità", McGraw-Hill



Introductory notions

Calculability

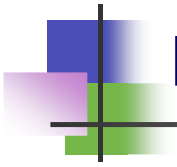
Computational calculabity theory

- the investigation of the problems that can be solved using reasonable computers
- the reasonable computers are those … that can be constructed (now or in the future)

Complexity

Computational complexity theory

- the investigation of the time, memory, or other resources required for solving computational problems
- the resources are usually measured as a function of of the input dimension

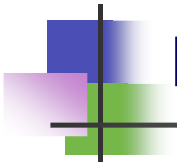


Big O-Notation

It will be extremely convenient to use the following 'order-notation' to express our complexities.

Definition: Let f and g be two functions $N \to R^+$.
We say and write $f(n) = O(g(n))$ if and only if there are two positive constant c and n_0 such that
$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

"g(n) is an (asymptotic) upper bound on f(n)".



Little o-Notation

Definition: Let f and g be two functions $N \to R^+$.
We say and write $f(n) = o(g(n))$ if and only

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

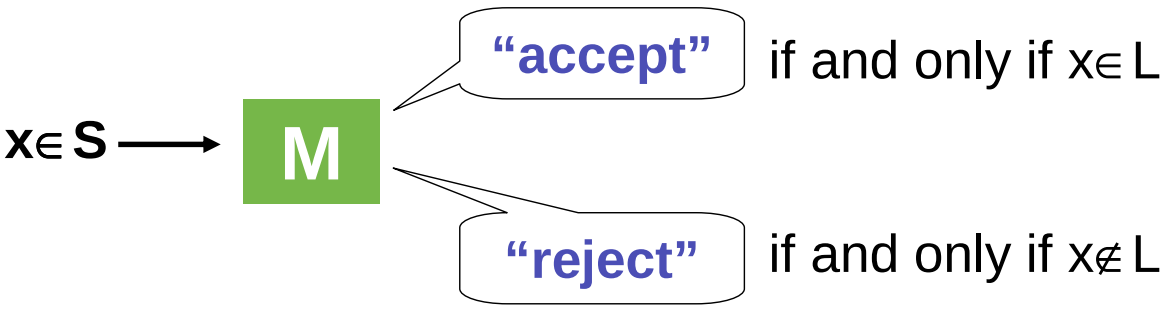
- big-O is about "less-or-equal-than",
- little o is about "strictly less than".

Languages

- Given an alphabet Σ, we can make a word or string by concatenating the letters of Σ.
- A language L is a set of words, i.e.
  - $L \subseteq \Sigma^*$, where * is the kleene operator
- Examples
  - Σ={0,1,.}, the set of decimal binary numbers
  - Σ={a,b,c,d,e,f,…}, the set of italian words

Accepted Languages

- Let L be a language $\subseteq$ S
- a machine M accept L if



Regular languages

- Given an alphabet Σ, any expression that can be obtained by

1. R = a, with $a \in \Sigma$ (a symbol of the alphabet)
2. R = ε (the null string)
3. R = $\varnothing$ (the void expression)
4. R = (R_1 | R_2), the alternation
5. R = (R_1 R_2), the concatenation
6. R = (R_1*), the kleene closure

- A language is regular if it can be defined by a regular expression

Regular languages: examples

- Binary positive numbers
  (1 | 0) *

- A decimal number with one integer digit and two decimal digits
  (+|-|ε)(0|1|2|...|9)* . (0|1|2|...|9) (0|1|2|...|9)

- identifiers of a programming language
  (a|b|c| ... |z|) (a|b|c| ... |z|0|1| ...|9|)*

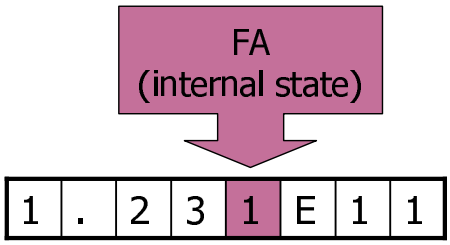
Finite Automaton and regular languages

Alternative definition

A language is regular if and only if it can be accepted by a finite automaton

A finite (state) automata

- a simple machine that reads a single input character at every time step
- has an internal state that can assume a finite number of different values
- internal state are of two different types (accept, reject) that define whether the read string is accepted or not



Finite Automaton and regular languages

Formally

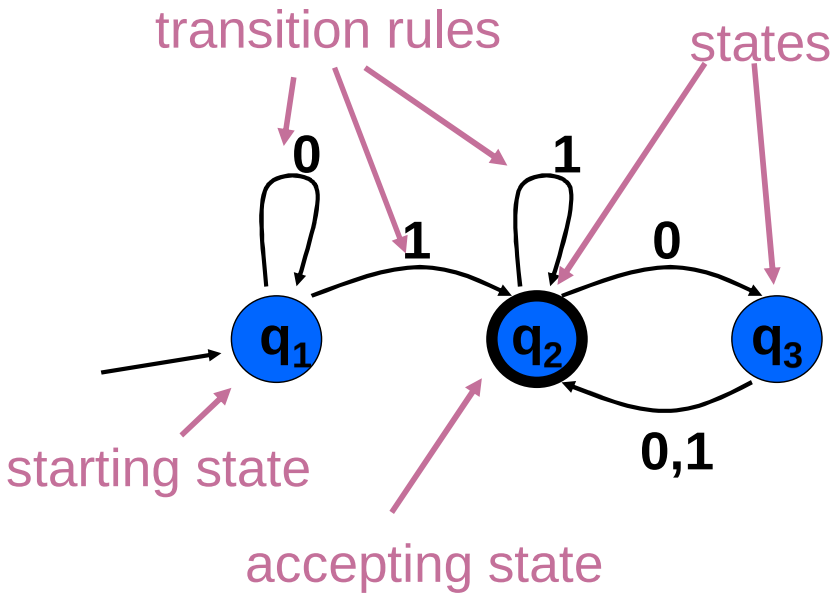
A nondeterministic finite automaton (FA) M is defined by a 5-tuple $M=(Q,\Sigma,\delta,q_0,F)$, with

Q: finite set of states
Σ: finite alphabet
δ: transition function $\delta: Q \times \Sigma_\varepsilon \rightarrow P(Q)$
$q_0 \in Q$: start state
$F \subseteq Q$: set of accepting states

The automata is called deterministic or non deterministic according whether $\delta(q,a)$ is a singleton or not for each q,a

Finite automaton

The deterministic automaton that recognizes the language:
0*1((1|0) (0|1))*



Some properties of FA and regular languages

- Deterministic FA (DFA) and non deterministic FA (NFA) are equivalent:
  - they accept all and only all the regular languages
  - there exist algorithms to transform a DFA into an equivalent FA, and vice versa …
- FA and regular expression are equivalent
  - there exist algorithms to transform a FA to the equivalent regular expression and, vice versa,…
- Regular languages are closed under union, intersection, complementation, concatenation
  - there exist algorithms to transform a FA to another FA that accepts the complement language

Context free languages

Definition A context free grammar G=(V,Σ,R,S) is defined by

- V: a finite set variables
- Σ: finite set terminals (with $V \cap \Sigma = \varnothing$)
- R: finite set of substitution rules $V \rightarrow (V \cup \Sigma)^*$
- S: start symbol $\in V$

- Definition The language of grammar G, denoted by L(G), is the set of strings of terminal symbols that can be obtained by applying the substitution rules from the start symbol

- $L(G) = \{ w \in \Sigma^* \mid S \Rightarrow^* w \}$

Example: the boolean algebra

G=(V,Σ,R,S) with

- V = {S,Z}
- Σ = {0,1,(,),¬,∨,∧}
- R:
  - $S \rightarrow 0$
  - $S \rightarrow \neg(S)$
  - $S \rightarrow (S) \vee (S)$
  - $S \rightarrow (S) \wedge (S)$

- Some elements of L(G):

  0

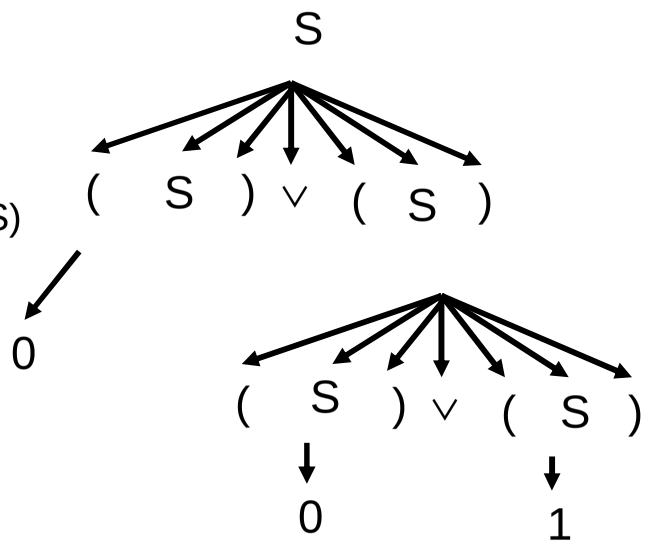
  $\neg((\neg(0)) \vee (1))$

  $(1) \vee ((0) \wedge (0))$

Parse Trees

The parse tree for (0)∨((0)∧(1))

- S → 0 | 1 | ¬(S) | (S)∨(S) | (S)∧(S)



Example: a simple programming language

G=(V,Σ,R,S) with

- V = {S,C,E,B}
- Σ = {id,val,=,(,),if,then,==}
- R: S → C;S | ε
  C → id=E
  C → if (B) then S end
  E → id|val
  B → E==E

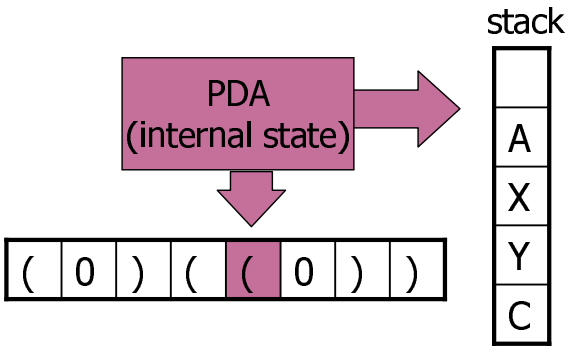
A program in L(G):

```
id=val;
id=id;
if (id==val) then
        id=val
        end
```

Pushdown automata

A pushdown automata

- is a machine composed by
  - a finite state control unit
  - a stack that contain symbols
- it can read one input character at each step
- it can add, read and remove symbols from the stack



Pushdown automata can recognize (decide) context free languages

More about context free languages

- A context free language
  - can be decided by a parsers (automata that use a stack)
  - can be decided in $O(n^3)$ ($O(n^2)$ if the language is not ambiguous)
- Context free languages strictly contains regular languages
  - for example, the language $\{a^n b^n | n>0\}$ is context free languages, but not a regular language

More languages

Context-sensitive grammars

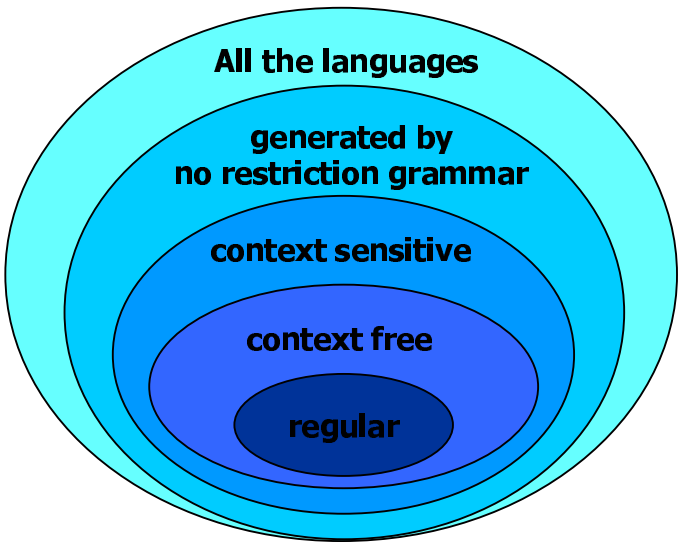
- the rules are in the form
  - $\alpha A \beta \rightarrow \alpha \gamma \beta$
  - where A is a non terminal and α,β,γ are strings of terminals and non terminals, γ cannot be null
- the rule $S \rightarrow \varepsilon$ is allowed if *S* does not appear on the right side of any rule
- no algorithm is known to decide Context-sensitive languages in polynomial time

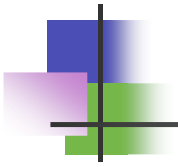
Unrestricted grammars

- the rules are any form
- no algorithm is known to decide languages generated by unrestricted grammars

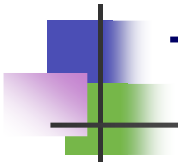
Chomsky language gerarchy

- All inclusions are strict!!





Turing machines



Turing Machines

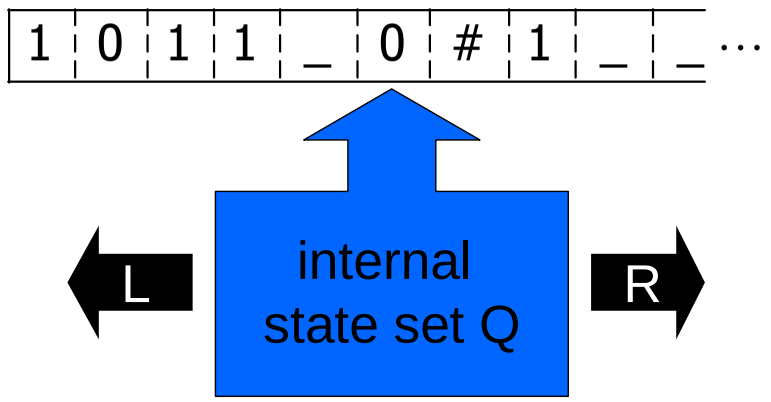
Alan M. Turing (1912–1954)

In 1936, Turing introduced his abstract model for computation in his article "*On Computable Numbers, with an application to the Entscheidungsproblem*".

At the same time, Alonzo Church published similar ideas and results.

The Turing model has become the standard model in theoretical computer science.

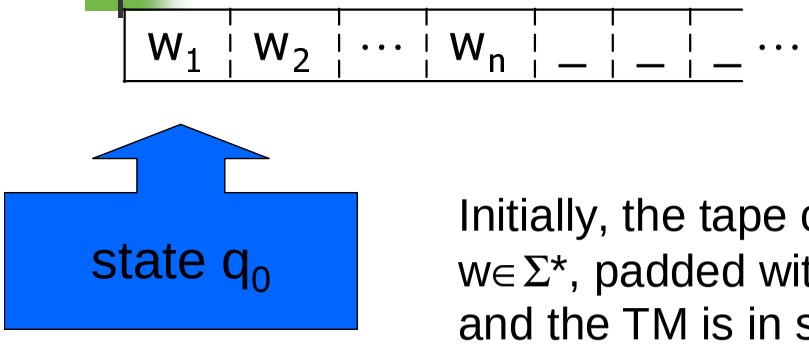
Informal Description TM



At every step, the head of the TM M reads a letter x_i from the one-way infinite tape.

Depending on its state and the letter x_i, the TM
- writes down a letter,
- moves its read/write head left or right, and
- jumps to a new state.

Input Convention

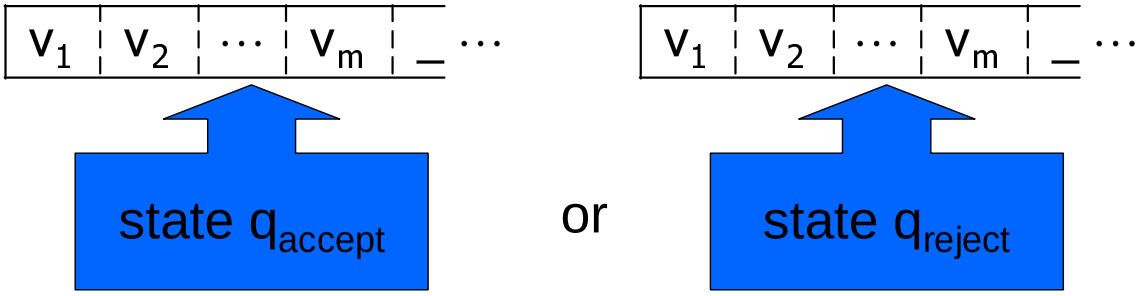


Initially, the tape contains the input $w \in \Sigma^*$, padded with blanks "_", and the TM is in start state q_0.

During the computation, the head moves left and right (but not beyond the leftmost point), the internal state of the machine changes, and the content of the tape is rewritten.

Output Convention

The computation can proceed indefinitely, or the machines reaches one of the two halting states:



Turing Machine (Def. 3.3)

A Turing machine M is defined by a 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

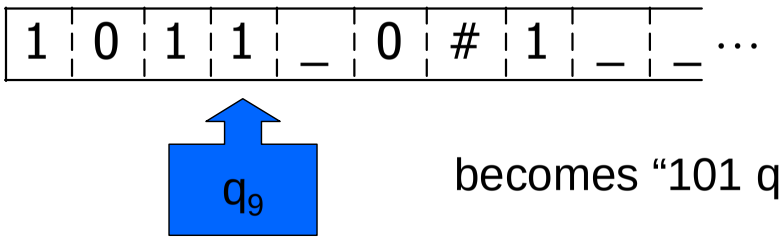
- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{ \_ \} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in$ Q
- q_{accept} accept state $\in$ Q
- q_{reject} reject state $\in$ Q
- δ the transition function

$$\delta: Q\backslash\{q_{accept},q_{reject}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$$

Configuration of a TM

The configuration of a Turing machine consists of
- the current state $q \in Q$
- the current tape contents $\in \Gamma^*$
- the current head location $\in \{0,1,2,\ldots\}$

This can be expressed as an element of $\Gamma^* \times Q \times \Gamma^*$:



becomes "101 q_9 1_0#1"

An Elementary TM Step

Let $u,v \in \Gamma^*$; $a,b,c \in \Gamma$; $q_i,q_j \in Q$, and M a TM with transition function δ.

We say that the configuration "ua q_i bv" yields the configuration "uac q_j b" if and only if:

$$\delta(q_i,b) = (q_j,c,R).$$

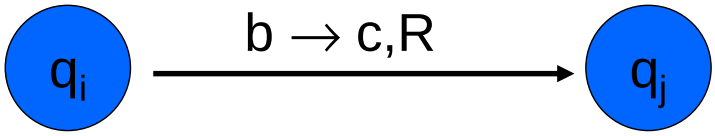
Similarly, "ua q_i bv" yields "u q_j acb" if and only if

$$\delta(q_i,b) = (q_j,c,L).$$

State diagrams of TMs

Like with PDA, we can represent Turing machines by (elaborate) diagrams.

If transition rule says: $\delta(q_i,b) = (q_j,c,R)$, then:



Terminology

starting configuration on input w: "q_0w"

accepting configuration: "$uq_{accept}v$"

rejecting configuration: "$uq_{reject}v$"

The accepting and rejecting configurations are the *halting configurations*.

Example: A = { 0^j | $j=2^n$ }

Goal: To build a TM that accepts all and only the strings in A = { 0^j | $j=2^n$ }

Approach: If j=0 then "reject"; If j=1 then "accept"; if j is even then divide by two; if j is odd and >1 then "reject". Repeat if necessary.

1. Check if j=0 or j=1, accept/reject accordingly
2. Check, by going left to right if the string has even or odd number of zeros
3. If odd then "reject"
4. If even then go back left, erasing half the zeros
5. goto 1

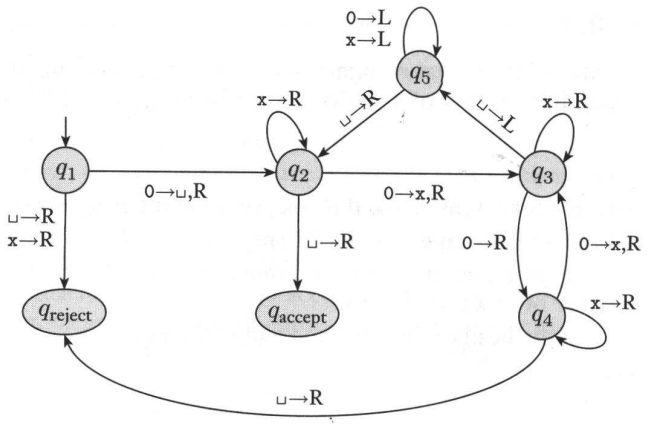


FIGURE **3.4**
State diagram for Turing machine M_2

A sample run of M_2 on input 0000:

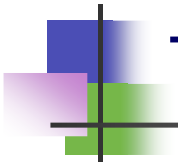
$q_1$0000	␣q_5x0x␣	␣x$q_5$xx␣
␣$q_2$000	q_5␣x0x␣	␣$q_5$xxx␣
␣x$q_3$00	␣q_2x0x␣	q_5␣xxx␣
␣x0$q_4$0	␣x$q_2$0x␣	␣$q_2$xxx␣
␣x0xq_3␣	␣xxq_3x␣	␣x$q_2$xx␣
␣x0q_5x␣	␣xxxq_3␣	␣xxq_2x␣
␣x$q_5$0x␣	␣xxq_5x␣	␣xxxq_2␣
		␣xxx␣q_{accept}

Accepting TMs

A Turing machine M accepts input $w \in \Sigma^*$ if and only if there is a finite sequence of configurations $C_1, C_2, \ldots, C_k$ with

- C_1 the starting configuration "$q_0 w$"
- for all i=1,…,k–1 C_i yields C_{i+1} (following M's δ)
- C_k is an accepting configuration "$uq_{accept}v$"

The language that consists of all inputs that are accepted by M is denoted by L(M).



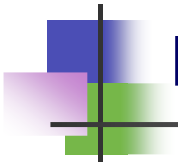
Turing recognizable

A language L is Turing-recognizable if and only if there is a TM M such that L=L(M).

Also called: a recursively enumerable language.

Note: On an input $w \notin L$, the machine M can halt in a rejecting state, or it can 'loop' indefinitely.

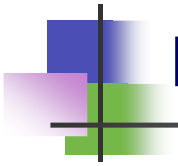
How do you distinguish between a very long computation and one that will never halt?



Enumerating Languages

When a TM E generates the words of a language,
E is an enumerator (cf. “recursively enumerable”).

A Turing machine E enumerates the language L
if it prints an (infinite) list of strings on the tape
such that all elements of L will appear on the tape,
and all strings on the tape are elements of L.
(E starts on an empty input tape. The strings
can appear in any order; repetition is allowed.)



Enumerating = Recognizing

Theorem: A language L is TM-recognizable
if and only if L is enumerable.

Proof: (“if”) Take the enumerator E and input w.
Run E and check the strings it generates.
If w is produced, then “accept” and stop,
otherwise let E continue.
(“only if”) Take the recognizer M.
Let $s_1, s_2, \ldots$ be a listing of all strings $\in \Sigma^* \supseteq L$.
For j=1,2,… run M on $s_1, \ldots, s_j$ for j time-steps.
If M accepts an s, print s. Keep increasing j.

Turing Decidable (Def. 3.5)

A language L=L(M) is decided by the TM M if on every w, the TM finishes in a halting configuration. (That is: q_{accept} for $w \in L$ and q_{reject} for all $w \notin L$.)

A language L is Turing-decidable if and only if there is a TM M that decides L.

Also called: a recursive language.

Multitape Turing Machines

A k-tape Turing machine M has k different tapes and read/write heads. It is thus defined by the 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{ \_ \} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in$ Q
- q_{accept} accept state $\in$ Q
- q_{reject} reject state $\in$ Q
- δ the transition function

$$\delta: Q\backslash\{q_{accept},q_{reject}\} \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$

k-tape TMs versus 1-tape TMs

Theorem: For every multi-tape TM M, there is a single-tape TM M' such that L(M)=L(M'). Or, for every multi-tape TM M, there is an equivalent single-tape TM M'.

Proving and understanding these kinds of robustness results, is essential for appreciating the power of the Turing machine model.

From this theorem follows:
A language L is TM-recognizable if and only if some multi-tape TM recognizes L.

Proof sketch

Let $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$ be a k-tape TM.
Construct 1-tape M' with expanded $\Gamma' = \Gamma \cup \underline{\Gamma} \cup \{\#\}$

Represent M-configuration

$$u_1 q_j a_1 v_1, \quad u_2 q_j a_2 v_2, \quad \ldots, \quad u_k q_j a_k v_k$$

by M' configuration

$$q_j \# u_1 \underline{a}_1 v_1 \# u_2 \underline{a}_2 v_2 \# \ldots \# u_k \underline{a}_k v_k$$

(The tapes are seperated by #, the head positions are marked by underlined letters.)

Proof sketch

On input $w=w_1 \ldots w_n$, the TM M' does the following:

- Prepare initial string: $\#\underline{w}_1 \ldots w_n \#\_\#\cdots\#\_\#\_ \cdots$
- Read the underlined input letters $\in \Gamma^k$
- Simulate M by updating the input and the underlining of the head-positions.
- Repeat 2-3 until M has reached a halting state
- Halt accordingly.

PS: If the update requires overwriting a # symbol, then shift the part # ⋯_ one position to the right.

Nondeterministic TMs

A nondeterministic Turing machine M can have several options at every step. It is defined by the 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, with

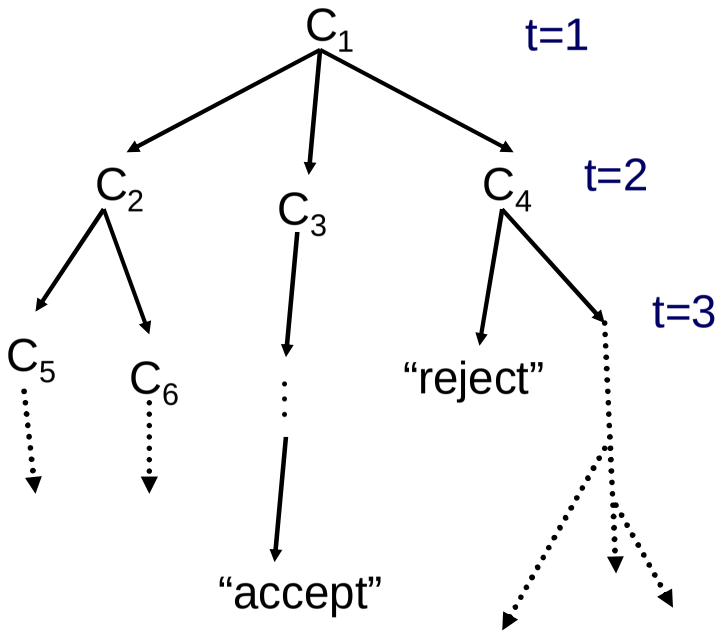
- Q finite set of states
- Σ finite input alphabet (without "_")
- Γ finite tape alphabet with $\{\_\} \cup \Sigma \subseteq \Gamma$
- q_0 start state $\in$ Q
- q_{accept} accept state $\in$ Q
- q_{reject} reject state $\in$ Q
- δ the transition function

$$\delta: Q\backslash\{q_{accept},q_{reject}\} \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L,R\})$$

Computing with Nondeterministic TMs

Evolution of the n.d. TM represented by a tree of configurations (rather than a single path).

If there is (at least) one accepting leave, then the TM accepts.



Simulating Nondeterministic TMs with Deterministic Ones

We want to search every path down the tree for accepting configurations.

Bad idea: "depth first". This approach can get lost in never-halting paths.

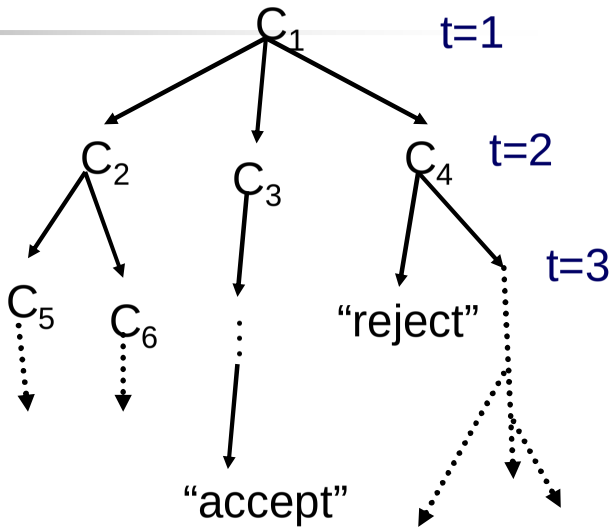
Good idea: "breadth first". For time step 1,2,… we list all possible configurations of the non-deterministic TM. The simulating TM accepts when it lists an accepting configuration.

Breadth First

Let b be the maximum number of children of a node.

Any node in the tree can be uniquely identified by a string $\in \{1,\ldots,b\}^*$.

Example: location of the rejecting configuration is (3,1).



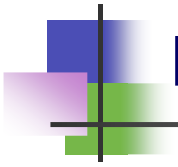
With the lexicographical listing ε, (1), (2),…, (b), (1,1), (1,2),…,(1,b), (2,1),… et cetera, we cover all nodes.

Proof of Theorem (simulating NTM)

Let M be the nondeterministic TM on input w.

The simulating TM uses three tapes:
T1 contains the input w
T2 the tape content of M on w at a node
T3 describes a node in the tree of M on w.

1) T1 contains w, T2 and T3 are empty
2) Simulate M on w via the deterministic path to the node of tape 3. If the node accepts, "accept", otherwise go to 3)
3) Increase the node value on T3; go to 2)



Robustness

Just like k-tape TMs, nondeterministic Turing machines are not more powerful than simple TMs:

Every nondeterministic TM has an equivalent 3-tape Turing machine, which –in turn– has an equivalent 1-tape Turing machine.

Hence: “A language L is recognizable if and only if some nondeterministic TM recognizes it.”

The Turing machine model is extremely robust.

Other Computational Models

We can consider many other ‘reasonable’ models of computation: neural networks, quantum computing…

Experience teaches us that every such model can be simulated by a Turing machine.

Church-Turing Thesis:

The intuitive notion of computing and algorithms is captured by the Turing machine model.

Random Access Machines

RAM Machines

- A set of R_1, R_2, R_3, ... of registers is available
  - Every register is a memory that can contain a positive integer
- Program contains a sequence of instructions in
  - Increment: R_i ++
    The registrer is increased by 1
  - Decrement: R_i --
    The register i decreased b1. If the register is already 0, the instruction has no effect
  - conditional jump: IF R_i GOTO L1
    If the register is not null, we jump at L1

The sum of the two registers R_1 e R_2

```
        IF R1 GOTO ciclo
ciclo:  R2++
        R1 - -
        IF R1 GOTO ciclo
fine:
```


Non-deterministic RAM

Is number in R_1 prime ?
We use a procedure DIV that check whether R_2 is a factor of R_1 ?

- In non-deterministic RAM, three other instructions are available
  - random choice: FORK
    takes in input two instructions and execute one of them
  - ACCEPT
    the machine accepts
  - REJECT
    the machine rejects
- An input is accept, if there is a path that allows the program to accept.

```
a       IF R1 GOTO b
        R1--
        FORK{  GOTO a,
               R2++ }
        GOTO a
        DIV
        IF R3 GOTO re
        ACCEPT
re      REJECT
```

μ-recursive functions

μ-recursive functions are defined as

- basic functions
  - constants, $f(x_1,..,x_k)=n$
  - increment, $S(x)=x+1$
  - projection, $U^s(x_1,..,x_k)=x_s$
- composed functions:

  if h, g_i, are μ-recursive functions then f is μ-recursive
  - composition, $f(x_1,..,x_k)=h(g_1(x_1,..,x_k),\ldots, g_r(x_1,..,x_k))$
  - recursion, $f(0,x_1,..,x_k)= g(x_1,..,x_k)$

    $f(y+1,x_1,..,x_k)=h(y,f(y,x_1,..,x_k), x_1,..,x_k)$

μ-recursive functions: multiplication

- constants, $f(x_1,..,x_k)=n$
- increment, $S(x)=x+1$
- projection, $U^s(x_1,..,x_k)=x_s$
- composition, $f(x_1,..,x_k)=h(g_1(x_1,..,x_k),\ldots, g_r(x_1,..,x_k))$
- recursion, $f(0,x_1,..,x_k)= g(x_1,..,x_k)$

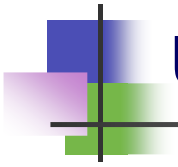
  $f(y+1,x_1,..,x_k)=h(y,f(y,x_1,..,x_k), x_1,..,x_k)$

$piu(0,b)=b$ ← base recursion: g is the projection operator

$piu(a+1,b)=piu(a,b)+1$ ← recursion: h is the successor

$per(0,b)=0$ ← base recursion: g is a constant

$per(a+1,b)=piu(per(a,b),b)$ ← recursion: h is the composition of piu and the projection



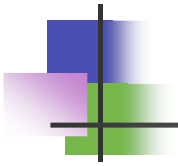
Universal programming languages

A universal programming language (for a RAM machine) must include at least

- increment and decrement operators
- instruction concatenation and
  - GOTO and IF or
  - recursion or
  - WHILE or
  - undefined FOR

Remark

- A FOR is defined if the values assigned to the FOR variable are defined before starting the execution of the FOR
  - f.i. for(i=0;i<10;i++)
- a defined FOR is not sufficient to obtain an universal machine. Why?



Decidibility

Hilbert's 10th Problem

In 1900, David Hilbert (1862–1943) proposed his *Mathematical Problems* (23 of them).

The Hilbert's 10th problem is: **Determination of the solvability of a Diophantine equation.**

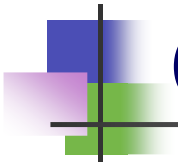
Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: *To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.*

Diophantine Equations

Let $P(x_1,\ldots,x_k)$ be a polynomial in k variables with integral coefficients. Does P have an integral root $(x_1,\ldots,x_k) \in Z^k$?

Example: $P(x,y,z) = 6x^3yz + 3xy^2 - x^3 - 10$ has integral root $(x,y,z) = (5,3,0)$.

Other example: $P(x,y) = 21x^2 - 81xy + 1$ does not have an integral root.

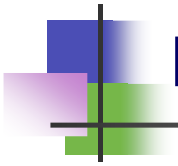


(Un)solving Hilbert's 10th

Hilbert's *"…a process according to which it can be determined by a finite number of operations…"* needed to be defined in a proper way.

This was done in 1936 by Church and Turing.

Matijasevič proved that Hilbert's 10th problem is unsolvable in 1970.



Decidability

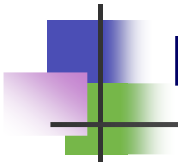
Thus, we are now ready to tackle the question:

What can computers do and what not?

We do this by considering the question:

Which languages are TM-decidable, Turing-recognizable, or neither?

Assuming the Church-Turing thesis, these are fundamental properties of the languages.



Describing TM Programs

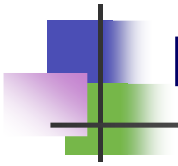
Three Levels of Describing algorithms:

- formal (state diagrams, CFGs, et cetera)
- implementation (pseudo-Pascal)
- high-level (coherent and clear English)

Describing input/output format:

TMs allow only strings $\in \Sigma^*$ as input/output.

If our X and Y are of another form (graph, Turing machine, polynomial), then we use <X,Y> to denote 'some kind of encoding $\in \Sigma^*$'.



Deciding Regular Languages

The acceptance problem for deterministic finite automata is defined by:

A_{DFA} = { <B,w> | B is a DFA that accepts w }

Note that this language deals with all possible DFAs and inputs w, not a specific instance.

Of course, A_{DFA} is a TM-decidable language.

A_{DFA} is Decidable

Proof: Let the input <B,w> be a DFA with
B=(Q, Σ, δ, q_{start}, F) and w∈Σ*.
The TM performs the following steps:

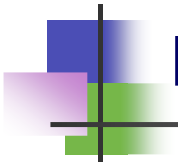
1) Check if B and w are 'correct', if not: "reject"
2) Simulate B on w with the help of two pointers:
   $P_q \in$ Q for the internal state of the DFA, and
   $P_w \in$ {0,1,…,|w|} for the position on the string.
   While we increase P_w from 0 to |w|, we change P_q according to the input letter w_{Pw} and the transition function value $\delta(P_q, w_{Pw})$.
3) If M accepts w: "accept"; otherwise "reject"

Regular Expressions

The acceptance problem
A_{REX} = { <R,w> | R is a regular expression that can generate w }
is a Turing-decidable language.

Proof Theorem. On input <R,w>:

1. Check if R is a proper regular expression and w a proper string
2. Convert R into a DFA B
3. Run earlier TM for A_{DFA} on <B,w>



Emptiness Testing

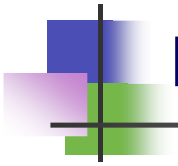
Another problem relating to DFAs is the emptiness problem:

E_{DFA} = {<A> | A is a DFA with L(A) = ∅ }

How to decide this language?

This language concerns the behavior of the DFA A on all possible strings.

Less obvious than the previous examples.



Proof for DFA-Emptiness

Algorithm for E_{DFA} on input $A=(Q,\Sigma,\delta,q_{start},F)$:

1) If A is not proper DFA: "reject"
2) Make set S with initially $S=\{q_{start}\}$
3) Repeat |Q| times:
   a) If S has an element in F then "reject"
   b) Otherwise, add to S the elements that can be δ-reached from S via:
      "If $\exists q_i \in S$ and $\exists s \in \Sigma$ with $\delta(q_i,s)=q_j$, then q_j goes into S_{new}"
4) If final $S \cap F = \emptyset$ "accept"

DFA-Equivalence

A problem that deals with two DFAs:
EQ_{DFA} = {<A,B> | L(A) = L(B) }

Theorem: EQ_{DFA} is TM-decidable.

Proof: Look at the *symmetric difference* between the two languages: $(L(A) \cap \bar{L}(B)) \cup (\bar{L}(A) \cap L(B))$

Note: "L(A)=L(B)" is equivalent with an empty symmetric difference between L(A) and L(B).

This difference is expressed by standard DFA transformations: union, intersection, complement.

Proof of Theorem (cont.)

Algorithm on given <A,B>:

1) If A or B are not correct DFA: "reject"
2) Construct a third DFA C that accepts the language $(L(A) \cap \bar{L}(B)) \cup (\bar{L}(A) \cap L(B))$
3) Decide with the TM of the previous theorem whether or not $C \in E_{DFA}$
4) If $C \in E_{DFA}$ then "accept";
   If $C \notin E_{DFA}$ then "reject"

Context-Free Languages

Similar languages for context-free grammars:

A_{CFG} = { <G,w> | G is a CFG that generates w }

E_{CFG} = { <G> | G is a CFG with L(G)=∅ }

EQ_{CFG} = { <G,H> | G and H are CFGs with L(G)=L(H) }

The problem with CFGs and PDAs is that they are inherently nondeterministic.

Chomsky Normal Form

Definition
A context-free grammar G = (V,Σ,R,S) is in Chomsky normal form if every rule is of the form

$A \to BC$ or $A \to x$

with variables $A \in V$ and $B,C \in V \setminus \{S\}$, and $x \in \Sigma$. For the start variable S we also allow "$S \to \varepsilon$"

Every context-free grammar can be transformed into an equivalent Chomsky NF grammar.

The derivation $S \Rightarrow^* w$ requires $2|w|-1$ steps (apart from $S \Rightarrow \varepsilon$).

Deciding CFGs (1)

Theorem: The language
A_{CFG} = { <G,w> | G is a CFG that generates w }
is TM-decidable.

Proof: Perform the following algorithm:

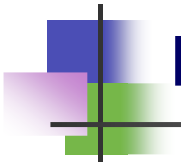
1) Check if G and w are correct, if not "reject"
2) Rewrite G to G' in Chomsky normal form
3) Take care of w=ε case via S→ε check for G'
4) List all G' derivations of length 2|w|–1
5) Check if w occurs in this list;
   if so "accept"; if not "reject"

Deciding CFGs (2)

Theorem: The language
E_{CFG} = { <G> | G is a CFG with L(G)=∅ }
is TM-decidable.

Proof: Perform the following algorithm:

1) Check if G is proper, if not "reject"
2) Let G=(V,Σ,R,S), define set T=Σ
3) Repeat |V| times:
   - Check all rules $B \rightarrow X_1 \ldots X_k$ in R
   - If $B \notin T$ and each $X_1 \ldots X_k$ is in T then add B to T
4) If $S \in T$ then "reject", otherwise "accept"



Equality CFGs

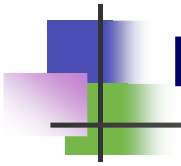
What about the equality language

EQ_{CFG} = { <G,H> | G and H are CFGs with L(G)=L(H) }?

For DFAs we could use the emptiness decision procedure to solve the equality problem.

For CFGs this is not possible… (why?)
… because CFGs are not closed under complementation or intersection.

Later we will see that EQ_{CFG} is not TM-decidable.



Deciding Languages

We now know that the languages:

A_{DFA} = { <B,w> | B is a DFA that accepts w }

A_{CFG} = { <G,w> | G is a CFG that generates w }

are TM decidable.

What about the obvious next candidate

A_{TM} = {<M,w> | M is a TM that accepts w }?

Is one TM capable of simulating all other TMs?

The Universal TM

Given a description <M,w> of a TM M and input w, can we simulate M on w?

We can do so via a universal TM U (2-tape):

1) Check if M is a proper TM
   Let M = $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$
2) Write down the starting configuration
   $< q_0w >$ on the second tape
3) Repeat until halting configuration is reached:
   - Replace configuration on tape 2 by next configuration according to δ
4) "Accept" if q_{accept} is reached; "reject" if q_{reject}

The Halting Problem

The existence of the universal TM U shows that
A_{TM} = {<M,w> | M is a TM that accepts w }
is TM-recognizable, but can we also *decide* it?

The problem lies with the cases when M does not halt on w. In short: the halting problem.

We will see that this is an insurmountable problem: in general one cannot decide if a TM will halt on w or not, hence A_{TM} is undecidable.

Are there undecidable languages?

For $\Sigma=\{0,1\}$, there are 2^k words of length k.

- A language is a set words: there are $2^{(2^k)}$ languages $L \subseteq \Sigma^k$.
- A TM M an be described by a string: there are 2^k TMs of length k.

An idea

Let us count languages and TMs, if TMs are less than languages then we know that some language cannot be computed!!

However, languages and TMs are infinite and we need a more sophisticate way to compare them

Cardinality

A set S has k elements if and only if there is a bijection possible between S and {1,2,…,k}.

S and {1,…,k} have the same cardinality.

If there is a surjection possible from {1,…,n} to S, then $n \geq |S|$.

We can generalize this way of comparing the sizes of sets to infinite ones.

Countable Sets

If there exists a surjective function F:S→N,

- the set S is infinite

  "The set N has not more elements than S."

If there exists a surjective function F:N→S

- the set S may not be infinite

  "The set S has not more elements than N."

If there exists a bijective function F:N→S.

- The set S is countable

  "The sets N and S are of equal size."

Some Countable Infinite Sets

One can make bijections between N and Z, N^2, {a}*, {a,b}*:

0	1	2	3	4	5	6	…
0	+1	−1	+2	−2	+3	−3	…
(0,0)	(0,1)	(1,0)	(0,2)	(1,1)	(2,0)	(0,3)	…
ε	a	aa	aaa	aaaa	aaaaa	aaaaaa	…
ε	a	b	aa	ab	ba	bb	…

A Big Countable Set

Consider the set N*, the set of finite sequences of numbers: $(0) \in N^*$, $(4,63) \in N^*$, $(1,0,\ldots,1) \in N^*$.

This set is also countable infinite.

How do make the bijection between N* and N?

1. There are infinitely many primes
   $p_1=2$, $p_2=3$, $p_3=5$, …
2. Every number $n \in \{2,3,\ldots\}$ has a unique prime factorization: $84 = p_1 \cdot p_1 \cdot p_2 \cdot p_4$

Bijection between N and N*

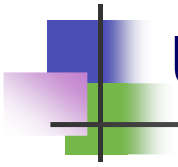
Let $(n_1,n_2,\ldots,n_k) \in N^*$

Consider the number $m = p_1^{n_1+1} \cdot p_2^{n_2+1} \cdots p_k^{n_k+1}$

Infinitely many primes: For every k this is possible.

Unique prime factorization: Given m, we can determine $(n_1,n_2,\ldots n_k)$

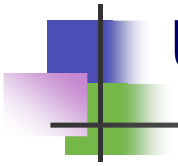
The "+1" enables us to make a distinction between (1,4,0) and (1,4).



Uncountable Sets

There are infinite sized sets that are not countable.
Typical examples are $\Re$, $\mathcal{P}(N)$ and $\mathcal{P}(\{0,1\}^*)$

We prove this by a diagonalization argument.
In short, if S is countable, then you can make a list $s_1, s_2, \ldots$ of all elements of S.
Diagonalization shows that given such a list, there will always be an element x of S that does not occur in $s_1, s_2, \ldots$



Uncountability of $\mathcal{P}(N)$

The set $\mathcal{P}(N)$ contains all the subsets of {0,1,2,…}.

There is a bijection between P (N) and $\{0,1\}^\infty$ (the set of infinite bit string).
Each subset $X \subseteq N$ can be identified by an infinite string of bits $x_0x_1x_2\ldots$ such that $x_j=1$ iff $j \in X$.

Proof by contradiction: Assume $\mathcal{P}(N)$ is countable.

Hence there must exist a bijection $F:N \rightarrow \{0,1\}^\infty$.
"There is a list of *all* infinite bit strings."

Diagonalization

Try to list all possible infinite bit strings:

0	0	0	0	0	0	⋯
1	1	1	1	1	1	⋯
2	1	0	0	0	0	⋯
3	0	1	0	1	0	⋯
⋮						⋱

Look at the bit string on the diagonal of this table: 0101… The negation of this string ("1010…") does not appear in the table.

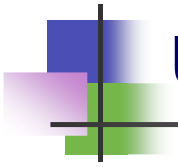
No bijection $\mathbf{N} \to \{0,1\}^{\infty}$

Let F be a function $N \to \{0,1\}^{\infty}$.
F(0),F(1),F(2),… are all infinite bit strings.

Define the infinite string $Y=Y_0Y_1Y_2\ldots$ by
Y_j = **NOT**(j-th bit of F(j))

On the one hand $Y \in \{0,1\}^{\infty}$, but on the other hand: for every $j \in N$ we know that $F(j) \neq Y$ because F(j) and Y differ in the j-th bit.

F cannot be a surjection: $\{0,1\}^{\infty}$ is uncountable.



Uncountability

We just showed that there it is impossible to have surjection from N to the set $\{0,1\}^\infty$.

Similar proofs are possible for the uncountability of the sets $\Re$, $\mathcal{P}(\{0,1\}^*)$, ….

Counting TMs

Observation: Every TM has a finite description; there is only a countable number of different TMs. (A description <M> can consist of a finite string of bits, and the set $\{0,1\}^*$ is countable.)

Our definition of Turing recognizable languages is a mapping between the set of TMs $\{M_1, M_2, \ldots\}$ and the set of languages $\{L(M_1), L(M_2), \ldots\} \subseteq \mathcal{P}(\Sigma^*)$.

Question: How many languages are there?

Counting Languages

There are uncountable many different languages over the alphabet Σ={0,1}: the set of languages is P ({0,1}*)

With the lexicographical ordering ε,0,1,00,01,… of Σ*, every L coincides with an infinite bit string via its characteristic sequence χ_L.

Example for L={0,00,01,000,001,…} with χ_L= 0101100…

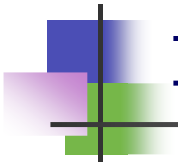
Σ*	ε	0	1	00	01	10	11	000	001	010	⋯
L		X		X	X			X	X	X	⋯
χ_L	0	1	0	1	1	0	0	1	1	1	⋯

Counting TMs and Languages

There is a bijection between the set of languages over the alphabet Σ={0,1} and the uncountable set of infinite bit strings $\{0,1\}^{\infty}$.

- There are uncountable many different languages L⊆{0,1}*.
- Hence there is no surjection possible from the countable set of TMs to the set of languages. Specifically, the mapping L(M) is not surjective.

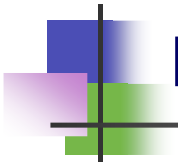
Conclusion: There are languages that are not Turing-recognizable. (A lot of them.)



Is This Really Interesting?

We now know that there are languages that are not Turing recognizable, but we do not know what kind of languages are non-TM recognizable.

Are there interesting languages for which we can prove that there is no Turing machine that recognizes it?



Proving Undecidability (1)

Consider –again– the acceptance language
A_{TM} = { <M,w> | M is a TM that accepts w }.

Proof that A_{TM} is not TM-decidable
(Contradiction) Assume that TM G decides A_{TM}:

$$G\langle M, w\rangle = \begin{cases} \text{"accept" if M accepts w} \\ \text{"reject" if M does not accept w} \end{cases}$$

From G we construct a new TM D that will get us into trouble…

Proving Undecidability (2)

The TM D works as follows on input <M> (a TM):

1) Run G on <M,<M>>
2) Disagree with the answer of G

(The TM D always halts because G always halts.)

In short: $D\langle M\rangle = \begin{cases} \text{"accept" if G rejects } \langle M, \langle M\rangle\rangle \\ \text{"reject" if G accepts } \langle M, \langle M\rangle\rangle \end{cases}$

Hence: $D\langle M\rangle = \begin{cases} \text{"accept" if M does not accept } \langle M\rangle \\ \text{"reject" if M does accept } \langle M\rangle \end{cases}$

Now run D on <D> ("on itself")…

Proving Undecidability (3)

Result: $D\langle D\rangle = \begin{cases} \text{"accept" if D does not accept } \langle D\rangle \\ \text{"reject" if D does accept } \langle D\rangle \end{cases}$

This does not make sense: D only accepts if it rejects, and vice versa.
(Note again that D always halts.)

Contradiction: **A_{TM} is not TM-decidable**.

This proof used diagonalization implicitly…

Review of Proof (1)

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	…
M_1	accept		accept		
M_2	accept	accept	accept	accept	
M_3					…
M_4	accept	accept			
⋮			⋮		⋱

'Acceptance behavior' of M_i on $\langle M_j \rangle$

Review of Proof (2)

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	…
M_1	accept	reject	accept	reject	
M_2	accept	accept	accept	accept	
M_3	reject	reject	reject	reject	…
M_4	accept	accept	reject	reject	
⋮			⋮		⋱

'Deciding behavior' of G on $\langle M_i, \langle M_j \rangle \rangle$

Review of Proof (3)

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	…	$\langle D \rangle$	…
M_1	accept	reject	accept	reject			
M_2	accept	accept	accept	accept			
M_3	reject	reject	reject	reject	…		
M_4	accept	accept	reject	reject			
⋮			⋮		⋱		
D	reject	reject	accept	accept	…		
⋮							

Disagreeing D has to occur in list as well…

Review of Proof (4)

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	…	$\langle D \rangle$	…
M_1	accept	reject	accept	reject			
M_2	accept	accept	accept	accept			
M_3	reject	reject	reject	reject	…		
M_4	accept	accept	reject	reject			
⋮			⋮		⋱		
D	reject	reject	accept	accept	…	?	
⋮							

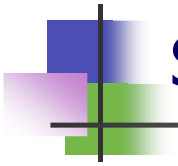
Contradiction for D on input <D>.

Another View of the Problem

The **"Self-referential paradox"** occurs when we force the TM D to disagree with itself.

On the one hand, D knows what it is going to do on input <D>, but then it decides to do something else instead.

"You cannot know for sure what you will do in the future, because then you could decide to change your actions and create a paradox."

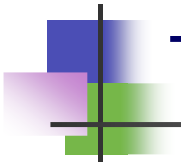


Self-Reference in Math

The diagonalization method implements the self-reference paradox in a mathematical way.

In logic this approach is often used to prove that certain things are impossible.

Kurt Gödel gave a mathematical equivalent of "This sentence is not true."

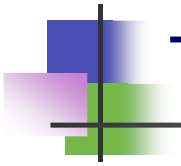


TM-Unrecognizable

A_{TM} is not TM-decidable, but it is TM-recognizable ?

Theorem: A_{TM} is recognizable

Proof: It is suffcient to simulate M. Run the M on w. If M accepts, then accept



TM-Unrecognizable

A_{TM} is not TM-decidable, but it is TM-recognizable. What about a language that is not recognizable?

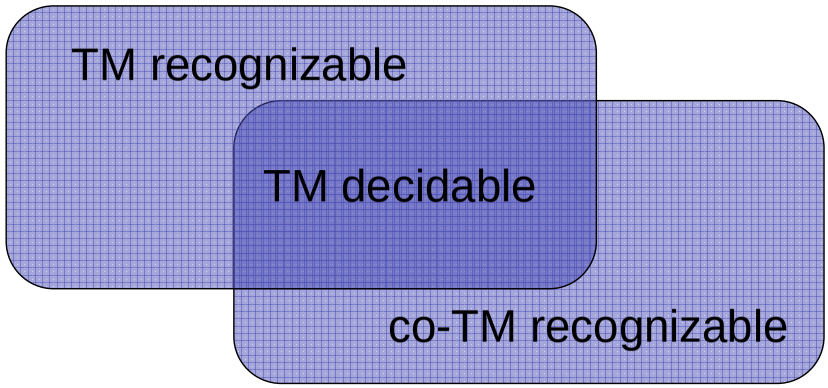
Theorem: If a language A is recognizable and its complement $\bar{A}$ is recognizable, then A is Turing machine decidable.

Proof: Run the recognizing TMs for A and $\bar{A}$ in parallel on input x. Wait for one of the TMs to accept. If the TM for A accepted: “accept x”; if the TM for $\bar{A}$ accepted: “reject x”.

$\bar{A}_{TM}$ is not TM-Recognizable

By the previous results it follows that $\bar{A}_{TM}$ cannot be TM-recognizable, because this would imply that A_{TM} is TM decidable.

We call languages like $\bar{A}_{TM}$ co-TM recognizable.



Things that TMs Cannot Do:

The following languages are also unrecognizable:

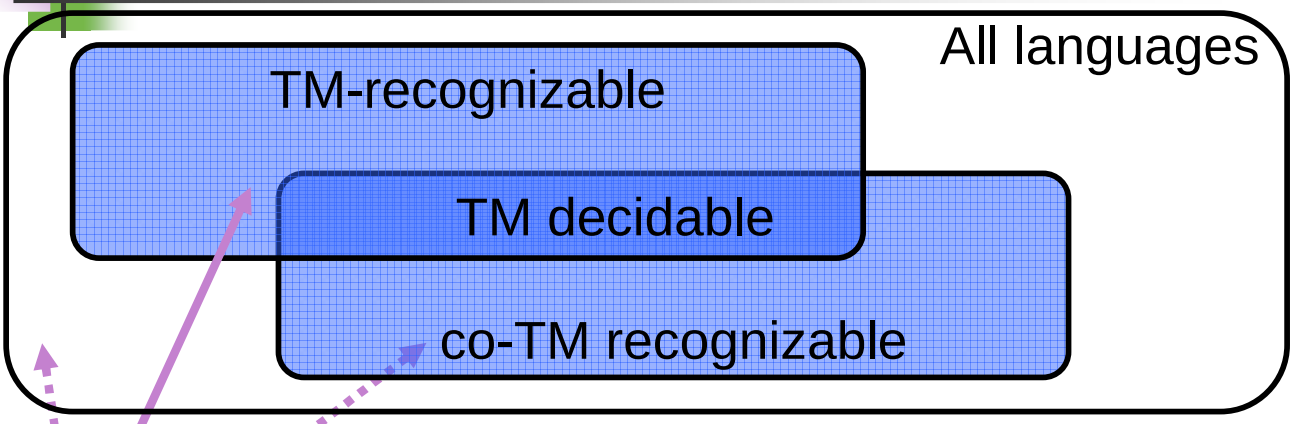
E_{TM} = { <G> | G is a TM with L(G)=∅ }

EQ_{TM} = { <G,H> | G and H are TMs with L(G)=L(H) }

To be precise:

- E_{TM} is co-TM recognizable
- EQ_{TM} is not even co-Turing recognizable

Summing up

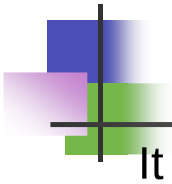


A_{TM} = { <M,w> | M is a TM that accepts w }

E_{TM} = { <M> | M is a TM with L(M)=∅ }

EQ_{TM} = { <G,H> | G,H TMs with L(G)=L(H) }

Reducibility



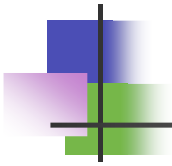
Reducibility

It required quite some effort to prove that A_{TM} is not TM-decidable.

But now we can build on this result as follows:

If "L is TM-decidable" implies "A_{TM} is decidable", then L is not decidable.

Typical proof outline:
Let M be the TM that decides L; with this M as a subroutine, the following TM [….] decides A_{TM}.
Conclusion: M is not TM-decidable.



Halting Problem Revisited

Theorem: The 'halting problem' language
$HALT_{TM}$ = { <M,w> | the TM M halts on input w }
is undecidable (but of course recognizable).

Proof: Let G be a TM that decides $HALT_{TM}$.
The following TM decides A_{TM}:

1. On input <M,w> run G to decide halting
2. If G rejected <M,w> then "reject"
3. If G accepted <M,w>
   then copy (reject/accept) output of M on w.

(Note that this TM always produces an output.)
A_{TM} is undecidable, hence such a G cannot exist.

Emptiness Testing (1)

Theorem: The language of non-accepting TMs
$E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM with } L(M)=\emptyset \}$
is not decidable (but co-TM recognizable).

Reduction proof: Let G be a TM that decides E_{TM}…

But how to deal with input <M> *versus* <M,w>?
Proof idea: Given <M,w>, we will make a different Turing machine P_{Mw} such that the answer to "$P_{Mw} \in E_{TM}$?" will also answer "$\langle M,w \rangle \in A_{TM}$?"

Emptiness Testing (2)

Proof: Given <M,w>, define the TM P_{Mw} by:
P_{Mw} = "On input x:

1. If $x \neq w$ then "reject"
2. If x=w then run M on w
   If M accepts w then "accept"

By construction:
Either $L(P_{Mw})=\{w\}$ if and only if M accepts w,
or $L(P_{Mw})=\emptyset$ if and only if M does not accept w.

Deciding "$P_{Mw} \in E_{TM}$?" decides "$\langle M,w \rangle \in A_{TM}$?"

Emptiness Testing (3)

Final proof: Let G be a TM that decides E_{TM}.
Consider the following TM on input <M,w>
1) Construct TM P_{Mw}
2) Run G on P_{Mw}
3) If G accepts P_{Mw} then "reject" <M,w>
4) If G rejects P_{Mw} then "accept" <M,w>

Because:
$<P_{Mw}> \notin E_{TM} \Longleftrightarrow$ M accepts w $\Longleftrightarrow <M,w> \in A_{TM}$
the above TM decides A_{TM}, this TM cannot exist.
Conclusion: E_{TM} is not TM-decidable.

Rice's Theorem

Consider the generic TM-related language
T= { <M> | M is a TM for which a property P holds }

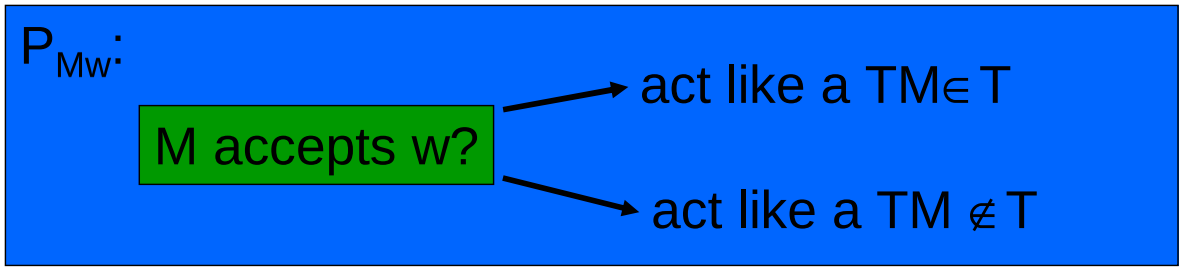
Rice's Theorem: **If** P is such that
1) $L(M_1) = L(M_2)$ implies "$<M_1> \in T \Longleftrightarrow <M_2> \in T$"
(That is: "$M \in T$?" depends only on L(M).)
2) There are two TMs M_1 and M_2 with
$<M_1> \in T$ and $<M_2> \notin T$
(The language T is non-trivial.)

Then the language T is undecidable.

Proof Idea Rice's Theorem

Assume that there is a TM G that decides T…

As with the emptiness language we will take an input <M,w> for an A_{TM} question, and turn it into a single TM P_w such that deciding "$<P_{Mw}> \in T$?" also decides "$<M,w> \in A_{TM}$?"



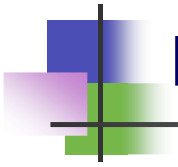
Proving Rice cont.

TMs with empty language are either in T or not.
Assume that empty language $\notin T$, and $<Q> \in T$.

Proof: Given <M,w>, define the TM P_{Mw} by:
P_{Mw} = "On input x:
1. Run M on w
2. If M rejected then "reject" x
3. If M accepted then run Q on x

By construction:
- If M on w rejects or never stops: $L(P_{Mw})=\varnothing$
- If M accepts w: $L(P_{Mw}) = L(Q)$

Deciding "$P_{Mw} \in T$?" decides "$<M,w> \in A_{TM}$?"



Final Proof Rice's Theorem

Let G be a TM that decides T.

The following TM would decide A_{TM} (input <M,w>):

1) Given M and w, make P_{Mw} as described before

2) Give same answer as G on $<P_{Mw}>$

Correctness proof:

- If M accepts w: $L(P_{Mw}) = L(Q)$, hence $<P_{Mw}> \in T$
- If M does not accept w, then $L(P_{Mw}) = \emptyset$: $<P_{Mw}> \notin T$

Consequences of Rice's Thm.

Almost any language property of Turing machines is undecidable:

$Regular_{TM}$ = { <M> | L(M) is a regular language }

$Finite_{TM}$ = { <M> | L(M) is a finite language }

CFG_{TM} = { <M> | L(M) is a CFG language }

Deciding Equality

As you would expect, the equality language
$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ TMs}, L(M_1)=L(M_2) \}$
is undecidable.

Proof Idea: Deciding EQ_{TM} for a fixed M_2 gives already contradictory conclusions.
Let $M_\emptyset$ be a TM with $L(M_\emptyset)=\emptyset$.
A TM that decides EQ_{TM} can also decide E_{TM} by deciding "$\langle M_1, M_\emptyset \rangle \in EQ_{TM}$?"

EQ_{TM} is not even TM or co-TM recognizable…

Limited Success thus far

Our reductions have been very straightforward: "A TM for this language can be transformed into a similar TM that decides another language"

As a result, the provable undecidable languages are very alike A_{TM}, EQ_{TM}, $HALT_{TM}$, et cetera.

For languages concerning questions not about TMs we have to use a more refined reduction.

Computation Histories

An accepting computation history for a TM M on a string w consists of a sequence of configurations $C_1, C_2, \ldots, C_k$ such that the following properties hold:

1. C_1 is the start configuration of M on w
2. Each C_{j+1} follows properly from C_j
3. C_k is an accepting configuration

Observation: Stating "$\langle M,w\rangle \notin A_{TM}$" is equivalent with stating "There is no accepting computation history $C_1, \ldots, C_k$ for M on w".

Hilbert's 10th Problem again

Decision problem:

Let $P(x_1, \ldots, x_n)$ be a polynomial in n variables
$\exists (x_1, \ldots, x_n) \in Z^n$: $P(x_1, \ldots, x_n) = 0$?

Express computation histories as sequences ($x \in Z^n$) of numbers and the statement "x encodes an accepting history for M on w" by a polynomial equation "$P_{Mw}(x) = 0$".

Then, deciding "$\langle M,w\rangle \in A_{TM}$?" is equivalent with deciding "$\exists x \in Z^n$: $P_{Mw}(x) = 0$?"

Context-Free Languages

Remember the language
$EQ_{CFG} = \{\langle G_1, G_2\rangle \mid G_1, G_2 \text{ CFGs } L(G_1)=L(G_2)\}$?

We can prove the undecidability of this language with the use of computation histories.

Goal: Linking A_{TM} with EQ_{CFG}

This will require some careful steps.

Initial Attempt

$\langle M,w\rangle \in A_{TM}$ equals
"There is an accepting history x of M on w."

Let's try to express this as:
$\langle x\rangle \in L_{CFG}$, where L_{CFG} is some CF language.

Problem: "$\langle x\rangle \in L_{CFG}$?" is decidable…

Second Attempt

$\langle M,w\rangle \notin A_{TM}$
equals
"There is no accepting history x of M on w."
equals
"All histories x are non-accepting for M on w"

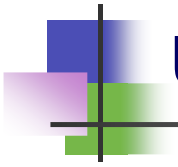
Let's try to express this as:
"All $\langle x\rangle \in L_{CFG}$",
where L_{CFG} is some CF language that contains all descriptions of non-accepting histories of M on w

A Specific CFG

Given a Turing machine M and input w, we have to make a context-free grammar G such that:
$x \in G$ if and only if x does not encode an accepting history of M on w

Let x encode the history $C_1,\ldots,C_k$, then $x \in G$ if
1) C_1 not proper starting configuration, or
2) There is an improper $C_j \rightarrow C_{j+1}$ transition, or
3) C_k is not an accepting configuration.

We can do this (CFL are closed under 'or').

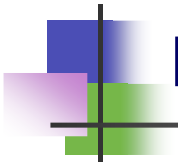


Undecidability CFG Properties

With the previous outline, we can prove that deciding the language
ALL_{CFG} = {<G> | G is CFG, L(G)=Σ* }
enables us to decide the undecidable A_{TM}

Theorem: ALL_{CFG} is undecidable

Corollary: The language is EQ_{CFG} undecidable.
Proof: Take G_* such that $L(G_*)= \Sigma^*$, and ask
"$<G_1,G_*> \in EQ_{CFG}$?"



Mapping Reducibility

Thus far, we used reductions informally:
If "knowing how to solve A" implied "knowing how to solve B", then we had a reduction from B to A.

Sometimes we had to negate the answer to the "$\in$ A?" question, sometimes not. In general, it was unspecified which transformations were allowed around the "$\in$ A?"-part of the reduction.

Here now comes rigor…

Computable Functions

A function $f:\Sigma^* \to \Sigma^*$ is a TM-computable function if there is a Turing machine that on every input $w \in \Sigma^*$ halts with just f(w) on the tape.

All the usual computations (addition, multiplication, sorting, minimization, etc.) are all TM-computable.

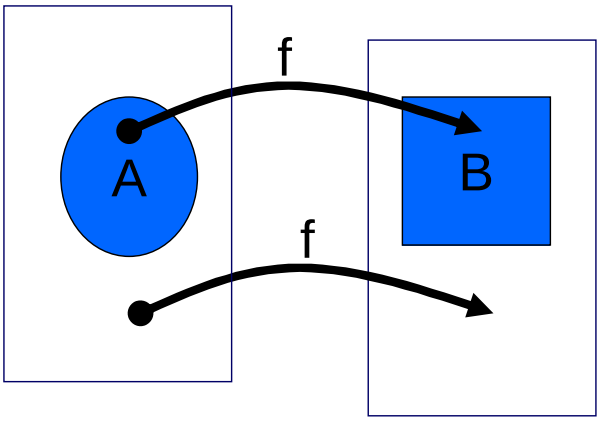
Important here is that alternations to TMs, like "given a TM M, we can make an M' such that…" can also be described by computable functions that thus have f(<M>) = <M'>.

Mapping Reducible

A language A is mapping reducible to a another language B if there is a TM-computable function $f:\Sigma^* \to \Sigma^*$ such that: **$w \in A \Longleftrightarrow f(w) \in B$** for every $w \in \Sigma^*$.

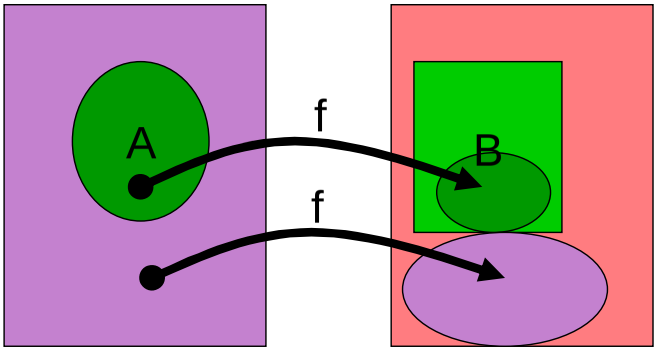
Terminology/notation:

- $A \leq_m B$
- function f is the reduction of A to B
- also called: "*many-one reducible*"



$A \leq_m B$

The language B can be more difficult than A.



Typically, the image f(A) is only a subset of B, and f(Σ*\A) a subset of Σ*\B.

"Image f(A) can be the easy part of B".

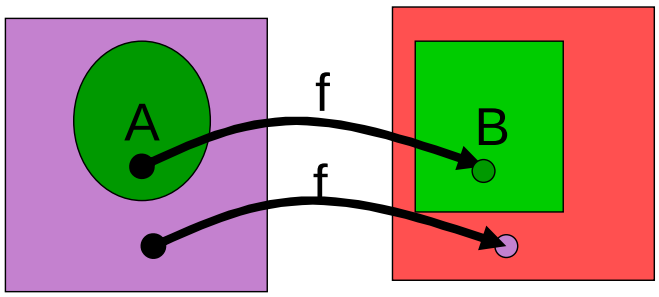
Decidable $A \leq_m B$

If A is a decidable language, then $A \leq_m B$ for every nontrivial B. (Let $1 \in B$ and $0 \notin B$.)

Because A is decidable, there exists a TM M such that M outputs "accept" on every $x \in A$, and "reject" on $x \notin A$.

We can use this M for a TM-computable function f with

$f(x)=1 \in B$ if $x \in A$ and $f(x)=0 \notin B$ if $x \notin A$



"The function f does all the decision-work"

Decidability obeys $\leq_m$ Ordering

Theorem: If $A \leq_m B$ and B is TM-decidable,
then A is TM-decidable.
Proof: Let M be the TM that decides B and f the reducing function from A to B. Consider the TM:
On input w:
1) Compute f(w)
2) Run M on f(w) and give the same output.

By definition of f: if $w \in A$ then $f(w) \in B$.
M "accepts" f(w) if $w \in A$, and
M "rejects" f(w) if $w \notin A$.

Undecidability obeys $\leq_m$ Order

Corollary: If $A \leq_m B$ and A is undecidable,
then B is undecidable as well.
Proof: Language A undecidable and B decidable contradicts the previous theorem.

Extra: If $A \leq_m B$, then also for the complements
$(\Sigma^* \backslash A) \leq_m (\Sigma^* \backslash B)$
Proof: Let f be the reducing function of A to B with $w \in A \Longleftrightarrow f(w) \in B$. This same computable function also obeys "$v \in (\Sigma^* \backslash A) \Longleftrightarrow f(v) \in (\Sigma^* \backslash B)$" for all $v \in \Sigma^*$

Recognizability and $\leq_m$

Theorem: If $A \leq_m B$ and B is TM-recognizable, then A is TM-recognizable.
Proof: Let M be the TM that recognizes B and f the reducing function from A to B. Again the TM:
On input w:
1) Compute f(w)
2) Simulate M on f(w) and give the same result.

By definition of f: $w \in A$ equivalent with $f(w) \in B$.
M "accepts" f(w) if $w \in A$, and
M "rejects" f(w)/does not halt on f(w) if $w \notin A$.

Unrecognizability and $\leq_m$

Corollary: If $A \leq_m B$ and A is not Turing-recognizable, then B is not recognizable as well.
Proof: Language A not TM-recognizable and B recognizable contradicts the previous theorem.

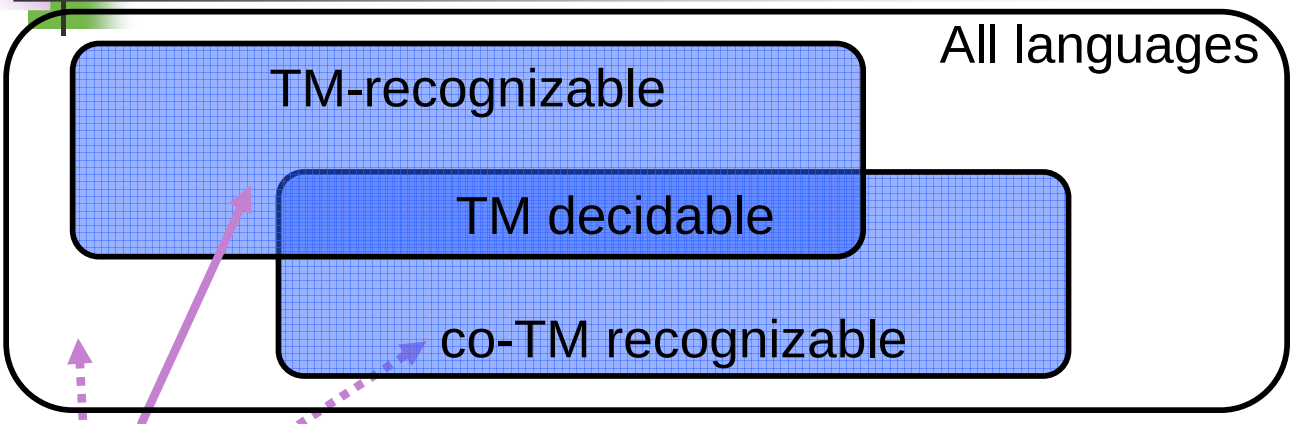
Extra: If $A \leq_m B$ and A is not co-TM recognizable, then B is not co-Turing-recognizable as well.
Proof: If A is not co-TM-recognizable, then the complement ($\Sigma^* \backslash A$) is not TM recognizable.
By $A \leq_m B$ we also know that $(\Sigma^* \backslash A) \leq_m (\Sigma^* \backslash B)$.
Previous corollary: ($\Sigma^* \backslash B$) not TM recognizable, hence B not co-Turing-recognizable .

An Old Result

Ad nauseam: The emptiness language
$E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM with } L(M)=\emptyset \}$
is not Turing recognizable.
Simple proof via ($\bar{A}_{TM} \leq_m E_{TM}$):
Let f on input <M,w> give <M'> as output with:
M': Ignore input
Run M on w
If M accepted w then "accept"
otherwise "reject"

Now: $\langle M,w \rangle \in \bar{A}_{TM} \Longleftrightarrow f(\langle M,w \rangle) = \langle M' \rangle \in E_{TM}$

Something is still missing ...



$A_{TM} = \{ \langle M,w \rangle \mid M \text{ is a TM that accepts } w \}$

$E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM with } L(M)=\emptyset \}$

$EQ_{TM} = \{ \langle G,H \rangle \mid G,H \text{ TMs with } L(G)=L(H) \}$

EQ_{TM} is not TM Recognizable

Proof (by showing $\bar{A}_{TM} \leq_m EQ_{TM}$):

Let f on input <M,w> give $<M_1,M_2>$ as output with:
M_1: "reject" on all inputs
M_2: Ignore input
Run M on w
"accept" if M accepted w

We see that with this TM-computable f:
$<M,w> \in \bar{A}_{TM} \Longleftrightarrow f(<M,w>) = <M_1,M_2> \in EQ_{TM}$

Because $\bar{A}_{TM}$ is not recognizable, so is EQ_{TM}.

EQ_{TM} is not co-TM Recognizable

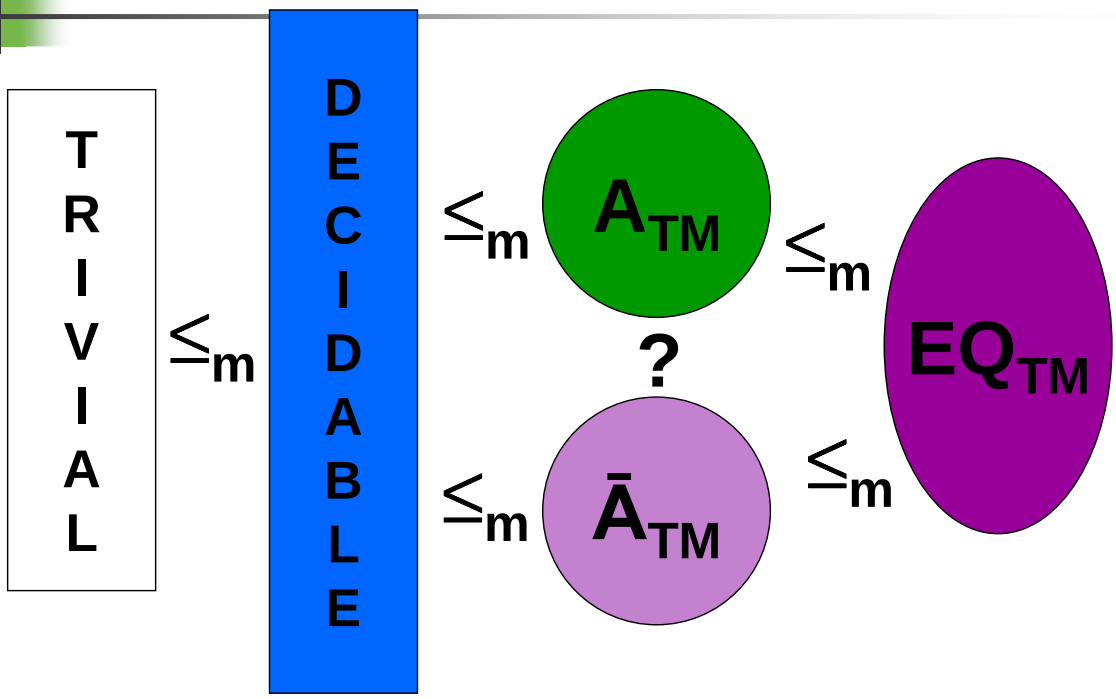
Proof (by showing $A_{TM} \leq_m EQ_{TM}$):

Let f on input <M,w> give $<M_1,M_2>$ as output with:
M_1: "accept" on all inputs
M_2: Ignore input
Run M on w
"accept" if M accepted w

We see that with this TM-computable f:
$<M,w> \in A_{TM} \Longleftrightarrow f(<M,w>) = <M_1,M_2> \in EQ_{TM}$

Because A_{TM} is not co-recognizable, so is EQ_{TM}.

Partial $\leq_m$ Ordering



About first order logic

First Order Logic

First Order Logic (FOL)

- extends propositional logic
- a FOL contains
  - constants: a,b,c… denote objects in asome domains
  - variables: x,y,… they can assume the value of an object of the domain
  - the existential quantifier: $\exists x$ stands for x exists …
  - the universal quantifier: $\forall x$ stands for each x **…**
  - predicates: P(x,y) is a fact about x and y that may be true or false
  - functions: f(x) is another object determined as function of x
  - operators of propositional logic: $\neg$, $\wedge$,$\vee$, $\rightarrow$
- examples of sentences in FOL
  - $\exists x \forall y \neg(P(x,y) \wedge Q(x,y))$
  - $\forall x \forall y\ P(x,y) \rightarrow Q(f(x),f(y))$

Truth in FOL

- A FOL sentence is true (valid) or not true (valid) with respect to a model and an interpretation
  - a model is a set of objects
  - an interpretation specifies the meaning of constant, variables, functions and predicates in the model
  - f.i. $\forall x \forall y\ y*(x+1)=y*x+y$, when the model is the set of integers and the predicate = and functions *, are given the usual interpretation
- logical validity
  - is a sentence is logically valid if it is true in every model and interpretation,
    f.i. $\forall x \forall y \neg P(x,y) \vee P(x,y)$, $\exists x \neg Q(x) \rightarrow \neg \forall x \neg Q(y)$,
  - a sentence true for some model and interpretation is called satisfiable, otherwise it is called unsatisfiable

Axioms, Inferences rules, theories

A theory contains inference rules and hypotheses

Hypotheses

- are sentences that are assumed to be true in the considered domains and interpretations

Inference rules

- relates a fixed number sentence (premises) to a sentence (conclusion)
- are used to prove other sentences (theorems)!!

Completeness and soundness

For a theory with hypotheses *H* and rules *R* let us denote

- H |= *A*
  if the sentence *A* is true for all the models and interpretations where *H* is true
- H $|-_R$ *A*
  if the sentence *A* can be deduced from *H* by the rules *R*

A theory is

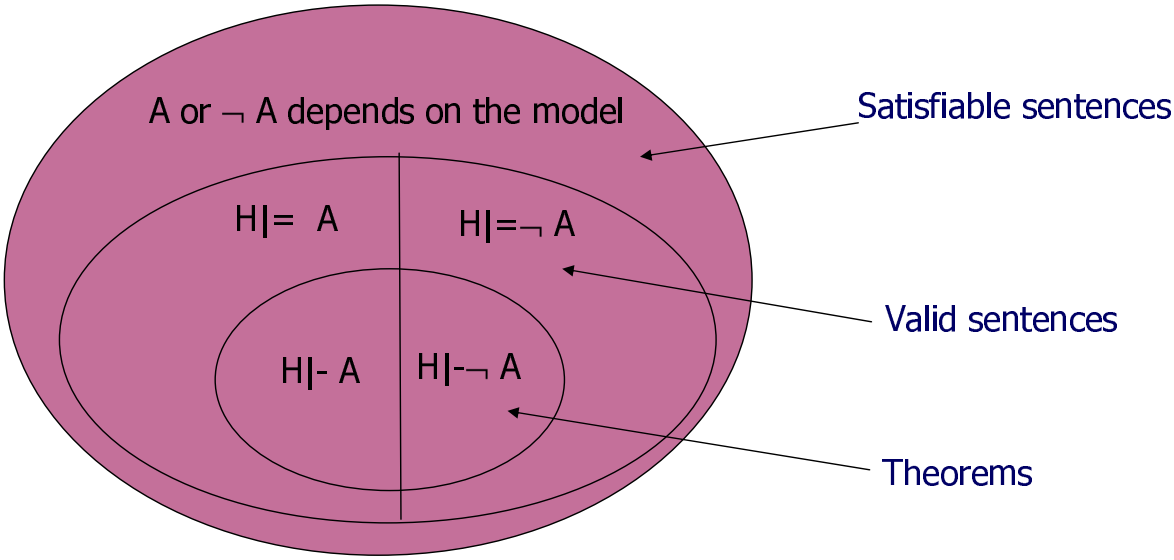
- sound, if H $|-_R$ *A* implies H |= *A*
- complete, if H |=*A* implies H $|-_R$ *A*

Of course,

- soundness is essential
- completeness is very desired ….!!!

Valid, unsatisfiable, satisfiable, sentences

When a theory is sound



Logical axioms

Consider the general FOL where only the logical symbols are defined

- only one rule exists,

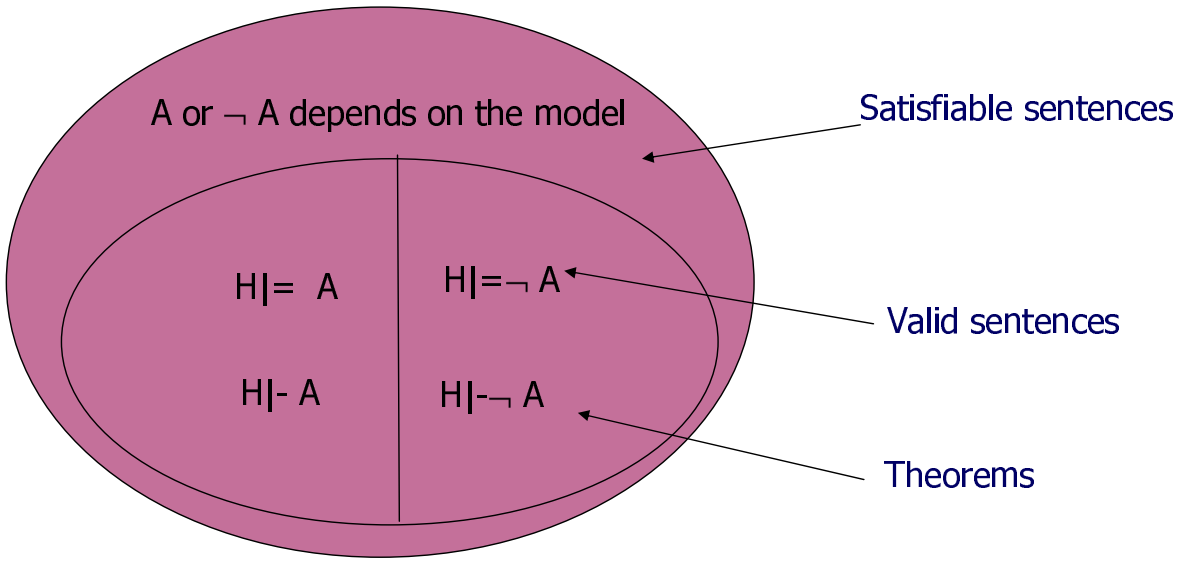
  if A→B and A are true, then deduce B
- only the (logical) axioms are considered, where t,r,s are terms
  - $(\forall x\ P(x)) \rightarrow P(t)$
  - $P(t) \rightarrow (\exists x\ P(x))$
  - $(\forall\ x\ (W \rightarrow P(x))) \rightarrow (W \rightarrow \forall\ x\ P(x))$
  - $(\forall\ x\ (P(x) \rightarrow W)) \rightarrow (\exists\ x\ P(x) \rightarrow W)$
  - $t \rightarrow (r \rightarrow s)$, $t \wedge r \rightarrow t$,.. and other axioms of propositional logic

Gödel completeness theorem states that

The above FOL theory is complete and sound!!

Valid, unsatisfiable, satisfiable, sentences in general FOL

When a theory is sound and complete



FOL and calculability

Let us consider the sets

- the theorems T={A| |- A} of the theory
- the valid sentences V={A| |= A} of the theory

The following holds

- T is recursively enumerable
- since, T=V, V is recursively enumerable, too
- We will see that T, V are not decidable

What does it happen when the theory has also non-logical hypothesis?

The theory of natural numbers is undecidable

Let us consider the theory of natural numbers with the operators + and *

$\forall q\ \exists p\ \forall x,y\ [\ p>q\ \&\ (x,y>1 \rightarrow xy \neq p)\ ]$
$\forall a,b,c,n\ [\ (a,b,c>0\ \&\ a^n+b^n=c^n) \rightarrow n \leq 2\ ]$
$\forall q\ \exists p\ \forall x,y\ [\ p>q\ \&\ (x,y>1 \rightarrow (xy \neq p\ \&\ xy \neq p+2))\ ]$

Let Th(N,+,*) denote the set of all valid sentences in such a theory

- Th(N,+,*) is undecidable!!

Notice that we have not defined a set of hypotheses and derivations rules, so we can simply assume that the hypotheses are just all the true sentences in natural number mathematics

Proof

Sketch:

For each TM M, it is possible (not described here) to define a sentence $\exists x\ R_{M,w,}$ in the FOL of the natural numbers with + and *, that is true if x, which is free in $R_{M,w}$, is a correct accepting history of w on M.

Thus, let M_2 be a machine that decides Th(N,+,*) . Then, we can define a TM machine M_3 that takes in input <M,w>, constructs $\exists x\ R_{M,w}$ and call M_2 on it. M_3 decides A_{TM}, which is absurd.

Gödel's incompleteness theorem

It is even worse:
natural number theory cannot be axiomatized

Gödel's incompleteness theorem

- No sound and complete set of axioms for the theory of natural number is recursively enumerable

Proof

Sketch:

Let us suppose, by reduction to absurd reasoning, that there is a recursively enumerable set of axioms for integer number theory.

Then the valid sentences of the theory are recursively enumerable. Since, the for each sentence A, either A or $\neg$ A is valid, then valid sentences are also decidable.

This fact is absurd due to the undecidability of Th(N,+,*).

Validity is undecidable

Using a similar approach as the one used for natural number, it is possible to prove that

- The set of valid sentences of basic FOL is undecidable

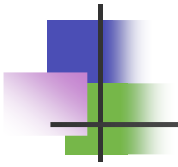
Notice the difference with natural number theory

- In natural number theory, for any sentence A, either A or $\neg$A is true, but there is no set of axioms that allow us to decide which one holds
- In general FOL, if A is valid (a theorem) A, we can decide whether either A or $\neg$A holds, but we cannot decide whether a sentence is valid!!

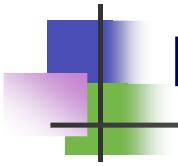
Particular results

Special results may hold for special theories, f.i

- Th(N,+) is a TM-decidable set
- Th(N,+,$\times$) is a not a TM-recognizable set
- Th(R,+,$\times$) is TM-decidable
- monadic predicate logic (predicates with only one parameter) is decidable
- validity of bounded sentences (f.i. $\exists x(x<3)\ldots$) is decidable



Minimum description length

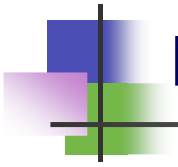


Measuring Information

Standard information theory considers each n bit-string in $\{0,1\}^n$ equal.

However, we feel a difference between the string "0101010101010101010101010101010101" and the string (of equal length) "0101110101001010001110001100011011".

We consider the first string more 'regular' than the second one.



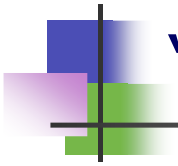
Regularity

We can give a short description of "0101010101 01010101010101010101010101" by "17 times 01".

For the other "01011101010010100011100011 00011011" this seems more problematic.

This suggests:

A regular string is a string that has a short description; an irregular one has no such summary.



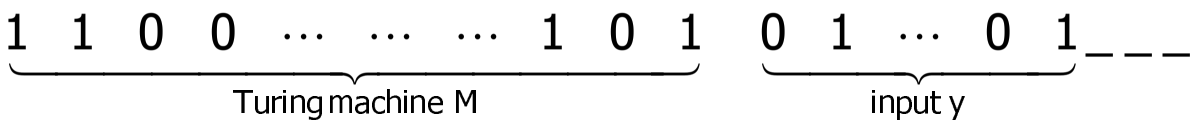
'Turing Describable'

Key idea: We allow a Turing machine M with input y as a description of a string x if the output of M on y equals x.

An x will have many different descriptions (M,y). We consider the size of the shortest description as an indication of the *intrinsic complexity* of x.

An Encoding for <M,y>

The description of the TM M and its input y is going to be one long bit-string:



How do we know where <M> stops and <y> starts?

We will use a *self-delimiting code* for <M>:
two bits “00” for ‘zero’,
two bits “11” for ‘one’, and
“01” for ‘end of string’.

Description Length of <M,y>

For the encoding of M and x we concatenate the self-delimiting/double bit description of M with y.

Hence from now on: <M,y> = <M>y.

For the length of <M,y> this implies:
|<M,y>| = |<M>| + |y|

Note that the $y \in \{0,1\}^*$ is encoded trivially.

Descriptive Complexity of x

(Fix a universal Turing machine U.)

The descriptive complexity K(x) of a string x is the length |d(x)| of its minimal description:

$$K(x) = \min_{<M>y} \left\{ \left| \langle M \rangle y \right| \ : \ U \text{ on } M \text{ and } y \text{ outputs } x \right\}$$

Also known as: algorithmic complexity, or Kolmogorov (Solomonoff-Chaitin) complexity.

Description length and information theory

Let us find a suitable coding to compress the information …

- Messages has to be sent through a communication channel.
- The messages are sequences of symbols in the alphabet $S=\{s_1,\ldots,s_N\}$..

Description length theory point of view

- To codify a message we send
  - a description of the alphabet S,
    it has a fix cost C_S
  - the index of each element of the alphabet
    each index can be codified using log(|S|) bits
- Hence $K(x) \leq c_S + m \log(|S|)$,

Description length and information theory

Shannon source coding theorem states

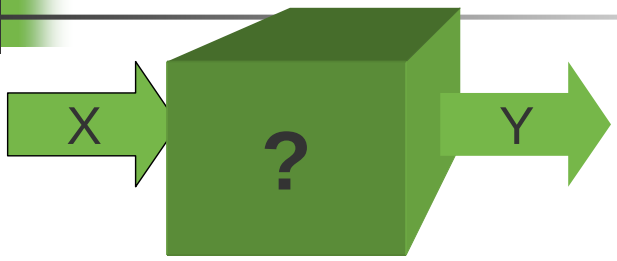
- that the optimal code satisfies

$$H(S) \leq EB(S) \leq H(S)+1$$

where H(S) is the entropy of alphabet S and EB(S) is the average number of bits used to codify the alphabet

- thus the average minimum description length EK(x) of a message is EK(x) ≤m (H(S)+1)

Description length and model selection



We want to define a model that explains a set of observations $X_1,...,X_n$ and $Y_1,...,Y_n$ of a system. If several models are available, which one has to be selected?

The goals

- the model should be able to explain well the observations
- the model should be simple

Get the model such that

- <M> is minimum
- M(X)=Y

Occam's Razor

"It is vain to do with more what can be done with fewer."

William of Ockham (1290–1349)

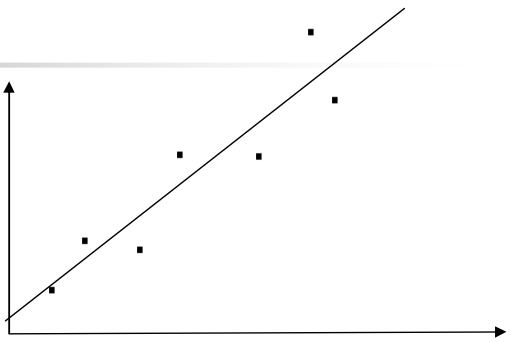
Alternatively:

"Entities should not be multiplied beyond necessity."

Description length and model selection

An observation X, Y of a system can be defined

- using a model M
- the difference between the predicted output and the actual output D=Y-M(X)



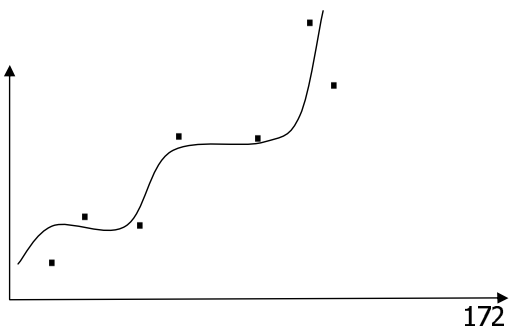
The goals

- the model should be able to explain well the observations
- the model should be simple

Get the model such that

- <M>D is minimum

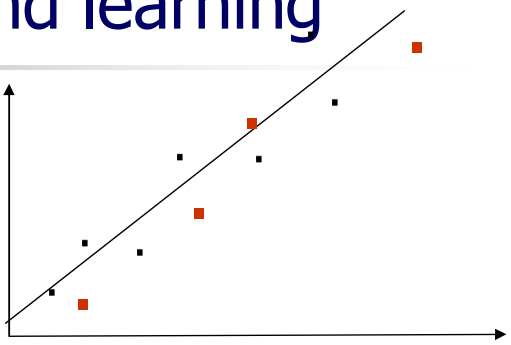
Which model is better ?



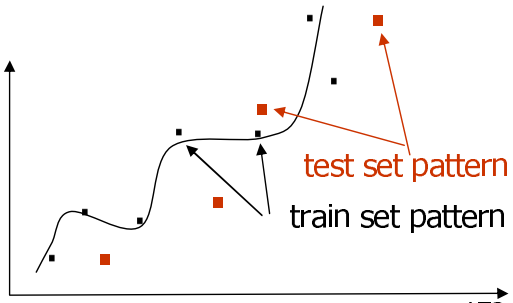
Description length and learning

In machine learning

- the generalization is the capability of a trained model to predict the behavior of a system on unseen patterns
- Usually,
  - the simpler a model is, the better generalization can be achieved
  - the simpler a model is, a lower performance is achieved on training set
- The description length may be adopted as a tool for improving generalization performance



Which model is better ?



Example: description length and neural networks

Assumption

- A source communicates to a receiver the model of a system. Source and receiver know the input data $X_1,\ldots,X_n$.
- The source sends
  - a neural network M

    its encoding <M> requires to encode each weight by a binary number
  - the difference between the predicted and the actual output for each input data

    the difference D it is encoded by a binary number

Example: description length and neural networks

The minimum description length theory suggests to minimize <M>D (an intuitive sketch)

- To minimize <M>

  The weights of the neural network must be kept small, in order to minimize the precision required for its representation

- To minimize D

  the expected difference between the predicted output and the actual output must be kept small, in order to minimize the precision required for its representation

Thus we have to minimize

$$\alpha \sum_{i=1}^{n} \log\left(\left|Y_i - M(X_i)\right|\right) + \beta \sum_{i=1}^{m} \log\left(\left|w_i\right|\right)$$

How Universal is K?

Recall: (Fix a universal Turing machine U.)

$$K(x) = \min_{<M>y} \left\{ \left| \langle M \rangle y \right| \ : \ U \text{ on } M \text{ and } y \text{ outputs } x \right\}$$

Problem: The function K depends on the universal U that is used: we should say K_U instead of K…

Maybe that for another TM V, the complexity measure K_V is much smaller than K_U?

Invariance Theorem

Theorem: Let U be a universal TM, then
for any other description method V, we have

$K_U(x) \leq K_V(x) + c$ for all strings x.

Note that the constant c depends on V and U,
but not on x.

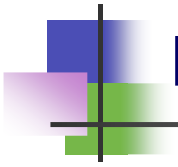
Proof: Because U is universal, we can give a
finite description to U how it should simulate V.
Let this description be of size c.

An Obvious First Result

Theorem: There exists a constant c, such
that $K(x) \leq |x| + c$, for every x. ("The complexity of
a string can never be much bigger than its length.")

Proof: Let M be the TM that simply outputs its
input string y: M(y)=y.
Then <M>x is a description of x, and hence
$K(x) \leq |\langle M \rangle| + |x|$. Let $c=|\langle M \rangle|$.

(Here we benefit from our way of encoding (M,y).



Data Compression

Theorem: There is a constant c such that
$K(xx) \leq K(x) + c$, for every string x.

Proof: Take the TM M that given input <N>x:
1) Calculate the output s of N on x
2) Output ss

Let d(x) be the minimum description <Q>r of x,
then <M>d(x) will give a description of xx.
Hence, $K(xx) \leq |\langle M \rangle| + |d(x)| = K(x) + c$.

Concatenation

You would expect that for all strings x and y:
$K(xy) \leq K(x) + K(y) + c$, for some c.
However, this is not true.

The problem lies –again– in the separation
between d(x) and d(y).

Instead, we have a constant c such that:
$K(xy) \leq K(x) + K(y) + 2\log(K(x)) + c$,
for all strings x and y.

Log Cost of Concatenation

Theorem: There is a c such that
$K(xy) \leq K(x) + K(y) + 2\log(K(x)) + c$, for all x,y.

Proof: Let m be the logarithm of |d(x)|, then the string "1^m0 <|d(x)|> d(x)" gives a self-delimiting description of x.
(We need 2m bits to indicate the length of d(x).)
Hence the input "1^m0 <|d(x)|> d(x) d(y)" gives an unambiguous description of xy.

Alternatively, we could double the representation of x, obtaining a less tight bound $K(xy) \leq 2K(x) + K(y) + c$

Incompressibility

Theorem: For every n there exists at least one incompressible string $x \in \{0,1\}^n$ with $K(x) \geq n$.

Proof:
– There are 2^n different strings x in $\{0,1\}^n$.
– There is one description of length 0, two descriptions of length 1,…, and 2^{n-1} descriptions of length n–1.
In total: 2^n–1 descriptions of length smaller than n.
Hence, there has to be an $x \in \{0,1\}^n$ that has a minimal description of at least n bits.

Incompressibility: more geneally

Theorem: For every n there exists at least $2^n-2^{n-c+1}+1$ strings that are incompressible by c, i.e. such that satisfy $k(x)>|x|-c$

Proof: As in the previous theorem, there is only $2^{n-c+1}-1$ descriptions (strings) having length smaller or equal to n-c

Counting Primes less than n

Q: How many primes are there less than n?
Let $p_1,\ldots,p_m$ be the m primes $\leq n$.
We know that we can describe n by:

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_m^{e_m}$$

Hence $<e_1,\ldots,e_m>$ gives a description of n.

- For each j we have: $e_j \leq \log(n)$.Thus $<e_1,\ldots,e_m>$ requires less than $m \cdot \log(\log(n))$ bits.
- By, incompressibility, there are n with $K(n) \geq \log(n)$.

Conclusion: $m \geq \log(n) / \log(\log(n))$ for those n.

More Carefully Counting of Primes

$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_m^{e_m}$. Hence <$e_1$,…,$e_m$> gives a description of N. For each j we have: $e_j \leq \log(n)$.

- There is an encoding y_n of e_1,…,e_m with $|y_n| \leq m \cdot \log(\log(n))$. The total description <M>y_n requires no more than $c + m \cdot \log(\log(n))$ bits.
- For n with $K(n) \geq \log(n)$, we thus have the bound: $\log(n) \leq m \cdot \log(\log(n)) + c$, which implies

$$m \geq \log(n)/\log(\log(n)) - c/\log(\log(n))$$

for arbitrary big n.

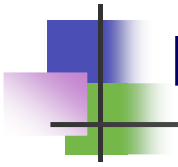
(Un)computability of K

To which extent can we know the value K(x)?

- $K(x) \leq n$ can be proven by a specific example <M>y (with total length $\leq n$) that produces x.
- For $K(x) \geq n$ we would have to prove that all <M>y (with length < n) do not produce x…

K can only be approximated 'from above'.

True statements like "$K(x) \leq n$" are recognizable, but not TM-decidable.

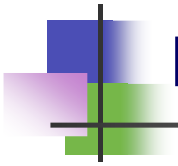


Richard-Berry Paradox

"The least number that cannot be defined in fewer than twenty words."

By formalizing this paradox we will see that the problem lies in recognizing that a number *cannot* be described in fewer than 20 words.

(There is no problem with formalizing "defined".)

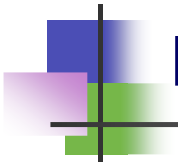


Richard-Berry Formalized

Richard-Berry Paradox Theorem:
The set C = { <x,n> | K(x) ≥ n } is not decidable.

Proof by contradiction: Assume C decidable.
Consider the following TM M (on input n):
1) Take s=ε
2) Decide <s,n> ∈ C?
3) If answer "no",
   then increase s lexicographically; go to 2)
4) If answer "yes", then print s and halt.

The descriptive length of <M>n is c_M+log(n)…



Richard-Berry Formalized

… The descriptive length of <M>n is c_M+log(n).
Take n big enough such that n > c_M+log(n).
(The original paradox has n equal "20 words".)

Then the length of <M>n is less than n, but it outputs/describes a string s with K(s) ≥ n.
Contradiction.

The set { <x,n> | K(x) ≥ n } is not decidable.
(But it is co-TM recognizable.)

Compression and Gödel (1)

The impossibility of calculating K gives a simple way of rephrasing Gödel's incompleteness thm.

Let A be an attempt to find the complete set of axioms and derivation rules for Th(N,+,×).

There exists an n_A which is a constant that depends only on the size of A in bits, such that…
… with A we *cannot* prove *any* of the true statements "K(x) > n_A".

Compression and Gödel (2)

For any string x, the statement "K(x) > n" can be expressed as a potential element of Th(N,+,×).

(We can formalize "K(x) > n" in the language of the theory of natural numbers, +, and ×.)

Consider the following program that uses A and n:
1) Enumerate a statements that follows from A
2) If this statement is of the form "K(x) > n",
   then print(x) and halt
3) Otherwise: generate next statement and go to 2)

Compression and Gödel (3)

1) Enumerate a statements that follows from A
2) If this statement is of the form "K(x) > n",
   then print(x) and stop
3) Otherwise: generate next statement and go to 2)

The above program can be expressed with less than 2|A| + 2log(n) + c bits.
(The constant c does not depend on A or n.)

Also, the program outputs a string x with K(x)>n.

Contradiction if n > 2|A| + 2log(n) + c.

Compression and Gödel (4)

Summary:
Consider a system of axioms/derivations A.
Take an n_A with $n_A > 2|A| + 2\log(n_A) + c$.
This program of length $2|A| + 2\log(n_A) + c$:

1) Enumerate a statements that follows from A
2) If this statement is of the form "$K(x) > n_A$",
   then print(x) and stop
3) Otherwise: generate next statement and go to 2)

would print an x with $K(x) > 2|A| + 2\log(n_A) + c$.

Contradiction: A cannot prove $K(x) > n_A$.

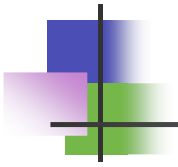
Compression and Gödel (5)

Summary:
Consider a system of axioms/derivations A.
Take an n_A with $n_A > 2|A| + 2\log(n_A) + c$.

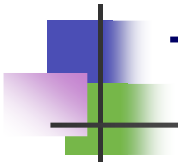
With A we cannot prove "$K(x) > n_A$" for any x.

This is a very strong result:
- For all but $2^{n_A+1}-1$ strings, "$K(x) > n_A$" is true.
- The statement "$K(x) > n_A$" does not use the content of A at all, only the size |A|.

"You can't prove a theorem of one kilogram with only one gram of axioms."



Time complexity



Time Complexity

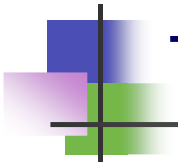
Let M be a Turing machine that halts on all inputs.

Definition: The running time or time complexity of M is the function f:N→N, defined by

$$f(n) = \max_{|x|=n}(\text{no. of time steps of M on x})$$

Note: Worst case and size of the input x in bits.

- *"f(n) is the running time of M"*
- *"M is an f(n) time Turing machine"*.



Time Complexity Classes

Definition: For the function $t:N \rightarrow N$,
the time complexity class TIME(t(n)) is the set of decision problems:

TIME(t(n)) = { L | there is a TM that decides
the language L in time O(t(n)) }

The Polynomial Time Class

Definition: The class of languages that can be decided by a single tape TM in polynomial time is denoted by **P**:

$$\mathbf{P} = \bigcup_{k=1,2,\ldots} \mathrm{TIME}(n^k)$$

The problems in P are considered efficiently solvable.

Examples

Problems in P

- given two nodes of a graph, is there path that connect them?

$$PATH = \{(G, n, v) \mid \text{there is a path from n to v in G}\}$$

- given two integer numbers, are they relatively prime?

$$RELPRIME = \{(a, b) \mid \text{a and b are relatively prime}\}$$

- given a string and a predefined context free language, does the string belongs to the language?

A_{CFG} = { <G,w> | G is a CFG that generates w }

- ...

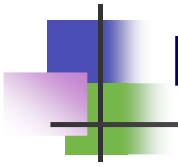
The Complement Polynomial Time Class

Definition: The class coP contains the languages whose complement can be decided in polynomial time

$$\mathbf{coP} = \{A \mid \overline{A} \in \mathrm{P}\}$$

Theorem: The complement polynomial class is equal to the polynomial class

$$\mathbf{coP} = \mathbf{P}$$

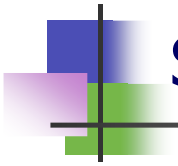


Proof

Proof

Let M be the DTM that decide on A. Build M2 such that

- M is run on the input w
- If M accepts, then reject
- If M rejects, then accept



Strong Church-Turing Thesis

All reasonable computational models are polynomial-time/space equivalent:

- It is possible to simulate one model by another model with only polynomial time/space overhead.

The answer to the question "A∈P?" does not depend on the model.

Relationships between Turing machines

The computation time on different Turing machines differs for only a polynomial time term

Theorem: Every t(n) time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing Machine

Theorem: Every t(n) time single tape Turing machine has an equivalent O(t(n)) RAM program

Theorem: Every t(n) time RAM program has an equivalent $O(t^3(n))$ single tape Turing machine

Proving that a multi-tape TM is equivalent to a single tape TM

Let $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$ be a k-tape TM.
Construct 1-tape M' with expanded $\Gamma' = \Gamma \cup \underline{\Gamma} \cup \{\#\}$

Represent M-configuration

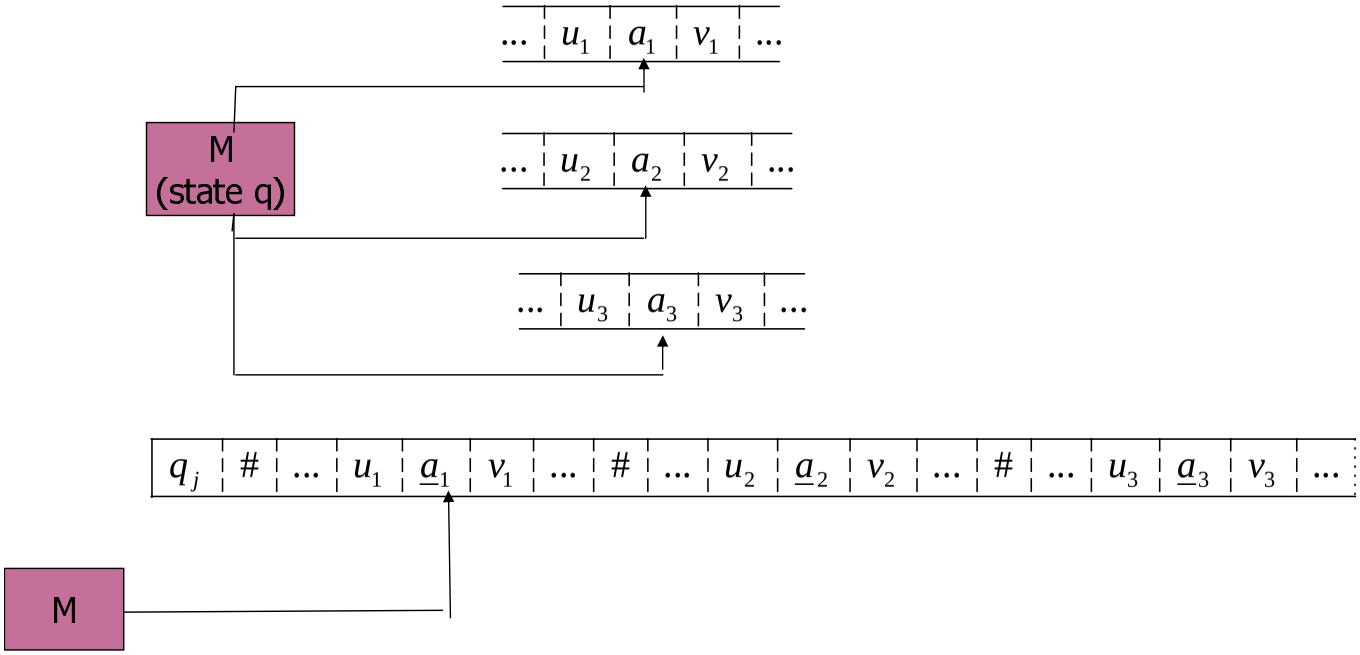
$$u_1 q_j a_1 v_1, \quad u_2 q_j a_2 v_2, \quad \ldots, \quad u_k q_j a_k v_k$$

by M' configuration

$$q_j \# u_1 \underline{a}_1 v_1 \# u_2 \underline{a}_2 v_2 \# \ldots \# u_k \underline{a}_k v_k$$

(The tapes are seperated by #, the head positions are marked by underlined letters.)

Proving that a multi-tape TM is equivalent to a single tape TM



Proving that a multi-tape TM is equivalent to a single tape TM

On input $w=w_1 \ldots w_n$, the TM M' does the following:

1. Prepare initial string: $\#\underline{w}_1 \ldots w_n \# \_ \# \cdots \# \_ \# \_ \cdots$
2. Read the underlined input letters $\in \Gamma^k$
3. Simulate M by updating the input and the underlining of the head-positions.
4. Repeat 2-3 until M has reached a halting state
5. Halt accordingly.

PS: If the update requires overwriting a # symbol, then shift the part $\# \cdots \_$ one position to the right.

Proving that a multi-tape TM is equivalent to a single tape TM

- The simulation of an operation of the multi-tape operation can cost at most m, where m is the dimension of the tape
- $m \leq t(n)$ holds, since the multi-tape TM cannot write more than a symbol at every operation
- thus, the number of operation runs by the single tape TM is smaller than $O(m\ t(n)) \leq O(t^2(n))$

Unreasonable Models?

Physical unrealistic requirements for the proper functioning of the machine.

- Machines using elementary components with infinite precision
- Unbounded parallel computing
- Machines that requires exponential space and energy.
- Time-travel computing

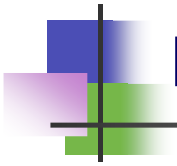
Verifiers and NP

Languages in P have poly-time deciders.
Languages in NP have poly-time 'verifiers'.

Definition: A verifier for a language A is a program V such that
A = {<w> | V accepts <w,c> for some c}

The string c is the certificate that proves w∈A.

Remark
- the length of c is polynomial w.r.t. w



Nondeterministic Polynomial Time

Definition: NP is the class of languages that have polynomial time verifiers.

The following is another equivalent definition

Definition: A language L is in NP if and only if L can be decided by a poly-time by a nondeterministic TM.

The problems in NP are considered difficult to be solved.

Proof: the two definitions are equivalent

Proof

Let $A \in NP$ have an $O(n^k)$ time verifier V.

A polynomial time NTM can guess the $O(n^k)$ certificate c that V has to use for $x \in A$ and simulate V on <x,c>.

Let N be the $O(n^k)$ time nondeter. decider for B.

The $O(n^k)$ guesses of N define a certificate c.

A polytime deterministic V can simulate N on <x,c> and verify "$x \in B$" for every such x.

A problem in NP: Boolean Formula Satisfiability (SAT)

A Boolean variable x can be TRUE or FALSE, which is also denoted by "1" or "0".

Standard Boolean operations are AND ($x \wedge y$), OR ($x \vee y$), and NOT ($\neg x$ or also $\bar{x}$).

Typical Boolean formula: $\phi(x,y) = (\neg x \vee y) \wedge (x \vee \neg y)$. This ϕ is satisfiable (by the assignments x=y=TRUE and x=y=FALSE).

SAT = {<ϕ> | ϕ is a satisfiable Boolean formula}

SAT is in NP:

- a certificate is a assignment of the variables that satisfies the formula.
- the certificate length is proportional to the number of variables

Examples of problems in NP

Example

- does a given graph contains a clique?

$$CLIQUE = \{(G,k) \mid \text{G contains a k - clique}\}$$

the certificate is a coding of the clique nodes

- is there a subset of integers in a given set that sum to a given number?

$$SUBSET - SUM = \left\{ (\text{S},\text{t}) \mid S = \{x_1,..,x_n\},\ \text{it exists } Y \subseteq S \text{ such that } t = \sum_{i \in Y} x_i \right\}$$

the certificate is a coding of the subset

Winning $1,000,000

- The Clay Mathematics Institute named seven open problems in mathematics the Millennium Problems
  - Anyone who solves any of these problems will receive $1,000,000
  - Proving whether or not P equals NP is one of these problems

About P=NP

- It is believed that P≠NP
- Several theorems in complexity theory are in the form of
  - if P≠NP then …
- There is an incredibly large number of open problems whose solution would allow to decide about P=NP
- Important cryptography algorithms are based on P≠NP

co-NP

- Definition: The class coNP contains the languages whose complement can be decided in polynomial time on a non deterministic TM

$$\textbf{coNP} = \left\{ A \mid \overline{A} \in \text{NP} \right\}$$

Example

Tautology is in coNP. Tautology equals complement of SAT. The certificate is given by an assignment of the variables for which the formula is false.

$$TAUTOLOGY = \{ A \mid A \text{ is a tautology (It is true for any assignament)} \}$$

Notice that…

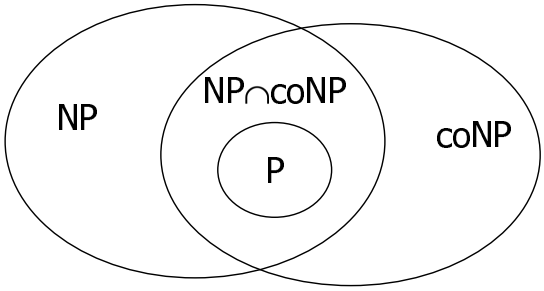
If a problem is in NP, then its complement is in coNP and vice versa…

co-NP, NP,...

Remark

- differently from P, it is unknown whether NP=coNP, actually it is supposed NP≠coNP

... we will come back later on this issue



co-NP, NP,...

Remark

- integer factorization (decision problem) is known to be both in NP and coNP, but no polynomial algorithm is available

$$INT-FACT=\{(n,k)\,|\,n \text{ has a factor smaller than } k\}$$

Proof

- INT-FACT is in coNP

  a certificate (that an integer has factors smaller than k) is given by a factor
- INT_FACT is in NP

  A polynomial certificate for prime numbers was produced by Pratt in 1975

Pratt certificate (sketch)

Fermat's little theorem converse

n is prime if and only if there exist r

1. r is prime to n
2. $r^{n-1}=1 \pmod n$
3. there is no integer $e<n-1$ such that $r^e= 1 \pmod n$

   (actually, it is sufficient to prove this for every $e=(n-1)/p_i$, where p_i is prime factor of n-1

Notice that

- r by itself is not a good certificate, because checking 3 requires to compute the factors of n-1, which cannot be done in polynomial time!!

A certificate is recursively defined: it contains

1. r
2. and the factors of n-1, along with their certificates

Pratt certificate: an example

7919 is prime because

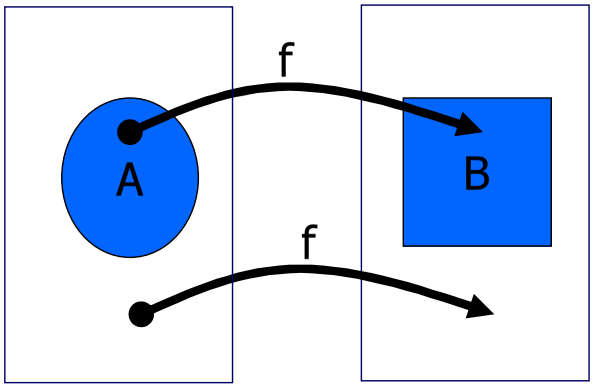
- $7^{7918}=1 \bmod 7919$
- $7918=2*37*107$
- and $7^{918/2} \neq 1 \bmod 7919$, $7^{918/37} \neq 1 \bmod 7919$, $7^{918/107} \neq 1 \bmod 7919$
  - 2 is prime (... by default)
  - 37 is prime because
    - $2^{36}=1 \bmod 37$
    - $36=3*3*2*2$, and $2^{36/2} \neq 1 \bmod 37$, $2^{36/3} \neq 1 \bmod 37$
    - 2 is prime (... by default)
    - 3 is prime because
      - $2^2=1 \pmod 3$ 7
      - $2=2$, ...ok
  - 107 is prime because
    - $2^{106}=1 \bmod 107$
    - $106=53*2$
    - .....

This reasoning can be memorized and provides the certificate

Polynomial Time Reducible

Definition A language A is polynomial time reducible to a another language B if there is a polynomial time computable function $f:\Sigma^* \to \Sigma^*$ such that:
$\mathbf{w \in A \iff f(w) \in B}$ for every $w \in \Sigma^*$.

Terminology/notation:
- $A \leq_P B$
- f is the polynomial time reduction of A to B



Poly-Time Functions

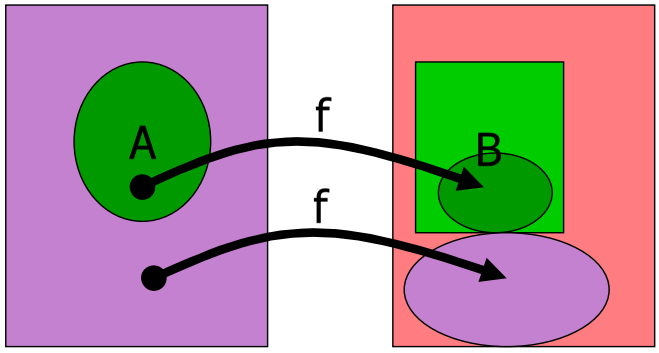
Definition A function $f:\Sigma^* \to \Sigma^*$ is a poly-time-computable function if there is a TM that on every input $w \in \Sigma^*$ halts after poly(|w|) steps with f(w) on the tape.

All the usual computations (addition, multiplication, sorting, minimization, etc.) are all poly-time.

Important here is that object transformations, like "Given a formula ϕ, make a graph G_ϕ" can almost always be done in poly-time.

$A \leq_P B$

The language B can be more difficult than A.



Typically, the image f(A) is only a subset of B, and f(Σ*\A) a subset of Σ*\B.

"Image f(A) can be the easy part of B".

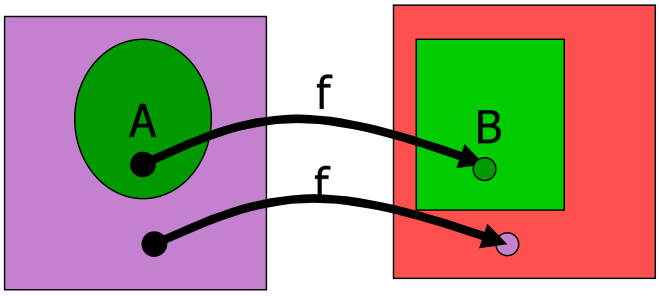
Polytime $A \leq_P B$

Theorem If A is in **P**, then $A \leq_P B$ for every nontrivial B.

Proof

Since A is in **P**, there exists a TM M such that M outputs "accept" on every $x \in A$, and "reject" on $x \notin A$ in polynomial time.
We can use this M for a TM-computable function f with

$f(x)=1 \in B$ if $x \in A$ and $f(x)=0 \notin B$ if $x \notin A$



"The function f does all the work"

P obeys $\leq_P$ Ordering

Theorem:
If $A \leq_P B$ and B is in **P**, then A is in **P** as well.
Proof: Let M be the poly-time TM for B and f the reducing function from A to B. Consider the TM:
On input w:
1) Compute f(w)
2) Run M on f(w) and give the same output.

By definition of f: $w \in A$ if and only if $f(w) \in B$.
M "accepts" f(w) in poly-time if $w \in A$, and
M "rejects" f(w) in poly-time if $w \notin A$.

NP-Completeness

Definition: A language B is **NP**-complete if
- B is in **NP**, and
- For every language $A \in$ **NP** we have $A \leq_P B$.

NP-complete problems are the most difficult problems in **NP**…

If we omit requirement 1 we define the set **NP**-hard.

Definition: A language B is **NP**-hard if
1. for every language $A \in$ **NP** we have $A \leq_P B$.

Why is NP-Completeness important?

If an efficient solution will be found for a NP-complete language, then an efficient solution will be found for each language in NP

Theorem If there is an NP-complete problem B that can be decided in deterministic polynomial time, then for all languages A∈ **NP** we also have A∈ **P**.

The question P=NP can be studied by analyzing just any NP-complete language

Theorem If B is NP-complete, then B∈ **P** if an only if P=NP

Cook-Levin Theorem

Cook-Levin Theorem proves that there a NP-complete language exists

Theorem: SAT is NP-complete

Proof Outline

Let A be a language in **NP**.

For every w, we want a (CNF) formula ϕ_w such that $w \in A \Longleftrightarrow \phi_w \in SAT$, with a poly-time function that calculates ϕ_w from w.

Let N be the nondeterministic **NP** that accepts A.
Key idea:
$w \in A \Longleftrightarrow \exists$ accepting path of N on w
$\Longleftrightarrow \exists\, x_1 \ldots x_m \, [\, \phi_w(x_1 \ldots x_m) = \text{TRUE} \,]$

More Proof Outline

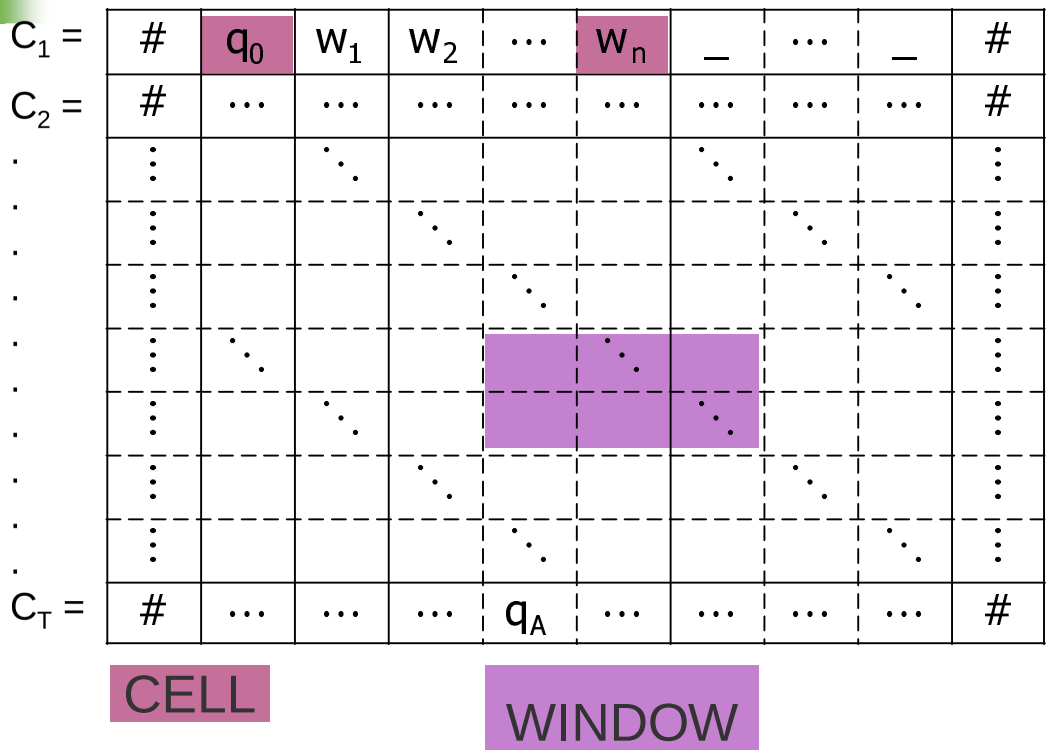
Specifically we will establish the chain:
$w \in A \Longleftrightarrow \exists$ accepting path of N on w

$\Longleftrightarrow$

There exists a sequence $C_1, \ldots, C_T$ of configurations with:
- C_1 the start configuration of N on w
- (C_j, C_{j+1}) a proper N transition for every j
- C_T an accepting configuration

$\Longleftrightarrow \exists x_1 \ldots x_m \, [\, \phi_z(x_1 \ldots x_m) = \text{TRUE} \,]$

Accepting Path of N on w

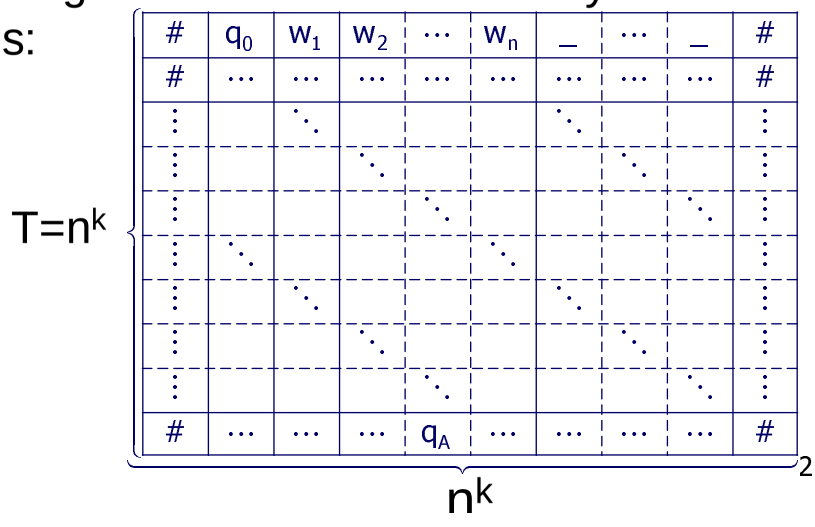


Size of Path of N on w

N is a poly-time nondeterministic TM:
The (accepting) path of N on w is limited by $O(n^k)$.

The sequence of configurations can described by a tableau $n^k \times n^k$ cells:

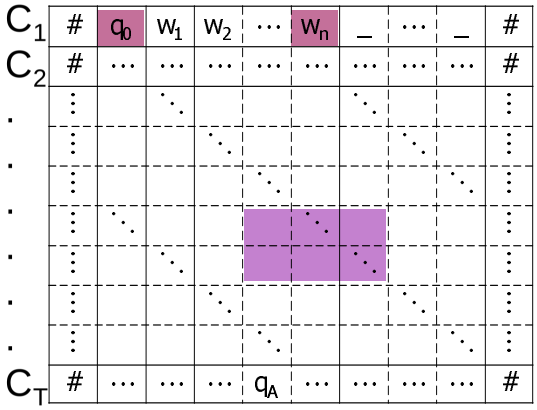
$\phi_w(x_1 \ldots x_m)$ describes this tableau…



How ϕ Describes the Tableau

The m Boolean variables in $\phi_w(x_1 \ldots x_m)$ have to describe:

- that each cell is proper
- that C_1 is the start configuration of N on w
- that the transitions (C_j, C_{j+1}) are proper
- that C_T is an accepting configuration



How ϕ Describes the Cells

The TM N has state set Q and tape alphabet Γ.
The content of a cell is in the set $Q \cup \Gamma \cup \{\#\}$.

Define the Boolean variables according to:

$$x_{(i,j,s)} = \begin{cases} \text{TRUE if cell}(i,j) = s \\ \text{FALSE otherwise} \end{cases}$$

In total there are
$n^k \times n^k \times |Q \cup \Gamma \cup \{\#\}|$
Boolean x-variables.
This is $O(n^{2k})$ polynomial in n.

ϕ_{cell} for Proper Cells

Not all x-assignments make sense.
We require that each cell(i,j) has a one description.
For every $1 \le i,j \le n^k$ there should be only one variable $x_{(i,j,s)}$ set TRUE.
The cell symbols s range from 1 to $c = |Q \cup \Gamma \cup \{\#\}|$.
Hence we define with the ONE-function:

$$\phi_{cell} = \bigcap_{i,j=1}^{n^k} ONE(x_{(i,j,1)}, \ldots, x_{(i,j,c)})$$

ϕ_{start} for the Start Configuration

The starting state of N is q_0, and the input string is $w_1,\ldots,w_n$.
(The rest is filled with _-spaces and marked on the left and right by #-symbols.)
Hence:

$$\phi_{start} = x_{(1,1,\#)} \wedge x_{(1,2,q_0)} \wedge$$
$$x_{(1,3,w_1)} \wedge \cdots \wedge x_{(1,n+2,w_n)} \wedge$$
$$x_{(1,n+3,\_)} \wedge \cdots \wedge x_{(1,n^k-1,\_)} \wedge x_{(1,n^k,\#)}$$

ϕ_{accept} for Accepting

After the TM N has entered the accepting state q_{accept}, it stays in this state.
We only have to check if the bottom row contains the accepting state:

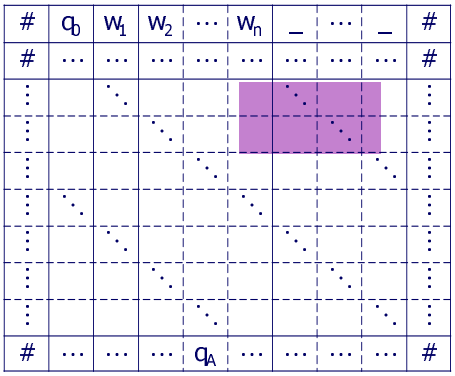
$$\phi_{accept} = \bigcup_{j=1}^{n^k} x_{(n^k, j, q_{accept})}$$

ϕ_{move} for Proper Transitions

The last requirement is that the sequence of configurations $C_1, C_2, \ldots, C_T$ are allowed by N.

We can check this by locally checking all 2×3 windows.

There are $(n^k - 1) \times (n^k - 2)$ of such windows:



$$\phi_{move} = \bigcap_{i=1}^{n^k - 1} \bigcap_{j=1}^{n^k - 2} (i, j) \text{ window legal}$$

Legal Windows

If there is no TM head, the tape should remain unchanged:

a	b	c
a	b	c

for all $a,b,c \in \Gamma \cup \{\#\}$.

For the transitions $\delta(q_1,a) = \{(q_2,b,R),(q_3,c,L)\}$:

a	q_1	a
a	b	q_2

and

a	q_1	a
q_3	a	c

et cetera.

More Legal Windows

The head can move in and out the window:

a	b	c
a	b	q_2

b	a	q_4
b	a	c

b	a	c
q_7	a	c

et cetera.

The tape and the TM remain stationary once an accepting state has been reached:

a	q_{accept}	b
a	q_{accept}	b

Some Illegal Windows

For various reasons the following windows indicate a mistake in the configuration sequence:

a	b	c
a	b	a

b	a	q_4
b	q_2	q_1

#	q_2	c
q_6	a	c

Most important, this window:
is illegal if $(q_9,c,R) \notin \delta(q_4,b)$

a	q_4	b
a	c	q_9

(i,j) Window Legal...

Let $L \subset (Q \cup \Gamma \cup \{\#\})^6$ be the set of legal windows.
Then "(i,j) window legal" can be expressed by

$$\bigcup_{(s_1,\ldots,s_6)\in L} (x_{(i,j,s_1)} \wedge x_{(i,j+1,s_2)} \wedge \cdots \wedge x_{(i+1,j+2,s_6)})$$

With these sub-formulas we can express

$$\phi_{move} = \bigcap_{i=1}^{n^k-1} \bigcap_{j=1}^{n^k-2} (i,j) \text{ window legal}$$

The Complete ϕ_w Formula

Together these four requirements give:

$$\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

By construction, ϕ_w is satisfiable if and only if there is an accepting nondeterministic path of N on input w:

$$w \in A \Longleftrightarrow \phi_w \in SAT$$

We have to check that $w \rightarrow \phi_w$ is poly-time…

Polytime Reduction Check 1

Fixed TM N with time complexity $O(n^k)$ on input $w_1,\ldots,w_n$ of length n.

There are $O(n^{2k})$ Boolean variables $x_{(i,j,s)}$ in ϕ_w.

Last issue: Can we describe ϕ_w in poly(n) time?

$$\phi_{cell} = \bigcap_{i,j=1}^{n^k} ONE(x_{(i,j,1)},\ldots,x_{(i,j,c)})$$

$O(n^{2k})$ time.

Polytime Reduction Check 2

Can we describe ϕ in poly(n) time?

$$\phi_{start} = x_{(1,1,\#)} \wedge x_{(1,2,q_0)} \wedge$$
$$x_{(1,3,w_1)} \wedge \cdots \wedge x_{(1,n+2,w_n)} \wedge$$
$$x_{(1,n+3,\_)} \wedge \cdots \wedge x_{(1,n^k-1,\_)} \wedge x_{(1,n^k,\#)}$$

… requires $O(n^k)$ time.

Polytime Reduction Check 3

Can we describe ϕ in poly(n) time?

$$\phi_{move} = \bigcap_{i=1}^{n^k-1} \bigcap_{j=1}^{n^k-2} (i,j) \text{ window legal}$$

… requires $O(n^{2k})$ time.

$$\phi_{accept} = \bigcup_{j=1}^{n^k} x_{(n^k,j,q_{accept})}$$

… complexity $O(n^k)$.

Polytime Reduction Check 4

Given $w_1,\ldots,w_n$, the construction of

$$\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

requires $O(n^{2k})$ time and space: poly(n).

The mapping from $w_1,\ldots,w_n$ to ϕ_w is a poly-time reduction from the **NP** problem A to SAT:
$A \leq_P SAT$ for all $A \in$ **NP**.

As SAT$\in$ **NP**, we see that SAT is **NP**-complete

Examples

Some NP-complete problems

- Is a given graph a k-clique?

$$CLIQUE = \{(G,k) \mid \text{G is a k - clique}\}$$

- Does the graph contains a Hamiltonian path?

$$HAMILTON = \{G \mid \text{G contains a Hamiltonian path}\}$$

- is there a subset of integers in a given set that sum to a given number?

$$SUBSET\text{–}SUM = \left\{(\text{S,t}) \mid S = \{x_1,..,x_n\},\ \text{it exists} Y \subseteq S \text{ such that} t = \sum_{i \in Y} x_i\right\}$$

Examples

Some NP-complete problems

- Has a given graph a cut of size k?

$$MAX-CUT=\{(G,k)\mid \text{G has cut of size k or more}\}$$

- Does an assignment of colors to the nodes of a graph exist such that no two adjacent nodes have the same colors?

$$3COLORS=\{G\mid \text{G can be colored with 3 colors}\}$$

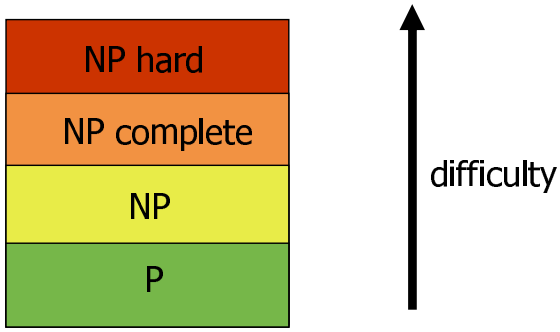
- has a given graph a k-node vertex cover ?

$$VERTEX-COVER=\{(G,k)\mid \text{G has k - vertex cover}\}$$

More about NP-complete

Remark

- NP-complete defines the most difficult problems in NP:
  - in fact any problem in NP can be reduced to NP-complete
- NP-hard problems may be even more difficult problem, however NP-hard contains problems outside NP



More about NP-complete

Remark

- It is unknown whether NP=coNP, but
- NP=P implies NP=coNP (by P=coP)
- and if NP-complete$\cap$coNP $\neq 0$ (or NP$\cap$coNP-complete $\neq 0$), then NP=coNP

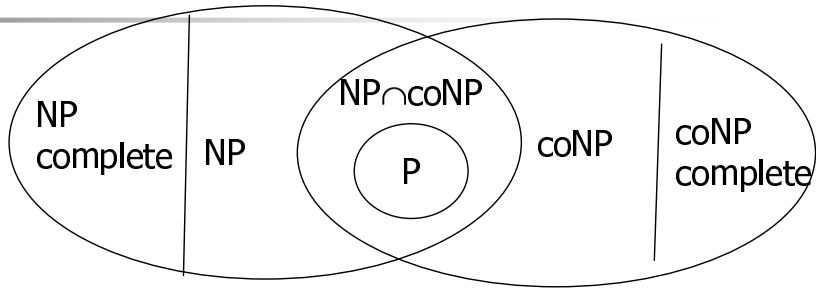
Proof

Assume there is a problem A is in NP-complete$\cap$coNP.

Since A$\in$ NP-complete, then for any problem B$\in$ NP, we have B$\leq_p$ A. Since, A$\in$ coNP, there is polynomial certificate for the complement of A. As a consequence, there is a polynomial certificate for B. Thus, NP$\subseteq$coNP.

Moreover, let C$\in$ coNP and consider its complement $\overline{C}$, which fulfills $\overline{C} \in$ NP. By, NP$\subseteq$coNP, it follows $\overline{C} \in$ coNP and, as a consequence, C$\in$ NP. Thus, coNP$\subseteq$NP

More about NP-complete



Remark

- the possible cases are
  - P=NP and NP=coNP
  - P≠NP and NP=coNP
  - P≠NP and NP≠coNP (the most believed)
- problems in NP $\cap$ coNP are believed to be easier than NP-complete and coNP-complete
  - f.i. integer factorization
  - it is unknown whether P= NP $\cap$ coNP

Are there problems whose complexity is in the middle between P and NP?

NP-Intermediate=NP-P

Ladner's Theorem
If P≠NP, then there are problems that do not belong both to P nor to NP-Complete

However, Ladner's Theorem does not provide a natural example of a NP-intermediate problem. Possible candidates

- graph isomorphism
- factoring
- discrete logarithm

A candidate for NP-intermediate

Graph isomorphism is a probable candidate for NP-intermediate

G_1,G_2 are isomorphs if there exist a bijection f from the nodes of G_1 to those of G_2 such that an edge (u,v) exists in G_1 if and only if edge (f(v),f(u)) exists in G_2.

Remarks

- Graph isomorphism is in NP: the certificate is any coding of f.
- No polynomial algorithm is known
- No reduction of a NP-complete problem is known

Graph isomorphism

the class of problems with a polynomial reduction to graph isomorphism (GI)

- contains
  - a subclass of maximun clique
  - finite automata isomorphism
  - regular graph isomorphism
  - directed acyclic graph isomorphism
- we will return on this issue later….

Language isomorphism

Polynomially isomorphic languages ($L_1, L_2 \subseteq \Sigma^*$)

- there is a bijection $h: \Sigma^* \rightarrow \Sigma^*$
- $w \in L_1$ if and only if $h(w) \in L_2$
- both h and h^{-1} are polynomial time computable

Remark

- All known NP-complete languages are polynomially isomorphic!!!
- … but the isomorphism among NP-complete languages has not been proved formally

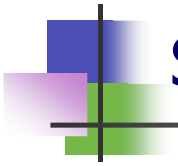
Sparse languages

Definition A language is sparse if the number of strings of size n is upper-bounded by a polynomial in n

Definition A language is unary if it contains only one symbol, i.e. L $\subseteq${0}*

Notice that

- NP-complete problems are not sparse languages
- The sparseness could provide another measure of the complexity of a decision problem:
  - perhaps simple languages with "few" strings can be easily recognized.
  - Notice, however, that unary languages can be even not computable
    - f.i., f(0^n)= accept if and only if <n> is the coding of a satisfiable boolean formula



Sparse languages

Sparse languages are important because

Theorem if P≠NP, then SPARSE⊆NP-intermediate

Theorem if P=NP, then any problem in NP is reducible to a sparse language

Theorem There is a sparse language in NP-intermediate if and only NP-intermediate contains a unary language

Thus we can study NP=P, by studying unary languages!!

Functions problems

- As far, we studied decision problems, what about function problems?

Let A be a language and R a relation such that
A={w| y exists s.t. R(w,y)}

Definition

The function problem corresponding to A is
"given w, find y such that R(w,y)"

Definition

FNP and FP are the classes of function problems computing in polynomial time on a DTM and a TM, respectively.

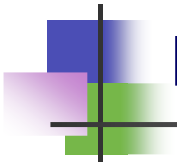
Functions problems

Remarks

- The complexity of a function problem is equal or larger than the complexity of the corresponding decision problem
  - the decision problem can be solved using the function problem
- we can define the concept of a polynomial reducibility also for the function problems
- we can prove that FSAT is FNP-complete
- we can prove that FSAT can be solved in polynomial time if and only if SAT can
  - FSAT is reducible to SAT

Theorem

FP=FNP if and only if P=NP



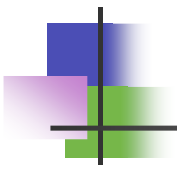
FSAT is polynomially reducible to SAT

FSAT= "Given a Boolean formula Φ, find an assignment that satisfy Φ"

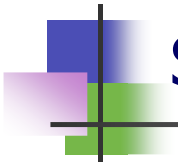
Proof

We give an algorithm for FSAT

1. check whether Φ is satisfiable: if it is not return no
2. else define $\Phi_1 = \Phi[x_1=\text{true}]$, $\Phi_2 = \Phi[x_1=\text{false}]$, where x_1 is a variable
3. check whether Φ_1, Φ_2 are satisfiable (one of them must be)
4. set $\Phi = \Phi_k$, where Φ_k is satisfiable
5. repeat 2-4 for all the variables



Space complexity



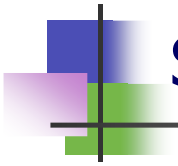
Space Complexity Classes

The space requirements of a computation are another important resource that we should be concerned about.

Definition: Let $f:N \to N$ be a function. The space complexity classes :

SPACE$(f(n))$ = { L | there is a TM that decides the language L in space $O(f(n))$ }

NSPACE$(f(n))$ = { L | there is a nondeterministic TM that decides the language L in space $O(f(n))$ }



Savitch's Theorem

Theorem: For any function $f:N \to N$ with $f(n) \geq \log n$, the inclusion

$$\textbf{NSPACE}(f(n)) \subseteq \textbf{SPACE}((f(n))^2)$$

holds.

Nondeterminism does not give you much extra for space complexity classes.

- Space behaves much nicer than 'time'.
- ***Space, unlike time, can be reused.***

Proof of Savitch's Theorem

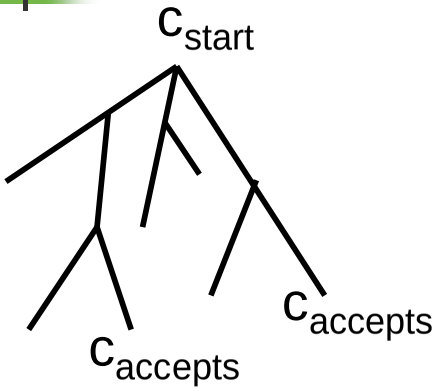
We could simulate the NTM by a TM (the TM emulate each brunch), but

- emulating a NTM would require to visit the NTM solution tree: during the visit we must keep track of the currently visited brunch
- an accepting brunch can be at most $2^{O(f(n))}$ long
  because, only $2^{O(f(n))}$ different memory configurations exist
- a stack with $2^{O(f(n))}$ locations is needed …. too much

On the other hand

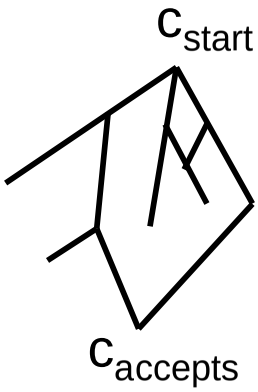
- also the number of brunches is at most $2^{O(f(n))}$
- and it may be possible to organize those brunches in another tree data structure having depth O(f(n)) …..

Proof of Savitch's Theorem



A **NPSPACE** tree
It has depth $2^{O(poly(n))}$, and
total number of nodes $2^{O(poly(n))}$
(note: *less than* 2^{depth}.)

First step of the proof:
all the accepting states are fused



Proof of Savitch's Theorem

- We define a program CANYIELD(c_1,c_2,t):
  - accept, if the state c_2 can be reached from c_1 in t steps on the NTM
  - reject, otherwise
- CANYIELD(c_{start},c_{accept}, $2^{f(n)}$) accept if and only if
  - the input string corresponding to c_{start} is accepted by the NTM
- the space used by of CANYIELD
  - each configuration c_i requires O(f(n))
  - the maximum stack depth is O(log(t)), where t= $2^{f(n)}$
  - total space is $O(f(n)^2)$

```
CANYIELD(c1,c2,t):
    1.  case t=0, accept if and only if c1==c2.
    2.  case  t>0, for each intermediate state cm
    3.          run CANYIELD(c1,cm,½t) and CANYIELD(cm,c2,½t)
    4.          if both steps 3 and 4 accept, then accept
    5.   if no intermediate state has been accepted, the reject
```


PSPACE

Definition: The class of languages that can be decided by a single tape TM in polynomial space is denoted by **PSPACE**:

$$\mathbf{PSPACE} = \bigcup_{k=1,2,\ldots} \mathbf{SPACE}(n^k)$$

Some Observations

What about the following relations?

- **P** and **PSPACE**
  P $\subseteq$ **PSPACE,** because a program that runs in polynomial time can access only a polynomial number of memory locations

- **NP** and **NPSPACE**
  NP $\subseteq$ **NPSPACE,** because each branch of the NTM has polynomial depth and can access only a polynomial number of memory locations

Some Observations

What about the following relations?

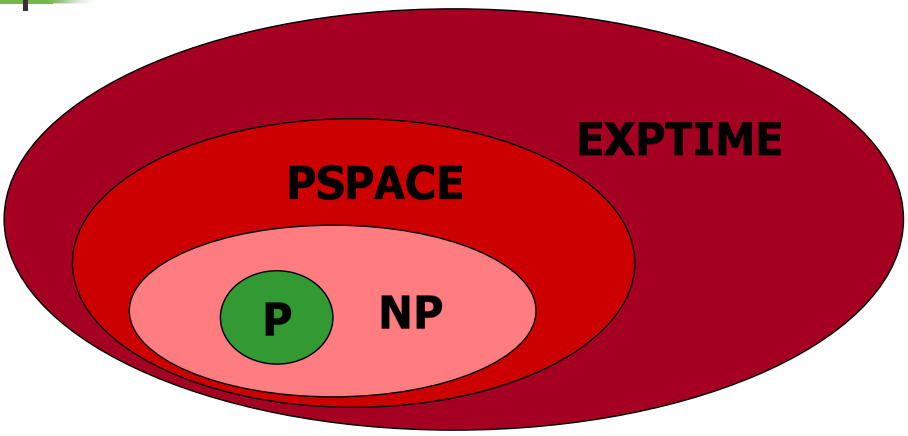
- **PSPACE** and **NPSPACE.**
  Savitch's theorem ensures that **PSPACE**=**NPSPACE**

- **PSPACE** and **EXPTIME**

  EXPTIME $= \bigcup_{k=1,2,\ldots} \mathbf{TIME}(2^{n^k})$

  PSPACE $\subseteq$ **EXPTIME**
  There are $2^{O(f(n))}$ different configurations for a **SPACE**(f(n)) computation…
  …hence **SPACE**(f(n))$\subseteq$**TIME**($2^{O(f(n))}$).

A Hierarchy of Classes



P $\subseteq$ **NP** $\subseteq$ **PSPACE**=**NPSPACE** $\subseteq$ **EXPTIME**

We don't know how to prove **P**$\neq$**PSPACE** or **NP**$\neq$**EXPTIME**.
But we do know: **P**$\neq$**EXPTIME.**

SPACE-Completeness

Definition: A language B is SPACE(f(n))-complete if

1) B is in PSPACE(f(n)), and

2) For every language A$\in$SPACE(f(n)) we have A$\leq_{ps}$B,

where $\leq_{ps}$ denotes poly-space reductions

Some authors require the reduction to be poly-time, not poly-space. Poly-time implies poly-space and decrease the possible reductions

Definition: A language B is SPACE(f(n)) -hard if

- for every language A$\in$ SPACE(f(n)) we have A$\leq_{s}$B.

TQBF

The typical **PSPACE**-complete language is true fully quantified Boolean formulas (TQBF):

$$TQBF = \{ \phi \mid \phi \text{ is a true fully q.b. formula} \}.$$

Some examples:

$$\exists x \forall y [x \Rightarrow y] \in TQBF$$

$$\forall x \exists y \exists z [(x \wedge y) \vee (x \wedge z)] \notin TQBF$$

Theorem TQBF is **PSPACE-Complete.**

TQBF

Theorem TQBF is **PSPACE-Complete.**

Proof:

TQBF$\in$**PSPACE**

This program uses linear space

- Algorithm Tqbf($Q_1 x_1,\ldots,Q_k x_k \Phi(x_1,\ldots, x_k)$)
- if k=0, return Φ
- if $Q_k = \exists$, then

  return Tqbf($Q_1 x_1,\ldots,Q_{k-1} x_k \Phi(x_1,\ldots, x_{k-1},\text{true})$)

  $\vee$ Tqbf($Q_1 x_1,\ldots,Q_k x_{k-1} \Phi(x_1,\ldots, x_{k-1},\text{false})$)

- if $Q_k = \forall$, then

  return Tqbf(Q1 x1,…,Qk-1 xk Φ(x1,…, xk-1,true))

  $\wedge$ Tqbf(Q1 x1,…,Qk xk-1 Φ(x1,…, xk -1,false))

proof: TQBF is PSPACE-Complete

If $A \in$ **PSPACE**, then $A \leq_{ps} B$

- Idea: Given M and w, we can make a QBF $\Phi_{M,w}$ such that: "$w \in A$ if and only if $\Phi_{M,w} \in$ TQBF".
- The TM that decides A takes time $2^{O(poly(|w|))}$ on input w.
- The question: YIELD($c_{start}, c_{accept}, 2^{O(poly(|w|))}$)?
- We define a QBF $\phi(c_1,c_2,t)$ of length $O(|c| \cdot \log(t))$:
- For $\phi(c_1,c_2,1)$ this is simple (as in Cook-Levin).
- For $\phi(c_1,c_2,2t)$ we could try:
  $\phi(c_1,c_2,2t) = \exists c_m [\phi(c_1,c_m,t) \wedge \phi(c_m,c_2,t)]$, but in this way, the length of the stack is $|\phi(c_1,c_2,t)| \sim t$, which is too big because t may equal $2^{O(poly(|w|))}$.

TQBF is PSPACE-Complete

- Instead, we use
  $$\phi(c_1,c_2,2t)=\exists c_m \forall (c_3,c_4) \in \{(c_1,c_m),(c_m,c_2)\} [\ \phi(c_3,c_4,t)\ ].$$
  Now the length of ϕ grows indeed like $O(|c| \cdot \log(t))$, which gives a linear stack even when $t = 2^{O(poly(|w|))}$.

- As a result, for every input w, we can make a QBF $\Phi_{M,w} = \phi(c_{start}, c_{accept}, 2^{O(poly(|w|))})$ such that:
  "M accepts w ($w \in A$)" if and only if "$\Phi_{M,w} \in$ TQBF",
  and size $|\Phi_{M,w}| = O(poly(n)^2)$.

The transformation $w \rightarrow \Phi_{M,w}$ is poly-time and, thus, it is also poly-space.

Space Complexity Classes

Remark

- in space complexity the role of the under linear classes is more important than in time complexity
- for example, search in trees, sorted vectors…..
- In the following we will discuss about
  - L= SPACE(log n)
  - NL= NSPACE(log n))

SUM and MULT belongs to L

- binary SUM $\in$ L

$$SUM = \{(a,b,c) \mid \mathrm{c} = \mathrm{a} + \mathrm{b}\}$$

during the sum, only the remainder must be stored

- binary MUL $\in$ L

$$MUL = \{(a,b,c) \mid \mathrm{c} = \mathrm{a} * \mathrm{b}\}$$

only two bits must be stored

Graph path belongs to NL

- PATH is in NL=NSPACE(log n)

$$PATH = \{(G, v, u) \mid \text{there is a path from n to v in G}\}$$

Input: G = (V; E), v, u ∈ V

1. x ← v
2. counter ← |V|
3. repeat
4. decrement counter by 1
5. guess a node y ∈ V s.t. (x,y) ∈ E
6. if y ∉ u then x ← y
7. until y = u or counter = 0
8. if y = u then accept, else reject

The algorithm search for the path
- guessing the node at each step
- memorizing only the last node

PATH is NL-complete

G is returned an edge at each step…. otherwise G will consume too much space
The machine that implements PATH will call the reduction any time it needs an input

∀L ∈ NL, L is log-space reducible to PATH

Input: an input string x, a TM M

Task: output a graph G=(V,E) and two nodes v, u ∈ V s.t. there is a path from v to u in G iff x is accepted by M

- compute n, the number of different configurations of M while computing input x
- for i = 1 to n
  for j = 1 to n
  if there is a transition of M from configuration i to configuration j, output an edge (i,j) of G
- output the number corresponding to the start and the accepting configuration

Immerman theorem and the complement class

Theorem

if there is A s.t. A∈ NL-complete and A∈ coNL, then NL=coNL

proof

- since A∈ NL-complete, then for each A' ∈ NL. A'$\leq_{log}$A and x ∈ A' if and only if f(x) ∈ A
- On other hand, let $\underline{A}$, $\underline{A'}$ be the complements of A and A'. x ∈ $\underline{A'}$ if and only if f(x) ∈ $\underline{A}$, so $\underline{A}$∈ NL implies $\underline{A'}$∈ NL (i.e. A∈ coNL implies A'∈ coNL)
- thus NL⊆coNL. The converse can be proved in a similar way

Immerman theorem and the complement class

Theorem (Immerman,1988)
PATH is NL-complete and belongs to coNL, thus NL=coNL

Immerman theorem can be extended to the other space complexity classess

Theorem:
∀s(n)≥log(n), NSPACE(s(n))=coNSPACE(s(n))

Log space problems are poly-time

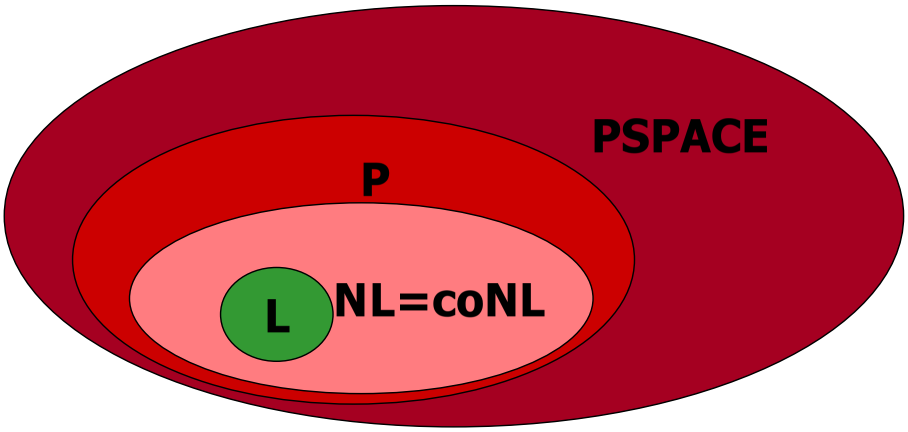
Theorem

$NL \subseteq P$

Proof

- any problem in NL is reducible to PATH
- the reduction can be carried out in O(log n) and, as a consequence in $O(2^{\log n})=O(n)$ time
- thus, any problem in NL is polynomial time reducible to PATH

Summing up



$L \subseteq NL = coNL \subseteq P \subseteq PSPACE$

Probabilistic algorithms

Probabilistic Turing machine

A probabilistic Turing machine M

- is a non deterministic machine with two legal moves
- each non deterministic step is called coin-flip step
- the probability of a branch b containing k coin-flip steps is $P(b) = 2^{-k}$
- the probability of accepting w is

$$P(\text{M accepts w}) = \sum_{\substack{\text{b is an} \\ \text{accepting branch}}} P(b)$$

- M decides a language A with error probability ε if

  $w \in A$ implies P(M accepts w)>1-ε

  $w \notin A$ implies P(M rejects w)>1-ε

Probabilistic polynomial time

Definition

- BPP is the class of languages that are recognized by probabilistic polynomial time Turing machines with error probability 1/3

Theorem

For any $\varepsilon<1/2$, any polynomial p(n), any probabilistic polynomial time Turing machine M_1 that operates with error ε, there is another polynomial time Turing machine M_2 that operates with an error $2^{-p(n)}$

Proof (sketch)

Proof

M_2 is

- we run k instances of M_1 in parallel
- if most of the outputs are accept then accept, otherwise reject

The probability that M_2 is wrong is

$$P(\mathrm{M}_2 \text{ is wrong }) \le \sum_{\substack{S \text{ is a bad} \\ \text{result sequence}}} P(S) \le 2^{2k}\varepsilon^k(1-\varepsilon)^k \le \left(4\varepsilon(1-\varepsilon)\right)^k$$

The theorem is proved if

$$k \ge \frac{p(n)}{\log_2 4\varepsilon(1-\varepsilon)}$$

The class BPP

- of course, $P \subseteq BPP$
- it is unknown whether $NP \subseteq BPP$ and/or $BPP \subseteq NP$!!
- It is supposed that $NP \not\subseteq BPP$, otherwise we could have reasonable algorithms for NP problems
- Somebody suppose also BPP=P
- BPP is closed under complement, BPP=coBPP

BPP and random number generation

A randomized algorithm needs a source for random numbers ....

- it is unknown whether perfect random numbers can be generated either by physical devices or mathematical tools
- a perfect random source generate a sequence of bit $b_1,b_2,\ldots$ where $P(b_1=y_1,..b_n=y_n)=2^{-n}$, for any $y_1,y_2,\ldots$

δ-BPP

- is class of languages that are recognized by probabilistic polynomial time Turing machines where the probabilty of each move is in $[\delta,1-\delta]$ instead of being ½ (intuitive definition)
- it can be proved that, for any $\delta<1/2$, δ-BPP=BPP!!

Primality

Primaility is

- is the number p prime?

- Primality is used by public key cryptographical algorithms (f.i. RSA)
- Primality has been recently proved to be in P $O(\log(n)^{12})$, however this algorithm is very slow in practice
- Primality can be tested using random algorithms

Miller-Rabin test

Miller-Rabin primality test for numbers

function Miller-Rabin(a, p)

- let *s, d* be integers where *d* is odd and that $p-1=2^s d$ holds
- **if** ($a^d = 1 (\mathrm{mod}\ p)$ and
- $a^{2^r d} = 1 (\mathrm{mod}\ p)$ for any $0 \le r \le s-1$) **return true**
- **else return false**

It can be proved that

if p is not prime, then the Miller-Rabin(a,p) test fails for at lest a half of the a, where $2 \le a \le p-1$

Testing primality

A method to test primality of p

- chose random $a_1,..,a_k$ in [2,p-1]
- check Miller-Rabin(a_i, p), for each a_i
- if some test fails then accept
  else reject

Remark

- the above test is carried out in polynomial time on a probabilistic TM
  - If the p is prime, the above algorithm always accept,
  - If p is composite, the probability of accepting is smaller than 2^{-k}

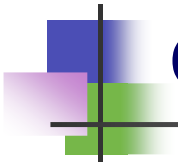
Integer factorization

Integer factorization is

- given an integer n, find its prime decomposition

Integer factorization properites

- it is considered much more difficult then primality
- it is unknown whether is in NP or in co-NP (no polynomial certificate is known)
- However
  - the corresponding decision problem "does *N* have a factor less than *M*?" is known to be in both NP and co-NP
  - there is a polynomial algorithm for integer factorization on quantun computers



Quantum computing

A quantum Turing machine (intuitive definition)

- a Turing machine where
  - the internal state is defined by a finite set of qubits
  - the tape is a sequence of qubits
  - ...

About QTMs

- each qubit can be in a state 0, 1, or in a superposition state, thus a quantum Turing machine can carry out parallel computations...
- in general, quantum computer can produce only a result having some probability of being correct
- a QTM can solve the same problems as a DTM, but in a faster way!

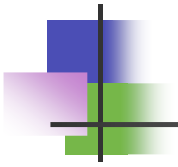
Quantum computing complexity classes

Exact quantum polynomial (EQP)

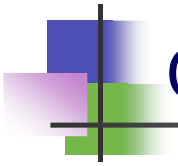
- languages that can be decided with probability 1 in polynomial time on a QTM
- it is the quantum class corresponding to P
- it has been proved P $\subset$ EQP, strictly!!

Bounded-errror quantum polynomial (BQP)

- languages that can be decided with probability larger than 1/3 in polynomial time on a QTM
- it is the quantum class corresponding to BPP
- it holds P $\subseteq$BPP $\subseteq$BQP $\subseteq$ PSPACE, but it is unknown whether the inclusion is strict for some inclusion!
- it is unknown the relation between NP and BQP
- it has been proved that integer factorization is in BQP (Shor algorithm)!!



Polynomial hierarchy



Oracles

Oracles

$L \in C_1^{C2}$:There exists a TM of class C_1 with access to an oracle in class C_2 that accepts L

Examples

- P^C = Languages accepted by *DTM* with access to an oracle *A* in class *C*.
- NP^C = Languages accepted by *NTM* with access to an oracle *A* in class *C*

Polynomial hierarchy

The polynomial hierarchy extends the definition of P, NP e coNP

Definition by indution

- $\Sigma_0 := P$
- $\Sigma_{i+1} := NP^{\Sigma_i}$
- $\Pi_i = co\Sigma_i$

Polynomial hierarchy (PH)

$$PH = \bigcup_i \Sigma_i$$

An alternative definition

- Deciding whether a formula is satisfiable belongs to

$\exists x_1 \ldots x_n (x_1 \vee \neg x_2 \vee x_8) \wedge \ldots \wedge (\neg x_6 \vee x_3)$
only existential quantifier

NP-complete

$\forall x_1 \exists x_2 \forall x_3 \ldots [(x_1 \vee \neg x_2 \vee x_8) \wedge \ldots \wedge (\neg x_6 \vee x_3)]$
existential & universal quantifiers

PSPACE-comple

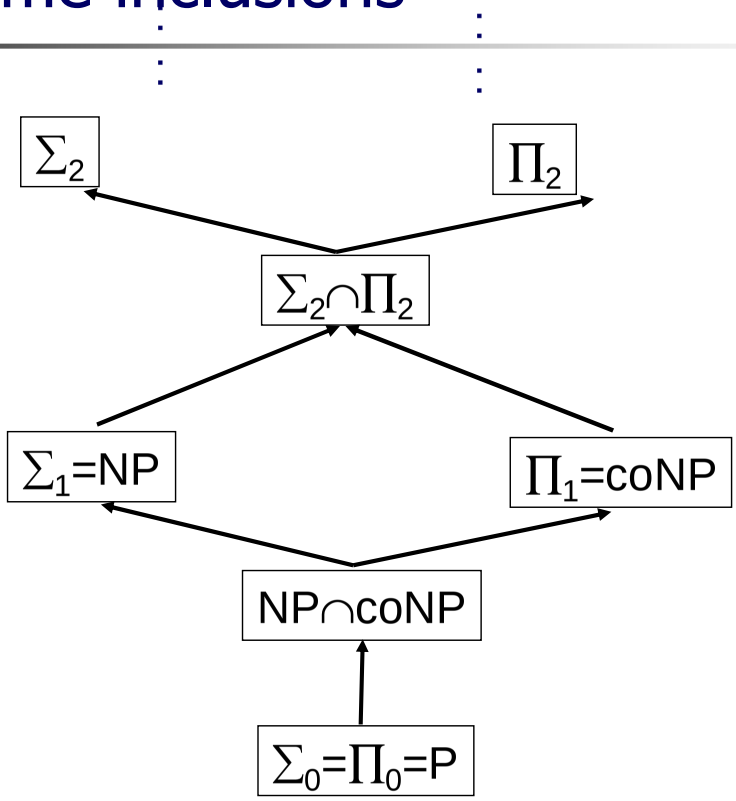
An alternative definition

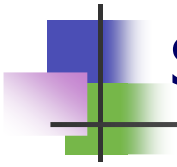
Alternative definition

Σ_i is the class of all languages reducible to deciding the sat. of a formula of type

$$\underbrace{\exists x_1 \forall x_2 \exists x_3 \ldots}_{i \text{ alternating quantifiers}} R(x_1, x_2, x_3, \ldots)$$

Some inclusions





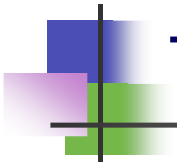
Some properties

PSPACE is an upper bound for the hierarchy PH:

- PH$\subseteq$PSPACE

M_2 contains the languages probabilistically decidable in polynomial time

- BPP $\subseteq\Sigma_2 \subseteq$PH
- notice that if it is unknown whether BPP $\subseteq\Sigma_1$=NP



The collapse of the hierarchy

- Its in unknown whether the inclusions of the hierarchy are strict, however ….

Theorem

- If for a k, $\Sigma_k = \Sigma_{k+1}$ or $\Sigma_k=\Pi_k$ then, for any r>k the hierarchy over k collapses, i.e. $\Sigma_r = \Sigma_{r+1} =\Pi_r= \Pi_{r+1}$
- In particular, if NP=coNP (P=NP), then the whole hierarchy collapses, i.e. PH=NP= $\Sigma_k =\Pi_k$