CS 4510 Automata and Complexity April 24, 2023

Lecture 22: The Polynomial Hierarchy

Lecturer: Abrahim Ladha *Scribe(s): Michael Wechsler*

1 Motivation

We will characterize the same thing three ways and hopefully prove just one theorem. Recall P is the class of languages we characterize as having a machine M which decides in polynomial time.

$$M(w) \text{ accepts} \iff w \in L$$

Recall NP is the class of languages verifiable in polynomial time, or decidable in nondeterministic polytime.

$$\exists x M(w, x) \text{ accepts} \iff w \in L$$

You may either think of x as the witness of a deterministic verifier or as a sequence of decisions or guesses that a nondeterministic machine makes. For M to run in polynomial time, we require that x be polysized. Recall that an NTM accepts if there exists ($\exists$) a computation branch. You may also recall that SAT is NP-complete, and we say that $\phi \in$ SAT if there exists ($\exists$) a satisfying assignment of the boolean variables of ϕ.

Recall coNP is the class of languages such that $\overline{L} \in \mathsf{NP} \iff L \in \mathsf{coNP}$. We can take the logical complement[1] of the definition of NP for a definition of coNP as

$$\forall x M(w, x) \text{ accepts} \iff w \in \overline{L}$$

Like SAT is NP-complete, coNP has its own complete problem called tautologies. TAUT $= \{ \phi \mid$ every assignment of ϕ is satisfying$\}$. We observe that NP and coNP have this interesting duality. A ying-yang structure emerges. NP is characterized by the existential quantifier $\exists$ ($\exists$ a witness, $\exists$ an assignment of SAT, or $\exists$ an accepting computation). coNP is characterized by the universal quantifier $\forall$ ($\forall$ witnesses, $\forall$ assignments of TAUT, or $\forall$ branches must be accepting). This duality motivates our discussion.

2 Generalizations of NP and coNP

For any class C, define the class $\exists C$ such that if M was a C-machine with a definition like

$$M(w) \text{ accepts} \iff w \in L \in C$$

then M' is a $\exists C$ machine such that

$$\exists x M'(w, x) \text{ accepts} \iff w \in L \in \exists C$$

[1] It should really be "$\forall x M(w, x)$ rejects" but we don't care abour rejection from L here, as we want to care about acceptance into $\overline{L}$. This distinction is arbitrary.

We naturally augment these deciders to take on witnesses[2]. We similarly define the class $\forall C$.

What is $\exists\mathsf{P}$? $\exists x M(w,x)$ accepts $\iff$ $w \in L$ and M runs in polytime. M is just a polytime verifier! So $\exists\mathsf{P}$ = classes of languages verifiable in polytime. Thus $\exists\mathsf{P} = \mathsf{NP}$. Similarly $\forall\mathsf{P} = \mathsf{coNP}$.

What is $\exists\exists\mathsf{P}$? $\exists x_1 \exists x_2 M(w, x_1, x_2)$ accepts and M is polytime? That's just two witnesses. It just complicates things. Each witness can be at most a polynomial number of bits anyway, so two witnesses is just a constant larger so $\exists\exists\mathsf{P} = \exists\mathsf{P} = \mathsf{NP}$. Similarly, $\forall\forall\mathsf{P} = \forall\mathsf{P} = \mathsf{coNP}$. Anytime we have a sequence of the same quantifier, we may compress them to one quantifier.

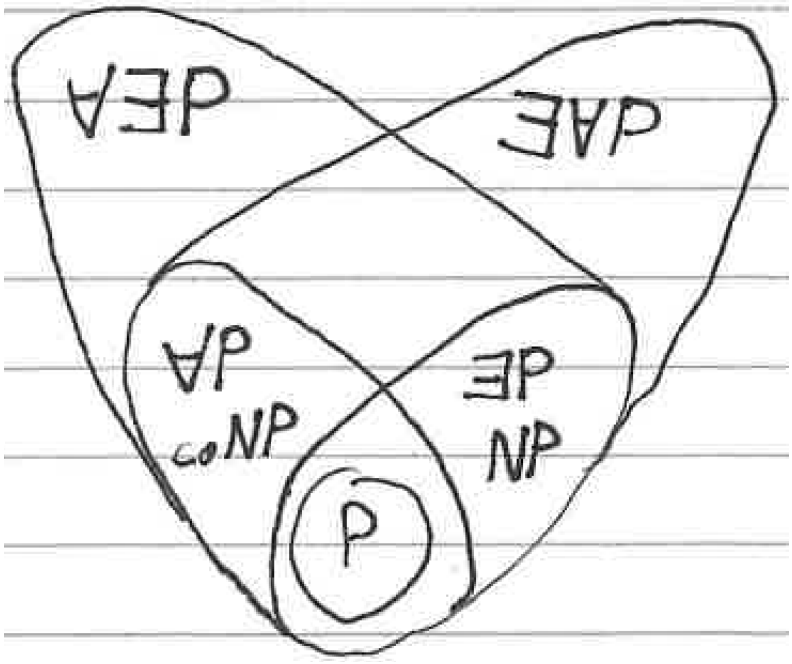
$$\exists\exists ... \exists\mathsf{P} = \exists\mathsf{P} = \mathsf{NP}$$

What about $\exists\forall\mathsf{P}$ and $\forall\exists\mathsf{P}$? Now things are getting interesting.

$$\exists x_1 \forall x_2 M(w, x_1, x_2) \text{ accepts } \iff w \in L \in \exists\forall\mathsf{P}$$

Note that we may add an $\exists$ quantifier to a $\forall\mathsf{P}$ machine which it ignores to show $\forall\mathsf{P} \subseteq \exists\forall\mathsf{P}$. We convert a $\forall\mathsf{P}$ machine to a $\exists\forall\mathsf{P}$ machine which ignores this witness. Adding an ignored parameter changes nothing of the program structure, so our machine still decides the same language. Since we can perform this surgery, $\forall\mathsf{P} \subseteq \exists\forall\mathsf{P}$. Similar logic can be used to show $\forall\mathsf{P} \subseteq \forall\exists\mathsf{P}$ and $\exists\mathsf{P} \subseteq \forall\exists\mathsf{P}$ and $\exists\mathsf{P} \subseteq \exists\forall\mathsf{P}$. Observe that $\forall\exists\mathsf{P}$ and $\exists\forall\mathsf{P}$ appear to be larger than $\exists\mathsf{P}$ and $\forall\mathsf{P}$.

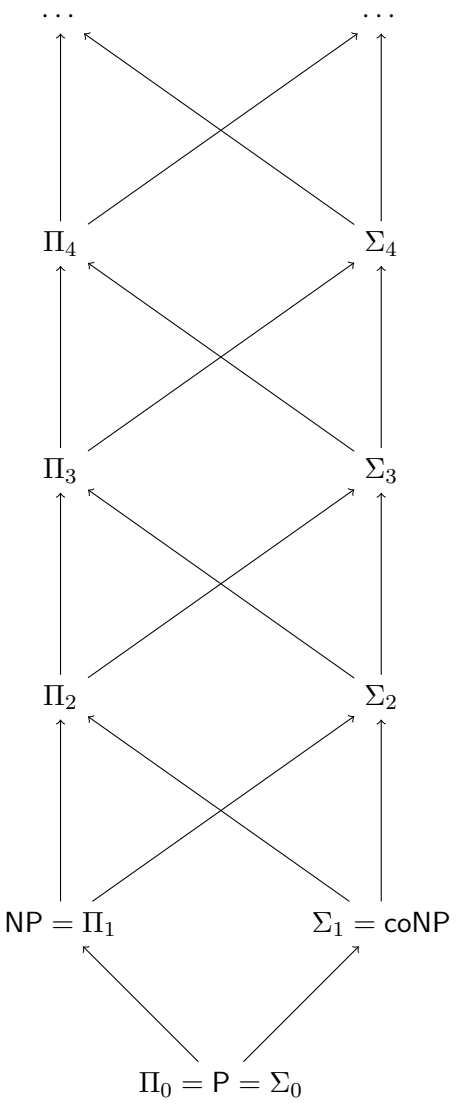
So does $\exists\forall\mathsf{P} = \forall\exists\mathsf{P}$? We don't know! $\exists\forall\mathsf{P}$ and $\forall\exists\mathsf{P}$ appear to have the same duality and dance that NP and coNP have.



[2]Although these auxilliary inputs maybe universally quantified and therefore not "witnessing" anything, we will still call them witnesses.

3 The Polynomial Hierarchy

We may repeat this argument on $\forall\exists\mathsf{P}$ and $\exists\forall\mathsf{P}$ as we did on $\exists\mathsf{P}$ and $\forall\mathsf{P}$. We see by an inductive or recursive argument this creates an infinite and alternating hierarchy. Countably many more generalizations of NP and coNP and their dualities. Here we use arrows to show containment to avoid too complex of a venn diagram.



Let $\Pi_0 = \Sigma_0 = \mathsf{P}$ and inductively define

$$\Sigma_i = \exists\Pi_{i-1} = \underbrace{\exists\forall\exists\forall\exists\ldots}_{\text{i}}\mathsf{P}$$

$$\Pi_i = \forall\Sigma_{i-1} = \underbrace{\forall\exists\forall\exists\forall\ldots}_{\text{i}}\mathsf{P}$$

It is important that the first quantifier of Σ_i is existential and the first quantifier of Π_i is universal. We define the polynomial hierarchy to be the class $\mathsf{PH} = \bigcup_{i=0}^{\infty}\Sigma_i = \bigcup_{i=0}^{\infty}\Pi_i$. We define a "level" of the polynomial hierarchy to be $\Pi_i \cup \Sigma_i$ for some i.

Note that by a generalizing the argument we made for NP and coNP, we see

$$\forall i\ \Pi_i \subseteq \Pi_{i+1}$$
$$\forall i\ \Sigma_i \subseteq \Sigma_{i+1}$$
$$\forall i\ \Sigma_i \subseteq \Pi_{i+1}$$
$$\forall i\ \Pi_i \subseteq \Sigma_{i+1}$$

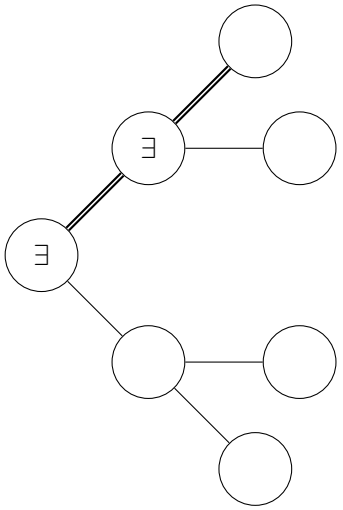
Whether or not these levels are strict is an open problem. Each level is defined only using finitely many quantifiers, so it would appear that $\mathsf{PH} \subseteq \mathsf{PSPACE}$ since $TQBF$ is a PSPACE-complete problem. If that containment is strict, it is also a an open problem. We would hope to show it is since $\mathsf{P} \subseteq \mathsf{PH} \subsetneq \mathsf{PSPACE} \Rightarrow \mathsf{P} \neq \mathsf{PSPACE}$. This is truly a beautiful class with beautiful structure of mostly theoretical interest.

We give two more quick equivalent characterizations of the polynomial hierarchy. First is using oracles:

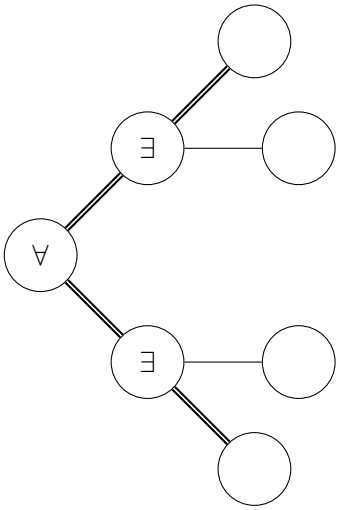
$$\Sigma_0 = \mathsf{P},\ \Sigma_1 = \mathsf{NP},\ \Sigma_2 = \mathsf{NP}^{\mathsf{NP}},\ \Sigma_3 = \mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}},\ \Sigma_i = \underbrace{\mathsf{NP}^{\mathsf{NP}^{\cdots^{\mathsf{NP}}}}}_{\text{i}},\ \ \Pi_i = \text{co-}\Sigma_i$$

We only mention this and leave it unjustified, but it is perhaps believable. The second characterization uses a generalized non-deterministic Turing Machine called an alternating

Turing machine. While a nondeterministic Turing machine accepts if just one branch accepts, an alternating Turing machine may pick from two transition functions at any step if it wants to require all branches to accept or just one:



NTM



ATM

For AP = alternating polynomial time, since $TQBF$ is PSPACE-complete, and an unbounded alternating polytime machine can simulate $TQBF$ problems alternating quantifiers. Like how SAT is NP-complete, $TQBF$ is AP-complete, and so AP = PSPACE.

We say a Σ_i-machine is one in which the first branch is at an existential one, and there are at most i existential or universal branching steps. We similarly define a Π_i-machine as an ATM which the first branching is a universal one, and there at most i universal or existential branching steps to depth i. Try to convince yourself that $\mathsf{NP} = \Sigma_i - \text{TIME(poly)}$, as you reformulate many guesses into just one. Rather than make a sequence of nondeterministic guesses, just make one bigger one. Why make two sequential coin flips when you can roll a four sided die.

$$\Sigma_i = \bigcup_{k=0}^{\infty} \Sigma_i\text{-TIME}(n^k)$$

$$\Pi_i = \bigcup_{k=0}^{\infty} \Pi_i\text{-TIME}(n^k)$$

Although the polynomial hierarchy may seem of a flamboyant, inapplicable interest, like other part of complexity, there are deep connections and ties. It may also be used to separate the complexity classes worth studying. Also most importantly, it looks cool.

4 Collapse

If for any i, $\Pi_i = \Sigma_i$ then $\mathsf{PH} \subseteq \Pi_i = \Sigma_i$. The polynomial hierarchy collapses to this i^{th} level. Each Π_i, Σ_i behave like a struct or pillar, supporting the levels above them. Were it the case that two distinct pillars were the same, our tower of Babel collapses. We prove a weaker idea, that if $\mathsf{P} = \mathsf{NP}$, then $\mathsf{PH} \subseteq \mathsf{P}$. That is, if $\Sigma_0 = \Sigma_1$, then the entire hierarchy collapses to Σ_0.

Assume $\mathsf{P} = \mathsf{NP}$. We proceed by induction. For $i = 1$, $\Pi_i = \mathsf{coNP}$ and $\Sigma_i = \mathsf{NP}$ are both $\subseteq \mathsf{P}$ by assumption. Now suppose that $\Pi_{i-1}, \Sigma_{i-1} \subseteq \mathsf{P}$. We prove $\Sigma_i \subseteq \mathsf{P}$. Since P is closed under complement and $\mathrm{co}\Sigma_i = \Pi_i$, this will also prove $\Pi_i \subseteq \mathsf{P}$ as desired. Let $L \in \Sigma_i$, then

$$w \in L \iff \underbrace{\exists x_1 \forall x_2 \ldots}_{i \text{ times}} M(w, x_1, x_2, \ldots) \text{ accepts}$$

Define

$$L' = \{\langle w, x_1 \rangle \mid \underbrace{\forall x_2 \exists x_3 \ldots}_{i-1 \text{ times}} M(w, x_1, x_2, \ldots) \text{ accepts } \}$$

Notice $L' \in \Pi_{i-1}$ since there are $i-1$ quantifiers and it begins with $\forall$. By our assumption that $\Pi_{i-1} \subseteq \mathsf{P}$ we see that $L' \in \mathsf{P}$. Thus, there exists a polytime algorithm for L' called M' such that

$$w \in L' \iff M'(w) \text{ accepts}$$

We may tranform this definition of L' into the original one for L to get

$$w \in L \iff \exists x_1 M'(w, x_1) \text{ accepts}$$

This is a polytime verifier, so $L \in \mathsf{NP}$. Since we assumed $\mathsf{P} = \mathsf{NP}$,we observe that $L \in \mathsf{P}$. Since L was any Σ_i language, then $\Sigma_i \subseteq \mathsf{P}$. So if $\mathsf{P} = \mathsf{NP}$, we may conclude that the polynomial hierarchy collapses to its 0^{th} level, to P, to ashes.

5 Karp-Lipton Theorem

This will be our final theorem of the class. Interpreting its statement is more difficult than the actual proof. It plainly states:

$$\mathsf{NP} \subseteq \mathsf{P/poly} \Rightarrow \mathsf{PH} \subseteq \Pi_2 \cup \Sigma_2$$

If SAT has a polynomial sized circuit family, then we do not have a polynomial hierarchy, it collapses to the second level. Originally Karp and Lipton proved it by a collapse to the third level, but Sipser improved it to the second level. The proof idea is to show $\Pi_2 \subseteq \Sigma_2$. Conversion of any $\forall\exists$-sentence to a $\exists\forall$-sentence means for any sentence higher in the hierarchy, we can repeatedly alternate and compress quantifiers until a sentence with many quantifiers is left with only two.

Let $\mathsf{NP} \subseteq \mathsf{P/poly}$, then SAT has a polynomial sized circuit family, $\{C_0, C_1, \ldots\}$. Each C_n takes as input an n-variable formula and outputs a single bit for yes/no if the input

formula was satisfiable. By a decision-to-search transformation, there exists a circuit family $\{C'_0, C'_1, \ldots\}$. Where each C'_n outputs n bits for not just if it was satisfiable or not, but the satisfying assignment itself. Since each C_n is polynomially-sized, so is each C'_n. This decision-to-search transformation should be believable, but to give you an example suppose we had a circuit to say if some formula ϕ was satisfiable or not. If x_1 is the first variable of ϕ, then $\phi \wedge x_1$ is a formula which is satisfiable if and only if ϕ was satisfiable, with $x_1 = 1$. We can play a hotter/colder game with the circuit families to infer not just if a formula was satisfiable, but what the actual satisfying assignment was. This decision-to-search transformation will incur only a polynomial overhead.

Let $L \in \Pi_2$. Then

$$w \in L \iff \forall x_1 \exists x_2 M(w, x_1, x_2) \text{ accepts}$$

We may convert M to a CNF φ_M using a Cook-Levin style construction. Note since M runs in a polynomial number of steps, φ_M is polynomially sized, and its construction takes polynomial time. Now, notice $\exists k$ such that $C'_k(\varphi_M, w, x_1) = x_2$. Since SAT has a polynomial sized circuit family, there exists a polysized circuit to search for this witness instead of quantifying over it. We can replace the existential quantification of x_2 with a computation of C'_k. Then rather than actually computing C'_k, we can just existentially quantify over it. So our definition of L has an equivalent statement:

$$\forall x_1 \exists x_2 M(w, x_1, x_2) \text{ accepts} \iff \exists C'_k \forall x_1 M(w, x_1, C'_K(\varphi_M, w, x_1)) \text{ accepts}$$

We may simply use existential quantification to guess the C'_k circuit. We converted a Π_2 sentence into a Σ_2 one! $L \in \Pi_2 \Rightarrow L \in \Sigma_2 \Rightarrow \Pi_2 \subseteq \Sigma_2$. We observe that if $\mathsf{NP} \subseteq \mathsf{P/poly}$, we collapse PH.

6 Further Study

I want to conclude with some advice on how to self study complexity theory. Your journey doesn't have to end here if you don't want it to. First, finish the Sipser book. It does not contain everything, but it does contain the best proofs of what it does cover. I wish it had a second volume. It's coverage of randomness, interaction, cryptography, and more may surprise you. Before you go further, you should definitely finish Sipser. After you finish Sipser, go through the first six chapters of the Arora-Barak book. All the other chapters (7+) cover an incredible breadth of material, and good pointers to other sources. These may include communication complexity, quantum complexity, the complexity of counting, and so on. Each of these chapters deserves (and has) their own books, but it's an introduction to these theories. You need to know what the things you don't know are called in order to google and learn them. Then go through Goldreich's and Papadimitriou's books. Goldreich has 400 pages of incredible notes. Finally, I recommend the books *The Nature of Computation* and Wigderson's *Mathematics and Computation*. Both of these are light on proofs, as a tradeoff for coming with incredible wisdom. If you want a proof, use the other books. If you want to what a proof means, use Wigderson's book. Of course, you may use me as a resource. If you have any questions, or come across anything in your own independent study, I would be happy to help and answer. Thank you for taking my class, I had a lot of fun.

- Introduction to the Theory of Computation, Michael Sipser 2012
- Computational Complexity: A Modern Approach, Sanjeev Arora and Boaz Barak, 2009
- Computational Complexity: A Conceptual Perspective, Oded Goldreich, 2008
- Computational Complexity, Christos Papadimitriou, 1994
- Mathematics and Computation, Avi Wigderson, 2019
- Goldreich's encyclopedic lecture notes. 375 pages across two semesters. An invaluable resource from 1999.
  `http://gen.lib.rus.ec/book/index.php?md5=fbb240574e6f5059fccdce95fab0ff38`
- Hatami's notes from 2022, also insanely useful.
  `https://www.cs.mcgill.ca/~hatami/comp531-F2022/files/Lectures.pdf`