Complexity relative to an oracle

Evgenij Thorstensen

V18

Oracles

"Our world can be characterized by the cruel fact that no computation whatsoever is free."

But what if some problems could be solved for free?

This subject is known as *relative computation*, that is, relative to the problem solvable for free.

Oracle machines

An oracle is a special black box that can solve any instance of a given problem in one step (decide $w \in \mathrm{L}$ in one step for any w).

A DTM or NTM with an oracle has a special transition that solves an appropriate problem on the tape.

For example, oracle for SAT, or for TQBF.

Can extend to oracle for a class: An oracle for P can solve any problem in P in one step.

Oracle machines, properly

An oracle TM (DTM or NTM) M^A has an extra read/write *query tape*, and three special states q_A, q_Y, and q_N.

Machine works as usual, except for the state q_A.

When the machine enters q_A, it moves to q_Y or q_N depending on whether the string w on the query tape is in A or not — in one step.

Time complexity measured as usual; space complexity has multiple definitions.

The power of oracles

Given an oracle, complexity classes are defined as usual. P^{SAT} is the class of problems

- decidable in polynomial time
- on a DTM with a SAT oracle.

Likewise NP^{SAT}, or even P^{NP}, or $\mathsf{NP}^{\mathsf{NP}}$.

Observation: $\mathsf{P}^{SAT} = \mathsf{P}^{\mathsf{NP}}$

Some observations

$\mathsf{P}^{\mathsf{P}} = \mathsf{P}$, but $\mathsf{NP}^{\mathsf{NP}} \supseteq \mathsf{NP}$.

This property is being self-low: A class A is low for itself if $A^A = A$.

Self-low classes can use their problems as subroutines.

Some observations

$\mathsf{P}^{\mathsf{P}} = \mathsf{P}$, but $\mathsf{NP}^{\mathsf{NP}} \supseteq \mathsf{NP}$.

This property is being self-low: A class A is low for itself if $A^A = A$.

Self-low classes can use their problems as subroutines.

For P^{P}, can simulate oracle queries in polynomial time.

However, $\mathsf{coNP} \subseteq \mathsf{NP}^{\mathsf{NP}}$ — can query the oracle, use answer!

Polynomial hierarchy

Recall TQBF with bounded quantifier alteration: we are only allowed a fixed number of quantifier changes.

Consider the formula $\forall X \exists Y \phi(X, Y)$.

Such formulas belong to coNP^{NP}.

Guess assignment to X, for each assignment call oracle about satisfiability of $\exists Y \phi$.

If formula is not true, exists a certificate (assignment to X), hence this is coNP with an NP oracle.

Polynomial hierarchy

We define $\Delta_0 = \Sigma_0 = \Pi_0 = P$.

- $\Delta_{i+1} = P^{\Sigma_i}$
- $\Sigma_{i+1} = NP^{\Sigma_i}$
- $\Pi_{i+1} = coNP^{\Sigma_i}$

Could define them using Π instead of Σ — oracles for NP and coNP are equivalent.

Easy to see that all of these are under PSPACE, and that $P = NP$ collapses them all to P.

$$PH = \bigcup_{k \in \mathbb{N}} \Sigma_i \cup \Delta_i \cup \Pi_i$$

Polynomial hierarchy, a picture

Courtesy of wikipedia. Arrows are inclusions.

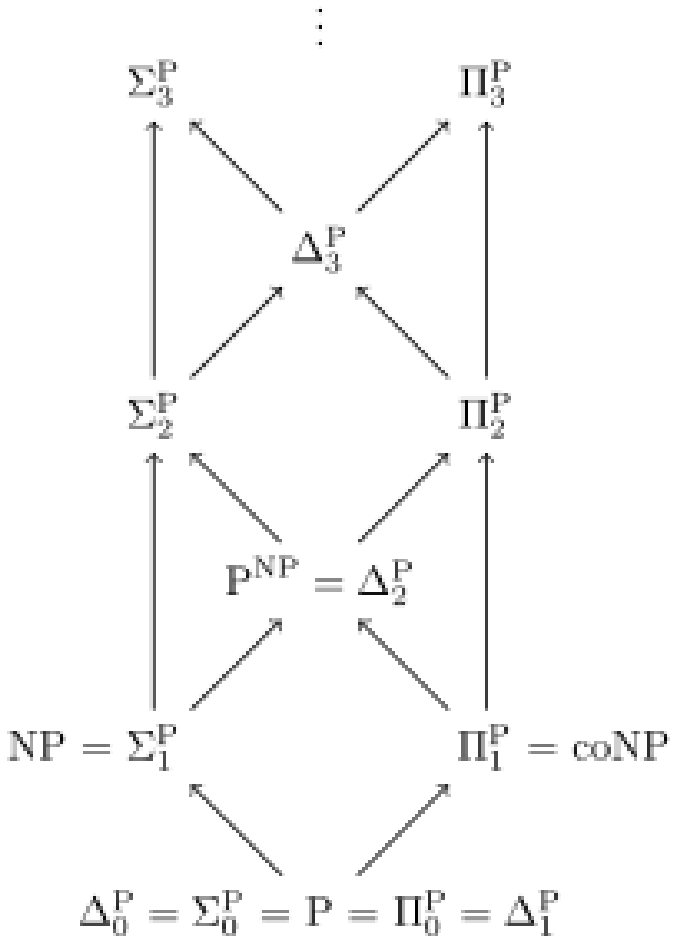

Figure: Polynomial hierarchy

PH gives a reason for NP $\neq$ coNP

If NP $=$ coNP, then PH $=$ NP $= \Sigma_1$.

Generalizes to PH $= \Sigma_i$ whenever $\Sigma_i = \Pi_i$.

Σ_2 is a practically interesting class.

Occurs in NP problems with minimality requirements over *all* solutions rather than some.

Even more oracles

How about a DTM M^H with a *halting oracle*?

M^H can solve the original halting problem: Given M, w, does $M(w)$ halt?

However, DTMs with a halting oracle have their own halting problem.

This problem they cannot solve — original proof goes through completely.

Assume a decider, feed it to itself, output wrong answer.

Relativization

The proof of the undecidability of the halting problem *relativizes* — it works relative to any oracle whatsoever.

Proof is based on a diagonalization argument.

There are other arguments that also relativize.

A problem, like $\mathsf{P} = \mathsf{NP}$, relativizes if $\mathsf{P}^A = \mathsf{NP}^A$ for any A.

Some problems, however, very much do not relativize.

A summary of results

Theorem (Baker, Gill, Solovay, 1975)

There exist oracles A *and* B *such that* $\mathsf{P}^{\mathrm{A}} = \mathsf{NP}^{\mathrm{A}}$ *and* $\mathsf{P}^{\mathrm{B}} \neq \mathsf{NP}^{\mathrm{B}}$.

In fact, there are oracles for all possible configurations of P, NP, and coNP.

Another interesting theorem, due to Sipser (yes, our Sipser), is that there are oracles for which $\mathsf{NP} \cap \mathsf{coNP}$ has complete problems and oracles for which it does not.

Oracle for which $\mathsf{NP} \cap \mathsf{coNP}$ has complete problems is the one for which $\mathsf{P} = \mathsf{NP}$.

Proving the BGS theorem

Finding the oracle A such that $\mathsf{P}^A = \mathsf{NP}^A$ is not too difficult.

Need something powerful that does not benefit from nondeterminism.

We know of such classes — PSPACE is such a class.

Let's try that one.

P and NP relative to PSPACE

Let $A = \text{PSPACE}$. We will show

$$\text{PSPACE} \subseteq \text{P}^A \subseteq \text{NP}^A \subseteq \text{NPSPACE} = \text{PSPACE}$$

$\text{NP}^A \subseteq \text{NPSPACE}$ since we can simulate each oracle query in polynomial space.

The NP machine uses at most polynomial space, and oracle queries can be computed in polynomial space too.

The other oracle

Hard part: Finding the other oracle. In fact, we will construct it.

We will construct a language L_A given an oracle.

Then we construct an oracle such that this language can't be done in polynomial time.

This we do, of course, by diagonalization.

Let $L_A = \{w \mid A(x) = 1, |x| = |w|\}$

Enumeration

Let $M_1^?, M_2^?, \ldots$ be an enumeration of oracle DTMs that run in polynomial time.

Since oracle machines query their oracle as a black box, can plug in any oracle.

Sipser: Assume for simplicity that $M_i^?$ has running time n^i.

We will build an oracle A so that none of these machines can decide L_A.

The construction

Inductive construction. We start with nothing, and at each stage we declare a finite set of strings to be in the language of A or out of it.

Goal: At stage i, make sure that $L(M_i^A)$ and L_A disagree on some string.

How do we do this? Well, L_A is all strings of same length as A accepts.

For a DTM to determine if $w \in L_A$, it will need to ask A about all strings of this length.

If we pick a large enough string, M_i^A won't have time to do this.

Stage i

Let M_i^A have running time n^i. Choose n larger than any string declared for A, such that $2^n > n^i$.

We are going to run M_i^A on 1^n. When M_i^A queries A with q, we

- Answer correctly if q has been declared,
- and answer NO otherwise.

If M_i^A accepts 1^n, we declare all strings of length n to be NO-strings. Then A has no YES-string of length n, and $1^n \notin L_A$.

If M_i^A rejects 1^n, we find a string of length n that M_i^A did not query. This exists, since $2^n > n^i$. Declare this string to be YES.

Finally, declare all undeclared strings of length up to n arbitrarily.

Stage 1, example

In stage 1, nothing has been declared yet.

We look at M_1^A, with running time n^1. We need $2^n > n^1$, 1 suffices, but pick 4.

Run M_1^A on the string 1111. Since nothing has been declared yet, answer NO to all queries.

If accept, then declare all strings of lenght 4 to be NO-strings.

If reject, M_1^A made 4 queries. Find a string among all of length 4 that has not been declared, declare it to be YES.

Finishing the proof

Our oracle is built to make the language L_A not decided by any DTM M^A with polynomial running time.

The oracle is well-defined — all strings accounted for.

However, L_A can easily be decided using an NTM with oracle A.

Hence, $L_A \in NP^A$ and $L_A \notin P^A$, as desired.