ITCS:CCT09 : Computational Complexity Theory Mar 18, 2009

Lecture 7

Lecturer: Jayalal Sarma M.N. *Scribe: Hao Song*

1 Non-deterministic Oracle Turing Machines

In the previous lectures, we have introduced the concept of deterministic oracle Turing machines. The concept of non-deterministic oracle Turing machines is just a direct extension of that to the non-deterministic case.

To be precise, a non-deterministic oracle Turing machine (sometimes abbreviated to "oracle NTM") is just like a deterministic oracle TM, but with a non-deterministic transition function (*with some restrictions which we will discuss below*). It has

- a finite state set, which contains two special states called the *query state* and the *answer state*, aside from those states of an ordinary NTM
- a finite alphabet, which contains two special symbols 0 and 1, among other things
- one input tape, which is read-only
- one output tape, which is write-only
- a finite number of working tapes
- a *query tape*, which is considered to be a special working tape, and is included in the computation of the machine's space consumption
- an oracle which computes a language L
- a non-deterministic transition function

The machine is allowed to query the oracle many times, and use the answers from the oracle to facilitate its own computation. Each of those queries works as follows

1. the machine writes a query string s onto the *query tape*, and enters the *query state.*
2. the oracle decides if s is in the language L, and if so, it replaces everything on the *query tape* with the symbol 1; otherwise, it replaces everything on the *query tape* with the symbol 0. After that, the oracle puts the machine into the *answer state.* This counts as one single step for the machine.

Please take note! To ensure that the only thing the NTM can do upon entering the *query state* is to query the oracle, we allow the transition function to be non-deterministic if and only if the current state is not the *query state*.

Just like any other computational model, we can define complexity classes based on oracle NTMs.

Definition 1 *For any complexity class* A *defined on NTMs and any complexity class* B*, we define the complexity class* A^{B} *to be the set of all languages* L *that satisfy the following conditions*

1. *there is an oracle NTM* ***M*** *that runs within the resouce constraints specified by* A
2. *there is an oracle* ***O*** *that's capable of computing a language* L' *in* B.
3. ***M*** *with access to* ***O*** *computes* L

2 The Polynomial-Time Hierarchy

Before introducing the main concept of this lecture, the polynomial-time hierarchy, we first need to present a lemma, which lays the groundwork for understanding the hierarchy.

Lemma 2 *we have the following relationships between oracle NTM complexity classes and ordinary DTM and NTM complexity classes*

1. $\mathsf{NP}^{\mathsf{P}} \subseteq \mathsf{NP}$
2. $\mathsf{NP}^{\mathsf{PSPACE}} \subseteq \mathsf{PSPACE}$

Proof

1. this is trivial
2. in previous lectures we have proved that for any (fully space-constructible) function $f(n) \geq n$, we have
$$\mathsf{NTIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n))$$
which implies that
$$\mathsf{NP} \subseteq \mathsf{PSPACE}$$
by *relativization*, we have
$$\mathsf{NP}^{\mathsf{PSPACE}} \subseteq \mathsf{PSPACE}^{\mathsf{PSPACE}}$$

which, combined with

$$\mathsf{PSPACE}^{\mathsf{PSPACE}} \subseteq \mathsf{PSPACE}$$

gives us the result we want.

■

2.1 Defining the Polynomial-Time Hierarchy with Oracle TMs

The polynomial-time hierarchy (or "PH" for short) is the polynomial analog of the arithmetic hierarchy in recursion theory. It consists of a series of complexity classes, and can be defined inductively by oracle TMs

Definition 3 *For non-negative integers n, we define complexity classes Σ_n^{P} and Π_n^{P} as follows*

- $\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$
- $\Sigma_{n+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_n^{\mathsf{P}}}$ *for all* $n \geq 0$
- $\Pi_{n+1}^{\mathsf{P}} = \mathsf{co} - \Sigma_{n+1}^{\mathsf{P}}$ *for all* $n \geq 0$

In addition, we define

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

In the following proposition, we summarize some of the basic properties of PH

Proposition 4 (Basic Properties of the Polynomial Time Hierarchy) *we have*

1. $\Sigma_1^{\mathsf{P}} = \mathsf{NP}$, $\Pi_1^{\mathsf{P}} = \mathsf{coNP}$
2. $\Sigma_i^{\mathsf{P}} = \mathsf{NP}^{\Sigma_{i-1}^{\mathsf{P}}} = \mathsf{NP}^{\Pi_{i-1}^{\mathsf{P}}}$ *for any integer* $i > 0$
3. $\Pi_i^{\mathsf{P}} = \mathsf{co} - \mathsf{NP}^{\Sigma_{i-1}^{\mathsf{P}}} = \mathsf{co} - \mathsf{NP}^{\Pi_{i-1}^{\mathsf{P}}}$ *for any integer* $i > 0$
4. *for any integer* $i > 0$, $\Sigma_{i-1}^{\mathsf{P}} \cup \Pi_{i-1}^{\mathsf{P}} \subseteq \Sigma_i^{\mathsf{P}} \cap \Pi_i^{\mathsf{P}}$
5. *for any integer* $i \geq 0$, $\Sigma_i^{\mathsf{P}} \subseteq \mathsf{PSPACE}$
6. *for any integer* $i \geq 0$, $\Pi_i^{\mathsf{P}} \subseteq \mathsf{PSPACE}$

7. $\mathsf{PH} \subseteq \mathsf{PSPACE}$

Proof

1. from the clause 1 of lemma 2, we know that
$$\mathsf{NP}^{\mathsf{P}} \subseteq \mathsf{NP}$$
and it's trivial to show that $\mathsf{NP} \subseteq \mathsf{NP}^{\mathsf{P}}$, thus
$$\Sigma_1^{\mathsf{P}} = \mathsf{NP}^{\mathsf{P}} = \mathsf{NP}$$
and
$$\Pi_1^{\mathsf{P}} = \mathsf{co} - \Sigma_1^{\mathsf{P}} = \mathsf{coNP}$$
2. from the definitions of oracle NTMs, it's obvious that for any complexity class C, we have
$$\mathsf{NP}^{\mathsf{C}} = \mathsf{NP}^{\mathsf{coC}}$$
Specifically, for any positive integer i
$$\Sigma_i^{\mathsf{P}} = \mathsf{NP}^{\Sigma_{i-1}^{\mathsf{P}}} = \mathsf{NP}^{\mathsf{co}-\Sigma_{i-1}^{\mathsf{P}}} = \mathsf{NP}^{\Pi_{i-1}^{\mathsf{P}}}$$
3. from the definition of PH and the previous clause, for any positive integer i
$$\Pi_i^{\mathsf{P}} = \mathsf{co} - \Sigma_i^{\mathsf{P}} = \mathsf{co} - \mathsf{NP}^{\Sigma_{i-1}^{\mathsf{P}}} = \mathsf{co} - \mathsf{NP}^{\Pi_{i-1}^{\mathsf{P}}}$$
4. this is an immediate corollary of the previous two clauses
5. this can be proved using an induction on i

   the base step for $i = 0$, this is just the fact that $\Sigma_0^{\mathsf{P}} = \mathsf{P} \subseteq \mathsf{PSPACE}$

   the induction step suppose the statement is true for $i = k$, for $i = k + 1$, we have, from clause 2 of lemma 2
$$\Sigma_{k+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_k^{\mathsf{P}}} \subseteq \mathsf{NP}^{\mathsf{PSPACE}} \subseteq \mathsf{PSPACE}$$
   Thus the statement is proved for all non-negative integers i
6. this follows from the previous clause and the fact that $\mathsf{PSPACE} = \mathsf{coPSPACE}$.
7. this follows from clause 5 and the definition of PH

■

2.2 Open Problems Related to the Polynomial-Time Hierarchy

There are several open problems concerning the basic structure of PH, with the notorious "P versus NP" being the most commonly known. Besides that, we just want to list two of them below, both weaker than "P versus NP".

Open Problem 1 *Is there some non-negative integer k such that $\mathsf{PH} = \Sigma_k^{\mathsf{P}}$?*

This phenomenon is commonly referred to as the "collapse" of the polynomial-time hierarchy.

Open Problem 2 *Is $\mathsf{P}^{\mathsf{NP}} \subseteq \mathsf{NP}$?*

You need to convince yourself that the containment is noway obvious from the definition. Actually we can define a (non-trivial) new member of the polynomial-time hierarchy because of this uncertainty

Definition 5 *For non-negative integers n, we define complexity classes Δ_n^{P} as follows*

- $\Delta_0^{\mathsf{P}} = \mathsf{P}$
- $\Delta_{n+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_n^{\mathsf{P}}}$ *for any integer $n \geq 0$*

2.3 The Alternative Definition of PH Based on Alternating Quantifiers

In this subsection, we want to develop an alternative formulation of the polynomial time hierarchy. In order to establish the equivalence between this formulation and the definition we have just described above, we need some theorems.

Theorem 6 *For any positive integer k, a language A is in the complexity class Σ_k^{P} if and only if there is a language B in Π_{k-1}^{P} and a polynomial $p(\cdot)$ such that*

$$x \in \mathsf{A} \quad \Leftrightarrow \quad \exists y, |y| \leq p(|x|) \text{ and } (x, y) \in \mathsf{B}$$

In order to prove this theorem, we'll need the following two lemmas, which we will list here without proofs. For the proofs, please read page 79 and 80 of Du and Ko's book [1]

Lemma 7 *Let $k \geq 0$*

1. *If* $\mathsf{A}, \mathsf{B} \in \Sigma_k^{\mathsf{P}}$, *then* $\mathsf{A} \cup \mathsf{B}, \mathsf{A} \cap \mathsf{B} \in \Sigma_k^{\mathsf{P}}$, *and* $\bar{\mathsf{A}} \in \Pi_k^{\mathsf{P}}$.
2. *If* $\mathsf{A}, \mathsf{B} \in \Pi_k^{\mathsf{P}}$, *then* $\mathsf{A} \cup \mathsf{B}, \mathsf{A} \cap \mathsf{B} \in \Pi_k^{\mathsf{P}}$, *and* $\bar{\mathsf{A}} \in \Sigma_k^{\mathsf{P}}$.
3. *If* $\mathsf{A}, \mathsf{B} \in \Delta_k^{\mathsf{P}}$, *then* $\mathsf{A} \cup \mathsf{B}, \mathsf{A} \cap \mathsf{B}, \bar{\mathsf{A}} \in \Delta_k^{\mathsf{P}}$.

Lemma 8 *For any set* A *and any* $m > 0$, *define* $A_m = \{\langle x_1, x_2, \ldots, x_m \rangle : (\exists i, 1 \leq i \leq m) x_i \in \mathsf{A}\}$ *and* $A'_m = \{\langle x_1, x_2, \ldots, x_m \rangle : (\forall i, 1 \leq i \leq m) x_i \in A\}$. *for any* $k, m > 0$

1. $A \in \Sigma_k^{\mathsf{P}} \quad \Rightarrow \quad A_m, A'_m \in \Sigma_k^{\mathsf{P}}$
2. $A \in \Pi_k^{\mathsf{P}} \quad \Rightarrow \quad A_m, A'_m \in \Pi_k^{\mathsf{P}}$

Proof [Proof of Theorem 6]

- **the "backward" direction**

  Suppose for the language A we can find a language B and a polynomial $p(n)$ that satisfy the specified condition, then we can construct an oracle NTM that computes A as follows: for an input x, first use the NTM to "guess" a witness string up to length $p(|x|)$, call this string y, then use an oracle that computes B to test whether $(x, y) \in \mathsf{B}$ is true, accepts x if and only if the oracle answers "yes".

  This is clearly an oracle NTM running in polynomial time and computes A. Thus we have
  $$\mathsf{A} \in \mathsf{NP}^{\Pi_{k-1}^{\mathsf{P}}} = \Sigma_k^{\mathsf{P}}$$

- **the "forward" direction**

  This proof is a little convoluted, but the main theme is clear, suppose the language A is in Σ_k^{P}, then there should be an oracle NTM that's able to compute it. A string x is in A if and only if there is an accepting path of this oracle NTM on the input x. Our job is to find an encoding scheme of possible accepting paths that meets the following criteria

  complete The encoding scheme should be able to correctly encode every possible accepting path of the oracle NTM on every possible input string.

  concise Given the fact that the oracle NTM is polynomial-time bounded, the length of the encoding of an accepting path is also polynomially bounded

  easy to verify Given an input string x and the encoding string of an accepting path y, we can verify that y is indeed a valid encoding of an accepting path of the oracle NTM on input string x, with the help of an oracle in Π_{k-1}^{P}.

Now the actual proof, it's a proof by induction

the base step when $k = 1$, it's just the definition of NP using P and witnesses. We have proved the validity of that before.

the induction step suppose the statement is proved to be true for everything less than a particular k, now we want prove it for k.

In this case, by definition, we have access to an oracle $\mathsf{D} \in \Sigma^{\mathsf{P}}_{k-1}$

Our first attempt to encode an accepting path is, not surprisingly, to capture the configuration sequence along the way. We denote the configuration sequence as $\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_m \rangle$, where $m \leq q(|x|)$, where x is the input string, and $q(\cdot)$ is the polynomial bounding the running time of the oracle NTM. α must satisfy the following conditions

1. α_1 is the initial configuration
2. α_m is an accepting configuration
3. for any $1 \leq i < m$, $\alpha_i \mapsto \alpha_{i+1}$ is a valid transition of the oracle NTM
4. for any $1 \leq i < m$, if α_i is in the query state
   - if the query string in α_i is in D, then the content on the query tape in α_{i+1} denotes a "yes" answer
   - if the query string in α_i is not in D, then the content on the query tape in α_{i+1} denotes a "no" answer

This is indeed a complete and concise encoding. And the first three conditions above can be verified in polynomial time. But unfortunately, there is no clear evidence that the last condition can be verified easily with a Π^{P}_{k-1} oracle. To address this issue, we need to evolve our encoding scheme a little bit, and transform the condition accordingly. We can collect all the query strings the machine has used along the way into two tuples, namely $u = \langle u_1, u_2, \ldots, u_l \rangle$ and $v = \langle v_1, v_2, \ldots, v_n \rangle$. Then we can replace condition 4 with the following conditions:

5. for any α_i in the query state, the query string in α_i is either in u or in v and
   - if it's in u, then the content of the query tape in α_i denotes a "yes" answer
   - if it's in v, then the content of the query tape in α_i denotes a "no" answer
6. for any $1 \leq i \leq l$, $u_i \in \mathsf{D}$
7. for any $1 \leq i \leq n$, $v_i \notin \mathsf{D}$

note that we can certainly make $l, n \leq m \leq q(|x|)$. And the length of each query string in u and v can be polynomially bounded. Condition 5 can be verified in polynomial time. And according to clause 2 of lemma 2, we can construct a language B_2 in Π^{P}_{k-1} as follows

$$\mathsf{B}_2 = \{v : v \text{ satisfies condition 7}\}$$

We know that for condition 6 to hold, every string in u must be in D, which is a $\Sigma^{\mathsf{P}}_{k-1}$ class language. By our induction hypothesis, we have a language $\mathsf{B}'_3 \in \Pi^{\mathsf{P}}_{k-2}$ and a polynomial $r(\cdot)$ such that

$$u_i \in \mathsf{D} \quad \Leftrightarrow \quad \exists w_i, |w_i| \leq r(|u_i|), \langle u_i, w_i \rangle \in \mathsf{B}'_3$$

We can collect all the w_i into one single tuple $w = \langle w_1, w_2, \ldots, w_l \rangle$. Again using clause 2 of lemma 2, we know the language defined by

$$\mathsf{B}_3 = \{\langle u, w \rangle : (\forall i, 1 \leq i \leq l) \langle u_i, w_i \rangle \in \mathsf{B}'_3\}$$

is a language in $\Pi^{\mathsf{P}}_{k-2} \subseteq \Pi^{\mathsf{P}}_{k-1}$.
We can collect all the conditions mentioned above that can be verified in polynomial time into one language

$$\mathsf{B}_1 = \{\langle x, \alpha, u, v \rangle : \text{conditions 1, 2, 3, 5 hold}\}$$

then clearly $\mathsf{B}_1 \in \mathsf{P} \subseteq \Pi^{\mathsf{P}}_{k-1}$.
Finally we can define the Π^{P}_{k-1} language B contained in the statement we want to prove in the first place

$$\mathsf{B} = \{\langle x, y \rangle : y = \langle \alpha, u, v, w \rangle, \langle x, \alpha, u, v \rangle \in \mathsf{B}_1, v \in \mathsf{B}_2, \langle u, w \rangle \in \mathsf{B}_3\}$$

according to clause 2 of lemma 7, $\mathsf{B} \in \Pi^{\mathsf{P}}_{k-1}$. And the length of the witness $|y|$ can be polynomially bounded by $|\alpha|$, $|u|$, $|v|$ and $|w|$, which, in turn, can all be polynomially bounded by $|x|$.
This concludes our induction step.

And that concludes our prove for the "forward" direction

Thus the equivalence. ■

Corollary 9 *For any positive integer k, a language A is in the complexity class Π^{P}_k if and only if there is a language B in $\Sigma^{\mathsf{P}}_{k-1}$ and a polynomial $p(n)$ such that*

$$x \in \mathsf{A} \quad \Leftrightarrow \quad \forall y, |y| \leq p(n) \text{ and } (x, y) \in \mathsf{B}$$

Corollary 10 (Alternation of Quantifiers) *For any positive integer k*

- *a language A is in Σ^{P}_k if and only if there exists a language $\mathsf{B} \in \mathsf{P}$ and a polynomial p such that for any string x*

$$\begin{aligned} x \in \mathsf{A} \quad \Leftrightarrow \quad & (\exists y_1, |y_1| \leq p(|x|))(\forall y_2, |y_2| \leq p(|x|)) \\ & \ldots (Q_k y_k, |y_k| \leq p(|x|)) \langle x, y_1, y_2, \ldots, y_k \rangle \in \mathsf{B} \end{aligned}$$

  where Q_k is the quantifier $\exists$ if k is odd, and it is $\forall$ if k is even.

- *a language* A *is in* Π_k^{P} *if and only if there exists a language* $\mathsf{B} \in \mathsf{P}$ *and a polynomial* p *such that for any string* x

$$\begin{aligned} x \in \mathsf{A} \quad \Leftrightarrow \quad & (\forall y_1, |y_1| \le p(|x|))(\exists y_2, |y_2| \le p(|x|)) \\ & \ldots (Q_k y_k, |y_k| \le p(|x|))\langle x, y_1, y_2, \ldots, y_k\rangle \in \mathsf{B} \end{aligned}$$

  where Q_k *is the quantifier* $\forall$ *if* k *is odd, and it is* $\exists$ *if* k *is even.*

Thus we can redefine the complexity classes Σ_k^{P} and Π_k^{P} of the polynomial time hierarchy directly from the familiar class P, without the need to introduce oracle NTMs. And as we have just proved, this definition is equivalent to our original definition based on oracle NTMs.

3 Alternating Turing Machines

In light of the alternating quantifier formulation of PH, we can define a new computational model called the *alternating Turing machines* (or NTM for short). They can be seen as a combination of non-deterministic Turing machines and co-non-deterministic Turing machines.

The defining feature of an *alternating Turing machine* is that its states are divided into two groups, called $\forall$-states and $\exists$-states. Like a non-deterministic TM, the computation of an *alternating Turing machine* on an input string can be conceptualized as a configuration tree. Each node in the tree corresponds to a possible configuration the machine may enter during the computation on the given input string. A configuration can be classified as a $\forall$-configuration or a $\exists$-configuration, according to the state it's in. A $\forall$-configuration accepts if and only if all children configurations accept; an $\exists$-configuration accepts if and only if any children configuration accepts.

We can define complexity classes based on *alternating Turing machines.* Like DTMs and NTMs, we use terms like ATIME and ASPACE to describe these complexity classes.

We have the following relationships between complexity classes defined on ATMs and complexity classes defined on DTMs and NTMs

Theorem 11 *For any (fully time constructible) function* t*, with* $t(n) \ge n$*,* $\mathsf{ATIME}(t(n)) \subseteq \mathsf{DSPACE}((t(n))^2)$*.*

This is theorem 3.17 on page 91 of Du and Ko's book [1]. You can read that for the proof.

Theorem 12 *For any (fully space constructible) function* s*, with* $s(n) \ge n$*,* $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{ATIME}((s(n))^2)$*.*

This is theorem 3.18 on page 92 of Du and Ko's book [1]. You can read that for the proof.

A unique resource measure for ATMs is the number of alternations (from $\forall$-configurations to $\exists$-configurations or vice versa) on any computational path. If this number is bounded by a constant k, then we will get complexity classes Σ_k^{P} and Π_k^{P}.

For more connections between the ATMs and the polynomial time hierarchy, we have the following relationships

Theorem 13 $\cup_{k>0}\mathsf{ATIME}(n^k) = \mathsf{PSPACE}$

Again the scriber refers to Du and Ko's book [1] (corollary 3.19 on page 93) for the proof.

Theorem 14 $\cup_{c>0}\mathsf{ASPACE}(c \log n) = \mathsf{P}$

Please see corollary 3.22 clause (a) on page 95 of Du and Ko's book [1] for the proof.

References

[1] Ding-Zhu Du and Ker-I Ko. *Theory of Computational Complexity.* John Wiley & Sons, Hoboken, New Jersey, 2000.