CS 6810: Theory of Computing **Fall 2023**

Lecture 17: The Polynomial Hierarchy Continued

Lecturer: Eshan Chattopadhyay *Scribe: Ruqing Xu*

1 Alternating Turing Machines

Definition 1.1. *Alternating Turing Machines (ATMs) can be seen as generalized NDTMs with the following features:*

- *Like an NDTM, an ATM has two transition functions δ_0 and δ_1.*
- *Unlike an NDTM, every state $q \in Q$ in an ATM (except q_{accept} and q_{halt}) has an associated label (either $\exists$ or $\forall$).*

An ATM M accepts a string x if the following conditions hold. Let $G_{M,x}$ be the configuration graph of M on input x. We iteratively define a subset of the nodes in $G_{M,x}$ called ACCEPT*:*

- *If a node is in state q_{accept}, add it to the set* ACCEPT*.*
- *If a node is in a state labeled $\exists$, add it to* ACCEPT *if at least one of its neighbors is in* ACCEPT*.*
- *If a node is in a state labeled $\forall$, add it to* ACCEPT *if both of its neighbors are in* ACCEPT*.*

M accepts an input x if and only if the starting configuration C_{start} is in ACCEPT *when the procedure stops.*

Remark 1.2. *A NDTM can be seen as a special case of ATMs where all states are labeled $\exists$. Therefore, a NDTM accepts a string if there is any path in the configuration graph from C_{start} to nodes with state q_{accept}.*

2 ATM Definitions of the Polynomial Hierarchy

Definition 2.1. *For any function $T : \mathbb{N} \to \mathbb{N}$ and any language $L \subset \{0,1\}^*$, we say $L \in$ $\mathsf{ATIME}(T(n))$ if there exists an ATM M that computes L in $O(T(n))$ time.*

Definition 2.2. *The complexity class* AP *of alternating polynomial time is defined as*

$$\mathsf{AP} = \bigcup_{C \in \mathbb{N}} \mathsf{ATIME}(n^C)$$

Observation 2.3. $\mathsf{AP} = \mathsf{PSPACE}$.

The complexity class ATIME imposes no restrictions on the ATMs used to compute the language. Next, we consider various restrictions on the ATMs and show that this establishes the relationship between ATMs and the polynomial hierarchy.

Definition 2.4. *For any function $T : \mathbb{N} \to \mathbb{N}$ and any language $L \subset \{0,1\}^*$, we say $L \in \Sigma_i\mathsf{TIME}(T(n))$ if there exists an ATM M that computes L in $O(T(n))$ time with the following restrictions: q_{start} is labeled $\exists$, and for any input x and any sequence of choices of transition functions, the machine switches from states with one label and states with the other label at most $i-1$ times.*

Similarly, we can define class $\Pi_i\mathsf{TIME}(T(n))$ with the restriction that q_{start} is labeled $\forall$.

Observation 2.5. $\Sigma_i = \bigcup_{C\in\mathbb{N}} \Sigma_i\mathsf{TIME}(n^C)$ *and* $\Pi_i = \bigcup_{C\in\mathbb{N}} \Pi_i\mathsf{TIME}(n^C)$.

This observation shows that we can think of the levels of polynomial hierarchy as languages computable by the corresponding restricted ATMs. The intuition for the equivalence condition is that the choices of transition functions after each "swtich" of quantifiers of a ATM can be seen as non-deterministic guesses for the certificate following the corresponding quantifier in the decision problem. Then, the acceptance condition of the ATM ensures that a string is in the language if and only if the ATM accepts the string.

3 Oracle Definitions of the Polynomial Hierarchy

Recall that Oracle Turing Machines are Turing Machines that are given oracle access to some language and can query whether a string is in the language during its computations. Also recall that the complexity class NP^O is the set of languages that a NDTM with oracle access to O can compute in polynomial time.

Definition 3.1. *For a boolean formula ϕ and i vectors of boolean variables,*

$$\phi \in \Sigma_i\mathsf{SAT} \iff \exists x_1 \forall x_2 \exists \ldots Q_i x_i \quad \phi(x_1, \ldots, x_i) = 1.$$

where Q_i is $\forall$ or $\exists$ depending on whether i is even or odd. $\Sigma_i\mathsf{SAT}$ is a complete problem for Σ_i.

Claim 3.2. *For all $i \geq 1$, $\Sigma_{i+1} = \mathsf{NP}^{\Sigma_i}$.*

Proof. We exemplify this by proving $\Sigma_2 = \mathsf{NP}^{\Sigma_1}$. Without loss of generality, we consider the oracle access to the complete language in Σ_1, i.e. $\mathsf{NP}^{\Sigma_1\mathsf{SAT}}$.

We first prove the direction that $\Sigma_2 \subset \mathsf{NP}^{\Sigma_1\mathsf{SAT}}$. This is equivalent to proving $\Sigma_2\mathsf{SAT} \subset \mathsf{NP}^{\Sigma_1\mathsf{SAT}}$. Recall that a formula $\phi \in \Sigma_2\mathsf{SAT}$ if and only if $\exists x \forall y, \phi(x,y) = 1$. We want to show that the language $\Sigma_2\mathsf{SAT}$ can be determined by a polynomial-time NDTM M with oracle access to $\Sigma_1\mathsf{SAT}$, which is just SAT. This machine M works like follows: for any input ϕ, first non-deterministically guess a certificate x. For this fixed x, consider the negation of the original problem: $\exists y, \neg\phi(x,y) = 1$. This problem is in SAT and thus can be answered by the oracle. If the oracle answers "Yes", let the machine M reject on this path. If the oracle answers "No", let the machine M accept on this path. If the machine accepts on any of the non-deterministic paths, then $\phi \in \Sigma_2\mathsf{SAT}$. Therefore, machine M decides $\Sigma_2\mathsf{SAT}$ in polynomial time.

Next, we prove the direction that $\mathsf{NP}^{\Sigma_1\mathsf{SAT}} \subset \Sigma_2$. Consider a language $L \in \mathsf{NP}^{\Sigma_1\mathsf{SAT}}$. This means that there exists a polynomial-time verifier V such that $x \in L \iff \exists y\ V^{\mathsf{SAT}}(x,y) = 1$. The process of running V involves making some oracle queries $\phi_1, \ldots, \phi_m$, each accompanied with an answer $a_1, \ldots, a_m$. For each $i = 1, \ldots, m$, if $a_i = 1$, we know that $\exists u_i$ such that $\phi_i(u_i) = 1$. If

$a_i = 0$, we know that $\forall v_i$ it must be that $\phi_i(v_i) = 0$. Therefore, we can equivalently write that

$$\begin{aligned} x \in L \iff & \exists y, \phi_1, \ldots, \phi_m, a_1, \ldots, a_m, u_1, \ldots, u_m, \\ & \forall v_1, \ldots, v_m, \\ & V \text{accepts } (x, y) \text{ using the guessed queries and answers } \{\phi_i, a_i\} \text{ AND} \\ & \forall i = 1, \ldots, m, \quad a_i = 1 \implies \phi_i(u_i) = 1, a_i = 0 \implies \forall v_k, \phi_i(v_k) = 0. \end{aligned}$$

The right-hand-side of the equivalence defines an ATM with quantifiers $\exists, \forall$, which nondeterministically guesses a certificate y, a series of queries $\phi_1, \ldots, \phi_m$, correct answers $a_1, \ldots, a_m$, and assignments $u_1, \ldots, u_m$, and accepts x if on one of the non-deterministic paths, the guesses of all the above strings are sufficient for the machine and are consistent with each other. There can only be polynomially many queries, and all the guesses and checks for consistencies on $\phi(u_i)$ take polynomial time. Therefore, L can be determined in polynomial time by such machines, i.e., $L \in \Sigma_2$. □

4 Karp-Lipton Theorem

Theorem 4.1. *If* $\mathsf{NP} \subset \mathsf{P/poly}$*, then* $\mathsf{PH} = \Sigma_2$*, i.e., the polynomial hierarchy collapses at level 2.*

Proof. If we can show that $\Pi_2\mathsf{SAT} \in \Sigma_2$, we will have $\Pi_2 \subset \Sigma_2$. Then, for any $L \in \Sigma_2$, we know $\overline{L} \in \Pi_2 \subset \Sigma_2$. Therefore we also have $\overline{L} \in \Sigma_2$ and $L \in \Pi_2$. This shows that $\Sigma_2 \subset \Pi_2$ and thus $\Sigma_2 = \Pi_2$. By a claim we proved in last lecture, $\mathsf{PH} = \Sigma_2$.

Now we prove that $\Pi_2\mathsf{SAT} \in \Sigma_2$. For a string x, define a language L_x as $\phi \in L_x \iff \exists y \phi(x, y) = 1$. Clearly $L_x \in \mathsf{NP}$. If $\mathsf{NP} \subset \mathsf{P/poly}$, then there exists a polynomial-sized circuit family $\{C_l(x, \cdot)\}_{l \in \mathbb{N}}$ such that $C_l(x, \phi) = 1 \iff \phi \in L_x$. Yet, using the search to decision reduction (see Theorem 2.19 in Arora & Barak), we know that there exists another polynomial-sized circuit family $\{\tilde{C}_l(x, \cdot)\}_{l \in \mathbb{N}}$ such that if $\phi \in L_x \iff \phi(x, \tilde{C}_l(x, \phi)) = 1$. In other words, if there is an assignment y such that $\phi(x, y) = 1$, $\tilde{C}_l(x, \phi)$ outputs the assignment.

We build the new circuit family as follows. If the decision version of the problem is true, i.e., $C_l(x, \phi) = 1$, then there must be a string y such that $\phi(x, y) = 1$. We can guess each digit of y and use the circuit family $\{C_l(x, \cdot)\}_{l \in \mathbb{N}}$ to check the guess. For example, if $C_l(x \circ 0, \phi) = 1$ (where $x \circ 0$ means x concatenated with 0), then we know that for some possible assignments the first digit of y is 0. If instead $C_l(x \circ 0, \phi) = 0$, then we know that the first digit of y is 1. In either case, we can proceed to the next digit. Importantly, for each digit we used only one circuit from $\{C_l(x, \cdot)\}_{l \in \mathbb{N}}$ to check one time. Therefore, the new circuit family that outputs the string y will make use of $\{C_l(x, \cdot)\}_{l \in \mathbb{N}}$ and will be polynomial-sized.

Taking together, we have $\exists y \phi(x, y) = 1 \iff \tilde{C}_l(x, \phi) = y \iff \phi(x, \tilde{C}_l(x, \phi)) = 1$. In other words, $\phi \in \Pi_2\mathsf{SAT} \iff \forall x \exists y, \phi(x, y) = 1 \iff \exists \{\tilde{C}_l(x, \cdot)\}_{l \in \mathbb{N}} \forall x, \phi(x, \tilde{C}_l(x, \phi)) = 1$. This circuit family is polynomial-sized in ϕ and can be guessed non-deterministically like a certificate. This corresponds to a decision problem in Σ_2, and therefore, we proved that $\Pi_2\mathsf{SAT} \in \Sigma_2$. □