PRINCETON UNIVERSITY COS 522: COMPUTATIONAL COMPLEXITY

Lecture 3: Diagonalization

Lecturer: *Sanjeev Arora* Scribe:*scribename*

To separate two complexity classes we need to exhibit a machine in one class that is different (namely, gives a different answer on some input) from *every* machine in the other class. This lecture describes *diagonalization*, essentially the only general technique known for constructing such a machine. We also indicate why this technique has been unable thus far to resolve $\mathbf{P} =? \mathbf{NP}$ and other interesting questions.

1 Time Hierarchy Theorem

The Time Hierarchy Theorem shows that allowing Turing Machines more computation time strictly increases the class of languages that they can decide.

THEOREM 1
If f, g are running times satisfying $f(n) \log f(n) = o(g(n))$, then

$$\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n)) \tag{1}$$

To showcase the essential idea of the proof of Theorem 1, we prove the simpler statement $\mathbf{DTIME}(n) \subsetneq \mathbf{DTIME}(n^2)$. We use diagonalization. Suppose $M_1, M_2, M_3, \ldots$ is a numbering of all Turing Machines, where the description of M_i can be produced from i in time $O(\log i)$. (Such numberings exist. For example, one can order TMs according to the number of states in their transition diagrams, and use lexicographic ordering among then all TMs that have the same number of states. Note that we allow machines that do not halt on all inputs.)

Consider the following Turing Machine, D: "*On input x, if $x = 0^j 1^k$ for some j, k then construct M_k and simulate it on x for $|x|^{1.5}$ steps. If M_k halts and accepts, reject. If M_k halts and rejects, accept. In every other situation (for example if M_k does not halt), accept.*"

This machine runs in time at most $2n^2$. Specifically, it has to maintain a timer that keeps tracks of the number of steps in the simulation of M_k. Maintaining this counter introduces an overhead so the running time of the modified machine will be $O(n^{1.5} \log n) = o(n^2)$.

Now we show that language accepted by D is not in $\mathbf{DTIME}(n)$. Suppose M_k is a machine that runs in linear time, specifically, in time at most $cn + d$. Then for every integer j satisfying $j + k > \max\{(c+1)^2, d\}$, the input $y = 0^j 1^k$ is such that D's computation on this input involves simulating M_k on the same input and flipping the answer. Since $|y|^{1.5} > c|y| + d$, the diagonalizer has enough time to complete its simulation of M_k and determine its answer.

The proof of Theorem 1 is similar, and involves the observation that the diagonalizing Turing machine D wants to simulate a machine that runs in time $f(n)$, and also maintain a counter to keep track of the running time. The overhead of maintaining the counter increases the running time of D by a factor $O(\log f(n))$.

2 Nondeterministic Time Hierarchy Theorem

The analogous hierarchy theorem for nondeterministic computation is even tighter than for deterministic computation: it drops the extra log term.

TBE later: why no extra log term?

THEOREM 2
If f, g are running times satisfying $f(n) = o(g(n))$, then

$$\mathbf{NTIME}(f(n)) \subsetneq \mathbf{NTIME}(g(n)) \tag{2}$$

Again, we just showcase the main idea of the proof by proving $\mathbf{NTIME}(n) \subsetneq \mathbf{NTIME}(n^2)$. The technique from the previous section does not directly apply. A nondeterminisitic machine that runs in $O(n)$ time may have $2^{O(n)}$ branches in its computation. It is unclear how to determine in $O(n^2)$ time whether or not it accepts and then flip this answer. Instead we use a technique called *lazy* diagonalization, which is only guaranteed to flip the answer on some input in a fairly large range.

For a pair of integers i, j, define $f(i, j)$ to be

$$f(i, j) = 2^{2^{2^{i}3^{j}}}. \tag{3}$$

Clearly, f is one-to-one. We say $(i, j) \precsim (k, l)$ if $f(i, j) < f(k, l)$. Then $\precsim$ is a linear ordering. Thus for any integer n, there is a largest pair i, j and smallest pair k, l such that $f(i, j) < n \leq f(k, l)$. We use the shorthands $L_n = f(i, j), H_n = f(k, l)$. Note that L_n, H_n are easily computed given n, certainly in $o(n^2)$ time. Furthermore, $H_n > 2^{(L_n)^2}$, so the interval $[L_n, H_n)$ is quite large. The diagonalizing machine tries to flip the answer in such large intervals.

Let $M_1, M_2, M_3, \ldots$ be an enumeration of all NDTMs. We construct the following NDTM D_1.

"*On input x, if $x \notin 1^*$, accept. If $x = 1^n$, then compute L_n, H_n. If $n = 1^{H_n}$, accept 1^n iff M_i rejects 1^{L_n+1} in $|L_n|^{1.5}$ time. (Note that this requires going through all possible $\exp(|L_n|^{1.5})$ nondeterministic branches of M_i on input 1^{L_n+1}.) Otherwise simulate M_i on input 1^{n+1} using nondeterminism in $n^{1.5}$ time and output its answer.* "

Clearly, D_1 runs in $O(n^2)$ time. We show that D_1 accepts a different language from any NDTM M_i that runs in $O(n)$ time. For contradiction's sake assume M_i runs in $cn + d$ time and accepts the same language as D_1. From the construction of M_i, we know that for all all input sizes n such that $f(i, j) = L_n$ for some j and $L_n < n \leq H_n$, D_1 accepts 1^n (and hence so does M_i) iff M_i accepts 1^{n+1}. Now if M_i and D_1 accept the same language, we conclude that M_i

accepts 1^n iff M_i accepts 1^{n+1}. This chain of implications leads to the conclusion that M_i accepts 1^{L_n+1} iff it accepts 1^{H_n}. But this leads to a a contradiction, since when j is large enough (specifically, larger than $c+d$), machine D_1 accepts 1^{H_n} iff M_i rejects 1^{L_n+1}. Hence M_i and D_1 cannot accept the same language.

3 Can diagonalization resolve P vs NP?

Quantifying the limits of "diagonalization" is not easy. Certainly, the diagonalization in Section 2 seems more clever than the one in Section 1 or the one that proves the undecidability of the halting problem.

For concreteness, let us say that "diagonalization" is any technique that relies upon the ability of one TM to simulate another. In order to identify the limitations of such techniques, we observe that they treat TMs as blackboxes: the machine's internal workings do not matter. In particular, the simulations also work if all Turing Machines are provided with the same oracle. (Whenever the TM being simulated queries the oracle, so does the simulating TM.) If we could resolve P vs NP —in whichever direction— using such techniques then the proof would also work in the presence of any oracle. However, we now exhibit oracles B, C such that $P^C = NP^C$ and $P^B \neq NP^B$, which implies that such a proof cannot exist.

For C we may take any **PSPACE**-complete problem, say TQBF, since $P^{\mathrm{TQBF}} = NP^{\mathrm{TQBF}} = \mathbf{PSPACE}$. Now we construct B. For any language A, let A_u be the unary language

$$A_u = \{1^n : \text{some string of length } n \text{ is in } A\}.$$

For every oracle A, the language A_u is clearly in NP^A. Below we construct an oracle B such that $B_u \notin P^B$. Hence $B_u \in NP^B \setminus P^B$, and we conclude $P^B \neq NP^B$.

Construction of B: Let $M_1, M_2, M_3, \ldots$ be all polynomial-time Oracle Turing Machines. (This enumeration need not be effective, since we are merely showing the *existence* of the desired oracle.) We construct B in stages, where stage i ensures that M_i^B does not decide B_u. We construct B by initially letting it be empty, and gradually deciding which strings are in it or outside it. Each stage determines the status of a finite number of strings.

Stage i: So far, we have declared for a finite number of strings whether or not they are in B. Choose n large enough so that it exceeds the length of any such string, and 2^n exceeds the running time of M_i on inputs of length n. Now run M_i on input 1^n. Whenever it queries the oracle about strings whose status has been determined, we answer consistently. When it queries strings whose status is undetermined, we declare that the string is not in B. We continue until M_i halts. Now we make sure its answer on 1^n is incorrect. If M_i accepts, we declare that all strings of length n are not in B, thus ensuring $1^n \notin B_u$. If M_i rejects, we pick a string of length n that it has not queried (such a string

exists because 2^n exceeds the running time of M_i) and declare that it is in B, thus ensuring $1^n \in B_u$. In either case, the answer of M_i is incorrect.

Thus our construction of B ensures that no M_i decides B_u.

Let us now answer our original question: Can diagonalization or any simulation method resolve **P** vs **NP**? Answer: Possibly, but it has to use some fact about TMs that does not hold in presence of oracles. Such facts are termed *nonrelativizing* and we will later encounter examples of such facts. Whether or not they have any bearing upon **P** vs **NP** (or other interesting complexity theoretic questions) is open.

Oracle Turing Machines

We give a quick introduction to oracle Turing machines, which were used above. If B is a language, then a machine M *with access to oracle* B, denoted M^B, is a machine with a special *query tape*. It can write down any string on this tape, and learn from the oracle in a single step whether or not the string is in the language B. We denote by $\mathbf{P}^B$ is the class of languages accepted by deterministic polynomial time machines that have access to oracle B.

EXAMPLE 1 $\overline{\text{SAT}} \in P^{\text{SAT}}$. To decide whether a formula $\varphi \in \overline{\text{SAT}}$, the machine asks the oracle if $\varphi \in \text{SAT}$, and then gives the opposite answer as its output.

Exercises

§1 Show that maintaining a time counter can be done with logarithmic overhead. (Hint??)

§2 Show that $\mathbf{P}^{\text{TQBF}} = \mathbf{NP}^{\text{TQBF}}$.

§3 Show that $\mathbf{SPACE}(n) \neq \mathbf{NP}$. (Note that we do not know if either class is contained in the other.)

§4 Say that a class C_1 is *superior to* a class C_2 if there is a machine M_1 in class C_1 such that for every machine M_2 in class C_2 and every large enough n, there is an input of size between n and n^2 on which M_1 and M_2 answer differently.

(a) Is $\mathbf{DTIME}(n^{1.1})$ superior to $\mathbf{DTIME}(n)$?

(b) Is $\mathbf{NTIME}(n^{1.1})$ superior to $\mathbf{NTIME}(n)$?

§5 Show that the following language is undecidable:

$$\left\{< M >:\ M \text{ is a machine that runs in } 100n^2 + 200 \text{ time}\right\}.$$