CAS CS 535: Complexity Theory

Lecturer: Mark Bun — Fall 2023

Lecture Notes 6:

Relativization, Space Complexity

Reading.

- Arora-Barak § 3.4, 4.1

Last time: Hierarchy theorems, padding, intro to oracles

1 Limits of Diagonalization

We've seen some very clever diagonalization arguments so far, e.g., for the nondeterministic time hierarchy theorem and Ladner's theorem. Could diagonalization be used to prove a landmark result like $P \neq \mathbf{NP}$?

There's a formal sense in which the answer turns out to be "no," but to state this, we have to define exactly what we mean by "diagonalization." We'll do so indirectly by examining what properties of the TM model we actually use to carry out these arguments, which turns out to not be very much. Specifically, we need the facts that

- TMs can be encoded by strings and enumerated
- There is a universal TM that can efficiently simulate any other TM.

Thus, our proofs all work just as well for any enhanced model with these properties, such as the model of oracle TMs.

Definition 1 (Oracle TM)**.** Let $A \subseteq \{0,1\}^*$ be a language. An TM with oracle access to A, denoted M, has:

1. An extra read/write "oracle tape"
2. The ability to, in 1 computational time step, obtain the answer to the question "Is $q \in A$?" where q is the string written on the oracle tape.

All of the complexity classes we've seen can be defined in the presence of oracles. For instance,

$$\mathbf{P}^A = \{ \text{ languages decided by poly-time deterministic TMs with oracle access to } A\}$$

$$\mathbf{NP}^A = \{ \text{ languages decided by poly-time nondeterministic TMs with oracle access to } A\}$$

So what do oracles do to these classes? Of course, it depends on the oracle.

Example 2. If $A \in \mathbf{P}$, then $\mathbf{P}^A = \mathbf{P}$. To see that $\mathbf{P} \subseteq \mathbf{P}^A$, note that an oracle TM M^A can always ignore its oracle. For the other direction, $\mathbf{P}^A \subseteq \mathbf{P}$, we can simulate each call to the oracle by deciding A in poly-time.

Example 3. $\mathbf{NP} \cup \mathbf{coNP} \in \mathbf{P}^{\mathsf{SAT}}$. (The latter is the class of languages that are Cook-reducible to SAT.)

A proof of a complexity class separation $\mathbf{C}_1 \neq \mathbf{C}_2$ (resp. collapse $\mathbf{C_1} = \mathbf{C_2}$) relativizes if it continues to hold in the presence of any oracle, i.e., for all oracles A, we have $\mathbf{C}_1^A \neq \mathbf{C}_2^A$ (resp. collapse $\mathbf{C}_1^A = \mathbf{C}_2^A$). Any conjecture that is inherently non-relativizing (i.e., because it's true in the presence of some oracle and false in the presence of another) requires non-relativizing techniques to resolve.

It turns out that the $\mathbf{P}$ vs. $\mathbf{NP}$ problem can't have a relativizing solution.

Theorem 4. *There exist oracles A, B such that*

$$\mathbf{P}^A = \mathbf{NP}^A, \qquad \mathbf{P}^B \neq \mathbf{NP}^B.$$

Proof. First, we construct the oracle A for which $\mathbf{P}^A = \mathbf{NP}^A$:

$$A = \{\langle M, x, 1^n \rangle \mid M \text{ accepts } x \text{ within } 2^n \text{ steps}\}.$$

One can show that A is $\mathbf{EXP}$-complete under poly-time reductions, so in particular, $\mathbf{EXP} \subseteq \mathbf{P}^A$.

Meanwhile, we can also see that $\mathbf{NP}^A \subseteq \mathbf{EXP}$. This is because in exponential time, we can brute-force over all the $\mathbf{NP}$ machine's nondeterministic choices, and simulate all of the queries to A in exponential time.

Now let's construct the oracle B for which $\mathbf{P}^B \neq \mathbf{NP}^B$. The first thing we'll actually do is describe the form of the *language* that is in $\mathbf{NP}^B$, but not in $\mathbf{P}^B$:

$$U_B = \{1^n \mid \exists x \in B, |x| = n\}.$$

Claim 1: For every oracle B, we have $U_B \in \mathbf{NP}^B$.

Proof of Claim 1: To decide if $1^n \in U_B$, nondeterministically guess a string $x \in \{0,1\}^n$ and use the oracle to check if $x \in B$.

Claim 2: There exists an oracle B such that $U_B \notin \mathbf{P}^B$.

Proof of Claim 2: This is the most interesting part of the proof, and perhaps ironically, constructs the oracle B by diagonalizing against all poly-time oracle TMs. Here's the gameplan. Let $M_1, M_2, \ldots$ be an enumeration of all TMs. We're going to construct the oracle B in stages:

$$\emptyset = B_0 \subseteq B_1 \subseteq B_2 \subseteq \ldots$$

such that for every stage i and every $j > i$, the TM $M_i^{B_j}$ does not decide U_{B_j} within $2^n/10$ steps. Our final oracle will be

$$B = \bigcup_{i=0}^{\infty} B_i.$$

Our stagewise construction will produce each B_i from B_{i-1}. Let n_i be large enough so that $|q| < n_i$ for every oracle query q made by every previous machine $M_j^{B_j}(1^{n_j})$ for $j < i$. Our goal is to construct B_i to force $M_i^{B_i}$ to mess up on input 1^{n_i}.

Run $M_i^{B_i}$ on input 1^{n_i} for $2^{n_i}/10$ steps, answering its oracle queries as follows:

- If q was queried in a previous round, answer consistently with the previous oracle B_{i-1}.
- If q is new, answer no, i.e., $q \notin B_i$.

After $2^{n_i}/10$ steps, there are two possibilities:

- If $M_i^{B_i}(1^{n_i})$ accepts, set $q \notin B_i$ for every $q \in \{0,1\}^{n_i}$. By the definition of B_i, this makes $1^{n_i} \notin U_{B_i}$, so $M_i^{B_i}$ fails to decide U_{B_i}.
- If $M_i^{B_i}(1^{n_i})$ does not accept, then let $q \in \{0,1\}^{n_i}$ be an arbitrary string not queried so far. Such a string exists because we stopped the simulation after only $2^{n_i}/10$ steps, so it could have only made this many queries. Set $q \in B_i \implies 1^{n_i} \in U_{B_i}$, which again ensures that $M_i^{B_i}$ does not decide U_{B_i}.

Finally, note that B_j is consistent with B_i on all of M_i's oracle queries for all $j > i$, so we conclude the stronger requirement that M_i^B does not decide U_B. $\square$

Remark 5. The structure of the construction of B doesn't actually have much to do with the fact that we're separating $\mathbf{P}$ from $\mathbf{NP}$. Really, it just uses the facts that 1) one can decide membership in U_B by making one nondeterministic query to B and 2) making fewer than $2^n/10$ deterministically-chosen queries to B does not suffice. An equivalent interpretation of the construction and analysis of U_B is that computing the OR function on 2^n inputs can be done with one nondeterministic query, but requires decision-tree depth $> 2^n/10$. If you want construct oracles that separate other complexity classes, a good way to do it is to identify the appropriate analogs of decision trees (equivalently, query algorithms) and prove lower bounds against those.

Barriers, more generally. If we're going to prove $\mathbf{P} \neq \mathbf{NP}$, we're going to need techniques that fail to relativize. By now, a few such techniques are available. One is circuit lower bounds, which compile TMs down to Boolean circuits and use more detailed combinatorial information about them. These, unfortunately, are often subject to the Razborov-Rudich "natural proofs" barrier. Another is algebraic techniques, which later in the course we'll see can be used to prove things like the $\mathbf{IP} = \mathbf{PSPACE}$ theorem. These too, however, are subject to the Aaronson-Wigderson "algebrization" barrier. So even if we're still far from proving $\mathbf{P} \neq \mathbf{NP}$, we have some good excuses for it!

2 Space Complexity

Space complexity aims to understand how much memory is required to perform a computation. Recall that we measure memory space usage as follows: A multi-tape TM M runs in space $S(n)$ if it accesses at most $S(|x|)$ cells of its $\underline{\text{work tapes}}$ for all $x \in \{0,1\}^*$.

Here are some examples of why this convention is useful.

1. Say we want to compute the function $f(x) = |x|$. We can do this by maintaining a counter on a work tape using $O(\log |x|)$ bits, and then copy the answer to the output tape. This uses space only $O(\log n)$.
2. To compute the function $f(x) = x$, we can copy the input bit-by-bit to the output tape, using no work space at all. This idea will be useful when we think about space-bounded reductions.

2.1 Space Complexity Classes

$$\mathbf{SPACE}(S(n)) = \{L \subseteq \{0,1\}^* \mid L \text{ is decidable by a deterministic TM in space } O(S(n))\}.$$
$$\mathbf{NSPACE}(S(n)) = \{L \subseteq \{0,1\}^* \mid L \text{ is decidable by an NTM in space } O(S(n))\}.$$

2.2 Time vs. Space

In general, space seems to be a more powerful resource than time. For example, computing $f(x) = |x|$ can be done using space $\log n$, but requires time n.

In general, $\mathbf{DTIME}(S(n)) \subseteq \mathbf{SPACE}(S(n))$, but space seems strictly more valuable that time since space can be "reused." In fact, it is known that

$$\mathbf{DTIME}(S(n)) \subseteq \mathbf{SPACE}(S(n)/\log S(n)) \subsetneq \mathbf{SPACE}(S(n)).$$

The first containment is a result of Hopcroft, Paul, and Valiant, and the second is just the space hierarchy theorem.

In somewhat broader strokes, the best known general relationships between time and space are as follows:

Theorem 6. *For space-constructible $S(n)$, we have*

$$\begin{aligned}\mathbf{DTIME}(S(n)) \subseteq &\mathbf{NTIME}(S(n)) \subseteq \mathbf{SPACE}(S(n)) \\ &\subseteq \mathbf{NSPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))}) =: \bigcup_{c=1}^{\infty} \mathbf{DTIME}(2^{cS(n)}).\end{aligned}$$

Proof. We'll prove the interesting containments:

$\mathbf{NTIME}(S(n)) \subseteq \mathbf{SPACE}(S(n))$. Let N be an NTM running in time $S(n)$. We'll simulate N using space $O(S(n))$ (deterministically) as follows:

On input x:

For every sequence of N's nondeterministic choices:

- Simulate N on x using that sequence
- Accept if the simulation accepts.

Reject.

The space usage is $O(S(n))$ for the simulation plus $O(S(n))$ to maintain a "counter" over the nondeterministic choices to simulate.

$\mathbf{NSPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))})$. Proving this result is a good excuse to make the following very useful definition. Given a TM M and input x, define the configuration graph $G_{M,x}$ as follows. The graph has a vertex for every triple of the form

$$C = (\text{state, head locations, work tape contents}),$$

and an edge $C \to C'$ if there exists a transition that takes C to C'. For example, if [pretending we're working with a one-tape TM for simplicity] the transition functions of an NTM specify $\delta_0(q_7, 0) = (q_4, 0, L)$ and $\delta_1(q_7, 0) = (q_3, 1, R)$, then vertex $C = (q_7, 3, 10010)$ has outgoing edges to $C'_0 = (q_4, 2, 10010)$ and $C'_1 = (q_3, 4, 10110)$.

What makes the configuration graph so useful is the following characterization:

$$M(x) \text{ accepts} \iff \text{there exists a path from } C_{\text{start}} \text{ to an accepting configuration.}$$

Without loss of generality, we can assume that M erases its worktapes and restores its heads to the left after halting, so there are unique accept and reject vertices in the configuration graph.

Now we're ready to simulate nondeterministic space-bounded computation with exponentially more time. Let N be an NTM running in space $S(n)$. Consider the following deterministic TM:

On input x:

1. Construct the configuration graph $G_{N,x}$
2. Run BFS to determine if there is a path from C_{start} to C_{accept} in $G_{N,x}$. Accept iff this is the case.

We can reason about the time needed to construct $G_{N,x}$ as follows. First, the size of each configuration is $O(S(|x|))$. This means that the number of possible configurations, i.e., the size of the graph, is $2^{O(S(n))}$. Therefore, one can materialize the whole graph and run BFS on it in time $2^{O(S(n))}$. $\square$