CS 4510 Automata and Complexity November 29, 2023

Lecture 24: The Polynomial Hierarchy (Draft)

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

1 coNP

Before we understand the polynomial hierarchy, we should understand coNP, so lets quickly review NP. NP is a class of languages with many characterizations. We may say $L \in \mathsf{NP}$ if:

- For a nondeterministic Turing machine which on input $w \in L$, *there exists* an accepting computation branch.
- For a polynomial time deterministic verifier $V(w, c)$, *there exists* a witness, or certificate to convince the verifier to accept the input.
- $L \leq_p SAT$ where $\phi \in SAT$ if *there exists* a satisfying assignment to ϕ.

Note the emphasis. These characterizations of NP all seem to involve existentiality ($\exists$).

Definition 1.1. We let coNP be the class of languages such that $L \in \mathsf{coNP} \iff \overline{L} \in \mathsf{NP}$.

We can characterize coNP analogously as we did NP. We care more about acceptance into $L \in \mathsf{coNP}$ than rejection from $\overline{L} \in \mathsf{NP}$. We may say $L \in \mathsf{coNP}$ if:

- We say a co-nondeterministic machine is one which accepts an input if all branches accept and and rejects an input if there exists a rejecting branch. For a co-nondeterministic Turing machine which on input $w \in L$, *all* branches must lead to an accept state.
- For a polynomial time deterministic machine $M(w, c)$, *for all* possible "witnesses", or certificates c, each one convinces the machine M to accept w.
- Both $\overline{SAT}$ and $TAUT$ are coNP-complete. $\phi \in TAUT$ if *every* assignment of ϕ is satisfying. Similarly, $\phi \in \overline{SAT}$ if *every* assignment of ϕ is unsatisfying.

Note the emphasis. These characterizations of coNP all seem to involve universality ($\forall$).

We believe that the relationship between $\mathsf{P}, \mathsf{NP}, \mathsf{coNP}$ looks like this.

PICTURE

We do not think that $\mathsf{P} = \mathsf{coNP}$. To verify SAT is quite easy, as the witness is simply a satisfying assignment. How could you verify $\overline{SAT}$? Every assignment needs to be unsatisfying, so how could you convince a verifier no assignment is satifying? It doesn't seem like there is a short witness you could provide for this. You could provide every assignment, but there 2^n of them, and a polytime verifier cannot read them all. We think $SAT \notin \mathsf{coNP}$ for similar reasons.

We know that P is closed under complement, and by definition, $\mathsf{NP} \cap \mathsf{coNP}$ is also closed under complement. But we do not know if $\mathsf{P} = \mathsf{coNP} \cap \mathsf{NP}$.

2 A Logical Definition of the Polynomial Hierarchy

The characterization of NP by $\exists$ and coNP by $\forall$ motivates this definition.

Definition 2.1. 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

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

We naturally augment these deciders to take on witnesses[1]. We similarly define the class $\forall C$ as

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

What is $\exists$P?

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

and M runs in polytime. M is just a polytime verifier! So $\exists$P = classes of languages verifiable in polytime. Thus $\exists \mathsf{P} = \mathsf{NP}$. Note that $|x|$ must also be a polynomial. Similarly what is $\forall$P? Well notice that $L \in \forall \mathsf{P}$ if $\overline{L} \in \exists \mathsf{P}$, so $\forall$P by definition is just coNP. The logical complement of the definition is the set complement of the language. Let us write out co$\exists$P:

$$w \notin L \iff w \in \overline{L} \iff \neg \exists x M(w,x) \text{ accepts} \iff \forall x M(w,x) \text{ accepts} = \forall \mathsf{P}$$

[2]

What is $\exists\exists$P?

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

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 $\exists\exists \mathsf{P} = \exists \mathsf{P} = \mathsf{NP}$. The intuition is there but lets try to prove it a little more rigorously

Theorem 1. $\exists\exists \mathsf{P} = \exists \mathsf{P}$

Proof. We first prove $\exists \mathsf{P} \subseteq \exists\exists P$. Take our $\exists$P machine, and modify its arguments to take on another witness which it simply does not read at all. This is now a $\exists\exists$P machine.

To show $\exists\exists \mathsf{P} \subseteq \exists \mathsf{P}$, take a $\exists\exists$P machine and create an $\exists$P machine. The witness to the $\exists$P machine will be an encoding of the two witnesses to the $\exists\exists$P machine. It splits its witness into two and then simulates the $\exists\exists$P machine.

```
def EP(w,[x_1,x_2]):
    simulate EEP(w,x1,x2)
```

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

[2] This should actually say $\forall x M(w,x)$ rejects but we care about acceptance into $\overline{L}$ rather than rejection from L

□

Similarly, $\forall\forall\mathsf{P} = \forall\mathsf{P} = \mathsf{coNP}$. Anytime we have a finite of the same quantifier adjacent to each other, we may compress them to into one.

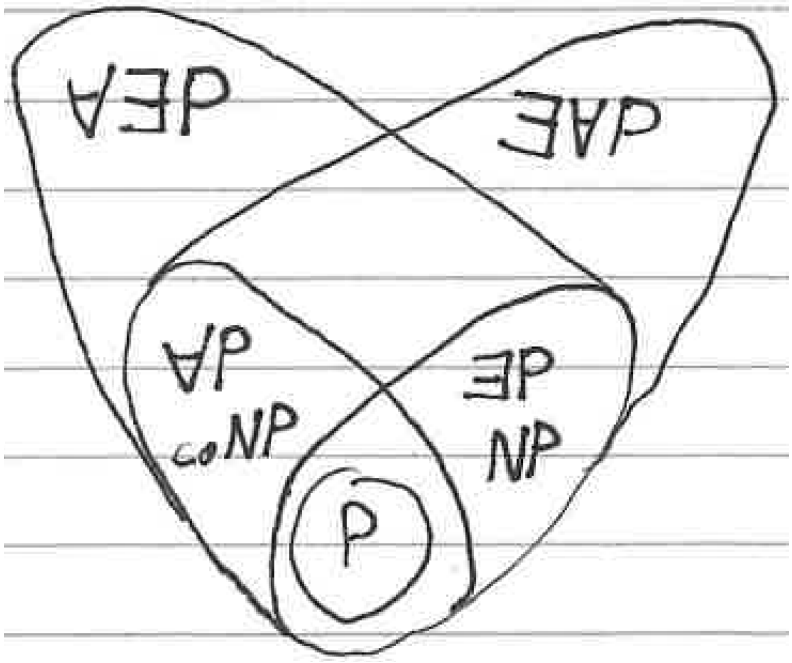
$$\exists\exists ... \exists\mathsf{P} = \exists\mathsf{P} = \mathsf{NP}$$

What about $\exists\forall\mathsf{P}$ and $\forall\exists\mathsf{P}$? Now things are getting interesting. First note that order matters, and in general, quantifiers may not commute. If you consider $\forall\exists\mathsf{P}$

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

This says for every x_1, there is an x_2 to make M accept. For each x_1 there may be a different x_2 but some x_2 must exist for each x_1. What is the relationship between $\mathsf{P}, \forall\mathsf{P}, \exists\mathsf{P}$ and $\forall\exists\mathsf{P}, \exists\forall\mathsf{P}$?

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.



What about Note that $\exists\exists\forall\mathsf{P} = \exists\forall\mathsf{P}$, so we really only want to study the classes which have alternating quantifiers. We are now ready to give a formal definition of the polynomial hierarchy. We define the levels of the polynomial hierarchy through a double induction.

Definition 2.2. Let $\Pi_0 = \Sigma_0 = \mathsf{P}$ and inductively define

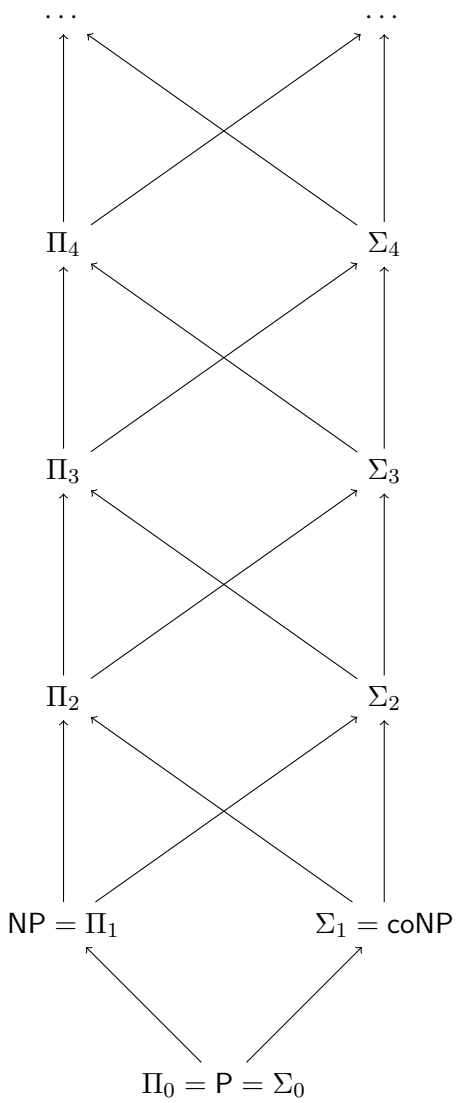
$$\Sigma_i = \exists\Pi_{i-1} = \underbrace{\exists\forall\exists\forall\exists\ldots}_{i}\mathsf{P}$$

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

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.



We prove that the polynomial hierarchy looks like this by proving the following relationships:

Theorem 2.

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

Proof. We simply generalize the argument we gave for why $\forall\exists\mathsf{P}$ contains $\forall\mathsf{P}$ and $\exists\mathsf{P}$. Given a Σ_{i+1} machine, it is defined like $\exists\forall\exists\forall\exists...\mathsf{P}$. Simply consider the machine which ignores its first universal quantifier. We may then compress the first and third quantifiers. This is a Σ_i machine so $\Sigma_i \subseteq \Sigma_{i+1}$. Consider the machine which ignores its first existential quantifier. This is a Π_i machine so $\Pi_i \subseteq \Sigma_{i+1}$. The proof follows identically to show Π_{i+1} contains Π_i and Σ_i. □

Whether or not these levels are strict is an open problem. This is truly a beautiful class with beautiful structure of mostly theoretical interest. We will discuss its connections and implications next time.

3 An Oracle Definition of the Polynomial Hierarchy

There exists an oracle characterization of the polynomial hierarchy as well.

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

Definition 3.1 (Polynomial Time Hierarchy)**.** Let $\Sigma_0 = \Pi_0 = \mathsf{P}$. Then inductively define the level Σ_i and Γ_i with an oracle to previous level.

$$\Sigma_i = \mathsf{NP}^{\Sigma_{i-1}}$$
$$\Pi_i = \mathsf{coNP}^{\Sigma_{i-1}}$$

This is a definition but we should prove its equivalence.

Proof. We proceed by induction. The base case holds. Let Σ_i be defined using the logical definition. We prove that $\Sigma_i = \mathsf{NP}^{\Sigma_{i-1}}$.

Let $L \in \Sigma_i$. So there is a $\exists\forall\exists\forall...Q_i\mathsf{P}$ machine to decide L. We give a $\mathsf{NP}^{\Sigma_{i-1}}$ machine to simulate it. Our logical definition of L is that there is a machine such that

$$\exists x_1 \forall x_2 \exists x_3 ... M(w, x_1, x_2, .., x_i) \text{ accepts } \iff w \in L$$

.

Suppose you didn't quantify over x_1 and it was simply fixed, hardcoded. The definition would look like

$$\forall x_2 \exists x_3 ... M'(w, x_2, .., x_i) \text{ accepts } \iff w \in L$$

.

This would actually be the negation of an Σ_{i-1} statement, a Π_{i-1} statement. We can bring back in x_1 by nondeterministically guessing it! Our $\mathsf{NP}^{\Sigma_{i-1}}$ machine to decide L will work as follows. On input w first it nondeterministically guesses x_1. Then it queries its Σ_{i-1} with $M(w, x_2, ..., x_i)$ with hardcoded x_1. If the oracle prophesizes no, then our machine will accept. If the query is not in Σ_{i-1}, it must be in Π_{i-1}, so our $\mathsf{NP}^{\Sigma_{i-1}}$ machine correctly simulates this $\exists\Pi_{i-1} = \Sigma_i$ machine.

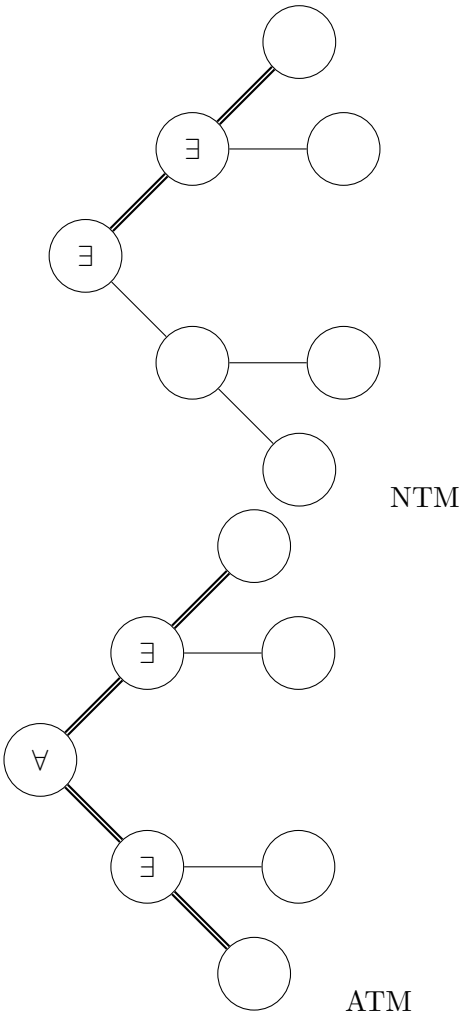
Next we prove $\mathsf{NP}^{\Sigma_{i-1}} \subseteq \Sigma_i$. This proof is TBD. Similar to the reason $\mathsf{NP} = \mathsf{NP}_v$.

□

4 An Alternation Definition of the Polynomial Hierarchy

The final characterization uses a generalized non-deterministic Turing Machine called an alternating Turing machine.

Definition 4.1 (Alternation)**.** 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.



Definition 4.2. Let AP be the class of languages which are decidable by an alternating Turing machine which halts within a polynomial number of steps.

Theorem 3. $\mathsf{AP} = \mathsf{PSPACE}$

Proof. First we prove $\mathsf{AP} \subseteq \mathsf{PSPACE}$. We can simulate an AP machine with only polynomial space with a similar space-reusing divide and conquer approach that we did for the proof that $TQBF \in \mathsf{PSPACE}$.

Next we prove $\mathsf{PSPACE} \subseteq \mathsf{AP}$ by showing TQBF $\in \mathsf{AP}$. If there is a $\exists$ quantifier, or a $\forall$ quantifier, our AP machine branches appropriately. Just like how SAT $\in \mathsf{NP}$ by guessing the assignment, here we show TQBF $\in \mathsf{AP}$ by by using the alternation super power to "guess" the correct answer. □

AP is a class where the machines run in alternating polytime, but are allowed to perform as many alternating moves as they want. What if they were only allowed to perform a finite amount of moves?

Definition 4.3. We say a Σ_i-machine is an alternating machine 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. Similarly $\Sigma_i-\mathsf{TIME}(f(n))$ is the class of language decidable by an alternating machine to depth i beginning with an $\exists$ step which uses at most $f(n)$ steps. The class $\forall_i - \mathsf{TIME}(f(n))$ is defined similarly.

Theorem 4. $\mathsf{NP} = \Sigma_1\mathsf{TIME}(poly)$

Proof. $\Sigma_1\mathsf{TIME}(poly) \subseteq \mathsf{NP}$ obviously, so we focus on the reverse containment.

The whole difficulty in understanding computation is that future steps may be dependent on all those that come previously. An NP machine is allowed to make many guesses, and perhaps future guesses are dependent on guesses made in the past. Or are they? I claim that any NP machine can be converted to one that makes at most a single nondeterministic choice. Simply nondeterminisitically guess the future guesses! 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.

This is analogous to our quantifier compression from the logical definition. □

We may generalize to give our final characterization of the polynomial hierarchy in terms of alternation.

Definition 4.4. For each i, let

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

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

Let us prove it is equivalent to our previous definitions

Proof. We proceed by induction. Notice that $\Sigma_0\mathsf{TIME}(poly)$ is an alternating machine which can make no alternating moves. Thats just a polytime Turing machine, so we see that $\Sigma_0 = \Pi_0 = \mathsf{P}$ and the base case holds.

Remaining proof TBD. □

Although the polynomial hierarchy may seem of a flamboyant, inapplicable interest, like other part of complexity, there are deep connections and ties. Three independent definitions[3] means it is not an imagined class, but a serious one. It may also be used to separate the complexity classes worth studying. Also most importantly, it looks cool.

[3] A fourth exists, using uniform circuit families