CS 710: Complexity Theory Date: Feb. 20th, 2024

Lecture 9: Oracles, Polynomial Hierarchy, BGS Theorem, and Circuits*

Instructor: Jin-Yi Cai Scribe: Hao Lin

1 Review

We have introduced oracles and polynomial hierarchy in the past two lectures.

The motivation is that although we already know NP $\subseteq$ PSPACE, we still have no idea if this is a proper containment, or say are there any other complexity classes in between which can separate NP from PSPACE?

- one of the most common concern in complexity theory is to ask whether we can separate class A from class B

Therefore, we want to discover (hopefully) more powerful complexity classes by adding more capabilities to the computation model, just as what we have seen the birth of the non-deterministic Turing machine with the magical parallel universe capability. And here comes the oracle and quantifier-based Turing machine.

1.1 Oracles, Relativization

Definition 1. Oracle is simply a set with strings, but when Turing machine M is equipped with an oracle, it can ask the oracle A with queries in the form of "is string $x \in A$, and the oracle will "magically" always reply the correct answer instantly. This procedure is called the relativization, and the new computation model is denoted as M^A.

Example: Let us take oracle A to be super powerful set in PSPACE-complete, let us say the quantifier boolean formula (QBF):

$$F = \exists X_1 \forall X_2 \exists X_3 \dots Q_k X_k [f(X_1, X_2, X_3, \dots, X_k)]$$

where $Q_k = \exists$ if k is odd, otherwise $Q_k = \forall$, $X_1, X_2, \dots, X_k \subseteq X$ is a partition of variables set X, f is the boolean formula. Deciding whether $F = 1$ is a PSPACE-complete problem.

Then, a P Turing machine equipped with it, can easily solve any problem in PSPACE since it simply ask this oracle A, whether the input is in it, and if the answer is "yes", it accepts, and rejects if it's "no".

*Andrew Yao's work on *Separating Polynomial Hierarchy from PSPACE using Oracle* is not covered here.

But since it can ask at most polynomial queries (since it is a P machine, the path cannot be super-polynomial), it is expected to have a chance to show more power than the corresponding PSPACE Turing machine. However, this strict relationship is yet not known. ⊠

1.2 Polynomial Hierarchy

Next, we add quantifiers to the vanilla Turing machine.

Definition 2.

$$\Sigma_k^p = \{x \mid \exists^p y_1, \forall^p y_2, \ldots, Q^p y_k D(x, y_1, y_2, \ldots, y_k) = 1\}$$

where $k \in \mathbb{N}$, $Q^k = \begin{cases} \exists^p & k \text{ is odd} \\ \forall^p & k \text{ is even} \end{cases}$, D is a polynomial computable predicate,

$\exists^p y = \exists y : |y| \leq p(|x|)$,

$\forall^p y = \forall y : |y| \leq p(|x|)$,

specifically, $\Sigma_0^p = \{x \mid D(x) = 1\}$.

Definition 3. The counterpart is defined for the complement languages in Σ_k^p,

$$\Pi_k^p = \{x \mid \forall^p y_1, \exists^p y_2, \ldots, Q^p y_k D(x, y_1, y_2, \ldots, y_k) = 1\}$$

with the same definition as Σ_k^p, except branches in Q_k are altered.

We would show that a decision problem is easily captured by the Π_2^p complexity class, to imply that we *may* define a harder complexity class than NP, however this is just a belief not a proof.

Example: We want to decide whether a boolean formula is the most succinct, which is called minimal equivalent expression (MEE). For a formula φ, being the most succinct means that $\forall\psi\exists\sigma, |\varphi(\sigma)| > |\psi(\sigma)|$. This decision problem is simply the definition of Π_2^p. ⊠

Proposition 1. It is easy to see that $\Sigma_0^p = \Pi_0^p = \mathrm{P}$, since we can take a polynomial deterministic Turing machine that accepts input x in polynomial time, and in return take those accepted input as the language Σ_0^p.

Proposition 2. It is less obvious but still directly from the definition that $\Sigma_1^p = \mathrm{NP}$, since we can take a NDTM that accepts x in polynomial time as long as there exists a path y with length bounded by $p(|x|)$, and in return take those accepted input as the language Σ_1^p.

Proposition 3. By using the "guess, delay and verify" technique (which will be shown in the next section), we are able to prove that $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$.

Here we give the quantifier-based definition for polynomial hierarchy.

Definition 4. Polynomial hierarchy (PH) is defined as the union of all Σ_k^p.

$$\mathrm{PH} = \bigcup_k \Sigma_k^p$$

Corollary 1.

$$\mathrm{PH} = \bigcup_k \Sigma_k^p = \bigcup_k \Pi_k^p$$

Proof. Since $\Sigma_k^p \subseteq \Pi_k^p$ and $\Pi_k^p \subseteq \Sigma_{k+1}^p$, it is clear to see this result. □

1.3 The Power of Complement Complexity Class as Oracle is the Same

Before we move on to more hierarchies, we need to take a closer look at the concept and properties of the complement complexity class.

Definition 5. The complement of language L is denoted as L^c means for any string ω, the following holds

$$(\omega \in L \Rightarrow \omega \notin L^c) \wedge (\omega \notin L \Rightarrow \omega \in L^c)$$

Proposition 4. Therefore, we also have $(L^c)^c = L$, and

$$(\omega \in L^c \Rightarrow \omega \notin L) \wedge (\omega \notin L^c \Rightarrow \omega \in L)$$

Proof. If L can be decided by a Turing machine M, then there exists a complement Turing machine M^c gives an opposite answer. Specifically, M^c can decide L^c in such a way that: for any input $x \in L$, if M accepts it, then M^c rejects it; if M rejects it, then M^c accepts it. □

Corollary 2. For complexity class NP, we have any language L,

$$L \in \mathrm{NP} \Longleftrightarrow L^c \in \mathrm{coNP}$$

where L^c is the complement of language L, and coNP is the set of all languages decided by any M^c.

We also know for any input x, there exists p a polynomial function and D a polynomial predicate, such that

$$x \in L \Leftrightarrow \exists^p y, D(x, y) = 1$$

$$x \in L^c \Leftrightarrow \forall^p y, (1 - D)(x, y) = 1$$

Proof. The left hand side is expanded as, a language L belongs to class NP means, for every string x, there exists a polynomial function p and predicate D, such that

$$x \in L \Leftrightarrow \exists^p y, D(x, y) = 1$$

The contrapositive to it is

$$x \notin L \Leftrightarrow \forall^p y, 1 - D(x, y) = 1$$

and hence

$$x \in L^c \Leftrightarrow \forall^p y, 1 - D(x, y) = 1$$

so we have (p, D) witness $L \in \text{NP}$ is equivalent $(p, 1 - D)$ witness $L^c \in \text{coNP}$. □

It is trivial that $\text{P}^c = \text{P}$, by simply negating the decision of the Turing machine. However, it is still not known whether $\text{coNP} = \text{NP}$, but it is easy to see that $\text{P} \subseteq \text{coNP} \cap \text{NP}$, while whether they are separated is also an open question.

Theorem 1. For any complexity class $\mathcal{C}$, its complement co-$\mathcal{C}$ has the same power as an oracle when the machine asking the query is at least polynomial computable.

Proof. If asking the oracle $\mathcal{C}$ with query string x, if there exists a language $L \in \mathcal{C}$ such that $x \in L$, we know from the above that $L^c \in$ co-$\mathcal{C}$. We can transform x into a string in L^c in polynomial time, therefore prove it. □

2 Equivalence of Quantifier-based and Oracle-based Definitions of Polynomial Hierarchy

We have seen that $\Sigma_2^p = \text{NP}^{\text{NP}}$, which connects the quantifier-based definition of polynomial hierarchy with the complexity classes relativized by oracles, so here comes a question to a more general proposition that can this hold for all k? The answer is yes.

Theorem 2. $\Sigma_k^p = \text{NP}^{\text{NP}^{\cdot^{\cdot^{\text{NP}}}}}$, where there are k NP on the right hand side.

The theorem states that the alternation of quantifiers is equivalent to asking oracles at the end of each path. We will prove this theorem with a series of claims. The basic idea is the "guess, delay and verify" simulation method.

Definition 6. Operator $\exists.$ is defined as, for any language L, and polynomial computable function p, we have

$$\exists.L = \{x \mid (\exists^p)y, \langle x, y\rangle \in L\}$$

where $\langle x, y\rangle$ is a pairing function which gives a one-to-one correspondence function $\mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, an example of pairing functions is Cantor pairing function with $\langle x, y\rangle = (x + y)(x + y + 1)/2 + x$.

Similar definitions for class $\mathcal{C}$ and operator $\forall$.

Claim 1. $\text{NP}^A = \exists.P^A$, for any oracle A.

Proof. $\exists.P^A \subseteq \mathrm{NP}^A$ since by definition of left hand side, for any relativized P machine NP^A, to accept any input x, there exists a decision path y of polynomial length (or say polynomial bounded), this is exactly the definition of NP^A.

On the other hand, for any NP^A machine, we will show that $\mathrm{NP}^A \subseteq \exists.P^A$. The idea is that asking queries at the end can be simply simulated by a P^A machine, so we just need to do an equivalent transformation to our NP^A machine such that all the queries are delayed to the end. We simply simulate the NP^A when there is no query. When there exists a query, simply guess its answer, and remember this query. At the end, we ask all the queries at once, and verify if all the guesses to the queries are correct in polynomial time simulation. This method is called "guess, delay and verify". □

Now a claim with two quantifiers.

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

Proof. Let us first show $\exists.\forall.P^A \subseteq \mathrm{NP}^{\exists.P^A}$. We can simply let our NP machine guess a path, and ask a negation of the question to that for the oracle $\forall.P^A$. This is because $L^c \in \forall.P^A \Leftrightarrow L \in \exists.P^A$ proved above (but here is the relativization version). Therefore, we know that the complement of complexity class as the oracle has the same power.

On the other hand, we want to show $\mathrm{NP}^{\exists.P^A} \subseteq \exists.\forall.P^A$. For an NP machine N, with oracle $B \in \exists.P^A$, we want to accept any input x that is accepted by N^B, by $\exists.\forall.P^A$. To do this, we can guess ($\exists.$) the path p in N, and the answers b_i to queries $\omega_i \in B$, and some positive answers y_i, such that ($\forall.$) for any negative answer z_j, we accept x if the following predicate holds in P^A: N accepts x for all queries with answers b_i in the first place; and then check all queries, if $b_i = 1$, then $D(x, y_i)$ holds; if $b_i = 0$, then $\neg D(x, z_i)$ holds. □

In summary, we now have two claims hold, for any oracle A,

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

Claim 3. For any oracle A, $\mathrm{NP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}} \subseteq \Sigma_k^{p,A}$, where there are k NP on the left hand side.

Proof. The first two claims prove the case for $k = 1, 2$. If $k \geq 3$, by applying claim 1, we have

$$\mathrm{NP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}} \subseteq \mathrm{NP}^{\exists.P^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}}}$$

where the number of NP on RHS decrease by 1.

By applying claim 2, we have

$$\mathrm{NP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}} \subseteq \forall.\exists.P^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}}$$

where the number of NP on RHS decrease by 2.

By apply these two claims alternatively, we prove it. □

Claim 4. For any oracle A, $\Sigma_k^{p,A} \subseteq \mathrm{NP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}}$, where there are k NP on the right hand side.

Proof. We prove by induction, let us assume this holds for $k-1 \geq 2$, we also have the complement version $\Sigma_{k-1}^{p,A} \subseteq \mathrm{coNP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}}$, where there are $k-2$ NP.

Since $\Sigma_k^{p,A} = \exists.\Pi_{k-1}^{p,A}$, we have $\Sigma_k^{p,A} \subseteq \exists.\mathrm{coNP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}} = \mathrm{NP}^{\mathrm{coNP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}}}$. As we have proved the simple version before, we know that the complement class as the oracle has the same power, therefore $\mathrm{NP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{NP}^A}}}}}$ with $k-1$ NP can capture $\Pi_{k-1}^{p,A}$, and hence prove it. □

These claims together prove the equivalence of polynomial hierarchy between quantifier-based and oracle-based definitions.

It implies that the super power we have created for oracles is nothing but an alternation of quantifiers built upon the vanilla Turing machine.

3 BGS Theorem: Why P vs. NP is Hard to Solve?

Here is the Baker-Gill-Solovay Theorem:

Theorem 3. There exists oracle A and B, such that

$$\mathrm{NP}^A = \mathrm{P}^A$$
$$\mathrm{NP}^B \neq \mathrm{P}^B$$

Remark 1. This means there cannot be relativizable proof for either P = NP or P ≠ NP, and hence the diagnolization method, which is the only powerful tool we have used so far (e.g. for proving time and space hierarchy theorems) cannot be used to solve P vs. NP problem.

Proof. It is easy to realize that we can just pick an oracle powerful enough to diminish the difference between P and NP machine. Let us just pick QBF as the oracle, so any question asked by NP machine can be simply answered by a P machine with one more ∃ quantifier in front of its QBF oracle, which is still a QBF oracle.

For the second statement, the ideas is still the diagonalization method. To separate these two classes, we need to find a sparse enough set B, such that it contains answers to queries that is not asked by P but asked by NP based on the intrinsic deficit of P being unable to ask super-polynomial many queries.

Let us consider a deterministic Turing machine M_i, we want to find a set B_i leading to a wrong answer for $M_i^{B_i}$ on some input, let us say 1^n but not for NP^{B_i}. Let us find a sufficiently large n_i such that $p(n_i) < 2^{n_i}$ by its asymptotically increment definition. Therefore, there will for sure have some questions not being asked by P but asked by NP, let us simply pick any of them, let us say ω_i, and diagonalize it:

If the answer for ω_i by $M_i^{B_i}(1^n)$ is 1, then we don't add ω_i in B_i, otherwise we add it.

For the rest of the queries, we simply don't add them to B_i, so B_i is super sparse, but enough to prevent M_i from being of the same power as any in NP^{B_i}.

Then, we simply generalize this to all p Turning machines using another diagonalization. Let us enumerate all p Turing machines M_i with their n_i in ascending order, and each one with a oracle set B_i as described above except that every simulation is based on B_{i-1}, and $B_i = B_{i-1}$ if M_i accepts; $B_i = B_{i-1} \cup \{\omega\}$ otherwise, so each Turing machine we only add at most one element incrementally to B_i, and then we get $B = \cup_i B_i$, which is sparse but enough to separate P^B from NP^B on input 1^n. □

4 Circuits

As we have seen from the BGS theorem, there is no way for us to distinguish P and NP if the proof can be relativized. Therefore, we need to seek other computation model.

We know that Turing machine cannot decide halting problems, as we can simply use diagonalization method to prove it. However, we have found that we can capture the halting problem using another model called circuit families, which may give a tool to solve the P vs. NP problem, bypassing the curse of relativization shown by the BGS theorem.

Definition 7. Boolean circuit C^n is a directed acyclic graph (DAG) with n input nodes as the boolean variables, and m boolean operations to connect some nodes with an output value assign to them, and the last node gives boolean value to this circuit. Here we usually constrain the boolean operations to be $\neg$, $\wedge$ and $\vee$.

The size of the boolean circuit is the number of nodes $|V|$ in this DAG. The depth of the circuit is the longest path from the input node to the output node.

Circuit families is the the set of such circuits with all input sizes, $C = \{C^n\}_{n=0}^{\infty}$.

Proposition 5. Boolean circuit families can capture the halting problem.

Proof. We can have a super sparse set containing only 1^n to indicate the halt condition where

n is asymptotically super large, say $n = 2^{2^k}$, so it is not computable, whereas the empty set for non-halting case, but we can have a simple circuit families with $C^n = (x_1 \wedge \overline{x_1}) \vee (x_1 \wedge x_2 \wedge \dots x_n)$. □

Let us now take a look at the parity function and to see that a problem can be captured by different circuit families.

Definition 8. Parity function is defined as follows:

$$\oplus(x_1, x_2, \dots, x_n) = \begin{cases} 1 & \text{there are odd number of 1} \\ 0 & \text{otherwise} \end{cases}$$

Proposition 6. This function can be captured by a depth 2 unbounded fan-in circuit families, with a CNF and DNF of an exponential size $O(2^n)$.

Proposition 7. By applying divide and conquer, we can capture this function with circuit families with fan-in of 2, depth of $O(\log n)$, and a linear size of $O(n)$.

Proof. The proof is simply using the master theorem. We have the recursive equation $T(n) = 2 \times T(n/2) + O(1)$, since we can divide the equations into two equal sizes and calculate them recursively, and then combine them with a single boolean operator. □

We would like to ask that can we find a constant depth, unbounded fan-in and polynomial size circuit families (called AC^0) for this parity function. The answer is no but the proof is beyond the scope here.

Why are we caring about the existence of such kind of circuit families? There is a connection between P vs. NP problem and the size, depth of the circuit. It is easy to see that any P problem can have a polynomial size circuit families, however, if we are able to prove the lower bound of the circuit families of a NP-complete problem is super-polynomial, then we are able to separate P and NP.

There is also an open question whether we can separate PH from PSPACE, where we have the similar intent using different size of circuit to distinguish them.

References

[1] Jin-Yi Cai, Draft Book on Complexity Theory. 2003 `http://pages.cs.wisc.edu/~jyc/710-draft-book.pdf`

[2] Terrance Tao, P=NP, relativisation, and multiple choice exams. 2009 `https://terrytao.wordpress.com/2009/08/01/pnp-relativisation-and-multiple-choice-exams/`

[3] MIT 6.841/18.405J: Advanced Complexity Theory. 2002 `https://people.seas.harvard.edu/~madhusudan/MIT/ST02/scribe/lect02.pdf`