Computational Complexity

Running time (or <u>time complexity</u>) of a T.M.

A T.M. has time complexity $T(n)$ if it halts in at most $T(n)$ steps (accepting or not) for all input strings of length n.

<u>Polynomial time</u>: $T(n) = O(n^c)$ for some fixed c

(fixed means independent from n, i.e. the input-size)

Examples:

$$\left.\begin{array}{l} O(n^2) \\ O(n \cdot \log n) \\ O(n^{3.14}) \end{array}\right\} \text{polynomial time}$$

$$\left.\begin{array}{l} O(n^{\log n}) \\ O(2^n) \end{array}\right\} \text{non-poly}$$

Complexity theory considers <u>tractable</u> all problems with poly-time algorithms.

Motivations:

1.) robustness wrt the computation model
(all general computation models can simulate each other in poly-time ⇒ they define the same class of tractable probl.

2.) robustness wrt combining algorithms
(a polynomial of a polynomial is still a polynomial)

3.) going from polynomial to non-polynomial is drastic also in practice (e.g. compare $10 \cdot n^4$ with $0.1 \cdot 2^n$, when n grows)

4) Most practically used algorithms that are polynomial are so with a low coefficient (i.e. $T(n) = O(n^c)$, with c typically ≤ 3.

Time complexity classes:

Definition: $P = \{ L \mid L = \mathcal{L}(M)$ for some poly-time DTM $M \}$
$NP = \{ L \mid L = \mathcal{L}(N)$ for some poly-time NTM $N \}$

Note: both DTMs and NTMs must be halting T.M.s

From the definition we have immediately: $P \subseteq NP$
(every NTM is also a DTM)

Note: being in P corresponds to the intuition that the problem can be solved efficiently.

Instead, being in NP means intuitively that, given a solution, we can check efficiently whether it is correct.

Satisfiability:

Boolean formula: operands: $x_1, \ldots, x_n$
operators: $\wedge, \vee, \neg$
formula $F(x_1, \ldots, x_n)$

Satisfiability problem: given a boolean formula $F(x_1, \ldots, x_n)$, is there a truth assignment (i.e., an assignment of true/false values) for $x_1, \ldots, x_n$ that satisfies F (i.e., makes F evaluate to true)?

Example: $F(x_1, x_2) = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2))$

is satisfiable: $x_1 = 1$, $x_2 = 1$

$F(x_1, x_2) = x_1 \wedge (\neg x_1 \vee x_2) \wedge \neg x_2$

is not satisfiable

We first show how we can convert it to a language problem:

- we must encode formulas as strings

$\Sigma = \{\wedge, \vee, \neg, (,), x, 0, 1\}$

variable x_i: $x\langle i\text{ in binary}\rangle$

e.g. x_5 is encoded as $x101$

$\Rightarrow$ we obtain that $F(x_1, \dots, x_n)$ can be encoded as a string over Σ.

$L_{SAT} = \{w \mid w \text{ encodes a satisfiable formula}\}$

Theorem: $L_{SAT} \in NP$ (i.e., satisfiability is in NP)

Proof:

It suffices to show a poly-time NTM N s.t. $\mathcal{L}(N) = L_{SAT}$

N runs in two steps:

1) "guess" a truth assignment F for $x_1, \dots, x_n$

2) evaluate F on truth assignment and whether it has value true.

We have: F satisfiable $\Leftrightarrow \exists$ satisfying T.A.

$\Leftrightarrow N$ has accepting execution

Running time: step 1) $O(n)$

step 2) $O(n^2)$ with multiple tapes $\Rightarrow O(n^4)$

q.e.d.

<u>Note</u>: - All decision problems can be converted to language problems, by encoding the input as a string.

- We know that $L_{SAT} \in NP$, but we do not know whether $L_{SAT} \in P$:
  - we cannot exploit the conversion NTM $\longrightarrow$ DTM, since it causes an exponential blowup in running-time
- under the standard NTM – DTM conversion, the DTM will have to try all possible truth-assignments (2^{n})

In fact: open whether $L_{SAT} \in P$

Special case of SAT: CSAT (
conjunctive normal form:
(note: we use + for $\vee$
and $\cdot$ for $\wedge$)

- literal: variable x_i or its negation $\overline{x_i}$
- clause: sum/or of literals: $C_j = x_1 + \overline{x_2}$
- CNF-formula: product/and of clauses: $F = C_1 \cdot \ldots \cdot C_m$

Thus $F = \prod_{j=1}^{m} C_j$ with $C_j = \sum_{i=1}^{k_j} x_{ji}$

CSAT-problem: given a CNF formula F,
decide whether F is satisfiable

Since SAT $\in$ NP, we have also CSAT $\in$ NP k-CNF-formula: each clause has exactly k literals

1-SAT : $(\bar{x}_1) \cdot (x_2) \cdot (x_3)$

2-SAT : $(x_1 + \bar{x}_2) \cdot (\bar{x}_1 + x_2)$

3-SAT :

⋮

Facts: 1-SAT $\in$ P (trivial)

2-SAT $\in$ P (not so easy – via graph reachability)

3-SAT $\in$ P is still open

There are many (thousands) problems like SAT and CSAT that can be easily established to be in NP as follows:

Step 1 : "guess" some solution S

Step 2 : verify that S is a correct solution

Note: Step 1 exploits nondeterminism, and is clearly polynomial (running time of a NTM)

Step 2, for the problem to be in NP, must be carried out deterministically in poly-time (polynomial verifiability)

Examples :

- Traveling salesman problem (TSP)

input : - graph $G=(V,E)$ with edge lengths $l(u,v)$
- integer k

problem : does G have a tour (visiting each node exactly once) of length $\leq k$?

TSP $\in$ NP

Step 1 : guess a tour

Step 2 : check that length of tour is $\leq k$

- Clique : input - graph $G = (V, E)$
  - integer k

  problem: does the graph have a clique of size k

  (a clique is a subgraph of G in which each pair of nodes is connected by an edge)

- Knapsack input : - set of items, each with an integer weight
  - capacity k of a knapsack

  problem: is there a subset of the items whose total weight matches the capacity k

This property explains why so <u>many practical problems are in NP</u>:
- problems ask for the design of mathematical objects (paths, truth assignments, solutions of equations, VLSI-routes, ...)
- sometimes we look for the best solution, (or a solution that matches some condition) that matches the specification
- the solution is of small (polynomial) size, otherwise it would be useless
- it is simple (poly-time) to check whether it matches the spec.

but, there are exponentially many possible solutions

If we had $P = NP$, all these problems would have efficient (poly-time) solutions.

But we currently believe that $P \neq NP$.

<u>Assuming $P \neq NP$</u>, how do we determine which problems of NP are not in P (i.e., we know they don't have an efficient algorithm)?

NP-completeness

Key idea: we define NP-completeness in such a way that if we show that an NP-complete problem is in P, then all problems in NP would be in P.
(i.e., we would have $P = NP$)

It follows: assuming $P \neq NP$, an NP-complete problem cannot be in P

<u>Poly-time reduction</u>:

Problem X reduces to problem Y in poly-time ($X <_{poly} Y$) if there is a function R (the poly-time reduction) s.t.

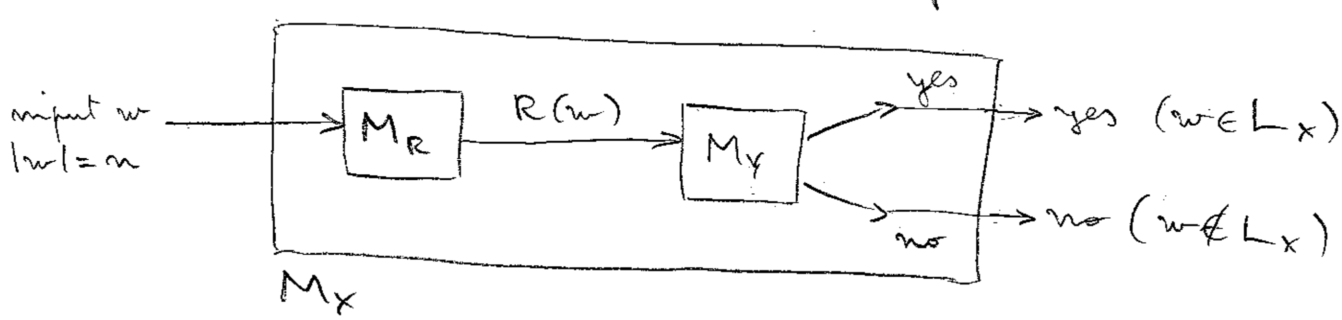
1) $w \in L_X \iff R(w) \in L_Y$
2) R is computable by a poly-time DTM

(L_X is the language encoding of problem X)

<u>Theorem</u>: $X <_{poly} Y$ and $Y \in P \implies X \in P$

Proof: let M_R be a poly-time DTM for R
M_Y — " — Y

We construct a DTM M_X for X as follows



Running time of M_X:
suppose: M_R runs in time $T_R(n) \leq n^a$
M_Y — " — $T_Y(n) \leq n^b$

Let $|w| = n$

Then $|R(w)| \leq n^a$

$\Rightarrow M_X$ runs in time

$$T_X(n) \leq T_R(n) + T_Y(T_R(n)) =$$
$$= n^e + (n^a)^b = O(n^{e \cdot b})$$

q.e.d.

Corollary: $X <_{poly} Y$ and $X \notin P \Rightarrow Y \notin P$

Definition: Problem Y (or language L_Y) is NP-hard if $\forall X \in NP$ we have $X <_{poly} Y$

Intuitively: an NP-hard problem is at least as hard as any problem in NP

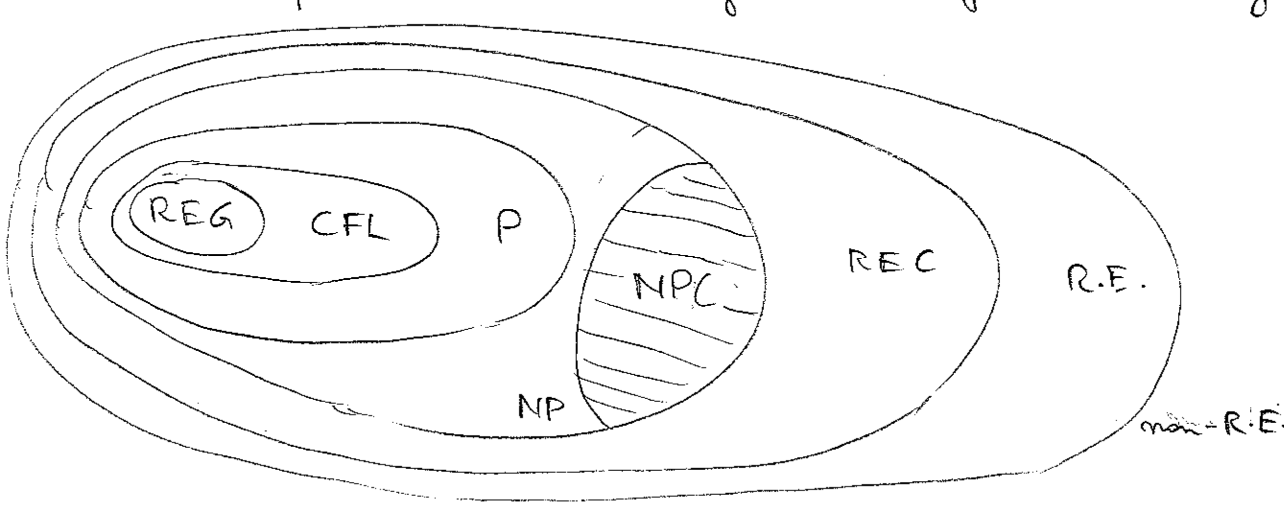
Immediate: Y is NP-hard and $Y \in P \Rightarrow P = NP$

Definition: Y is NP-complete if
1) $Y \in NP$ and
2) Y is NP-hard

Intuitively: NP-complete problems are the hardest problems in NP. If one of them is in P, then all problems in NP are in P.

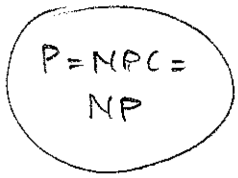
Hence: NP-completeness is a strong evidence of intractability.

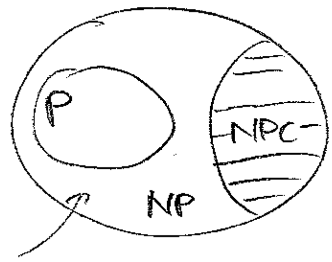
Languages:



Note: relationship between P, NPC and NP

either $P = NP$ or $P \neq NP$





in this case we know there are problems in NP that are neither in P nor NPC (proof is complicated)

How do we prove problems to be NP-complete?

<u>Theorem</u>: X is NP-hard and $X <_{poly} Y \Rightarrow Y$ is NP-hard

Proof: NP $<_{poly} X <_{poly} Y$

But, to exploit this result, we need a first NP-hard problem:

<u>Cook's theorem:</u> CSAT is NP-hard

Proof idea: we must show: $\forall L \in NP : L <_{poly} L_{CSAT}$

Fix $L \in NP$ and let M_L be a poly-time NTM for L.

We must show a poly-time reduction R_L:
input: string w
output: CNF formula F s.t.

$$w \in \mathcal{L}(M_L) \Leftrightarrow F \text{ is satisfiable}$$

Idea: F encodes the computation of M_L on w.

Suppose $w \in \mathcal{L}(M_L)$ and $|w| = n$.

then there exists a sequence of IDs of M_L:

$$ID_0 \vdash ID_1 \vdash \cdots \vdash ID_\tau$$

with $ID_0 = q_0 w$

ID_τ is an accepting ID (i.e. M_L is in a final state.)

$\tau \leq P(n)$

We assume that $\tau = P(n)$ by adding

$ID_{\tau+1}, ID_{\tau+2}, \ldots, ID_{P(n)}$ same as ID_τ

Idea: encode computation as matrix X

TAPE $\longrightarrow$

TIME $\downarrow$

	1	2	3		n	$n+1$	$n+2$			$P(n)$
0	q_0/w_1	w_2	w_3	....	w_n	$\not{b}$	$\not{b}$	....	$\not{b}$	$\not{b}$
1	x	q_1/w_2	w_3	...	w_n	$\not{b}$	$\not{b}$	...	$\not{b}$	$\not{b}$
2	x	y	q_3/w_3	...	w_n	$\not{b}$	$\not{b}$	...	$\not{b}$	$\not{b}$
⋮										
$P(n)$	x_1	x_2	x_3	...	x_n	x_{n+1}	x_{n+2}		q/y_1	y_2

M cannot use more than $P(n)$ cells

x_{it} : contents of tape cell i in ID_t

except for composite symbol $\boxed{q/x}$ to denote state and head position

We have that $w \in \mathcal{L}(M_L)$ iff

a) X is properly filled in

b) row 0 is ID_0

c) row $P(n)$ has final state

d) successive rows are related through legal transitions of M_L

M_L is NTM. Let k be the maximum degree of nondeterminism, i.e., for all q, X: $|\delta(q,X)| \leq k$.

To encode which of the possible transitions is chosen when going from ID_i to ID_{i+1} for the accepting sequence.

We use an array C of $P(n)$ elements (clue array)

TIME ↓

0	c_0
1	c_1
	$\vdots$
$P(n)-1$	c_{n-1}

$1 \leq c_i \leq k$

To represent X and C we use boolean variables

x_{itA} = true if cell i in ID_t contains A

c_{tl} = true if $c_t = l$

when $1 \leq i \leq P(n)$

$0 \leq t \leq P(n)$

$A \in \Gamma' = \Gamma \cup \underbrace{\Gamma \times Q}_{\text{composite symbols}}$

$1 \leq l \leq k$

Total number of variables is $O(P(n)^2)$, i.e., polynomial

To construct the CNF formula F we use 4 types of clauses

type a) X and C are properly filled in:

UNIQUE(i,t): for each i and t, cell i in ID_t is uniquely filled

$$\left(\sum_{A \in \Gamma'} x_{itA}\right) \cdot \prod_{\substack{A,B \in \Gamma' \\ A \neq B}} \left(\overline{x_{itA}} + \overline{x_{itB}}\right)$$

UNIQUEC(t): for each t, $C[t]$ is uniquely filled

$$\Big(\sum_{l\in\{1,\dots,k\}} C_{t,l}\Big)\cdot \prod_{\substack{l,m\in\{1,\dots,k\}\\ l\neq m}} \left(\overline{C}_{tl}+\overline{C}_{tm}\right)$$

$\Rightarrow O(P(n)^2)$ clauses, which is still polynomial (since $1\leq i\leq P(n)$ and $0\leq t\leq P(n)$)

type b) $ID_0 = q_0 w$

INIT: $x_{10\,[q_0/w_1]}\cdot x_{20w_2}\cdots x_{n0w_n}\cdot$

$x_{n+1,0,\not b}\cdot x_{n+2,0,\not b}\cdots x_{P(n),0,\not b}$

$\Rightarrow O(P(n))$ clauses, each of length 1

type c) $ID_{P(n)}$ is accepting

ACCEPT: $\displaystyle\sum_{\substack{q\in F\\ A\in\Gamma'\\ i\in\{1,\dots,P(n)\}}} x_{i,P(n),[q/A]}$

$\Rightarrow$ 1 clause of length $O(P(n))$

type d) legal transitions

consider ID_t and ID_{t+1}

t	A_1	A_2	$\dots$	q/A_j	A_{j+1}	$\dots$	t	C_t
$t+1$	B_1	B_2	$\dots$	B_j	p/A_{j+1}			

In ID_{t+1} cell i depends only on 3 cells above it and on C_t

A_{j-1}	A_j	A_{j+1}
	B_j	

Various cases:

1) A_{j-1}, A_j, A_{j+1} are not composite symbols

then $B_j = A_j$

2) A_{j-1} is $\boxed{q/X}$ and c_t'th move in $\delta(q,X)$ is (r, Y, R)

then $B_j = \boxed{r/A_j}$

3) A_j is $\boxed{q/X}$ and c_t'th move in $\delta(q,X)$ is $(r, Y, -)$

then $B_j = Y$

4) A_j is $\boxed{q/X}$ and c_t'th move in $\delta(q,X)$ is (r, Y, L)

then $B_j = \boxed{r/A_j}$

We use clauses that forbid illegal moves: $LEGAL(t,j)$

$$\prod_{D,E,F,G,H} \left(\overline{C}_{t,D} + \overline{X}_{j-1,t,E} + \overline{X}_{j,t,F} + \overline{X}_{j+1,t,G} + \overline{X}_{j,t+1,H} \right)$$

s.t. with choice D and $\begin{array}{|c|c|c|}\hline E & F & G \\ \hline & H & \\ \hline\end{array}$ we have an illegal move

(NB. the illegal moves are those that do not correspond to 1–4 above)

$\Rightarrow O(P(n)^2)$ clauses

(since $0 \le t < P(n)$, $1 \le j \le P(n)$)

Formula F is the conjunction off all above clauses.
We can prove that $w \in \mathcal{L}(M_L)$ iff F is satisfiable.
It is easy to see that the reduction is poly-time q.e.d.

For a collection of NP-complete problems with discussion of variants see

Garey & Johnson.
Computers and Intractability. A Guide to the Theory of NP-completeness
Freeman & Co. 1979

coNP-completeness

Let us consider the complement of a problem in NP.
E.g. unsatisfiability
$$\text{UNSAT} = \{ F \mid F \text{ is a propositional formula that is not satisfiable} \}$$

Given a prop. formula F, how can we check whether $F \in \text{UNSAT}$?
- try all possible truth assignments for the vars in F
- if for none of these F evaluates to true, answer yes

Intuitively, this is very different from a problem in NP.
Note: in general, a NTM cannot answer yes to such a problem in polynomial time

<u>Definition</u>: $\text{coNP} = \{ L \mid \overline{L} = \Sigma^* \setminus L \in NP \}$

Note: many problems in coNP do not seem to be in NP.

We might conjecture NP $\neq$ coNP

This conjecture is stronger than P $\neq$ NP.

- indeed, since P = coP, we have that NP $\neq$ coNP implies P $\neq$ NP
- but we might have P $\neq$ NP, and still NP = coNP

The following result shows a strong connection between NP-complete problems and the conjecture that NP $\neq$ coNP.

<u>Theorem</u>: If for some NP-complete problem/language L we have $\bar{L} \in NP$ (i.e., $L \in coNP$), then NP = coNP.

Proof: Assume $L \in NPC$ and $\bar{L} \in NP$.

1) We show NP $\subseteq$ coNP.

Let $L' \in NP$. We show $L' \in coNP$, i.e. $\bar{L'} \in NP$.

Since $\bar{L} \in NP$, there is a poly-time NTM $N_{\bar{L}}$ s.t. $\mathcal{L}(N_{\bar{L}}) = \bar{L}$.

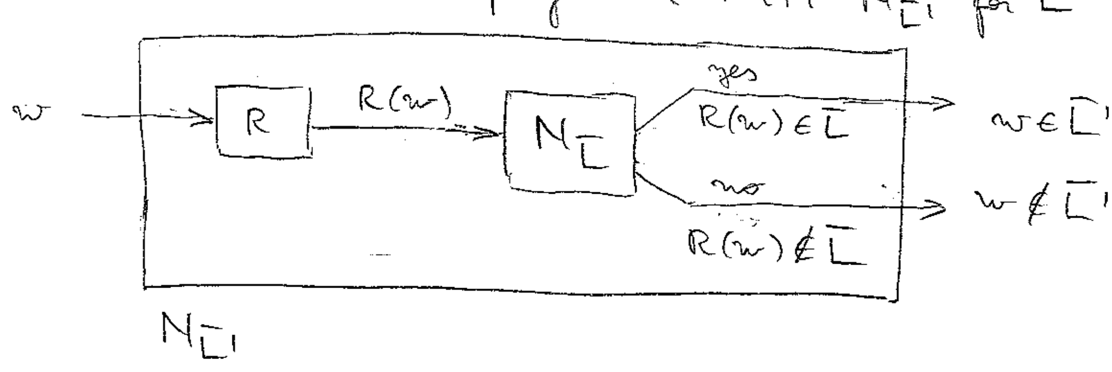
Since $L' \in NP$ and $L \in NPC$, $L' <_{poly} L$, i.e.

there is a polytime reduction R s.t.

$w \in L' \iff R(w) \in L$ i.e.

$w \in \bar{L'} \iff R(w) \in \bar{L}$

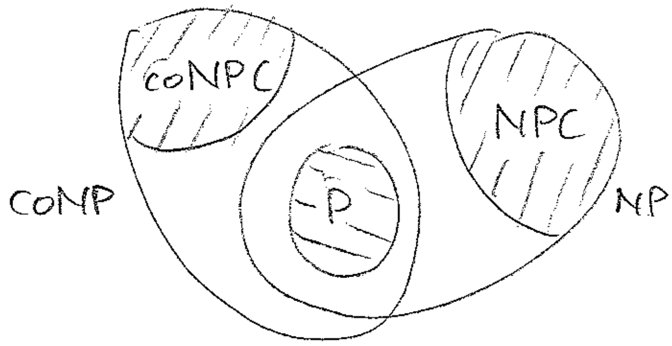
We can construct a poly-time NTM $N_{\bar{L'}}$ for $\bar{L'}$



2) coNP $\subseteq$ NP. Similar

q.e.d.

We get the following picture (assuming $P \neq NP$ and $NP \neq coNP$)



Note: It may or may not be that $P = NP \cap coNP$

The polynomial hierarchy

There are many classes of problems that are more complex than problems in NP or coNP, but not "arbitrarily" complex.

- problems related to regular expressions / languages
  - e.g. - containment of regular expressions
    - universality of reg. expr.
- games in which players alternate moves on a board, generalized to an $n \times n$ board
- problems related to special kinds of logics (that are more expressive than prop. logic, but less expressive than first-order logic)

Can we better characterize the comp. complexity of such problems?

A first step is to resort to oracle TMs. (OTMs)

We define OTMs informally.

- let g be a function $\Sigma^* \to \Sigma^*$ (which we use as an oracle)

- an OTM M_g that uses oracle g is a TM with two tapes and a special oracle state o:
  - an ordinary tape
  - an oracle tape on which the TM can read and write normally, but also consult the oracle g at the cost of a single transition

- to consult the oracle, M_g:
  - writes the input string x for g on the oracle tape
  - enters the oracle state o
  - this activates the oracle, which replaces x with $g(x)$ on the oracle tape and places the head at the beginning of $g(x)$ (all in one step)
  - after consulting the oracle, M_g leaves the oracle state, but can use $g(x)$ on the oracle tape

- M_g accepts as usual, by entering a final state

Oracles can give TMs a lot of power.

Let us consider a class $\mathcal{C}$ of TMs computing functions:

<u>Definition</u>: $P^{\mathcal{C}} = \{ L \mid L$ is accepted by a (deterministic) poly-time OTM with an oracle in $\mathcal{C}\}$

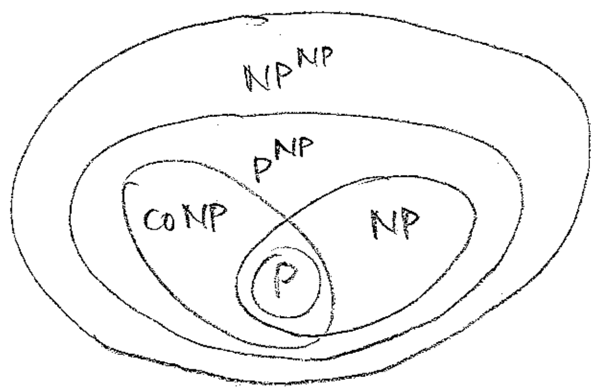
$NP^{\mathcal{C}} = \{ L \mid L$ is accepted by a non-deterministic poly-time OTM with an oracle in $\mathcal{C}\}$

Example: Consider $\mathcal{C} = NP$, i.e. the oracle is a poly-time NTM (that leaves its result on the oracle tape)

P^{NP} includes both NP and $coNP$

To solve a problem in NP (resp. coNP) a single call to the oracle is sufficient.

We get



Note: we do not know whether $P^{NP} \neq NP^{NP}$

Exploiting this idea, we can define a hierarchy of classes of greater and greater apparent difficulty:

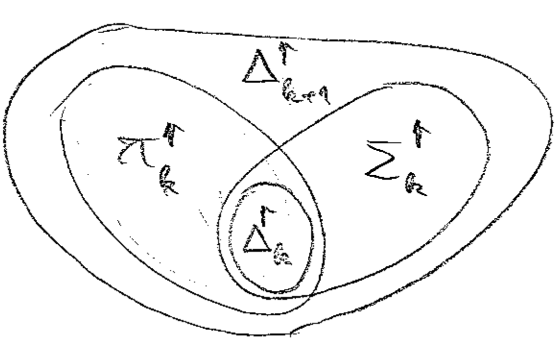
$$\Sigma_0^P = \Pi_0^P = \Delta_0^P = P$$

and for all $k \geq 0$:

$$\Delta_{k+1}^P = P^{\Sigma_k^P}$$

$$\Sigma_{k+1}^P = NP^{\Sigma_k^P}$$

$$\Pi_{k+1}^P = co\text{-}\Sigma_{k+1}^P$$



Note: $\Sigma_1^P = NP^{\Sigma_0^P} = NP^P = NP$

$\Pi_1^P = co\Sigma_1^P = coNP$

Graphically:

$$\begin{tikzcd}[column sep=small]
\Pi_3^{\uparrow} & & \Sigma_3^{\uparrow} \\
& \Delta_3^{\uparrow} \arrow[ul, dash] \arrow[ur, dash] & \\
\Pi_2^{\uparrow} \arrow[ur, dash] & & \Sigma_2^{\uparrow} \arrow[ul, dash] \\
& \Delta_2^{\uparrow} \arrow[ul, dash] \arrow[ur, dash] & \\
coNP = \Pi_1^{\uparrow} \arrow[ur, dash] & & NP = \Sigma_1^{\uparrow} \arrow[ul, dash] \\
& P = \Delta_0^{\uparrow} = \Sigma_0^{\uparrow} = \Pi_0^{\uparrow} = \Delta_1^{\uparrow} \arrow[ul, dash] \arrow[ur, dash] &
\end{tikzcd}$$

We define the polynomial hierarchy $PH = \bigcup_{j=1}^{\infty} \Sigma_j^{\uparrow}$.

It is not known whether the hierarchy is truly infinite, but if it collapses at one level, then it collapses also above.

<u>Theorem:</u> If for some $k \geq 1$, we have $\Sigma_k^{\uparrow} = \Pi_k^{\uparrow}$, then

$$\Sigma_j^{\uparrow} = \Pi_j^{\uparrow} = \Sigma_k^{\uparrow} \text{ for all } j \geq k.$$

In particular, if $P = NP$, then $NP = \Sigma_1^{\uparrow} = \Pi_1^{\uparrow}$ and so $\Sigma_j^{\uparrow} = P$ for all $j \geq 0$, i.e. $PH = P$.

We can define completeness for the versions $\Sigma_i^{\uparrow}$, $\Pi_i^{\uparrow}$, $\Delta_i^{\uparrow}$ as we did for NP-completeness.

Are there natural problems that are complete for $\Sigma_i^{\uparrow}$, $\Pi_i^{\uparrow}$?

Quantified boolean formulae (QBF)

let X be a set of boolean variables partitioned into

$X = X_1 \dot{\cup} \cdots \dot{\cup} X_i$

and let F be a propositional formula over X.

Then $\phi = \exists X_1 \forall X_2 \exists X_3 \cdots \underset{\substack{\uparrow \\ \text{either } \exists \text{ or } \forall}}{Q} X_i \; F$ is a quantified boolean formula with i alternations of quantifiers (QBF_i)

ϕ is satisfiable if:

- there is an assignment to the variables in X_1 s.t.
  for all — " — X_2
  there is — " — X_3 s.t.
  ⋮
  F is true

$QSAT_i = \{ \phi \mid \phi \text{ is a } QBF_i \text{ and } \phi \text{ is satisfiable} \}$

<u>Theorem</u>: For all $i \geq 1$ $QSAT_i$ is Σ_i^P-complete.

Note: games where players alternate i moves can be encoded as a formula of QBF_i

Space and time bounded TMs

It turns out that all problems in PH can be solved by a TM that uses at most polynomial space

PSPACE = { L | L = $\mathcal{L}$(M) for some DTM M that uses at most space that is polynomial in its input }

Examples of PSPACE-complete problems

- universality of a regular expression
- emptiness of the intersection of n DFAs (n is part of the input)
- satisfiability of quantified boolean formulas, i.e. QSAT
- board games with a polynomially bounded number of moves (existence of a winning strategy)

We said that $QSAT_i \in \Sigma_i^p$-complete
and QSAT $\in$ PSPACE-complete

In fact, we have that

Theorem: PH $\subseteq$ PSPACE

It is not known whether the inclusion is proper.

In fact, it is not known whether P = PSPACE!

We can define: NPSPACE as PSPACE, using NTM.

Theorem PSPACE = NPSPACE

Similarly, we can define

EXPTIME = $\{L \mid L = \mathcal{L}(M)$ for some exptime DTM $M\}$

EXPSPACE = $\{L \mid$ — " — expspace — " — $\}$

kEXPTIME = $\{L \mid L = \mathcal{L}(M)$ for some DTM M with running time $T(n) = \underbrace{2^{\cdot^{\cdot^{2}}}}_{k \text{ times}}{}^{O(n)}\}$

kEXPSPACE = $\{L \mid L = \mathcal{L}(M)$ for some DTM M that, on input of length n, use space that is at most $\underbrace{2^{\cdot^{\cdot^{2}}}}_{k \text{ times}}{}^{O(n)}\}$

We can define NkEXPTIME

NkEXPSPACE

as the deterministic classes, using NTM instead of DTM.

We have:

kEXPTIME

$\subseteq$ $\xleftarrow{\;?\;}$ open whether inclusion is strict

NkEXPTIME

$\subseteq$

kEXPSPACE = NkEXPSPACE

$\subseteq$

$(k+1)$EXPTIME.

Natural problems in these classes are logic related

Note: EXPTIME is the first provable intractable class, i.e. we know P $\neq$ EXPTIME.