Computational Complexity Lecture Notes

Lecture 2

"*The Polynomial Hierarchy*"

Draft version 0.4

2.1 Oracle Turing Machines

We can equip a generic Turing Machine with arbitrary access to a language, treated as a black box. This enables us to visit "parallel computational universes", where the solution to a specific problem, or a class of problems, is free. Enter oracle worlds!

At first, we define such a model:

Definition 2.1

For an arbitrary language $A \subseteq \Sigma^*$, a Turing Machine M^A with *oracle* A is a multi-string TM with a special tape, called *query tape*, and three special states: q_q (*query state*), q_{yes} and q_{no} (*answer states*). The computation of the oracle machine M^A proceeds like an ordinary TM, but it has the special ability to write a string on the query tape, and enter the query state q_q: *From q_q it moves to either q_{yes}, q_{no}, depending on whether the current query string is in A or not.*

- ▶ The machine M^A can ask during its computation process several questions $x \overset{?}{\in} A$, and use this answer to its further computation.
- ▶ By the above definition, the machine M^A has access to A's characteristic function $A(x) = \chi_A(x)$, and it can query this function and obtain the answer at one computational step.
- ▶ The number of queries is bounded by the overall running time of the TM. For example, a polynomial-time TM can ask at most a polynomial number of queries.

2.1.1 Oracle Complexity Classes

Definition 2.2

Let $\mathcal{C}$ be a time complexity class (deterministic or nondeterministic). Define $\mathcal{C}^A$ to be the *class* of all languages decided by machines of the same sort and time bound as in $\mathcal{C}$, only that the machines have now oracle access to A. Also, we define: $\mathcal{C}_1^{\mathcal{C}_2} = \bigcup_{L \in \mathcal{C}_2} \mathcal{C}_1^L$.

For example, $\mathbf{P}^{\mathbf{NP}} = \bigcup_{L \in \mathbf{NP}} \mathbf{P}^{L}$. Note that $\mathbf{P}^{\text{SAT}} = \mathbf{P}^{\mathbf{NP}}$. Also, since the oracle is considered a black-box, we have free answers also for the complementary language. For example, $\mathbf{P}^{\mathbf{NP}} = \mathbf{P}^{\text{SAT}} = \mathbf{P}^{\overline{\text{SAT}}} = \mathbf{P}^{co\mathbf{NP}}$.

2.1.2 Enumerations

Recall that we can encode TMs as strings, just by encoding the TM's description using an alphabet, and that this encoding is *not* unique. So, every machine is represented by *infinitely many* strings, and every string can potentially encode *a* TM[1]. So, there exists a function $e(x)$ mapping strings to TMs, such that:

1. For every $x \in \Sigma^*$, $e(x)$ represents a TM.
2. Every TM is represented by at least one $e(x)$.
3. The code of the TM $e(x)$ can be easily decoded.

Such a function is called an *enumeration* of TMs (deterministic or nondeterministic).

When we consider classes like **P** or **NP**, we can easily enumerate only these machines, a *subclass* of all DTMs (NTMs respectively): if a function is *time-constructible*, then by definition, there exists a DTM halting after exactly $t(n)$ moves. Such a machine is called a $t(n)$*-clock machine*. For any DTM M_1, we can attach a $t(n)$-clock machine M_2 and obtain a "product" machine $M_3 = \langle M_1, M_2 \rangle$, which halts if either M_1 or M_2 halts, and accepts only if M_1 accepts.

Now, consider the functions $p_i(n) = n^i, i \geq 1$. If $\{M_x\}$ is an enumeration of DTMs, let $M_{\langle x,i \rangle}$ be the machine M_x attached with a $p_i(n)$-clock machine. Then, $\{M_{\langle x,i \rangle}\}$ is an *enumeration of all polynomial-time clocked machines*, and it is an enumeration of languages in **P**, such that:

- Every machine $M_{\langle x,i \rangle}$ accepts a language in **P**.
- Every language in **P** is accepted by at least a machine in the enumeration (in fact, by infinite number of machines).

Remark 2.1

This list will *not* contain *all* the polynomial-time bounded machines! Remember that it is undecidable to determine whether a given TM halts in polynomial time on all inputs, due to Rice's Theorem.

The same holds for **NP**, just by enumerating all poly-time alarm clocked NTMs. Also, we can do the same trick with *space*, using a *yardstick*, a DTM that halts after visiting *exactly* $s(n)$ memory cells. We can also enumerate all the functions in **FP**, and all polynomial-time *oracle* DTMs or NTMs.

[1] In the case of an invalid encoding, we can easily map this string to the empty TM M_0, which has an empty program and always rejects ($M_0(x) = 0$ for every $x \in \Sigma^*$).

2.1.3 Relativizations of the P vs NP question

Theorem 2.1

There exists an oracle $A \subseteq \Sigma^*$, for which $\mathbf{P}^A = \mathbf{NP}^A$.

Proof. Take A to be a **PSPACE**-complete language.Then:

$$\mathbf{PSPACE} \subseteq \mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{PSPACE}^A = \mathbf{PSPACE}^{\mathbf{PSPACE}} \subseteq \mathbf{PSPACE}$$

- Since A is **PSPACE**-complete, $\mathbf{PSPACE} \subseteq \mathbf{P}^A$, because we can reduce in polynomial-time a **PSPACE** computation to a query to a **PSPACE**-complete language.
- Trivially, $\mathbf{P}^A \subseteq \mathbf{NP}^A$ (*note that it actually holds for any oracle* A).
- $\mathbf{NP}^A \subseteq \mathbf{PSPACE}^A$, because we can simulate each path in the nondeterministic tree, reusing the same (polynomial) space, and asking the same oracle questions to A when the machine enters a query state. Notice that it is the same technique we used to prove $\mathbf{NP} \subseteq \mathbf{PSPACE}$, extended to oracle machines.
- $\mathbf{PSPACE}^{\mathbf{PSPACE}} \subseteq \mathbf{PSPACE}$, since a **PSPACE** machine can resolve the **PSPACE** queries by itself, as a subroutine, again by reusing the (polynomial) space needed for each query question subroutine.[2]

□

But, on the other hand:

Theorem 2.2

There exists an oracle $B \subseteq \Sigma^*$, for which $\mathbf{P}^B \neq \mathbf{NP}^B$.

Proof. We will try to find a language $L \in \mathbf{NP}^B \setminus \mathbf{P}^B$, and a good candidate is:

$$L = \{1^n \mid \exists x \in B \text{ with } |x| = n\}$$

It is easy to see that $L \in \mathbf{NP}^B$: a NPTM with oracle B can guess all strings y of length n in Σ^*, and for each y it asks the oracle if y is in B. The difficult part is to define the oracle $B \subseteq \Sigma^*$ such that $L \notin \mathbf{P}^B$. Without loss of generality, take $\Sigma = \{0, 1\}$. Let $M_1^{\cdot}, M_2^{\cdot}, \dots$ an enumeration of all PDTMs with oracle, such that every machine appears *infinitely many* times in the enumeration. We will define B iteratively: $B_0 = \emptyset$, and $B = \bigcup_{i \geq 0} B_i$, where $B_i = \{x \in B \mid |x| \leq i\}$. Let also X denote a set, the set of *exceptions*, which we will use during the proof.

In the i^{th} stage, we simulate $M_i^B(1^i)$ for $i^{\log i}$ steps. During the simulation, the oracle machine may enter the query state, so we have to determine how to answer questions "*Is x in B?*". Note that we have already defined B_{i-1}, so:

[2]This also holds for **P**, i.e. $\mathbf{P}^{\mathbf{P}} = \mathbf{P}$.

- ▶ **If** $|x| < i$, we look for x in B_{i-1}:
  - **If** $x \in B_{i-1}$, M_i^B goes to q_{yes}
  - **Else** M_i^B goes to q_{no}
- ▶ **If** $|x| \geq i$, M_i^B goes to q_{no}, and $x \to X$, in order to remember this answer, else the definition of B would be inconsistent.

Suppose now, that after at most $i^{\log i}$ steps the simulation stops. The simulated machine either accepted, rejected or was stopped before reaching a final state.

- ▶ If the machine *rejects*, we define
$$B_i = B_{i-1} \cup \{x \in \{0,1\}^* : |x| = i, x \notin X\}$$
that is, we add to B_i all the strings of length i that are *not* in X (recall that we enforced above that the members of X are not in B). Hence, $1^i \in L$, and $L(M_i^B) \neq L$, so we made sure that the machine does *not* decide the language correctly. Of course, for this idea to work, we have to ensure that $\{x \in \{0,1\}^* : |x| = i, x \notin X\} \neq \emptyset$, but that is easy, since each machine is simulated for $i^{\log i}$ steps, so it hasn't time to add all the 2^i strings to X, thus $\{x \in \{0,1\}^* : |x| = i, x \notin X\} = \{0,1\}^i \setminus X \neq \emptyset$.
- ▶ If the machine *accepts*, we define $B_i = B_{i-1}$, so that $1^i \notin L$, and $L(M_i^B) \neq L$. The machine is made to err again.
- ▶ If the machine *fails to halt* in the allotted time, that is if its polynomial bound is greater than $i^{\log i}$, we set $B_i = B_{i-1}$. We have not ensured that $L(M_i^B) \neq L$, but, remember that each machine appears *infinitely many* times in the enumeration, so we know that this machine will be simulated again (and again), so there exists an index i' large enough such that $i'^{\log i'}$ will surpass the polynomial bound of the machine, and it will enter in a final state, so we will fall in the above two cases.

□

2.1.4 A First Barrier: The Limits of Diagonalization

As we saw, an oracle can transfer us to an alternative computational "*universe*". Recall that we created a universe where $\mathbf{P} = \mathbf{NP}$, and another where $\mathbf{P} \neq \mathbf{NP}$. One can ask when and how we can traverse these universes. If we prove an inclusion between complexity classes, can it be transferred to some, or all, oracle worlds?

Also, notice that diagonalization is a technique that relies on two facts: Firstly, that TMs are (effectively) represented by strings, and secondly, that a TM can simulate another TM without much overhead in the resources. But these two properties of TMs must also stand for oracle TMs: A TM with oracle access to a language A can also be represented as a string, and a TM with oracle access to A can simulate another TM with oracle access to A (it just asks the query questions of the simulated machine to its own oracle). Hence, diagonalization or any other proof technique that relies only on these two facts, holds also for *every* oracle. Such results are called *relativizing results*. For example, $\mathbf{P}^A \subseteq \mathbf{NP}^A$, for every $A \in \{0,1\}^*$.

The above two theorems indicate that **P** vs. **NP** is a *nonrelativizing* result, so diagonalization and any other relativizing method doesn't suffice to prove it. We obviously need something more.

This rules out the possibility of a super-clever diagonalization technique for proving **P** vs. **NP** that could elude us.

So, the first barrier we come across is **Diagonalization**. During the course, we will stumble on two more.

2.1.5 Cook Reductions

Using oracles, we can define more general and useful reductions:

Definition 2.3

A problem A is *Cook-Reducible* to a problem B, denoted by $A \leq_T^p B$, if there is an oracle DTM M^B which in polynomial time decides A (*making at most polynomial many queries to* B).

- ▶ Observe that the above definition is equivalent to $A \in \mathbf{P}^B$.
- ▶ If $A \leq_m^p B \Rightarrow A \leq_T^p B$. This means that Cook reductions are more general than Karp reductions, or, that Karp reductions are very specific and restricted Cook reductions.
- ▶ By definition, we have that $\overline{A} \leq_T^p A$.

Theorem 2.3

P, **PSPACE** are closed under $\leq_T^p$.

Remark 2.2

Is **NP** closed under $\leq_T^p$? (*cf. Problem Sets!*)

2.1.6 Relativized Results

There are many other oracle worlds:

Theorem 2.4

There exists $C \subseteq \Sigma^*$ such that:

$$\mathbf{P}^C \neq \mathbf{NP}^C = co\mathbf{NP}^C$$

Theorem 2.5

There exist $D, E \subseteq \Sigma^*$ such that:

$$\mathbf{NP}^D \neq co\mathbf{NP}^D \text{ and } \mathbf{P}^D = \mathbf{NP}^D \cap co\mathbf{NP}^D$$

$$\mathbf{NP}^E \neq co\mathbf{NP}^E \text{ and } \mathbf{P}^E \neq \mathbf{NP}^E \cap co\mathbf{NP}^E$$

2.1.7 Random Oracles

In the above sections, we proved that $\exists A \subseteq \Sigma^* :\ \mathbf{P}^A = \mathbf{NP}^A$, and also that $\exists B \subseteq \Sigma^* :\ \mathbf{P}^B \neq \mathbf{NP}^B$. One can ask about quantifying this question, e.g. "*how many oracles make* **P** *equal to* **NP**, *and how many make them differ?*", or even better "*what happens if we choose an oracle at random?*". A naïve way to choose a random oracle is to traverse all the (countable) elements of Σ^* in lexicographic order, and add each element to the oracle set with probability $1/2$.

Now, consider the set $\mathcal{U} = Pow(\Sigma^*)$, and the sets:

$$\{A \in \mathcal{U} :\ \mathbf{P}^A = \mathbf{NP}^A\}$$

$$\{B \in \mathcal{U} :\ \mathbf{P}^B \neq \mathbf{NP}^B\}$$

Can we compare these two sets, by defining a kind of *measure*, and find which is *larger*?

Theorem 2.6 (Bennet, Gill)

$$\mathbf{Pr}_{B\subseteq\Sigma^*}\left[\mathbf{P}^B \neq \mathbf{NP}^B\right] = 1$$

The above result states than *almost all* oracles (in a measure-theoretic notion) make **P** different from **NP**. This means that if we travel to a computational parallel universe at random, then *almost surely* in this universe **P** will be different from **NP**. These kind of results are called measure one results, and there are many in Complexity Theory, all proved in the effort to approach unsolved questions, by relativizing them to other oracle worlds. For a detailed guide to measure one results, you can see [VW97].

These efforts culminated to the formalization of the *Random Oracle Hypothesis* by Bennett and Gill, which stated informally that "every statement about relativized complexity classes that holds with probability one relative to a random oracle, also holds in the unrelativized case". This hypothesis was quite intuitive, but was disproved by S. Kurtz, who presented two counterexamples in [Kur83].

2.2 The Polynomial Hierarchy

We defined interesting classes such as $\mathbf{P}^{\mathbf{NP}}$. We can also define its nondeterministic analogue $\mathbf{NP}^{\mathbf{NP}}$, and its complement $co\mathbf{NP}^{\mathbf{NP}}$. These are well defined complexity classes, so we can use them as oracles to other classes, like $\mathbf{P}^{\mathbf{NP}^{\mathbf{NP}}}$, $\mathbf{NP}^{\mathbf{NP}^{\mathbf{NP}}}$ and so on. This forms an hierarchy, known as the Polynomial-Time Hierarchy, the polynomial analogue of Kleene's Arithmetical Hierarchy in recursion theory.

Definition 2.4 (Polynomial Hierarchy)

- $\Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathbf{P}$
- $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$
- $\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$
- $\Pi_{i+1}^p = co\mathbf{NP}^{\Sigma_i^p}$
- $\mathbf{PH} \equiv \bigcup_{i \geqslant 0} \Sigma_i^p$

So, intuitively, the i^{th} layer Σ_i^p is defined as $\mathbf{NP}$ with an oracle to the previous layer class Σ_{i-1}^p, its complementary class is Π_i^p, and Δ_i^p is $\mathbf{P}$ with an oracle to the previous layer class Σ_{i-1}^p.
It can be easily shown that these properties hold:

- $\Sigma_i^p, \Pi_i^p \subseteq \Sigma_{i+1}^p$
- $A, B \in \Sigma_i^p \Rightarrow A \cup B \in \Sigma_i^p$, $A \cap B \in \Sigma_i^p$
- $A \in \Pi_i^p \Rightarrow \overline{A} \in \Sigma_i^p$
- $A, B \in \Delta_i^p \Rightarrow A \cup B$, $A \cap B$ and $\overline{A} \in \Delta_i^p$

$$\begin{tikzcd}
\vdots & & \vdots \\
& \Delta_3^p = \mathbf{P}^{\mathbf{NP}^{\mathbf{NP}}} \arrow[ul] \arrow[ur] & \\
\Pi_2^p = co\mathbf{NP}^{\mathbf{NP}} \arrow[ur] & & \Sigma_2^p = \mathbf{NP}^{\mathbf{NP}} \arrow[ul] \\
& \Delta_2^p = \mathbf{P}^{\mathbf{NP}} \arrow[ul] \arrow[ur] & \\
\Pi_1^p = co\mathbf{NP} \arrow[ur] & & \Sigma_1^p = \mathbf{NP} \arrow[ul] \\
& \Delta_0^p = \Sigma_0^p = \\ = \Pi_0^p = \Delta_1^p = \mathbf{P} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

This is an hierarchy of oracles, and as in recursion theory, we can show that it can be viewed as an hierarchy of *alternating quantifiers*! To this end, we prove that we can "jump" from Π_{i-1}^p to the next Σ_i^p by adding an $\exists$ quantifier. This is a generalization of the $\mathbf{NP}$ quantifier characterization. Recall that a relation R is called polynomially balanced if $(x, y) \in R \Rightarrow |y| \leq |x|^k$, for some $k \in \mathbb{N}$.

Theorem 2.7

Let L be a language , and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in Π_{i-1}^p and

$$L = \{x : \exists y, s.t. : (x, y) \in R\}$$

Proof. We will use induction on i:

- For $i = 1$, we have that $\{x; y : (x, y) \in R\} \in \mathbf{P}$, so $L = \{x | \exists y : (x, y) \in R\} \in \mathbf{NP}$

(certificate characterization of **NP**).

▶ For $i > 1$, suppose that the result holds for $i-1$:

$$A \in \Sigma_{i-1}^p \Rightarrow \exists B \in \Pi_{i-2}^p : (z \in A \Leftrightarrow \exists w : (z, w) \in B) \tag{1}$$

We will prove the two directions of the theorem. If there exists such an $R \in \Pi_{i-1}^p$, we must show that $L \in \Sigma_i^p$, i.e. there exists NTM with Σ_{i-1}^p oracle deciding L. This machine, on input x, guesses a y and asks the Σ_{i-1}^p oracle whether $(x, y) \notin R$ (*we ask the complementary question since* $\Sigma_i^p = co\Pi_i^p$).

Conversely, if $L \in \Sigma_i^p$, we must show the existence of R. Since $L \in \Sigma_i^p$, there exists an NTM M^A, $A \in \Sigma_{i-1}^p$, which decides L. Using (1), there exists $B \in \Pi_{i-2}^p : (z \in A \Leftrightarrow \exists w : (z, w) \in B)$.

We must describe a relation R, such as in the case of **NP**. The relation must verify a certificate for the instance. The main difference here is that we have an oracle machine M^A, which during its computation can make oracle calls to the Σ_{i-1}^p oracle A.

Observe that for each query z_i that A responds "yes", we have a certificate w_i such that $(z_i, w_i) \in B$. So, we can define R as:

> $R(x, y) =$"$(x, y) \in R$ *iff* y *records an accepting computation of* M^A *on* x *, together with a certificate* w_i *for each* **yes** *query* z_i *in the computation.*"

Now we have to show that $\{x; y : (x, y) \in R\} \in \Pi_{i-1}^p$. What must R do to be a correct verifier for L? First, it must check that all steps of M^A are legal. This isn't different from an **NP** verifier, and it takes polynomial time. Secondly, it must check that for every "yes" oracle answer z_i, it holds that $(z_i, w_i) \in B$. But $B \in \Pi_{i-2}^p$, and thus is in Π_{i-1}^p. And also for all "no" queries z_i', it has to check that $z_i' \notin A$, which is another Π_{i-1}^p question, so $\{x; y : (x, y) \in R\} \in \Pi_{i-1}^p$.

□

We can prove the same thing for Π_{i-1}^p, that we can "jump" from Σ_{i-1}^p to the next Π_i^p by adding an $\forall$ quantifier:

Corollary 2.1

Let L be a language , and $i \geq 1$. $L \in \Pi_i^p$ iff there is a polynomially balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in Σ_{i-1}^p and

$$L = \{x : \forall y, |y| \leq |x|^k, s.t. : (x, y) \in R\}$$

And if we expand this result i times, until we get an R in **P**, we have a characterization of a Σ_i^p (respectively Π_i^p) language with i alternating quantifiers in front of a **P** verifier:

Corollary 2.2

Let L be a language , and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced, polynomially-time decicable $(i+1)$-ary relation R such that:

$$L = \{x : \exists y_1 \forall y_2 \exists y_3 ... Q_i y_i, s.t. : (x, y_1, ..., y_i) \in R\}$$

where the i^{th} quantifier Q_i is $\forall$, if i is even, and $\exists$, if i is odd.

The above justify the use of the quantifier notation:

Remark 2.3

$\Sigma_i^p = (\underbrace{\exists\forall\exists\cdots Q_i}_{i \text{ quantifiers}} / \underbrace{\forall\exists\forall\cdots Q_i'}_{i \text{ quantifiers}})$ $\Pi_i^p = (\underbrace{\forall\exists\forall\cdots Q_i}_{i \text{ quantifiers}} / \underbrace{\exists\forall\exists\cdots Q_i'}_{i \text{ quantifiers}})$

Example 2.1. $\Sigma_3^p = (\exists\forall\exists / \forall\exists\forall)$, $\Pi_2^p = (\forall\exists / \exists\forall)$

Theorem 2.8

If for some $i \geq 1$, $\Sigma_i^p = \Pi_i^p$, then for all $j > i$:

$$\Sigma_j^p = \Pi_j^p = \Delta_j^p = \Sigma_i^p$$

Or, the polynomial hierarchy *collapses* to the i^{th} level.

Proof. It suffices to show that: $\Sigma_i^p = \Pi_i^p \Rightarrow \Sigma_{i+1}^p = \Sigma_i^p$.

Let $L \in \Sigma_{i+1}^p$, so there exists an $R \in \Pi_i^p$ such that $L = \{x | \exists y : (x, y) \in R\}$. But $\Pi_i^p = \Sigma_i^p$ by assumption, so $R \in \Sigma_i^p$. By expanding the characterization again, we have that $(x, y) \in R \Leftrightarrow \exists z : (x, y, z) \in S$, $S \in \Pi_{i-1}^p$. But y, z are polynomial-length certificates, thus their concatenation $y; z$ will also be polynomial. So we proved that $x \in L \Leftrightarrow \exists y; z : (x, y, z) \in S$, $S \in \Pi_{i-1}^p$, and this implies that $L \in \Sigma_i^p$.

□

Corollary 2.3

If **P=NP**, or even **NP**=co**NP**, the Polynomial Hierarchy collapses to the first level.

We can define the $QSAT_i$ problems, which are generalization of SAT with i alternating quantifiers applied on the formula:

Definition 2.5 (QSAT$_i$ **Definition)**

Given expression ϕ, with Boolean variables partitioned into i sets X_i,is ϕ satisfied by the overall truth assignment of the expression:

$$\exists X_1 \forall X_2 \exists X_3 .....Q X_i \phi$$

where Q is $\exists$ if i is *odd*, and $\forall$ if i is even.

As expected:

Theorem 2.9

For all $i \geq 1$ QSAT$_i$ is Σ_i^p-complete.

These are the "canonical" complete problems for every level of the polynomial hierarchy. One can ask what if the whole polynomial hierarchy has a complete problem, but this does not seem plausible:

Theorem 2.10

If there is a **PH**-complete problem, then the polynomial hierarchy collapses to some finite level.

Proof. Let L is **PH**-complete. Since $L \in$ **PH**, $\exists i \geq 0 : L \in \Sigma_i^p$. But any $L' \in \Sigma_{i+1}^p$ reduces to L, since it is **PH**-complete. **PH** is closed under reductions, so we imply that $L' \in \Sigma_i^p$, so $\Sigma_i^p = \Sigma_{i+1}^p$.

□

The QSAT$_i$ problems we defined above are of course special cases of the TQBF problem, which is **PSPACE**-complete. From this we immediately have:

Theorem 2.11

PH $\subseteq$ **PSPACE**

It is an open question whether **PH** $\overset{?}{=}$ **PSPACE**. Note that if they are equal, then **PH** has complete problems, so it collapses to some finite level.

2.2.1 Relativized Results

Let's see how the inclusion of the Polynomial Hierarchy to Polynomial Space, and the inclusions of each level of **PH** to the next relativizes:

Theorem 2.12 (Yao 1985, Håstad 1986)

PH$^A \neq$ **PSPACE**^{A} relative to *some* oracle $A \subseteq \Sigma^*$

Theorem 2.13 (Cai 1986, Babai 1987)

$\mathbf{Pr}_A[\mathbf{PH}^A \neq \mathbf{PSPACE}^A] = 1$

Theorem 2.14 (Yao 1985, Håstad 1986)

$(\forall i \in \mathbb{N})\ \Sigma_i^{p,A} \subsetneq \Sigma_{i+1}^{p,A}$ relative to *some* oracle $A \subseteq \Sigma^*$.

Theorem 2.15 (Rossman-Servedio-Tan, 2015)

$\mathbf{Pr}_A[(\forall i \in \mathbb{N})\ \Sigma_i^{p,A} \subsetneq \Sigma_{i+1}^{p,A}] = 1$

2.3 The Complexity of Optimization Problems

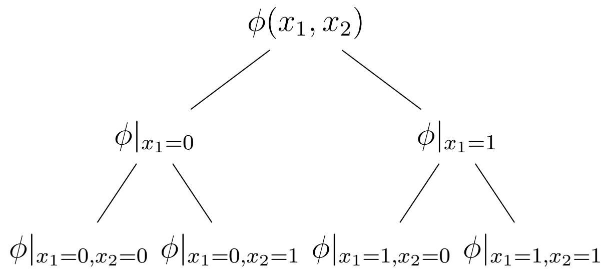
Let ϕ be a Boolean formula with n variables.. It is easy to see that we can reduce the satisfiability of ϕ to the satisfiability of two formulas with $n-1$ variables:

$$\phi \in \text{SAT} \Leftrightarrow (\phi|_{x_1=0} \in \text{SAT}) \vee (\phi|_{x_1=1} \in \text{SAT})$$

This property is called *self-reducibility* of SAT. Many other problems are self-reducible, which means that they can be reduced to "smaller" instances of themselves.

Imagine there was a SAT oracle. Then, we could ask it questions of type $(\phi|_{x_i=j} \overset{?}{\in} \text{SAT})$. So, we could traverse the self-reducibility tree, asking the oracle at every node, and determine the path depending on the oracle answers. We would only need $2n$ oracle calls to the *alleged* SAT oracle.

Example 2.2 (Self-Reducibility Tree of depth n)**.**



Let **FP** be the function analogue of **P**: it contains functions computable by a DTM in polynomial time. Also, define the function problem FSAT:

Definition 2.6 (FSAT)

Given a Boolean expression ϕ, if ϕ is satisfiable then return a satisfying truth assignment for ϕ. Otherwise return "no".

The above shows that:

$$\text{FSAT} \in \mathbf{FP} \Leftrightarrow \text{SAT} \in \mathbf{P}$$

since if SAT $\in$ **P**, we can use the self-reducibility property to fix variables one-by-one, and retrieve a solution. On the other hand, if we have a solution, we know that a solution exists. The above indicate that we can extract the solution of the optimization problem by using an oracle to the corresponding decision problem.

2.3.1 What about TSP?

As before, we can solve TSP using a hypothetical algorithm for the **NP**-complete decision version of TSP (denoted as TSP_D):

- ▶ We can find the cost of the optimum tour by *binary search* in the interval $[0, 2^n]$, where n is the input length. Obviously, the optimal cost will be somewhere in this interval. We need n oracle calls for this.
- ▶ When we find the optimum cost C, we fix it, and start changing intercity distances one-by one, by setting each distance to $C+1$.
- ▶ We then ask the **NP**-oracle if there still is a tour of optimum cost at most C:
  - If there is, then this edge is not in the optimum tour.
  - If there is not, we know that this edge is in the optimum tour.
- ▶ After at most n^2 (polynomial) oracle queries, we can extract the optimum tour, and thus have the solution to TSP.

2.3.2 The Classes $\mathbf{P}^{\mathbf{NP}}$ and $\mathbf{FP}^{\mathbf{NP}}$

Recall that $\mathbf{P}^{\text{SAT}}$ is the class of languages decided in polynomial time with a SAT oracle. We can ask a polynomial number of adaptive queries. Since SAT is **NP**-complete, it holds that $\mathbf{P}^{\text{SAT}}$=$\mathbf{P}^{\mathbf{NP}}$.

Now, $\mathbf{FP}^{\mathbf{NP}}$ is the class of *functions* that can be computed by a poly-time DTM with a SAT oracle. It is easy to see (by using the procedures we described) that:

$$\text{FSAT}, \text{TSP} \in \mathbf{FP}^{\mathbf{NP}}$$

What about completeness in $\mathbf{FP}^{\mathbf{NP}}$? It is a well-defined syntactic class, so it deserve to have complete problems. But we need reductions between function problems. A generic form of such reductions is:

Definition 2.7 (Reductions for Function Problems)

A function problem A reduces to B if there exists $R, S \in \mathbf{FL}$ such that:

- ▶ $x \in A \Rightarrow R(x) \in B$.
- ▶ If z is a correct output of $R(x)$, then $S(z)$ is a correct output of x.

Using this reductions, it can be shown that:

Theorem 2.16

TSP is $\mathbf{FP}^{\mathbf{NP}}$-complete.

2.4 Summary

- ▶ Oracle TMs have one-step oracle access to some language.
- ▶ There exist oracles $A, B \subseteq \Sigma^*$ such that $\mathbf{P}^A = \mathbf{NP}^A$ and $\mathbf{P}^B \neq \mathbf{NP}^B$.
- ▶ Relativizing results hold for *every* oracle.
- ▶ A Cook reduction $A \leq_T^p B$ is a poly-time TM deciding A, by using B as an oracle.
- ▶ The Polynomial Hierarchy can be viewed as:
  - Oracle hierarchy of consecutive **NP** oracles.
  - Quantifier hierarchy of alternating quantifiers.
- ▶ If for some $i \geq 1$ $\Sigma_i^p = \Pi_i^p$, or there is a **PH**-complete problem, then PH collapses to some finite level.
- ▶ Optimization problems with decision version in **NP** (such as TSP) are in $\mathbf{FP}^{\mathbf{NP}}$.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1st edition, April 2009.

[DK00] Ding-Zhu Du and Ker-I Ko. *Theory of Computational Complexity*. Wiley-Interscience, January 2000.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 1st edition, April 2008.

[Kur83] Stuart A. Kurtz. On the random oracle hypothesis. *Information and Control*, 57(1):40–47, 1983.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[VW97] Heribert Vollmer and Klaus W. Wagner. Measure one results in computational complexity theory. In *Advances in Algorithms, Languages, and Complexity*, 1997.