CS 710: Complexity Theory Date: Feb. 18th, 2020

Lecture 9: Polynomial Hierarchy Theorem and Alternating Turing Machine

Instructor: Jin-Yi Cai Scribe: Jialu Bao

The document has not been scrutinized as peer-reviewed works would have. The scribe is responsible for any errors within it.

1 Theorem $NP^{NP^{\cdot^{\cdot^{\cdot^{NP}}}}} = \Sigma_k^p$

We want to prove that for any oracle A, any k, the "tower" of height k with A at the top is equivalent to $\Sigma_k^{P^A}$. Formally,

$$\underbrace{NP^{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}}_{\text{k}} = \Sigma_k^p$$

In last lecture, we have proved that

Claim 1. *For any oracle A, $NP^A = \exists.P^A$*

Claim 2. *For any oracle A, $NP^{\exists.P^A} = \exists.\forall.P^A$*

where $\exists.\mathcal{C}$ and $\forall.\mathcal{C}$ are defined as

$$\exists.\mathcal{C} = \{\{x \in \Sigma^* \mid (\exists^P y)\langle x, y\rangle \in L\} \mid L \in \mathcal{C}\}$$
$$\forall.\mathcal{C} = \{\{x \in \Sigma^* \mid (\forall^P y)\langle x, y\rangle \in L\} \mid L \in \mathcal{C}\}$$

For instance,

$$\begin{aligned}
\exists.\forall.P &= \{\{x \in \Sigma^* \mid (\exists^P y \forall^P z)\langle x, y, z\rangle \in L\} \mid L \in P\} \\
&= \{\{x \in \Sigma^* \mid (\exists^P y \forall^P z) D(x, y, z)\}\} \text{ s.t. the decision problem } D(x,y,z) \text{ is in } P \\
&= \Sigma_2^P \qquad (1) \\
\underbrace{\exists.\forall.\ldots}_{k} P^A &= \{\{x \in \Sigma^* \mid (\exists^P x_1 \forall^P x_2 \ldots \forall^P / \exists^P x_k)\langle x_0, x_1, \ldots, x_k\rangle \in L\} \mid L \in P^A\} \\
&= \{\{x \in \Sigma^* \mid (\exists^P x_1 \forall^P x_2 \ldots \underbrace{\forall^P}_{\text{if } k \text{ odd}} / \underbrace{\exists^P}_{\text{if } k \text{ even}} x_k) D(x_0, x_1, \ldots, x_k)\}\} \text{ s.t. } D(x_0, x_1, \ldots, x_k) \text{ is in } P^A \\
&= \Sigma_k^{P^A} \qquad (2)
\end{aligned}$$

Using claim 1 and claim 2 and the fact that $\exists.P = NP, \exists.\forall.P = \Sigma_2^P$, we can build up the proof for the theorem.

Claim 3.

$$\underbrace{NP^{NP^{\cdot^{\cdot^{\cdot^{NP}}}}}}_{k} A \subseteq \Sigma_k^{P^A} \qquad (3)$$

Proof. We prove this by induction.

$k = 1, NP = \Sigma_1^P$**:** Trivial.

$k = 2, NP^{NP} = \Sigma_2^P$: Based on claim 1, we can substitute NP^A for $\exists.P^A$ in the left hand side of claim 2 and get

$$NP^{NP} = \exists.\forall.P^A$$

Since $\exists.\forall.P^A = \Sigma_2^P$ according to eq. (1), it follows that $NP^{NP} = \Sigma_2^P$.

$k > 2$**:** By letting the oracle A in claim 1 be $NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}$ that has $k-2$ level of NP towering, we derive

$$\begin{aligned} NP^{\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2}} &= \exists.P^{\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2}} \\ \Longrightarrow NP^{\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-1}} &= NP^{\exists.P^{\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2}}} \end{aligned} \tag{4}$$

Then by letting the oracle A in claim 2 be $NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}$ that has $k-2$ level of NP towering, we derive

$$NP^{\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-1}} = NP^{\exists.P^{\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2}}} = \exists.\forall.P^{\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2}} \tag{5}$$

Here we take a seemingly reckless move to relaxing the base class from P to NP and derive that

$$P^{\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2}} \subseteq NP^{\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2}} = \underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-1} = \Sigma_{k-1}^P \tag{6}$$

where the second to the last step follow from the inductive hypothesis that

$$\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-1} = \Sigma_{k-1}^P$$

At this stage, a sanity check may disappoint us, as it seems we cannot prove what we wanted:

$$
\begin{aligned}
\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-1} &= NP^{\exists.P\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2}} \quad \textbf{(by eq. (4))} \qquad (7)\\
&= \exists.\forall.P\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2} \quad \textbf{(by eq. (5))}\\
&= \exists.\forall.NP\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2} \quad \textbf{(By } P \subseteq NP\textbf{)} \qquad (8)\\
&= \exists.\forall.\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-1} \quad \textbf{By eq. (6)}\\
&= \exists.\forall.\Sigma_{k-1}^P = \Sigma_{k+1}^P
\end{aligned}
$$

This proves the left hand side of eq. (3) is a subset of Σ_{k+1}^P, while we want to prove that it is subset of Σ_k^P, which is tighter. Fortunately, for any oracle A, P^A is closed under complement, so for any oracle A and $k' \geq 0$, $P^A \subseteq \Sigma_{k'}^P$ implies that $P^A \subseteq \Pi_{k'}^P$ (See lemmas below for a more detailed proof). Therefore, from eq. (6) we can also derive that

$$
P\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-2} \subseteq \Pi_{k-1}^P \qquad (9)
$$

Substitute eq. (9) into the right hand side of eq. (8), we get

$$
\begin{aligned}
\underbrace{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}_{k-1} &\subseteq \exists.\forall.\Pi_{k-1}^P\\
&= \exists.\forall.\forall.\Sigma_{k-2}^P = \Sigma_k^P
\end{aligned}
$$

where the last step follows from collapsing two $\forall$ together.

This completes the proof for claim 3. □

Lemma 1. *If complexity class $\mathcal{C}_1 \subseteq \mathcal{C}_2$, and co-$\mathcal{C}_1$, co-$\mathcal{C}_2$ are respectively their their complement classes, then co-$\mathcal{C}_1 \subseteq$ co-$\mathcal{C}_2$.*

Proof. For any L, if $L \in$ co-$\mathcal{C}_1$, then $L^C \in \mathcal{C}_1$ as $\mathcal{C}_1$ is the complement of co-$\mathcal{C}_1$. Since $\mathcal{C}_1 \subseteq \mathcal{C}_2$, it must $L^C \in \mathcal{C}_2$ too. By co-$\mathcal{C}_2$ is the complement of $\mathcal{C}_2$, $L = (L^C)^C \in$ co-$\mathcal{C}_2$ too. Thus, co-$\mathcal{C}_1 \subseteq$ co-$\mathcal{C}_2$. □

Lemma 2. *For any oracle A, $P^A =$ co-(P^A).*

Proof. Given any A, if $L \in P^A$, then there exists a polynomial time turing machine TM to decide L with access to oracle A. Let TM' be the same as TM except adding one step flipping the "yes", "no" answer in the end, then TM' can decide L^C. Since TM' is still a polynomial time turing machine with access to oracle A, $L^C \in P^A$ too. Thus, co-$(P^A) \subseteq P^A$.

Then, for any $L \in \mathrm{P}^A$, it must $L^C \in$ co-$P^A \subseteq P^A$. Taking the complement again, we have $L = (L^C)^C \in$ co-P^A. So $P^A \subseteq$ co-(P^A).

Therefore, $P^A =$ co-(P^A) □

Thus, $P^A \subseteq \Pi_{k'}^P$ follows from co-$P^A \subseteq \Pi_{k'}^P$ according to lemma 1, which according to lemma 2 follows from the precondition that $P^A \subseteq \Sigma_{k'}^P$.

Claim 4.

$$\underbrace{NP^{NP^{\cdot^{\cdot^{\cdot^{NP}}}}A}}_{k} \supseteq \Sigma_k^{P^A} \tag{10}$$

Proof. According to eq. (2),

$$\Sigma_k^P = \exists.\Pi_{k-1}^{P^A} \tag{11}$$

By inductive hypotheses,

$$\underbrace{NP^{NP^{\cdot^{\cdot^{\cdot^{NP}}}}A}}_{\text{k-1}} \supseteq \Sigma_{k-1}^{P^A} \tag{12}$$

which by lemma 1 implies that co-$(\Sigma_{k-1}^{P^A}) = \Pi_{k-1}^{P^A} \subseteq$ co-$(NP^{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}})$, where the right hand side has $k-1$ level of NP towering. Thus,

$$\begin{aligned}
\Sigma_k^P &\subseteq \exists.\Pi_{k-1}^{P^A} \subseteq \exists.\text{co-}(NP^{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}) \\
&= \{\{x \in \Sigma^* \mid (\exists^P y)\langle x, y\rangle \in L\} \mid L \in \text{co-}(NP^{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}})\} \\
&= NP^{\text{co-}(NP^{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}})}
\end{aligned}$$

Note that $NP^L = NP^{L^C}$, since any non-deterministic turing machine(NTM) having access to L^C can always be simulated by a NTM having access to oracle L by flipping the answer given by the oracle, and vice versa. Therefore,

$$\Sigma_k^P \subseteq NP^{\text{co-}(NP^{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}})} = NP^{NP^{NP^{\cdot^{\cdot^{\cdot^{NP^A}}}}}}$$

This completes the proof for claim 4. □

Combined claim 3 and claim 4, we can conclude that

$$\underbrace{NP^{NP^{\cdot^{\cdot^{\cdot^{NP}}}}A}}_{\text{k}} = \Sigma_k^{P^A}$$

2 Alternating Turing Machine

We can think non-deterministic turing machine as a machine where every state is an OR-state, **i.e.** , at every state, it accepts if there exists a legal move to eventually accepting or it's already an accepting state. So it can be viewed as a disjunction of assertions that moves following from it will

eventually accept. In comparison, every state in a deterministic turing machine is an AND-state, **i.e.** , at every state, it accepts if all legal moves from it eventually accepts or it's already an accepting state. The determinism says that there's only one legal move following it, so we can still view it as a conjunction of assertions that all following legal moves accepts.

Inspired by this perspective, we can imagine an **Alternating Turing Machine**, where every state is either OR or AND state. It is a turing machine where every state inhabits one of types: OR, AND, ACCEPT, REJECT. Formally, a (one-tape) algternating Turing machine is a 5-tuple $M = (Q, \Sigma, \delta, q_0, g)$ where

- Q is the finite set of states
- Σ is the finite alphabet
- $\delta : Q \times \Sigma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ is the transition function
- $q_0 \in Q$ is the initial state
- $g : Q \to \{$ OR, AND, ACCEPT, REJECT$\}$ specifies the type of each state.

and the decision of the machine on input is based on the type of state as described in the last paragraph. As a result, an AND state with no following legal moves accepts no matter if its current state has type ACCEPT or not, and an OR state with no following legal moves rejects as long as its current state is not ACCEPT.(See [1] for more information)

The concept of alternating Turing machine is somewhat analagous to game trees, which the definition here does not require an OR state always followed by an AND state and an AND state always followed by an OR state.

A couple of results with regard to Alternating Turing machines are

1. ALogspace(ALternating Turing Machine with Log space) = PTIME
2. AP= PSPACE
3. APSPACE = EXPT
4. AEXPT = EXPSPACE

3 Karp-Lipton Thoerem

We define a seemingly strange set of classes below

Definition 1. *Given a class $\mathcal{C}$, define $\mathcal{C}/_{Poly}$ consists of languages L that is decidable in $\mathcal{C}$ with advice only dependent on the size of input and has polynomial length to that size. Formally, $L \in \mathcal{C}/_{Poly}$ if and only if there exists $L_0 \in P$, such that for any n, there exists $y(n) \in \{0,1\}^n$ such that $x \in L \Leftrightarrow \langle x, y(|x|)\rangle \in L_0$.*

$\mathcal{C}/_{Poly}$ is strange in that it can contain undecidable problems. Given a set of enumerable Turing machines $M_n(\sqcup)$, $\{1^n \mid M_n(\sqcup)$ halts $\}$, is a variation of the famously undecidable halting problem. However,$\{1^n \mid M_n(\sqcup)$ halts $\}$ is in $P/_{Poly}$ because if a machine can resort to some miraculous advice y such that $y(n) = 1$ if and only if $M_n(\sqcup)$ halts, then the machine could just accept whenever the input x is all 1's and $y(|x|) = 1$, which could be accomplished by a deterministic Turing Machine in polynomial time.

Claim 5. *$L \in P/_{Poly}$ if and only if there exists a family of boolean circuits $\{C_n\}$ of polynomial size such that for any $x \in \{0,1\}^n$, $x \in L$ is equivalent with $C_n(x) = 1$.*

Theorem 1 (Karp-Lipton Theorem)**.** *If $NP \subset P/_{Poly}$ then $\Sigma_2^p = \Pi_2^p$, and the whole polynomial hierarchy PH collapsed to $\Sigma_2^p = \Pi_2^p$.*

As we noted before, $P/_{Poly}$ contain undecidable problems, which are clearly not in NP, so $NP \neq P/_{Poly}$. By claim 5, there is an equivalent way of stating the Karp-Lipton Theorem: if any problem L in NP can be solved with a family of Boolean Circuits $\{C_n\}$ – solved means that $\forall x \in \{0,1\}^n$, $x \in L \equiv C_n(x) = 1$, then the polynomial hierarchy collapsed.

References

[1] https://en.wikipedia.org/wiki/Alternating Turing machine

[2] J. Cai. Lectures in Computational Complexity, 2003