Komplexitätstheorie

Strukturelle Komplexitätstheorie

Formale Grundlagen

Helmut Veith, veith@in.tum.de

Structural Complexity Theory

Structural Complexity Theory deals with resource-bounded computation for particular models of computation. It arose from the notion of tractable (feasible) computability.

Problems (rather than algorithms) are grouped into complexity classes by inherent complexity.

Studying the mathematical structures of complexity classes and their relationships, e.g., the question **P** =(?) **NP**, is the main interest of Structural Complexity Theory.

Basic Definitions

Definition 1 An *alphabet* is a finite set of symbols.
Unless stated explictly, it will be denoted by Σ.

Given an alphabet Σ, Σ^* denotes the set of all finite strings of elements of Σ, including the empty string ϵ.

A set $L \subseteq \Sigma^*$ is called a *language*.

Given a language L, its *complement* $\overline{L} = \Sigma^* - L$ consists of all strings not belonging to L.

Definition 2 A *class* is a set of languages. Given a class C, its complement class co-$C = \{L | \overline{L} \in C\}$.

Definition 3 If $M \subseteq D$, the *characteristic function* $\chi_M : D \rightarrow \{1, 0\}$ of M (with respect to D) is defined

$$\chi_M(x) = 1 \iff x \in M$$

and

$$\chi_M(x) = 0 \iff x \in D - M$$

Problem Representation

Objects (data) must be formally represented in order to be processed by a machine.

Definition 4 Given a set S, a representation of S in Σ^* is a suitable function $f : S \to \Sigma^*$; define that $L_S = f(S)$.

f corresponds to the internal data representation in common programming languages.

We shall identify S with L_S.

f "suitable":

- *if $x \neq y$, then $f(x) \neq f(y)$ (injective).*

- *L_S should be easily recognizable (i.e., χ_{L_S} should be "easy" to compute).*

Example: Represent an undirected (finite) graph $G = (V, E)$, (i.e., S consists of all such graphs).

Many possibilities; E. g., encode G by a string

- describing $|V|$ and the adjacency matrix

- describing $|V|, |E|$, and the vertex-edge incidence matrix

- describing $|E|$ and the list of the edges, where nodes are represented by numbers 1, ..., $|V|$.

- ...

Question: Which representation should be chosen ?

Rule: Any "natural" representation is reasonable. Such representations can be usually transformed into each other efficiently (in polynomial time).

Complexity of problem solving can be affected by

- high (exponential) overhead in encoding
- extremely succinct (highly compressed) encoding

Assumption: Numbers are represented in binary notation.

E.g., 5 is encoded by 101.

"Unary" notation (e.g., '11111' for 5) is exponentially longer!

Problem Description

Definition 5 *A* decision problem Π *consists of a set D_Π of* instances *and a subset $Y_\Pi \subseteq D_\Pi$ of* yes-instances.
The complementary problem, co-Π, has instances D_Π and yes-instances $D_\Pi - Y_\Pi$.

Assumption: encodings of generic instances can be "easily" recognized by a TM (i.e. $\chi_{L_{D_\Pi}}$ is easy to compute).

Standard problem description:

INSTANCE: A generic problem instance I, i.e. $I \in D_\Pi$.
QUESTION: yes-no question "$I \in Y_\Pi$?".

Example: Satisfiability problem

SAT:
INSTANCE: A well-formed Boolean (propositional) formula F.
QUESTION: Is F satisfiable?

E.g., $F = (x_1 \wedge \neg x_2) \vee \neg(x_3 \vee x_2)$

Search problems: a solution (value) for a problem instance is sought.

Example: Compute the greatest common divisor $gcd(n, m)$ of two integers n, m.

Decision problems can be seen as special search problems (compute "yes" or "no").

Example: FSAT

Given a Boolean formula F, find a truth assignment σ to the variables that satisfies F (i.e. F has value 1).

One can often solve a search problem "easily" with a subroutine for a suitable associated decision problem.

Example: Solve **FSAT** using associated decision problem **SAT**.

Traditionally, complexity theory considers decision problems.

Turing Machines

The Turing Machine is the computation model we shall define complexity upon.

Definition 6 A *deterministic Turing machine (DTM)* with k tapes is a five-tuple

$$M =< Q, \Sigma, \delta, q_0, F >$$

where

1. Q is the finite set of internal states;
2. Σ is the tape alphabet;
3. $q_0 \in Q$ is the initial state;
4. $F \subseteq Q$ is the set of final states, and
5. $\delta : Q \times \Sigma^k \to \Sigma^{k-1} \times Q \times \{-1, 0, +1\}^k$ is a partial function called the transition function of M.

Remark *The first tape (input tape) is assumed to be read-only.*

Definition 7 If M is a k-tape TM, a *configuration* of M is a $k+1$-tuple

$$(q, x_1, x_2, \ldots, x_{k-1}, x_k)$$

where q is the current state of M, and each $x_j \in \Sigma^*\#\Sigma^*$ represents the current contents of the j^{th} tape. The symbol "#" is supposed not to be in Σ, and precedes the position of the tape head.

Definition 8 The *initial configuration* of a TM M on an input w is $(q_0, \#w, \#, \ldots, \#)$.

Definition 9 An accepting configuration is a configuration $(q, w_1, \ldots, w_k)$ where $q \in F$

It is usually sufficient to use the following 1-tape model, where the unique work tape is also used as input tape:

Definition 10 A *deterministic Turing machine (DTM)* is a five-tuple

$$M =< Q, \Sigma, \delta, q_0, F >$$

where

1. Q is the finite set of internal states;
2. Σ is the tape alphabet;
3. $q_0 \in Q$ is the initial state;
4. $F \subseteq Q$ is the set of final states, and
5. $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, +1\}$ is a partial function called the transition function of M.

Definition 11 Given a TM M, a *computation* is a sequence of configurations which

1. obeys the transition function
2. starts with the initial configuration
3. ends in a configuration, where no more step can be performed

Definition 12 An input word $w \in \Sigma^*$ is *accepted* by a TM M, if the computation of M on input w **halts** in an accepting configuration. The language accepted by the TM M, denoted by $L(M)$, is the set of words accepted by M.

The Invariance Thesis

To obtain general complexity results, we rely on the

Invariance Thesis
All common computation models simulate each other with polynomial time overhead.

Example: A k-tape TM simulates a Random Access Machine in quadratic time, while RAM simulation of a TM costs only a logarithmic factor.

Therefore, we shall describe algorithms informally or in a legible Pascal-style formalism rather than by TMs.

Nondeterministic Turing Machines

Definition 13 A *nondeterministic Turing machine (NDTM)* with k Tapes is a five-tuple

$$M =< Q, \Sigma, \delta, q_0, F >$$

where

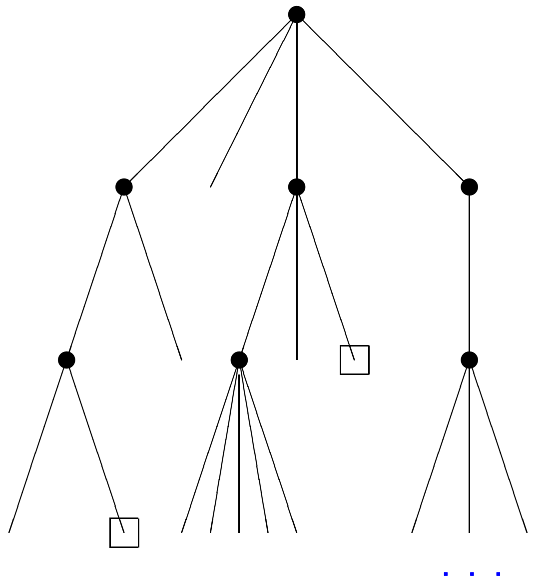
1. Q is the finite set of internal states;
2. Σ is the tape alphabet;
3. $q_0 \in Q$ is the initial state;
4. $F \subseteq Q$ is the set of final states, and
5. $\delta : Q \times \Sigma^k \to \mathcal{P}(\Sigma^{k-1} \times Q \times \{-1, 0, +1\}^k)$ is a partial function called the transition function of M where $\mathcal{P}(A)$ denotes the power set of a set A.

Note that the only difference to our former definition is the transition function. In each step the NDTM chooses within the set of possible transitions. Therefore we have to modify our notion of word acceptance:

Definition 14 An input word $w \in \Sigma^*$ is *accepted* by a NDTM M, if there exists a computation which ends in an accepting configuration.

The language accepted by the machine M, denoted by $L(M)$, is the set of words accepted by M.

nondeterministic computation tree



... accept

Alternative nondeterministic TM

A 'guess & check' TM operates in two phases:

1. nondeterministically write a sequence S of $\{0,1\}^*$ on a prespecified tape
2. If 1. stops, start deterministic computation (which may access S)

Such Guess & Check-TMs are as powerful as the previously defined NDTMs:

Simulation of NDTM by Guess & Check-TM:

Interpret S as an accepting computation of the NDTM on input w; accept if S is an accepting computation.

Simulation of Guess & Check-TM by NDTM:

Easy. "Guessing states" S_0, S_1 for phase (1); states for phase (2) + "subprogram" for lookup of S.

Simluation is quite efficiently possible.

Oracle Turing Machines

Model computations with calls to subprograms (Boolean functions in Pascal). The machine has a fixed language (*oracle set*) $O \subseteq \Sigma^*$ modelling the subroutine.

Definition 15 An *oracle Turing machine (DOTM or NOTM)* is a multitape Turing machine (DTM or NDTM) with one designated tape, the *oracle tape* or *query tape*, and where $\{QUERY, YES, NO\} \subseteq Q$.

If the machine is in state $QUERY$, a call to the oracle is performed:

1) If the string w on the query tape is in O, change state to YES, else to NO.

2) erase the content of the query tape.

An oracle call counts as a single step. Thus, the OTM can evaluate the characteristic function χ_O in unit time.

Example: Use the language of satisfiable Boolean formulas as an oracle.

Definition 16 The language accepted by an OTM M *relative to an oracle set* A, denoted by $L(M, A)$, is the set of all words accepted by the OTM using the oracle set A.

An oracle for language A is as good as an oracle for $\overline{A}$.

Theorem 17 *Given an OTM M with oracle set A, there is an OTM M' with oracle set $\overline{A}$ accepting the same language.*

Proof: Define transition function δ' of M' as δ of M but exchange transitions for YES and NO:

$$\delta'(NO, \ldots) = \delta(YES, \ldots), \quad \delta'(YES, \ldots) = \delta(NO, \ldots)$$

M' acts on the result of a subprogram call opposite to M.

$\Rightarrow$ (double complement) M' basically acts like M.

NONDETERMINISTIC PROGRAMS

We extend pseudocode programs by nondeterministic commands:

`guess`$(v_1, \ldots, v_n)$:
assign nondeterministically values to $v_1, \ldots, v_n$.

`choice`$(\mathtt{stat}_1 | \mathtt{stat}_2 | \ldots | \mathtt{stat}_n)$:
execute nondeterministically one of the statements $\mathtt{stat}_1, \ldots, \mathtt{stat}_n$.

Additional statements `succeed` and `fail`:

`succeed`:
if any computation branch reaches this point, stop and accept.

`fail`:
stop the computation.

Example: Programs for SAT

Let $Var(F) = \{x_1, \ldots, x_n\}$ be the variables occurring in the Boolean formula F. : Is E satisfiable?

DTM:

```
For each truth assignment τ to Var(F) do /*
2^n times!  */
  if τ satisfies E then succeed
fail
```

NDTM:

```
For i := 1 to n do
  choice(τ(x_i) := true | τ(x_i) := false);
if τ satisfies E then succeed;
fail
```

alternative guess and check algorithm:

```
guess(τ); /* guess a truth assignment */
if τ satisfies E then succeed;
fail
```


Time and Space Bounded Computation

Define classes of problems solvable by some machine within certain time bound.

Definition: *The size of problem instance I, $|I|$, is the number of symbols of the word x representing I. (I.e., $x = f(I)$.)*

Definition: *The running time of a DTM M on input I, $time_M(I)$, is the number of steps until M halts. (Undefined if M does not halt.)*

The running time of a NDTM M on input I, $time_M(I)$, is the minimum number of steps over all accepting computations and 1, if no accepting computation exists.

Analogous definition for OTMs.

For integer $n \geq 0$, the running time of machine M for n, $time_M(n)$, is the maximum over all $time_M(I)$ where $|I| = n$.

Definition 18 For a function $t(n) \geq n + 1$,

1) DTIME$(t) =$ class of all sets accepted by DTMs M with $time_M(n) \leq t(n)$, for $n \geq 0$.

2) NTIME$(t) =$ class of all sets accepted by NDTMs with $time_M(n) \leq t(n)$, for $n \geq 0$.

Remarks

- *t should be time-constructible.*

$$\begin{aligned}
\mathbf{P} &= \bigcup_{i \geq 0} \mathrm{DTIME}(n^i) \\
\mathbf{NP} &= \bigcup_{i \geq 0} \mathrm{NTIME}(n^i) \\
\mathbf{EXPTIME} &= \bigcup_{i \geq 0} \mathrm{DTIME}(2^{n^i}) \\
\mathbf{NEXPTIME} &= \bigcup_{i \geq 0} \mathrm{NTIME}(2^{n^i})
\end{aligned}$$

Similar definitions for space- bounded computations.

Definition:

The space used by a DTM M on input I, $space_M(I)$, is the number of different cells of the work tape visited until M halts. (Undefined if M does not halt.)

The space used by a NDTM M on input I, $space_M(I)$, is the minimum number of different work tape cells visited in an accepting computation, over all accepting computations, and 1, if no accepting computation exists.

Analogous definition for OTMs

assumption: oracle tape counts as work tape (exception: sublinear work space restrictions.)

Note: if the oracle space is unrestricted, then

$$\mathbf{PSPACE}^{\mathbf{P}} = \mathbf{EXPTIME}.$$

For integer $n \geq 0$, the space used by machine M for n, $space_M(n)$, is the maximum over all $space_M(I)$ where $|I| = n$.

Definition 19 For a function $t(n) \geq 1$,

1) DSPACE$(t) =$ class of all sets accepted by DTMs M with $space_M(n) \leq t(n)$, for $n \geq 0$.

2) NSPACE$(t) =$ class of all sets accepted by NDTMs with $space_M(n) \leq t(n)$, for $n \geq 0$.

Remarks

- *t should be space-constructible.*

$$\begin{aligned}
\mathbf{LOG} &= \bigcup_{c\geq 1} \text{SPACE}(c \cdot \log n) \\
\mathbf{NLOG} &= \bigcup_{c\geq 1} \text{NSPACE}(c \cdot \log n) \\
\mathbf{PSPACE} &= \bigcup_{i\geq 0} \text{SPACE}(n^i) \\
\mathbf{NPSPACE} &= \bigcup_{i\geq 0} \text{NSPACE}(n^i)
\end{aligned}$$

Properties & Relationships

- Each deterministic class is closed under complementation.
- Each deterministic class is included in its nondeterministic counterpart.
- $\mathbf{P} \subseteq_{=?} \mathbf{NP} \subseteq_{=?} \mathbf{PSPACE}$
- $\mathbf{PSPACE} = \mathbf{NPSPACE}$
- $\mathbf{LOG} \subseteq_{=?} \mathbf{NLOG} \subseteq_{=?} \mathbf{P} \subseteq_{=?} \mathbf{PSPACE}$
- $\mathbf{NLOG} \subset \mathbf{PSPACE} \subseteq_{=?} \mathbf{EXPTIME}$
- $\mathbf{P} \subset \mathbf{EXPTIME}$
- $\mathbf{NP} \subset \mathbf{NEXPTIME}$

$\mathcal{C}$-Completeness

Polynomial Reductions

Definition 20 Given two languages A_1 and A_2, A_1 is *polynomial time many-one reducible (m-reducible or Karp reducible)* to A_2 ($A_1 \leq_m^P A_2$) iff there exists a function $f\colon \Sigma^* \to \Sigma^*$, such that $f(w)$ is computable in time polynomial in $|w|$ and $\chi_{A_1}(x) = \chi_{A_2}(f(x))$ for all $x \in \Sigma^*$.

Intuitively, $A \leq_m^P B$ means that A is not harder than B in the following sense: An algorithm for B is sufficient for solving A as well with only polynomial time overhead for encoding A in terms of B.
One speaks of many-one reducible, *since several (often isomorphic) instances of A may be mapped on the same instance of B.*

Important properties of $\leq_m^P$:

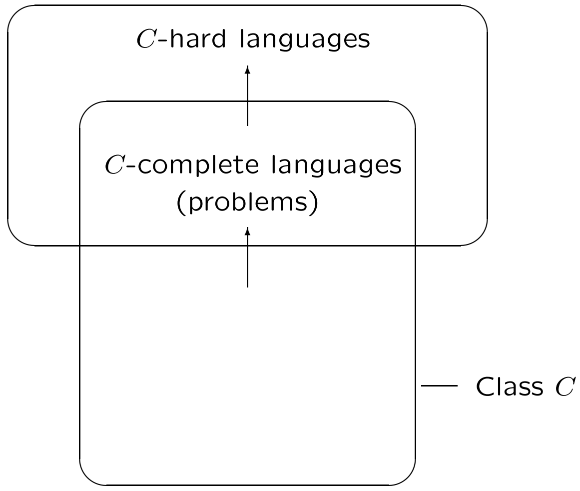
- $\leq_m^P$ is reflexive and transitive, i.e. a *preorder*
- $A \leq_m^P B \iff \overline{A} \leq_m^P \overline{B}$ (proof via the characteristic function)

Hardness and Completeness

Definition 21
Given a class C, a set A is *m-hard for C* (or *C-hard*) if for every $B \in C$, it holds that $B \leq_m^P A$.

A set A is *m-complete for C* (or C-complete), iff it is m-hard for C and $A \in C$.

The Notion of Completeness:



$\longrightarrow$: polynomial-time many one reducible ($\leq_m^P$)

Trivial consequences:

- A is C-hard, $A \leq_m^P B \Rightarrow B$ is C-hard
- A is C-complete, $B \in C$, $A \leq_m^P B \Rightarrow B$ is C-complete.
- A is C-hard $\Rightarrow \overline{A}$ is co-C-hard

Strategy to prove C-completeness of A:

(1) prove $A \in C$

(2) show $S \leq_m^P A$, where S is known to be C-complete.

Satisfiability Problems

Theorem 22 (Cook/Levin)
SAT is **NP***-complete.*

CNF-SAT:
INSTANCE: A Boolean formula E in conjunctive normal form
QUESTION: Is E satisfiable?

k-CNFSAT (k-SAT)
INSTANCE: A Boolean formula E in k-CNF, i.e., every clause consists of k literals.
QUESTION: Is E satisfiable?

Theorem 23 *CNF-SAT is* **NP***-complete.*

Theorem 24 *3SAT is* **NP***-complete.*

The proofs are based on constructing equivalent formulas in a boolean algebra in polynomial time. 2SAT, however, is in **P***.*

A Classical Reduction

Vertex Cover

Definition 25 Given an undirected graph $G = (V, E)$, $V' \subseteq V$ is a *vertex cover* (vc) iff $\forall (v, w) \in E : v \in V'$ or $w \in V'$.

VC:
INSTANCE: An undirected graph $G = (V, E)$ and an integer $K \leq |V|$
QUESTION: Is there a vertex cover of size $\leq K$?

Theorem 26 *VC is* **NP***-complete.*

Proof:
Membership: Guess a set $V' \subseteq V$ of vertices of size $\leq k$ and test in polynomial time if it is a vertex cover.

Proof (cont.) VC is **NP**-complete

Hardness: We show that 3SAT $\leq_m^P$ VC:

Let $F = F_1 \wedge F_2 \wedge \ldots \wedge F_q$, where $F_i = (\alpha_{i,1} \vee \alpha_{i,2} \vee \alpha_{i,3})$, be an instance of 3SAT.

Construct a graph $G = (V, E)$, where

$$\begin{aligned} V &= \{v_{i,j} \mid 1 \leq i \leq q, j = 1, 2, 3\} \\ E &= \{\, \{v_{i,j}, v_{i,k}\} \mid j \neq k\} \\ &\quad \cup \{\, \{v_{i,j}, v_{k,l}\} \mid \alpha_{i,j} \equiv \neg \alpha_{k,l}\} \end{aligned}$$

The $v_{i,j}$ represent the literals $\alpha_{i,j}$; the edges connect all pairs α_{i,j_1}, α_{i,j_2} of literals from the same clause F_i and all pairs of opposite literals (x and $\neg x$).

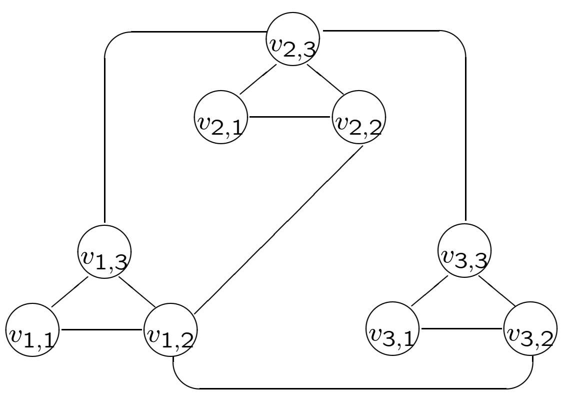
Intuition: vc C corresponds to literals $\alpha_{i,j}$ that are discarded ("deleted"). Complement $V - C$ describes a set of literals that can simultaneously have value true.

Example:

Instance of 3SAT:

$$F = (a \vee b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge (a \vee \neg b \vee c)$$

Constructed Instance of VC ($q = 3$):



Vertex cover $C = \{\, v_{1,1}, v_{1,3}, v_{2,2}, v_{2,3}, v_{3,1}, v_{3,2}\,\}$
$\Leftrightarrow$ reduced formula $F' = b \wedge a \wedge c$
$\Leftrightarrow$ sat.ass. σ where $b = \mathit{true}$, $a = \mathit{true}$, $c = \mathit{true}$

Proof (cont.): VC is **NP**-complete

Claim:
G has a vc of size $\leq 2q \iff$ F is satisfiable

$\Leftarrow$: choose a truth assignment σ to the variables that makes F true.

Select from every clause F_i one literal $\alpha_{i,\sigma(i)}$ whose value in σ is true.

The set of $2q$ vertices $C = \{v_{i,j} \mid j \neq \sigma(i)\}$ forms a vc.
Indeed, C includes

1) two vertices v_{i,j_1}, v_{i,j_2} for each $i = 1, \ldots, q$, and

2) at least one of the vertices $v_{i,k}$ and $v_{j,l}$ for each pair of opposite literals $\alpha_{i,k}$ and $\alpha_{j,l}$ since only one of $\alpha_{i,k}$ and $\alpha_{j,l}$ can be true in σ (but not both).

Claim: G has a vc C of size $\leq 2q \iff$ F is satisfiable

$\Rightarrow$: The vc C must contain:

1) at least one of of the vertices $v_{i,k}$ and $v_{j,l}$ for each pair of opposite literals $\alpha_{i,k}$ and $\alpha_{j,l}$. (The edge $\{v_{i,k}, v_{j,l}\}$ must be covered.)

2) exactly two of the vertices $v_{i,1}$, $v_{i,2}$, and $v_{i,3}$ for each $i = 1, \ldots, q$. Indeed, the edges $\{v_{i,1}, v_{i,2}\}$, $\{v_{i,2}, v_{i,3}\}$, $\{v_{i,1}, v_{i,3}\}$ must be covered; on the other hand, $|C| \leq 2q$.

Define a truth value assignment σ by

$$\sigma(x) = \begin{cases} \mathit{true}, & \alpha_{i,j} = x \text{ and } v_{i,j} \notin C \\ & \text{for some } i \text{ and } j; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

σ is well-defined and makes F true:
By 1), σ makes *each* $\alpha_{i,j}$ true s.t. $v_{i,j} \notin C$;
by 2), at least one such $\alpha_{i,j}$ exists for each $i = 1, \ldots, q$.

The reduction is computable in polynomial time. Thus, VC is **NP**-complete.

Co-NP-Completeness

Recall: co-$C = \{\overline{L} \mid L \in C\}$; thus co-**NP**$= \{\overline{L} \mid L \in \mathrm{NP}\}$.

Theorem 27 *Given a set L, L is* ***NP****-complete iff $\overline{L}$ is* ***co-NP****-complete.*
A decision problem Π is ***NP****-complete iff co-Π is* ***co-NP****-complete.*

We obtain a **co-NP**-complete problem:

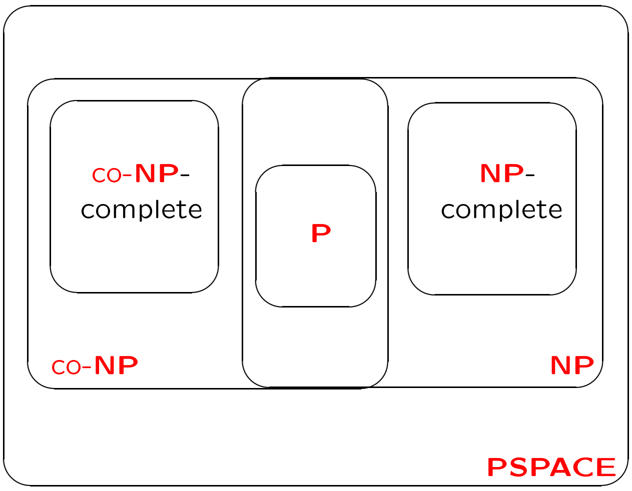
UNSAT:
INSTANCE: A Boolean formula E
QUESTION: Is E unsatisfiable?

Theorem 28 *UNSAT is* ***co-NP****-complete.*

Proof: E is unsatisfiable iff E is not a Yes-instance of SAT. Thus, UNSAT is the complementary problem of SAT (co-SAT).

Another **co-NP**-complete problem:
TAUT:
INSTANCE: A Boolean formula E
QUESTION: Is E a tautology?

The world of **NP** and co-**NP**:



(Assuming $\mathrm{P} \neq \mathrm{NP}$ and $\mathrm{NP} \neq$ co-NP)

The Polynomial Hierarchy

NP-Turing-Hardness

The polynomial reductions used in proving **NP**-completeness are akin to exhibiting (polynomial) algorithms that call a subroutine for a known **NP**-complete problem A. If the subroutine takes only polynomial time, the whole algorithm is polynomial.

This remains true if we would allow multiple calls to the subroutine for A. This idea is used in oracle computations. An OTM may consult an oracle (a subroutine) for another problem which answers in unit time.

Turing Reductions

Definition 29 A problem X is *polynomial time Turing reducible (Cook reducible)* to a problem Y, $X \leq_T Y$, if there is a polynomial DOTM for X with access to an oracle for Y.

A problem X is **NP**-*Turing-hard*, if there is an **NP**-complete problem Y s.t. $Y \leq_T X$

A problem X is **NP**-*easy*, if for some problem $Y \in \mathbf{NP}$, $X \leq_T Y$. (We call such problems easy, since they are not much harder than **NP**-complete problems).

Example: Given a satisfiable formula F, find an actually satisfying truth assignment.

A problem X is *polynomial time nondeterministic Turing reducible* to a problem Y, $X \leq_{NT} Y$, if there is a polynomial NOTM for X with access to an oracle for Y.

These notions can be extended to whole classes of problems:

Definition 30 Let C be a class of languages.

$$\begin{aligned} \mathbf{P}^C &= \{L \mid \exists L' \in C \colon L \leq_T L'\} \\ \mathbf{NP}^C &= \{L \mid \exists L' \in C \colon L \leq_{NT} L'\} \end{aligned}$$

Alternatively, one could define:

Definition 31 Given a complexity class C, $\mathbf{P}^C$ denotes the class of languages acceptable by a polynomial time bounded DOTM using an oracle for C.

Definition 32 Given a complexity class C, $\mathbf{NP}^C$ denotes the class of languages acceptable by a polynomial time bounded NOTM using an oracle for C.

Remark: Having an oracle for A is equivalent to having one for $\overline{A}$. Hence, $\mathbf{P}^A = \mathbf{P}^{\overline{A}}$, e.g., $\mathbf{P}^{\mathbf{NP}} = \mathbf{P}^{\text{co-}\mathbf{NP}}$.

The Polynomial Hierarchy

Observe that this process of defining new classes in terms of old ones can be iterated:

Definition 33 The *polynomial hierarchy* consists of classes Σ_k^{P}, Π_k^{P}, Δ_k^{P}, defined as follows:

$$\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \Delta_0^{\mathsf{P}} = \mathbf{P}$$

and for $k \geq 0$:

$$\begin{aligned} \Delta_{k+1}^{\mathsf{P}} &= \mathbf{P}^{\Sigma_k^{\mathsf{P}}} \\ \Sigma_{k+1}^{\mathsf{P}} &= \mathbf{NP}^{\Sigma_k^{\mathsf{P}}} \\ \Pi_{k+1}^{\mathsf{P}} &= \text{co-}\Sigma_{k+1}^{\mathsf{P}} \end{aligned}$$

The *polynomial hierarchy* **PH** is defined as:

$$\mathbf{PH} = \bigcup_{k=0}^{\infty} \Sigma_k^{\mathsf{P}}$$

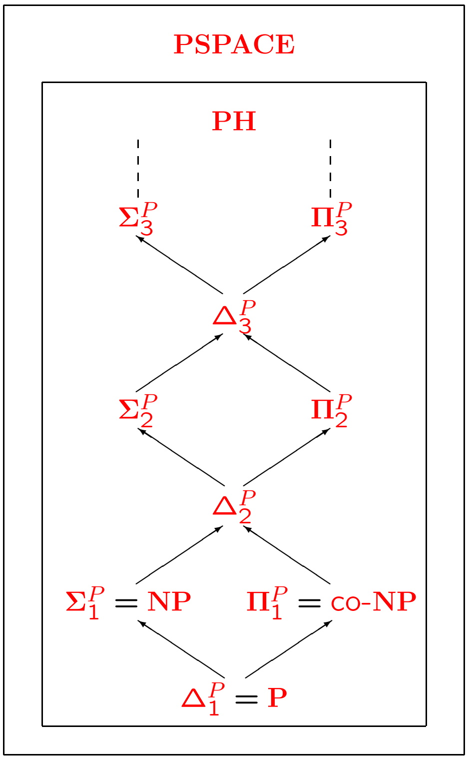
In particular:

$$\begin{aligned} \Delta_1^{\mathsf{P}} &= \mathbf{P} & \Delta_2^{\mathsf{P}} &= \mathbf{P}^{\mathbf{NP}} \\ \Sigma_1^{\mathsf{P}} &= \mathbf{NP} & \Sigma_2^{\mathsf{P}} &= \mathbf{NP}^{\mathbf{NP}} \\ \Pi_1^{\mathsf{P}} &= \text{co-}\mathbf{NP} & \Pi_2^{\mathsf{P}} &= \text{co-}\mathbf{NP}^{\mathbf{NP}} \end{aligned}$$

Theorem 34 *The following containment relationships hold:*

$$\Delta_k^{\mathsf{P}} \subseteq \Sigma_k^{\mathsf{P}} \cap \Pi_k^{\mathsf{P}} \qquad \Sigma_k^{\mathsf{P}} \cup \Pi_k^{\mathsf{P}} \subseteq \Delta_{k+1}^{\mathsf{P}}$$

The Structure of PH



Complete problems for Σ_k^{P} and Π_k^{P}

Quantified Boolean Formulas

QBF$_{k,\exists}$:

INSTANCE: A well-formed Boolean expression E in the variables $x_{i,j}$, $1 \le i \le k$, $1 \le j \le m_i$ for $m_1, \ldots, m_k \ge 1$.

QUESTION: Is the quantified Boolean expression

$$(\exists \vec{x}_1)(\forall \vec{x}_2)(\exists \vec{x}_3) \cdots (\mathsf{Q}_k \vec{x}_k) E$$

true, where $\vec{x}_i = x_{i,1}, \ldots, x_{i,m_i}$ and Q_k is $\exists$ if k is odd and $\forall$ otherwise?

This problem is easily seen to lie in Σ_k^{P}, and is in fact complete for this class. The dual problem **QBF**$_{k,\forall}$ is complete for Π_k^{P}.

Remark: $\mathbf{QBF} = \bigcup_i (\mathbf{QBF}_{i,\exists} \cup \mathbf{QBF}_{i,\forall})$ is complete for **PSPACE**.

A complete problem for Δ_k^{P}

Theorem 35 *The following problem is complete for* Δ_2^{P}*:*

MSA$_{\mathsf{odd}}$:

INSTANCE: A Boolean formula E in the variables x_1, . . . , x_n

QUESTION: Is $x_n = true$ in the lexicographically maximum truth assignment to $x_1 \ldots x_n$ that satisfies E ?

Generalization to $\Delta_{k+2}^{\mathsf{P}}$-complete problem:

INSTANCE: A quantified Boolean expression

$$\forall \vec{p}_1 \exists \vec{p}_2 \cdots Q_k \vec{p}_k\, E(\vec{x}, \vec{p}_1, \ldots, \vec{p}_k)$$

with free variables $\vec{x} = x_1, \ldots, x_n$.

QUESTION: Is $x_n = true$ in the lex. maximum truth assignment ϕ to $\vec{x}$ such that $F[\vec{x}/\phi(\vec{x})]$ is valid?

The Class PSPACE

Recall that the class **PSPACE** is the set of all languages recognizable by polynomial-*space* bounded TMs.

$$\mathbf{PSPACE} = \bigcup_{i \ge 0} \mathsf{DSPACE}(n^i)$$

Every problem solvable in polynomial time is also solvable in polynomial space, since no more tape cells can be used than steps are performed.

Theorem 36 $\mathbf{PH} \subseteq \mathbf{PSPACE}$

We prove that $\Sigma_k^{\mathsf{P}} \subseteq \mathbf{PSPACE}$ by induction on k:

(Basis) $\Sigma_0^{\mathsf{P}} = \mathbf{P} \subseteq \mathbf{PSPACE}$.

(Induction) Assume that $\Sigma_k^{\mathrm{P}} \subseteq \mathbf{PSPACE}$.
Show: $\Sigma_{k+1}^{\mathrm{P}} \subseteq \mathbf{PSPACE}$.

We have $\Sigma_{k+1}^{\mathrm{P}} = \mathbf{NP}^{\Sigma_k^{\mathrm{P}}} \subseteq \mathbf{NP}^{\mathbf{PSPACE}}$.

Any polynomial-time bounded computation of a NOTM can be written out in polynomial space. By cycling through all possible such computations, an accepting computation can be found (if one exists).

Thus, $\mathbf{NP}^{\mathbf{PSPACE}} \subseteq \mathbf{PSPACE}^{\mathbf{PSPACE}}$

($\mathbf{PSPACE}^{\mathbf{PSPACE}}$ = languages accepted by OTM with oracle set from $\mathbf{PSPACE}$ in polynomial space.)

Clearly, $\mathbf{PSPACE}^{\mathbf{PSPACE}} = \mathbf{PSPACE}$.

Hence, $\Sigma_{k+1}^{\mathrm{P}} \subseteq \mathbf{PSPACE}$.

PSPACE-Complete Problems

QBF:
INSTANCE: A quantified Boolean formula

$$F = (\mathrm{Q}_1 x_1) \ldots (\mathrm{Q}_n x_n) E$$

where Q_i is either $\exists$ or $\forall$ and E is a Boolean expression in $x_1, \ldots, x_n$
QUESTION: Is F true?

Membership: Recursive algorithm using

$$(\exists x_1) \ldots (\mathrm{Q}_n x_n) E \equiv (\mathrm{Q}_2 x_2) \ldots E|_{x_1=0} \vee (\mathrm{Q}_2 x_2) \ldots E|_{x_1=1}$$

$$(\forall x_1) \ldots (\mathrm{Q}_n x_n) E \equiv (\mathrm{Q}_2 x_2) \ldots E|_{x_1=0} \wedge (\mathrm{Q}_2 x_2) \ldots E|_{x_1=1}$$

Simulate recursion by a stack; algorithm runs in space quadratic in the size of input F.

Many 2-person games (e.g., generalized GO) are **PSPACE**-complete.

Literatur

M.R.Garey, D.S.Johson: Computers and Intractability. W.H.Friedman and Company, 1979.

C.Papadimitriou: Computational Complexity. Addison-Wesley, 1994.

U.Schöning: Theoretische Informatik - kurzgefaßt. Spektrum Akademischer Verlag, 1994.

J.L.Balcazar, J.Diaz, J.Gabarro: Structural Complexity I and II. Springer-Verlag 1988.

J.van Leeuwen, ed.: Handbook of Theoretical Computer Science, Volume A. MIT Press/Elsevier, 1990:

Chapter 1 (van Emde Boas): Machine Models and Simulation
Chapter 2 (Johson): A Catalog of Complexity Classes
Chapter 14 (Boppana, Sipser): The Complexity of Finite Functions

+ viele andere Kapitel