CS 6810: Theory of Computing **Spring 2026**

Lecture 4: February 3, 2026

Lecturer: Eshan Chattopadhyay *Scribe: Amanda Wang*

1 Introduction

We saw previously how to use diagonalization to show the existence of uncomputable functions, as well as to prove hierarchy theorems for **TIME** and **NTIME**. Another interesting result involving diagonalization is Ladner's Theorem. In brief, Ladner's says that if we believe $\mathbf{P} \neq \mathbf{NP}$, there exist **NP**-*intermediate* problems between the classes of **P** and **NP**-complete problems.

Theorem 1.1 (Ladner's Theorem [Lad75])**.** *Suppose that* $\mathbf{P} \neq \mathbf{NP}$*. Then there exists a language* $L \in \mathbf{NP} \setminus \mathbf{P}$ *that is not* **NP**-*complete.*

The proof idea goes as follows. We construct a variant of SAT which we call SAT_H consisting of satisfiable boolean formulae padded with a string of 1's. For each formula ψ, the length of the padding will depend on $n = |\psi|$ and some polytime computable function $H(n)$. H is cleverly chosen so that we have $\Omega(\text{poly}(n))$ length padding. In particular, we pick H such that:

- If any TM $M = M_i$ claims to solve SAT_H efficiently, the padding length must be polynomial in n for any sufficiently large input ψ. Thus, SAT_H is just SAT with polynomial padding $\implies \mathsf{SAT}_H \notin \mathbf{P}$ unless $\mathbf{P} = \mathbf{NP}$.
- Our choice of H implies a superpolynomial amount of padding. Then any polytime reduction from length n instances of SAT to $\text{poly}(n)$-length instances of SAT_H implicitly gives a reduction from length n to length $o(n)$ SAT instances. But iteratively applying this reduction until we get a size $\mathcal{O}(1)$ instance would let us solve SAT in polytime, which contradicts $\mathbf{P} \neq \mathbf{NP}$. So no such polytime reduction exists, and SAT_H cannot be **NP**-complete.

See Arora & Barak (Section 3.4, [AB09]) for more details.

Where did we use diagonalization in this proof? From the first condition, we can intuit that M_i must appear somewhere in our definition of H, but it's clear that this construction is not quite as straightforward as the one we used to prove the **TIME**-hierarchy theorem. In general, we'll think of *diagonalization* as any technique which only uses the following key facts about TM's:

I. The existence of representations of TM's as strings that we can enumerate over, and

II. The existence of a Universal TM that can simulate other TM's with minimal overhead.

Given our earlier success with using diagonalization for structural results, one might ask the following question: **Can we settle P vs. NP using just diagonalization?** Throughout the rest of this lecture, we'll show that the answer to this question is **no**—using diagonalization!

Note: There are other barriers we can show; e.g., that natural proofs [RR97] and algebrization [AW09] are also provably insufficient. These could be good topics to explore for the final project!

2 Oracle TMs

We begin by defining *oracle Turing machines*, which will be key toward showing limitations of diagonalization.

Definition 2.1 (Oracle Turing machines)**.** *An oracle TM M^O is a multitape TM with oracle access to $O \subseteq \{0,1\}^*$, through a special oracle tape and additional states $\{q_{query}, q_{yes}, q_{no}\}$. Whenever M transitions to q_{query}, it queries O on input y, and then transitions in one time step to either q_{yes} if $y \in O$ or q_{no} if $y \notin O$.*

The mental model here is that an oracle TM captures computability in a world where the language O is cheaply computable. For instance, the complexity class $\mathbf{P}^O$ is the collection of all languages decidable by M^O, where M is a deterministic TM which runs in polynomial time. Similarly, we can define $\mathbf{NP}^O$ as the class of languages decidable by M^O for nondeterministic M.

Let's see some concrete examples. Say we pick any language $L \in \mathbf{P}$. The classes $\mathbf{P}^L$ and $\mathbf{NP}^L$ are exactly $\mathbf{P}$ and $\mathbf{NP}$, since oracle access to L doesn't buy us any additional computational power. To give another example, co-$\mathbf{NP} \subseteq \mathbf{P}^{\text{3-SAT}}$, since we can just run the oracle and negate its output. As a reminder, co-$\mathbf{NP}$ is just the complement of $\mathbf{NP}$, meaning that $L \in$ co-$\mathbf{NP}$ iff $\overline{L} \in \mathbf{NP}$. In upcoming lectures, we'll see how to use oracle TMs to define the polynomial hierarchy.

3 Limitations of Diagonalization

With these definitions in hand, we are now ready to state the main theorem.

Theorem 3.1 (Baker, Gill, Solovay [BGS75])**.** *There exists oracles A and B such that $\mathbf{P}^A = \mathbf{NP}^A$ and $\mathbf{P}^B \neq \mathbf{NP}^B$.*

One interpretation of this result is that A is a such a powerful oracle that if the class of deterministic polynomial-time machines is given access to A, then it can gain no additional power from nondeterminism. On the other hand, B is a much weaker oracle; even with access to B, deterministic polytime machines still can't do everything that nondeterministic polytime machines can do. This suffices to show a barrier for using diagonalization to settle $\mathbf{P}$ vs. $\mathbf{NP}$, since both facts **I** and **II** hold for any oracle TM. Therefore, any proof using only diagonalization must hold given both oracle access to A and oracle access to B. Such proofs are said to *relativize*, meaning the proof holds relative to any oracle. As such, this theorem actually yields a slightly stronger result, which is a barrier for any technique which relativizes (including diagonalization). In the next two sections, we'll see how to construct the oracles A and B.

3.1 $\mathbf{P}^A = \mathbf{NP}^A$

Define the class $\mathbf{EXP} = \cup_{c \geq 0} \text{DTIME}(2^{n^c})$ and the oracle

$$A = \{(\lfloor M \rfloor, x, 1^n) : M \text{ accepts } x \text{ within } 2^n \text{ steps.}\}$$

Observe that $A \in \mathbf{EXP}$ and $\mathbf{P}^A \subseteq \mathbf{NP}^A$. We'll show that $\mathbf{P}^A = \mathbf{NP}^A = \mathbf{EXP}$ by sandwiching $\mathbf{EXP} \subseteq \mathbf{P}^A$ and $\mathbf{NP}^A \subseteq \mathbf{EXP}$.

Claim 3.2. $\mathbf{EXP} \subseteq \mathbf{P}^A$.

Proof. Suppose $L \in \mathbf{EXP}$. Then there exists some TM M that runs in 2^{n^c} time and decides L. On input x, query $(\lfloor M \rfloor, x, 1^{|x|^c}) \in A$ in one step. Output $x \in L$ iff the oracle outputs yes. Thus, L is also in $\mathbf{P}^A$. □

Claim 3.3. $\mathbf{NP}^A \subseteq \mathbf{EXP}$.

Proof. First observe that $\mathbf{NP} \subseteq \mathbf{EXP}$, since we can traverse all computation paths in exponential time to check if any path accepts. Now, consider any nondeterministic polytime machine M^A with oracle access to A. Such a machine may query A at any point along any of its computation paths. But it can only make exponentially many queries in total over all computation paths, and each query can only be polynomial in size. Since $A \in \mathbf{EXP}$, we can simulate each query in exponential time. We can therefore handle exponentially many such queries in exponential time, so $\mathbf{NP}^A \subseteq \mathbf{EXP}$. □

Thus, we showed the chain $\mathbf{EXP} \subseteq \mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{EXP} \implies \mathbf{P}^A = \mathbf{NP}^A$.

3.2 $\mathbf{P}^B \neq \mathbf{NP}^B$

Given any language $L \subseteq \{0,1\}^*$, we can define its unary counterpart as

$$U_L = \{1^n : \exists y \in \{0,1\}^n \text{ s.t. } y \in L\}.$$

Observe that for any L, $U_L \in \mathbf{NP}^L$ since y is a polytime verifiable certificate for $1^n \in U_L$, given oracle access to L. The rest of this section is devoted to carefully constructing a language B such that the corresponding unary language $U_B \notin \mathbf{P}^B$. Combining the previous observation with the following construction suffices to show that $\mathbf{P}^B \subsetneq \mathbf{NP}^B$.

Let $M_1, M_2, \ldots, M_i, \ldots$ be an enumeration of deterministic TM's. Our construction of B takes place over countably many stages: in each stage i, we irrevocably decide membership in B for finitely many strings. The objective is to decide membership in such a manner that the oracle TM M_i^B must decide U_B incorrectly on some input 1^{n_i}. Therefore, since every deterministic TM is represented by some M_i in our enumeration, we conclude that no deterministic TM correctly computes U_B, and $U_B \notin \mathbf{P}^B$.

Consider some stage i. Let n_i be the smallest number such that every string y of length n_i is still undecided. Such a n_i must exist, since we have only decided membership for finitely many strings. We now simulate M_i^B on 1^{n_i} for $2^{n_i}/10$ steps. Note that this is more steps than we strictly need for showing $U_B \notin \mathbf{P}^B$, but it makes the analysis clean. Recall from the definition of U_B that M_i^B should accept 1^{n_i} iff some string $y \in \{0,1\}^{n_i}$ is in B. Roughly, what we'll show is that since none of the strings in $\{0,1\}^{n_i}$ were decided prior to this stage, we can decide membership for strings in $\{0,1\}^{n_i}$ so that M_i^B must be incorrect on 1^{n_i}.

Of course, there's the somewhat tricky matter of simulating oracle access to a language B which we are actively constructing. It turns out that this is actually not an issue, since we can answer these queries in a consistent manner that defers the question "$1^{n_i} \in U_B$?" until after the simulation is complete. In particular, whenever the machine queries the oracle B on an undecided string y, we always answer $y \notin B$. Otherwise, if y was decided in an earlier stage, we just answer consistently.

Suppose the simulation halts within $2^{n_i}/10$ steps and accepts. Recall that for any length n_i oracle query made during the simulation, we answered $y \notin B$. We can now ensure $1^{n_i} \notin U_B$ by deciding $y \notin B$ for all remaining strings of length n_i.

Suppose instead that the simulation halts within $2^{n_i}/10$ steps and rejects. Notice that we could have made at most $2^{n_i}/10$ oracle queries during that time, and no strings of length n_i were decided before this stage, so there must exist some fraction of length n_i strings which have not yet been decided. We can now decide $y \in B$ for these remaining strings, ensuring that $1^{n_i} \in U_B$.

Thus, whenever the simulation halts, we can decide membership for all unqueried strings in $\{0,1\}^{n_i}$ such that M_i^B must compute U_B incorrectly on input 1^{n_i}. If the simulation doesn't halt

within $2^{n_i}/10$ steps, we conclude that M_i^B fails to compute U_B in polynomial time. Finally, for all remaining undecided strings which are no longer than the longest string queried thus far, we can arbitrarily decide their membership in B before moving on to the next stage.

There is a subtle nuance with this proof as stated: we seem to be making *asymptotic* claims from only finite-length instances. More concretely, we can't assert that a machine M^B doesn't compute U_B in polytime just because it fails to halt within $2^n/10$ steps on some input of length n. Indeed, for any fixed n, we could always pick some constants c and d such that $c \cdot n^d > 2^n/10$. However, recall that each Turing machine has M infinitely many string representations. Thus, if every string representation M_i of M fails to halt on its input 1^{n_i} within $2^{n_i}/10$ steps, then we can safely conclude that M runs in superpolynomial time. Of course, if any string representation M_i of M does halt within $2^{n_i}/10$ steps, then our construction guarantees that M computes $U_B(1^{n_i})$ incorrectly, so the proof holds.

References

[BGS75] Theodore Baker, John Gill, and Robert Solovay. "Relativizations of the P =? NP question." In: *SIAM Journal on Computing* 4.4 (Dec. 1975), pp. 431–442. DOI: 10.1137/0204037.

[Lad75] Richard E. Ladner. "On the Structure of Polynomial Time Reducibility". In: *J. ACM* 22.1 (Jan. 1975), pp. 155–171. ISSN: 0004-5411. DOI: 10.1145/321864.321877. URL: https://doi.org/10.1145/321864.321877.

[RR97] Alexander A. Razborov and Steven Rudich. "Natural proofs." In: *Journal of Computer and System Sciences* 55.1 (Aug. 1997), pp. 24–35.

[AW09] Scott Aaronson and Avi Wigderson. "Algebrization: A New Barrier in Complexity Theory". In: *ACM Trans. Comput. Theory* 1.1 (Feb. 2009). ISSN: 1942-3454. DOI: 10.1145/1490270.1490272. URL: https://doi.org/10.1145/1490270.1490272.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. 1st. USA: Cambridge University Press, 2009. ISBN: 0521424267.