CS 810: Complexity Theory 1/29/2007

Lecture 4: Time-Bounded Nondeterminism

Instructor: Dieter van Melkebeek Scribe: Baris Aydinlioglu

Last time we presented the strong connection among the problems that are in NP, which followed from the proof that the SAT problem is NP-complete under polynomial time mapping reductions. We strengthened this result by showing an even more efficient reduction that runs in quasi-linear time, establishing the NQLIN-completeness of SAT. We mentioned that almost all of the known natural problems in NP for which a polynomial-time algorithm was unknown were eventually shown to be NP-complete, and even NQLIN-complete. We finished the last lecture by asking two questions: If P differs from NP, then i) are there any complete problems in NP that are not in NQLIN, ii) are there problems in NP that are not NP-complete? In the first part of today's lecture we tackle these two questions in order. We answer the first question by obtaining hierarchy results for non-deterministic polynomial time, and the second by explicitly constructing an NP-intermediate language.

In the remainder of the lecture we discuss the concept of relativization, which will help explain why the P vs NP question has defied the efforts of many researchers for decades.

1 Nondeterministic Time Hierarchy

We start by showing that in the nondeterministic setting, similar to the deterministic case, with more time we can do strictly more.

Theorem 1. *For any two time bounds $t', t : \mathbb{N} \to \mathbb{N}$ such that $t(n) = \omega(t'(n+1))$ and t is time-constructible, we have* $\mathrm{NTIME}(t'(n)) \subsetneq \mathrm{NTIME}(t(n))$.

Proof. Recall why we cannot directly apply the diagonalization technique that we have used in separating deterministic time. In the proof of the deterministic time hierarchy, with polynomially more time in our hands, we were able to use the universal DTM to simulate the computation of a DTM with a lower time bound and then flip the result, hence differ from all deterministic computations with the lower time bound. In the nondeterministic setting, on the other hand, although we can efficiently simulate an NTM, we do not know how to negate the output of an NTM without using exponential time[1]. Since a hierarchy result with exponential jumps in time is not interesting, we need to come up with a different technique.

What we use is a modified version of the diagonalization technique from the proof of the deterministic time hierarchy, named *delayed diagonalization*, which is based on the following idea: just as in the deterministic setting, we try to diagonalize against all machines running in the lower time bound t', one by one, but instead of ensuring that we disagree with each machine on one particular input, we ensure that we disagree somewhere in an interval of inputs (we don't care exactly which particular input we disagree on, in fact we won't know).

To be able to simulate a machine on an interval of inputs, in contrast with the deterministic case where we mapped input i to machine M_i, here we map intervals of inputs I_i to machine M_i.

[1]Specifically, the problem lies in the case where the simulated machine accepts on some branches and rejects on others.

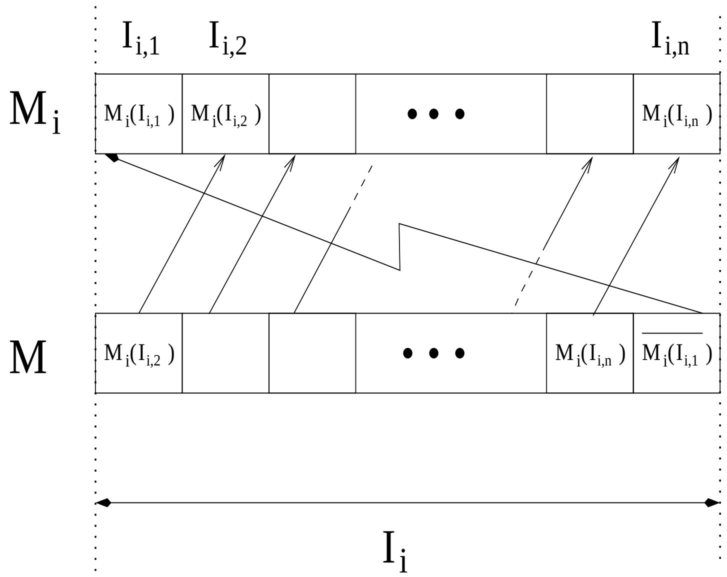

Figure 1: The delayed diagonalization of M on M_i in interval I_i. For each element of the interval except the last one, M nondeterministically simulates M_i on the next element of the interval. For the last element of the interval, M deterministically simulates M_i on the first element of the interval and flips the result.

On interval I_i, M attempts to diagonalize against machine M_i, and while doing so it will not use any more time than it is allowed, which is defined by the function t. As was the case for the deterministic time hierarchy, it must be the case that an infinite number of intervals is allocated for each machine M' - this ensures that the asymptotic behavior of t and t' guarantee that M has enough time to complete the construction for at least one of the intervals associated with M'. This can be achieved, although we skip these details in the present discussion. For a detailed example on how this mapping can be done, see p67 of the book by Arora & Barak.

The diagonalization is done as follows. Let the interval I_i contain n elements (we will find out what n must be shortly). Let $I_{i,k}$ denote the k^{th} element of I_i. The intervals are set up so that each element in the interval is 1 larger in size than the previous element so that for $1 \leq j \leq n-1$ we have $|I_{i,j}| = |I_{i,j+1}| - 1$ (for example, we could use $I_{i,1} = 0^m$ and $I_{i,k} = 0^{m+k-1}$ for some integer m and each $2 \leq k \leq n$). For $1 \leq j \leq n-1$, on $I_{i,j}$, M simulates M_i on $I_{i,j+1}$. Notice that for at least one of the intervals associated with M_i, M has enough time to simulate M for t' time steps on each $I_{i,j+1}$ since $t(|I_j|) = \omega(t'(|I_{j+1}|))$ and we can do universal nondeterministic simulation with a constant factor overhead in time. The behavior of M on the last element $I_{i,n}$ of the interval is quite different, and dictates the value of n. On $I_{i,n}$, M *deterministically* simulates M_i on $I_{i,1}$, and negates M_i's result. Notice that in order to be able to do this using a brute force deterministic simulation, M needs exponentially more time than how long it takes for M_i to halt on $I_{i,1}$. Therefore, the size of the interval, n, must be large enough so that $t(|I_n|) > 2^{t'(|I_{i,1}|)}$. See Figure 1.

We claim that M disagrees with M_i on some element of the interval. To see why, suppose towards a contradiction that for $1 \leq j \leq n$ we have $M(I_{i,j}) = M_i(I_{i,j})$. In particular, we have $M(I_{i,n}) = M_i(I_{i,n})$. By the way M was constructed, we also have $M(I_{i,n-1}) = M_i(I_{i,n})$ so $M(I_{i,n-1}) = M(I_{i,n})$. By repeated application of this argument, we get $M(I_{i,1}) = M(I_{i,n})$ and therefore $M_i(I_{i,1}) = M(I_{i,1}) = M(I_{i,n})$. However, M is constructed so that $M(I_{i,n}) = \overline{M_i(I_{i,1})}$, and we have reached a contradiction.

So we have shown that $L(M) \notin \mathrm{NTIME}(t'(n))$, and the proof is complete. $\square$

Theorem 1 tells us in particular that we can do strictly more in quadratic time than what we can do in quasi-linear time. This brings us to the answer of the first question we had raised, the proof of which we leave as an exercise.

Corollary 1. *There exist languages that are* NP*-complete but not NQLIN-complete.*

2 Existence of NP-Intermediate Problems

We now obtain the answer to our second question.

Theorem 2. *If* $\mathrm{P} \neq \mathrm{NP}$ *then there are* NP*-intermediate problems.*[2]

Proof. Notice that the statement of the theorem does not give any restrictions on the type of reductions involved. Therefore our proof must show this result in its strongest possible interpretation, namely under polynomial time oracle reductions. Hence we will display the existence of a language L in NP\P, such that L is not hard for NP even under $\leq_O^P$.

The main idea behind the proof is similar to the delayed diagonalization technique we employed in Theorem 1. We construct a language L that differs from all languages that have polytime DTM's, i.e. from all languages in P, by making L disagree with any possible polytime DTM on some input. Moreover, we make L not NP-hard by ensuring that the language for SAT disagrees on some input with any possible DOTM that uses an oracle for L. And we do these in such a way that L is in NP.

The first thing to be done then is to enumerate all polytime DTM's, and all polytime DOTM's with access to an L-oracle. We can do this as follows. Consider the set of all strings, $\Sigma^* = \{N_1, N_2, N_3, ...\}$. Given a member N_i of Σ^*, we interpret it (or parse it, if you will) in two different ways:

(1) as a pair $\langle M_j, n^k \rangle$, where M_j is a DTM that is clocked by time n^k. For brevity, call this interpretation $\Gamma(N_i)$.

(2) as a pair $\langle M_j^L, n^k \rangle$, where M_j^L is a DOTM that has access to an L-oracle and that is clocked by time n^k. Call this interpretation $\Psi(N_i)$.

It should be clear that any reasonable interpretation would give us the enumeration desired.[3]

Now, we want to construct $L \in \mathrm{NP}$ such that for $i \geq 1$ two conditions are met:

(1) $\Psi(N_i)$ fails to serve as a reduction for the language SAT.

(2) $\Gamma(N_i)$ decides a language different from L.

[2]The converse is also true, but is trivial.

[3]For example, given a string, we can parse its first half as the description of a TM and the second half as a time bound.

By meeting condition (1) we ensure that L is not NP-hard. For if L were NP-hard then there would be some polynomial time oracle reduction that reduces SAT to L, and meeting condition (1) over $i \geq 1$ explicitly rules out all such reductions. Similarly, with condition (2) we realize $L \notin$ P. For reasons that will become clear later, name condition (1) as C_{2i-1}, and condition (2) as C_{2i}.

In order to satisfy these two conditions, we construct L so that it somehow "interpolates" between some language in P and the language SAT. By interpolate, we mean on some intervals of inputs L agrees with the language in P, and on some others it agrees with SAT. On an interval where L agrees with the language in P, we make the interval large enough so that it contains a string on which the DOTM corresponding to that interval disagrees with SAT. This satisfies condition (1). On an interval where L agrees with SAT, we make the interval large enough that it contains a string on which the DTM corresponding to that interval disagrees with SAT. This satisfies condition (2). It doesn't matter which language in P we pick, so we just take the empty set, $\emptyset$.

We first present a construction that makes L meet the two conditions. Later we modify this construction so that L ends up in NP.

(1) $L \longleftarrow \emptyset$, $y \longleftarrow \varepsilon$(the empty string)
Construct L in phases. In phase i, realize conditions C_{2i-1} and C_{2i}, respectively:

(3) **foreach** phase $i = 1, 2, 3, \ldots$

(4) Since $L \in$ P, for any polytime DOTM M^L equipped with an L-oracle, there are infinitely many inputs on which M^L disagrees with SAT. In particular, this is the case for $\Psi(N_i)$. Let w be the lexicographically smallest string that comes after y such that $\Psi(N_i)$ disagrees with SAT on y.

(5) $L \longleftarrow (L \cap \Sigma^{\leq |w|}) \cup (SAT \cap \Sigma^{>|w|})$. In words, make L agree with SAT starting with strings of length $|w + 1|$ and larger. Notice that now $L \notin$ P.

(6) Since $L \notin$ P, for any polytime DTM M, there are infinitely many inputs on which M differs from L. In particular, this is the case for $\Gamma(N_i)$. Let y be the lexicographically smallest string that comes after w such that $\Gamma(N_i)$ disagrees with SAT on w (and thus also with L.)

(7) $L \longleftarrow (L \cap \Sigma^{\leq |y|}) \cup (\emptyset \cap \Sigma^{>|y|})$. In words, make L agree with $\emptyset$ starting with strings of size $|y + 1|$ and larger. Notice that now $L \in$ P.

As mentioned, L is not quite in NP yet. But it is close. In order to decide the membership of a string x in L, the only difficulty is in efficiently finding which of SAT or $\emptyset$ L agrees with on x. Once this is found out, the rest is an NP-computation (in case L agrees with $\emptyset$, the computation is trivial).

Given x, if we were not constrained on time, we could find out which of SAT or $\emptyset$ L agrees with on x by just running the above procedure until we hit x. The problem lies in steps 4 and 6—it may take too long, time exponential in $|x|$, to find a suitable string y on which N_i (in the form $\Psi(N_i)$ in case of step 4, $\Gamma(N_i)$ in step 6) disagrees with SAT.

We fix this problem by employing a kind of delayed diagonalization. Recall that the idea behind delayed diagonalization is to spread out each interval so that towards the end of the interval, the elements are so much larger than those at the beginning that we can brute-force check an otherwise hard-to-compute condition on the earlier elements. Specifically, we modify our construction in steps 4 and 6 by "waiting long enough" so that it becomes easy to detect a disagreement between SAT and N_i. In other words, a new interval in L is not begun until we can verify that condition for the present interval has been satisfied on some smaller input. The modified construction for L is presented below. The procedure COND takes the length of the input string x and returns the index of the condition that L realizes on that length. If the returned value is odd, then we know that L realizes condition (1) on x, i.e. L agrees with $\emptyset$ on x. If it is even, then we know that L realizes condition(2) on x, i.e. L agrees with SAT on x.

(1) **if** COND($|x|$) is odd **then** reject
(2) **else** return SAT(x)

PROCEDURE COND($n \in \mathbb{N}$)
(1) **if** $n = 0$
(2) **return** 1
(3) Compute COND($n-1$)
(4) Check the first n strings in lex order to see whether they witness $C_{\text{COND}(n-1)}$. Clock each check for n steps only. If can't decide in the given time bound, then conclude witness was not found.
(5) **if** witness found
(6) **return** COND($n-1$)+1
(7) **else**
(8) **return** COND($n-1$)

Given n, the procedure COND recursively calls itself on decreasing values of n. In step (4), we must compute SAT on the first n strings in lex order to verify if N_i agrees with SAT. As these n strings are of length $O(\log n)$, this can be done with a brute force search in polynomial time. Given this fact, the reader can verify that COND runs in polynomial time, and thus $L \in$ NP as desired while still satisfying each condition C_{2i-1} and C_{2i}.

□

3 Relativization

Definition 1. Relativizing a *(complexity theoretic)* statement with respect to *(a language)* A *means giving each machine involved in that statement access to an A-oracle. We say that* a statement *(theorem, proof, etc.)* holds relative to A *if it holds when it is relativized with respect to A. We say that* a statement relativizes *if it holds with respect to any language.*

Example: Consider the statement

$$\text{NTIME}(n) \subsetneq \text{NTIME}(n^2), (*)$$

which follows from Theorem 1. There are two types of machines involved in this statement, those that run in time $O(n)$ and those that run in time $O(n^2)$. We relativize this statement with respect to an oracle for some language A, by giving all of those machines access to an A-oracle. We express this modification by writing NTIME(n) as NTIME$^A(n)$, and NTIME(n^2) as NTIME$^A(n^2)$.

We say that (*) holds relative to A if

$$\mathrm{NTIME}^A(n) \subsetneq \mathrm{NTIME}^A(n^2).(**)$$

We say that (*) relativizes if (**) holds for any language A.

We claim that (*) relativizes. To see why, notice that the proof of Theorem 1 applies almost verbatim. The only modification needed is, during its simulation,whenever M_i uses the A-oracle (by writing onto the query tape, by transitioning to the query state, or by reading the query output), the simulator M does the same for its A-oracle. To state succinctly, the proof of the theorem relativizes and thus so does its statement. ⊠

Example: Recall from lecture 2 the language K_N, containing all tuples $\langle M, x, 0^t \rangle$ such that the NTM M accepts on input x in $\leq t$ steps. We had shown that K_N is complete for NP under $\leq_m^{\log}$.

Define, given a language A, $K_N^A = \{\langle M, x, 0^t \rangle \,|M^A$ halts and accepts x in $\leq t$ steps$\}$. As in the previous example, the proof of NP-completeness of K_N relativizes. Therefore, given any language A, K_N^A is complete for NP^A under $\leq_m^{\log}$. ⊠

One conclusion we can draw from the last example is that for any given language A, there is a NOTM N such that $L(N^A) = K_N^A$. But we can observe more. Consider two relativizations of the proof of completeness of K_N, one with respect to a language A, and another to a language A'. In the two proofs, the only difference between N^A and $N^{A'}$ is their oracles. Hence we reach a stronger result: there exists a *fixed* NOTM N such that for *any* A we have $L(N^A) = K_N^A$. In fact, N is essentially our efficient universal NTM. This remark will be a key ingredient in the proof of the next theorem.

3.1 Limits of the simulation technique and the P vs NP question

A fact which turns out to be very revealing is that almost all the results in complexity theory relativize. In particular, the kinds of techniques that we have used so far, such as diagonalization and lazy diagonalization, all relativize. This is because at the core of these techniques lies simulation; if we equip both the simulator and the simulated with a certain oracle, then all of the statements in the proofs without the oracle carry us through to yield proofs with the oracle.

There are many other techniques that we will see through the semester, nevertheless almost all of them relativize. In fact, those techniques that do not relativize are so infrequent that the reader should assume (and verify) that a given statement relativizes unless told otherwise.

What makes this important is our next result: There exist oracles relative to which P differs from NP, but there also exist oracles relative to which P equals NP. In other words, neither of the statements "P = NP" and "P $\subsetneq$ NP" relativizes. Before we prove this, we note its significance.

The implication of this corollary is that none of the techniques that we have seen so far (in general those that relativize) are enough to solve the P vs NP question. To settle this problem we need a new idea.

Theorem 3. *There exist oracles A and B s.t.* $\mathrm{P}^A = \mathrm{NP}^A$ *and* $\mathrm{P}^B \neq \mathrm{NP}^B$.

Proof. (First half) We start with the construction of the oracle (i.e., the language) A, relative to which P equals NP. The usual proof of this fact uses a PSPACE-complete language as the oracle A. We give an alternate proof that illustrates the techniques used in oracle constructions. We mention the alternate proof in a later lecture. We will construct A in such a way that we can build a DOTM M^A that runs in polytime and that decides a NP^A-complete language, giving us the first half of the theorem. The NP^A-complete problem that M^A solves will be K_N^A.

The key idea is the following. Call the NOTM for K_N^A N^A. We want to somehow make M^A as powerful as N^A. We will achieve this by making A compensate for M^A's lack of nondeterministic power. Specifically, we will construct A so that it contains the results of N^A's computations.

This strategy contains a pitfall that must be avoided. At any point during the construction of A, whenever we store a computation result of N^A in the language A, we must be careful not to affect any of N^A's computations *prior* to that point in the construction.

In order to address this issue, we make a key observation. Given an input of length n, the longest oracle query that N^A can make is of length $n-1$. To see why, recall how N^A works: it simulates the NOTM encoded by the first part of its input, for at most the number of steps equal to the length of its input, n. But if the simulated machine has n steps to execute, the longest query it can make to the oracle is of length $n-1$, since it needs 1 step to actually perform the query.

This observation clears the way for the construction of A. Similar to the proof of Theorem 2, we will build A in phases. At the end of phase i, we will have assigned the membership to A of all strings of length $\leq i$. Here is the procedure:

(1) $A \leftarrow \emptyset$
(2) **foreach** phase $i = 0, 1, 2, ...$
(3) **foreach** $x \in \Sigma^i$
(4) **if** $N^A(x)$ accepts
(5) $A \leftarrow A \cup \{x\}$

We can now elaborate on how this procedure works as intended. Consider the fully constructed A. Let A_i stand for A at any point in phase i during its construction. Then we have, for any phase i, that the set $A \backslash A_i$ does not contain a string of length $< i$, i.e., whatever string we add to A in phase i or subsequent phases is of length i or longer. In light of the observation above regarding the length of oracle queries that N^A makes with respect to its input size, this means that the addition of a string x to A_i does not affect any of the computations of N^A on inputs of length $\leq |x|$. So we do in fact avoid the pitfall mentioned earlier. As a parenthetical remark, observe that when considering $x \in \Sigma^i$ we need not follow any particular order.

With A constructed this way, M_A becomes trivial. On input x, M_A simply queries the oracle with x and returns the result of the query.

We now argue that $L(M^A) = K_N^A$. To see the containment from left to right, suppose that $x \in L(M^A)$. By the way M^A was constructed, this implies that $x \in A$. By the way A was constructed, this implies that x is accepted by $N^{A'}$, where $A' \subset A$. Again by the construction of A, this implies that N^A accepts x.

For the containment from right to left, suppose that N^A accepts x. Then, by construction, A contains x and thus M^A accepts x.

This completes the proof of the first half of the theorem.

(Second half)

We prove the second half by constructing a language through exploiting the power of nondeterminism. Consider the language $L^B = \{0^n | (\exists x \in \Sigma^n) x \in B\}$. Regardless of what the language B is, $L^B \in \mathrm{NP}^B$: given x, a NOTM N^B can guess a string of length $|x|$ in B. In what follows we construct B such that $L^B \notin \mathrm{P}$.

Let $N_1, N_2, N_3, ...$ be an enumeration of all polynomial DOTM's, i.e. all DOTM's clocked with all running times of the form n, n^2, n^3, etc. As the case in previous proofs, we assume each machine appears in the enumeration infinitely often. Without loss of generality, let N_i have a running time of n^i.

We build B also in phases. In phase i, we realize the condition $C_i : L^B \neq L(N_i^B)$. In each phase, we fix the oracle on strings longer than those considered in the previous phase. This ensures that the computation results of a DOTM N_j^B from any previous phase $j < i$ remain unaffected. Here is the procedure for constructing B.

(1) $B \leftarrow \emptyset$
(2) $f(0) \leftarrow -1$
(3) **foreach** phase $i = 1, 2, 3, ...$
(4)   pick an integer k such that $k > f(i-1)$ and $k^i < |\Sigma|^k$
(5)   **if** $N_i^B(0^k)$ rejects
(6)   pick a string y of length k which N_i has not queried while deciding 0^k
(7) $B \leftarrow B \cup \{y\}$
(8) $f(i) \leftarrow k^i$

$f(i)$ keeps track of the maximum length of the string that N_i could have queried given the construction so far. In line 4, we pick k large enough so that strings of length k have not been set yet and so that N_i cannot query all strings of length k. Then we run N_i on 0^k and if it accepts, we do not put in B any string of length k, thereby realizing condition C_i. In case N_i rejects, we put into B a k-length string which N_i has not queried, hence realizing C_i.

No polynomial time DOTM decides L^B, which completes the proof of the second half of the theorem.

$\square$

4 Next lecture

In the next lecture we will move from our current time-bounded setting to the space-bounded one, where we investigate the power of computation with limits on the amount of space that can be used. We will see results that suggest that the space-bounded setting is better understood, for example that nondeterministic space is closed under complementation.