CS 6810: Theory of Computing **Fall 2021**

Lecture 5: Sep 9, 2021

Lecturer: Eshan Chattopadhyay *Scribe: Conan Gillis*

1 Oracle Turing Machines and Relativisation (Or, the Limits of Diagonalization)

Definition An **oracle** is a language $O \subseteq \{0,1\}^*$, and a **query** is a string $x \in \{0,1\}^*$.

Definition Given an oracle O, an **Oracle Turing Machine** M^0 is a multitape Turing Machine with the following:

1. An oracle tape.
2. Three additional states, $q_{\text{query}}, q_{\text{no}}, q_{\text{yes}}$.

The machine is able to write a string (say, x) on the oracle tape, then transition into q_{query}. If $x \in O$, the very next step the machine will transition to q_{yes}. If $x \notin O$, the machine will transition into q_{no}. The machine is allowed to rewrite the tape, then transition to q_{query} again and repeat the process, thus making multiple queries in a single execution. In any time/space analysis, we do not charge for the space needed to write on the tape, and the transition from q_{query} to q_{no} or q_{yes} only takes one computational step. In other words, all activities of the "oracle," including storing and reading the input, occur for free, at least with respect to the TM's time/space complexity. We do, however, charge for the time to actually write to the oracle tape.

Note: M itself need not be deterministic.

Definition Let O be an oracle. We define

$$P^O = \{L \subseteq \{0,1\}^* : \exists \text{ TM } M \text{ such that } M^O \text{ runs in polynomial time to compute } L\}$$

and

$$NP^O = \{L \subseteq \{0,1\}^* : \exists \text{ NDTM } M \text{ such that } M^O \text{ runs in polynomial time to compute } L\}.$$

Observation If an oracle $O \in P$, then $P^O = P$.

Proof: If $L \in P$, there exists an oracle TM M^O that calculates L in polynomial time. But, since $O \in P$, a Turing Machine can simply calculate $x \in O$ in polynomial time, without reference to an oracle. The oracle Turing Machine M^O will make only polynomially many oracle queries (since each query still costs one unit of computation time, and M^O needs to compute L in polynomial time), so replacing these queries with a polynomial-time computation will still give rise to a polynomial time for the overall execution, so $L \in P$. Thus, $P^O \subseteq P$. The reverse inclusion is clear, since we can create an oracle machine for a language in P that simply never checks its oracle, and operates identically to the deterministic machine that already computes L in polynomial time (this exists by the definition of P). Therefore, $P^O = P$.

Question If $O = \overline{\text{SAT}} = \{\phi : \phi \text{ is not satisfiable}\}$, then is $\text{SAT} \in P^{\overline{\text{SAT}}}$?

Answer Yes. We can make an oracle machine $M^{\overline{\text{SAT}}}$ that takes in a string, writes in on its oracle tape, and runs a query, returning the opposite of whatever the query returns. If the query returns no (i.e. q_{query} transitions to q_{no}), the input is not in $\overline{\text{SAT}}$, so it must be in SAT. If the query returns yes, the input is in $\overline{\text{SAT}}$, and hence not in SAT. Thus $M^{\overline{\text{SAT}}}$ will correctly compute SAT, and the only time it needs is the time to write on the oracle tape, which is linear, and we are done.

[Baker, Gill, Solovay] There exist oracles A, B such that $P^A = NP^A$ but $P^B \neq NP^B$.

Moral: A proof "relativizes" if a) you (the prover) enumerate over Turing Machines, and b) use a Universal Turing Machine to simulate other Turing Machines.

Observation Any diagonalization proof relativizes.

Example Given a time function t, suppose we want to show

$$\text{DTIME}\ ^{O}(t(n)) \subsetneq \text{DTIME}\ ^{O}(t(n)^2)$$

for any oracle O. We would simply go through the proof of

$$\text{DTIME}\ (t(n)) \subsetneq \text{DTIME}\ (t(n)^2)$$

without the oracles, writing $*^O$ wherever needed.

More formally, if a proof that relativizes shows for complexity classes C_1, C_2 that $C_1 \neq C_2$, in fact it implies $C_1^O \neq C_2^O$ for any oracle O. This, combined with the above theorem, shows that diagonalization alone will not be enough to solve P vs. NP.

We now begin the proof of the Theorem.

Proof. Let $A = \{\langle\ \lfloor M \rfloor, x, 1^n\rangle\ :\ M \text{ accepts } x \text{ in at most } 2^n \text{ steps }\}$, and recall that

$$\text{EXP}\ = \bigcup_{c \geq 1} \text{DTIME}\ (2^{o(n^c)}).$$

We claim that $P^A = NP^A = \text{EXP}$. First, observe that the inclusion $P^A \subseteq NP^A$ follows immediately from the definitions of P^A and NP^A. We next need to show $NP^A \subseteq \text{EXP}$ and $\text{EXP}\ \subseteq P^A$. By the transitivity of the $\subseteq$ relation, this will prove our claim.

We show the former inclusion first. Let $L \in NP^A$. Then there exists an NDTM N such that N^A computes L in polynonial time. At each step of the execution of N^A, there are two possible transitions, and some of these steps are oracle queries. Thus, there are $2^{poly(n)}$ possible sequences of executions the machine can make. (The $poly(n)$ exponent comes from the fact that N^A terminates in $poly(n)$ steps.) In each sequence, every step might be an oracle call, so there are $poly(n)$ oracle calls made for each sequence. By the definition of A, for any string s we can decide if $s \in A$ in $\leq 2^{|s|}$ time. For, if $s = \langle\ \lfloor M \rfloor, x, 1^n\rangle$, we simply simulate $M(x)$ for 2^n steps and check if M has accepted it or not, and we can reject s in polynomial time if it is not of this form.

Thus, we may construct a Turing Machine to brute-force simulate each transition sequence of N^A, and simulate the oracle calls deterministically. With $2^{poly(n)}$ sequences, with $poly(n)$ possible oracle calls taking $2^{poly(n)}$ time each, this will take a total of $O(2^{poly(n)}(poly(n) \cdot 2^{poly(n)})$ time. This will be exponential, hence $L \in \text{EXP}$.

All that remains is to show EXP $\subseteq P^A$. If $L \in \text{EXP}$, then by definition there exists a Turing Machine M_L that computes L in $2^{O(n^c)}$ time. To show $L \in P^A$, we construct a machine N^A as follows. We hardcode c and the description $\lfloor M_L \rfloor$ of M_L in N^A, so that it can be reproduced automatically by the machine whenever needed. Given an input x, N^A writes $\langle \lfloor M_L \rfloor, x, 1^{|x|^c} \rangle$ on its oracle tape, and transitions to q_{query}. Note that this takes $poly(|x|)$ time, since the description of M_L is hardcoded and thus only costs constant time to write. If, on the next step, the machine transitions to q_{no}, then M_L does not accept x in exponential time. Since M_L by assumption accepts *every* element of L in exponential time, this means $x \notin L$, so we make N^A also reject x. If the machine transitions to q_{yes}, then M_L does except L, hence $x \in L$, so we make N^A accept x. Thus, we see that N^A computes L in polynomial time, hence $L \in P^A$. This completes the proof that $P^A = NP^A$.

To complete our proof of this theorem, we must produce an oracle B such that $P^B \neq NP^B$. For any language $L \subseteq \{0,1\}^*$, let the "unary language" $U_L = \{1^n : \{0,1\}^n \cap B \neq \emptyset\}$. We claim that $U_L \in NP^L$ for any L. For, given an input 1^n an NDTM with access to L as an oracle can "guess" (using non-determinism) some $x \in \{0,1\}^n$ and check that $x \in L$ using the oracle. With this fact in hand, we can construct B in an iterative process.

Stage 0 Set $B = \emptyset$

Stage i Let us be given some enumeration $M_1^B, M_2^B, \ldots$ of Turing Machines with access to B as an oracle (or, at least, that part of B which has already been constructed). Up to this point, $y \in B$ or $y \notin B$ has been decided for only finitely many $y \in \{0,1\}^*$. So, we may let n be the smallest integer such that no element of $\{0,1\}^n$ has been decided. Recalling that we are at **stage** i, run the machine M_i^B on 1^n for at most $2^n/10$ steps.

If M_i^B makes an oracle query (i.e. asks $y \in B$? for some y), then we can decide now whether y is in B as follows: if y's fate has already been decided at some previous stage then we agree with the previous decision, otherwise we declare answer $y \notin B$. If M_i^B outputs 1 on 1^n, then B contains a string of length n, so declare all remaining strings $z \in \{0,1\}^n$ to not be in B. Otherwise, declare some arbitrary $z \in \{0,1\}^n$ to be in B, and all others to not be (z exists because M_i^B has taken $2^n/10$ steps, so at most $2^n/10 < 2^n = |\{0,1\}^n|$ strings have been considered). Continue on to the next step.

□