CS 710: Complexity Theory 1/28/2010

Lecture 4: Time-Bounded Nondeterminism

Instructor: Dieter van Melkebeek Scribe: Dmitri Svetlov and Baris Aydinlioglu

In the previous lecture, we discussed NP-completeness and gave some strong results pertaining to the complexity of SAT, *viz.* that it is complete for NP under a reduction computable in logarithmic space and polylogarithmic time, and that it is complete for NQLIN (the set of NP problems solvable in quasi- linear time) under quasi-linear time mapping reductions. The conclusion that all naturally occurring NP-complete problems are also NQLIN-complete raised two questions: are there any complete problems in NP that are not also in NQLIN, and if P $\neq$ NP, then are there NP-intermediate problems, *i.e.*, problems that are in NP $\setminus$ P but that are not NP-complete?

In this lecture, we investigate time-bounded nondeterminism more closely to address these questions. First, we present a time hierarchy theorem for non- deterministic computation analogous to that for deterministic computation; given more time, one can perform strictly more computation. Second, we show the existence of NP-intermediate problems by constructing a specific (artificial) example of one. Finally, we discuss the impact of relativization on the question of whether P = NP and how it can be answered.

1 Hierarchy Results for Nondeterministic Computation

We showed in the second lecture that for any functions t' and t, if t' is sufficiently smaller than t, then DTIME(t') $\subsetneq$ DTIME(t). Here we prove that an analogous result exists for NTIME.

Theorem 1. *If $t, t' : \mathbb{N} \to \mathbb{N}$ are time bounds such that at least one of them is time-constructible and $t'(n) = \omega(t(n) + t(n+1))$, then* DTIME($t'$) $\subsetneq$ DTIME(t).

Proof. In the deterministic setting, we used a diagonalization argument involving the clocking of machines to show time hierarchies. For nondeterministic computation, complementation is a complicating factor for this argument since we know of no easy way to do it efficiently; therefore, we cannot "flip" results of specific computations in the same way we did for DTMs. So we employ a *delayed diagonalization.* Here, we diagonalize all machines running within the lower time bound t', but rather disagreeing with each machine on some particular input, we only ensure that there is a disagreement somewhere in an interval of inputs.

Let M_i be the machine to be diagonalized, M the diagonalizing machine, and I_i an interval of inputs to M_i. On each I_i, M attempts to diagonalize against machine M_i (within its time bound t) as follows. Let $I_{i,k}$ denote the kth element of I_i and choose each I_i such that each element is larger than the previous element, *e.g.* such that $|I_{i_j+1}| = |I_{i_j}| + 1$. For all elements $I_{i,j}$ except the last one, M simulates M_i on $I_{i,j+1}$; on the last element $I_{i,n}$, M *deterministically* (*i.e.*, by brute force) simulates M_i on the first element of the interval, $I_{i,1}$, and negates the result. Thus, n must be chosen to be sufficiently large to allow M exponentially more time than M_i needs to halt on $I_{i,1}$. For a graphical representation of this construction (on the specific example of machine M_1 and input interval I_1, see Figure 1.

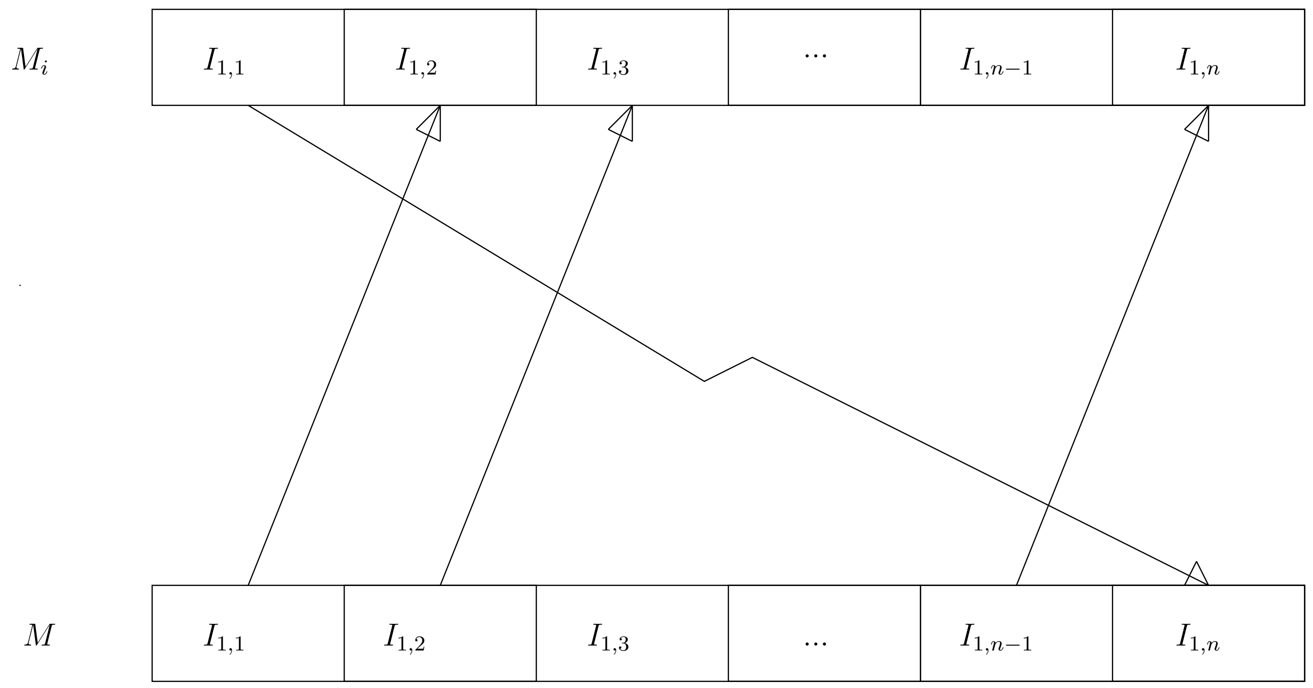

Figure. Simulation by M of M_i on interval I_1.

We now show that for some $I_{i,j}$, M disagrees with M_i. Suppose to the contrary that the machines agree everywhere. Then in particular, $M(I_{i,n}) = M_i(I_{i,n})$. But M was constructed such that also, $M(I_{i,n-1}) = M_i(I_{i,n})$, so $M(I_{i,n}) = M(I_{i,n-1})$. We can repeat this "crossing" of simulations to obtain that $M(I_{i,n}) = M(I_{i,1})$. By assumption, also $M(I_{i,1}) = M_i(I_{i,1})$. But then, $M(I_{i,n}) = M_i(I_{i,1})$, whereas by construction, $M(I_{i,n}) = \overline{M}_i(I_{i,1})$. So we obtain a contradiction, and therefore $L(M) \notin NTIME(t'(n))$. $\square$

2 Existence of NP-intermediate problems

We define the set of NP-intermediate problems as follows.

Definition 1. *$NPI \equiv NP \setminus (P \cup NPC)$, where NPC is the set of NP-complete problems.*

These problems exist only if P $\neq$ NP; otherwise, $NPI = \emptyset$. The notion of NP-intermediate problems is interesting in itself because for most NP problems of practical interest, it has been shown that the problem is in P or is NP-complete - with some notable exceptions, *e.g.* factoring and graph isomorphism. We will show the existence of such problems, assuming the inequality of P and NP, by constructing one (artificial) such problem that interpolates (in some sense) between a problem in P and some NP-complete problem, in such a way that it is everywhere either but overall neither.

Theorem 2. *If* P $\neq$ NP*, then there exists some L such that $L \in NPI$.* [1]

[1] The converse is trivially true.

Proof. The statement of this theorem does not specify the particular kind(s) of reduction involved, so this proof must demonstrate this result under the strongest possible interpretation, that of polynomial time oracle reductions. Therefore we must find a language L in NP that is not in P such that L is not hard for NP even under $\leq_O^P$.

To construct such a language, we use a technique similar to the delayed diagonalization used to prove Theorem 1. We ensure that $L \notin$ P, *i.e.* that L has no polytime DTM, by forcing L to disagree with every possible polytime DTM on some input, and also that L is not NP-hard by forcing it to disagree with the language for SAT on some input to any possible DOTM using an oracle for L. All that remains is to then show that $L \in$ NP.

We first enumerate all polytime DTMs and DOTMs (with access to an L-oracle) as follows. Consider the set of all input strings $\Sigma^* = \{N_1, N_2, N_3, ...\}$. Given a member N_i of Σ^*, we interpret (alternatively, parse) it in two different ways:

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

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

Note that reasonable interpretations should give the desired enumeration. [2]

We now construct $L \in$ NP such that for $i \geq 1$ we meet the conditions

(1) $\Psi(N_i)$ fails to serve as a reduction for SAT, and

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

By meeting (1) we ensure that L is not NP-hard, for if it were, then there would be some polytime oracle reduction from SAT to L, but meeting (1) over all i rules out such a reduction. By meeting (2), we ensure that $L \notin$ P. Name condition (1) as C_{2i-1} and condition (2) as C_{2i}.

The idea is that to meet these two conditions, we construct L so that it "interpolates" between some language in P (for simplicity, we will choose the empty set, $\emptyset$) and SAT, *i.e.* such that on some intervals of inputs L agrees with $\emptyset$ and on some others it agrees with SAT. On intervals where L agrees with $\emptyset$, we make the interval large enough that it contains some input on which the corresponding DOTM disagrees with SAT, satisfying condition (1), and on intervals where L agrees with SAT, we make the interval large enough that it contains some input on which the corresponding DTM disagrees with SAT, satisfying condition (2).

Below we give a construction that makes L satisfy these conditions; we will then modify it to ensure that $L \in$ NP.

[2]For instance, parse the first half of each string as the description of a TM and the second half as a time bound for it.

```
(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, ...
(4)       Since $L \in$ P, for any polytime DOTM $M^L$ (with an $L$-oracle), there are
          infinitely many inputs on which $M^L$ disagrees with SAT; in particular,
          this is true for $\Psi(N_i)$. Let $w$ be the lexicographically smallest string after
          $y$ such that $\Psi(N_i)$ disagrees with SAT on $y$.
(5)       $L \longleftarrow (L \cap \Sigma^{\leq|w|}) \cup (SAT \cap \Sigma^{>|w|})$. That is, make $L$ agree with SAT
          beginning with strings of length $|w+1|$ and larger. Now $L \notin$ P.
(6)       Since $L \notin$ P, for any polytime DTM $M$, there are infinitely many inputs
          on which $M$ disagrees with $L$; in particular, this is true for $\Gamma(N_i)$. Let $y$
          be the lexicographically smallest string after $w$ such that $\Gamma(N_i)$ disagrees
          with SAT (and so also with $L$) on $w$.
(7)       $L \longleftarrow (L \cap \Sigma^{\leq|y|}) \cup (\emptyset \cap \Sigma^{>|y|})$. That is, make $L$ agree with $\emptyset$ starting
          with strings of size $|y+1|$ and larger. Now $L \in$ P.
```

It remains to ensure that $L \in$ NP, which is not yet the case; in determining whether $x \in$ L, the difficulty lies in efficiently computing which of SAT or $\emptyset$ agrees with L on x. (The rest is then an NP-computation). Given x and absent any time constraints, we could determine this by simply running the above algorithm until x is reached. The problem is with steps (4) and (6)—it may take simply too long (possibly exponential in $|x|$) to find an input y on which N_i disagrees with SAT.

The power of delayed-diagonalization techniques in dealing with this costly computation is that if intervals can be sufficiently lengthened by the diagonalization, the later elements are so much larger than the earlier ones that it becomes possible to use brute force to check hard-to-compute conditions on those earlier inputs. So we modify steps (4) and (6) of the above construction by "waiting long enough" to allow easy checking of a disagreement between SAT and N_i; specifically, we do not begin a new interval in L until the appropriate condition for the *previous* interval has been satisfied on some smaller input. Below we give a modified construction of L using the checking procedure COND, which takes the length of the input x and returns the index of the condition that L realizes on that length. If the returned value is odd, then L realizes C_{2i-1} (for some i) on x, *i.e.* L agrees with $\emptyset$ on x. If it is even, then 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 lexicographical order to see if they
       witness $C_{COND(n-1)}$. Clock each check for only $n$ steps; if no
       decision is reached in this bound, conclude that no witness
       was found.
(5)    if a witness was found
(6)        return COND($n-1$)+1
(7)    else
(8)        return COND($n-1$)
```

On a given n, COND recursively calls itself on decreasing values of n. In (4), we must compute SAT on the lexicographically first n string to verify if N_i agrees with SAT. Since these n strings are of length $O(\log n)$, this can be done using a brute-force check in polynomial time. Thus the reader can verify that COND runs in polynomial time, and so $L \in$ NP while still satisfying each C_{2i-1} and C_{2i}. □

3 Relativization

Definition 2. Relativizing a *(complexity theoretic)* statement with respect to *(a language)* A *means giving 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. In this statement there are two types of machines - those running in time $O(n)$ and those running in time $O(n^2)$ - so to relativize this statement with respect to some language A, we give all these machines access to an A-oracle. We express this 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

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

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

We claim that (*) relativizes, and the proof is the same as that of Theorem 1, almost verbatim—except that whenever M_i uses its A-oracle, M does the same for its A-oracle. Thus the proof of the theorem relativizes and therefore, so does its statement. ⊠

Example: Recall from an earlier lecture the language K_N, containing all tuples $\langle M, x, 0^t \rangle$ such that the NTM M accepts on input x in no more than t steps. We showed then that K_N is complete for NP under $\leq_m^{\log}$.

Now, given a language A, we relativize K_N with respect to A by defining $K_N^A = \langle M, x, 0^t \rangle | M^A$ halts and accepts x in no more than t steps. As in the above example, it is clear that the proof of NP-completeness of K_N relativizes. Therefore, for any language A, K_N^A is complete for NP^A under $\leq_m^{\log}$. ⊠

From the second example, we conclude that there exists *some* NOTM N such that $L(N^A) = K_N^A$. Now consider two separate relativizations of the proof of the completeness of K_N, one with respect to a language A and one with respect to a different language $A\prime$. In the two modified proofs, the only difference between N_A and $N_{A\prime}$ is their oracles. So we obtain the stronger result that there exists a *fixed* NOTM N such that for *any* A we have $L(N^A) = K_N^A$. Indeed, this N is essentially our efficient, universal NTM. We will use this in proving the next theorem.

3.1 Limits of the simulation technique and the impact of relativization on the P versus NP question

Looking more broadly at the proof techniques used to demonstrate results in complexity theory, we find that almost of all of these techniques (and therefore the results obtained with them) relativize. For example, the techniques used thus far (diagonalization and delayed, or lazy, diagonalization) all relativize, because the conceptual core of all of them is simulation. When both the simulating and simulated machine are given access to a certain oracle, all proofs without using the oracle can be rewritten as proofs with the use of the oracle. It is perhaps surprising just how many proof techniques relativize—so many that one should assume (but also verify) that a given statement relativizes unless proven otherwise.

But two important statements that do not relativize are "P = NP" and "P $\subsetneq$ NP." This is because there exist oracles relative to which P differs from NP, but also ones relative to which P equals NP. This implies the disappointing result that none of the techniques we have seen so far—and perhaps none that have yet been discovered—are capable of solving the P versus NP question.

Theorem 3. *There exists oracles A and B such that* $\mathrm{P}^A = \mathrm{NP}^A$ *and* $\mathrm{P}^B \neq \mathrm{NP}^B$.

Proof. (First half) We start with the construction of the oracle (alternatively, language) A relative to which P equals NP. Whereas usual proofs of this fact use a PSPACE-complete language as A, we will use an alternate one to illustrate some techniques used in oracle construction. [3] We will construct A such that there exists a polytime DOTM M^A that decides an NP^A-complete language, namely, K_N^A.

Call the NOTM for K_N^A N^A. We want to make M^A as powerful as N^A. To do this we construct A so that it contains the results of the computations of N^A. In doing so, we must be careful that when storing each such result we do not affect any computations of N^A *prior* to that point. But we can take advantage of the fact that on an input of length n, the longest oracle query N^A can make is of length $n-1$. (Recall that N^A simulates the NOTM encoded by the first part of its input for a number of steps no greater than the length of its input. But since the query takes one step

[3] More detail about this proof will be given in a later lecture.

itself, at most $n-1$ steps can be used to write the query and thus only queries of *length* $n-1$ can be made.)

So we can construct A in phases (as in the proof of Theorem 2) such that at the end of phase i we will have assigned the membership to A of all inputs of length $\leq i$, as follows.

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

We can now verify that this construction works as we intend—that no addition of an input to the membership of A affects any prior step of the construction. Consider the addition of a given input x to A made at phase i. Note that after this phase, no input smaller than $|i|$ is added to A, since by phase i, all inputs of length less than i (*i.e.* up to $i-1$) have been queried by N^A and either added to or left excluded from A on the basis of the result. So the addition of string x to A affects no computations of N^A on inputs of length less than that of x, and we avoid the possible pitfall of this construction mentioned above.

It then follows that for all $x \in \Sigma^*, x \in K_N^A \Leftrightarrow x \in A$. With access to an oracle for A, M_A can simply query the oracle on input x and return the result. Since this can be done in linear time, $K_N^A \in \mathrm{P}^A$.

(Second half) We prove the second half of the theorem by constructing a language B that exploits the power of non-determinism in a way that no polytime DOTM can duplicate. Consider the language $L^B = \{O^n | (\exists x \in \Sigma^n)$ such that $x \in B\}$. Regardless of B, $L^B \in \mathrm{NP}^B$ since given x, an NOTM N^B can non-deterministically guess a string of length $|x|$ in B. So we construct B such that $L^B \notin \mathrm{P}^B$.

Let $N_1, N_2, N_3, ...$ be an enumeration of all DOTMs clocked at running times $n, n^2, n^3, ...,$ *i.e.* all polytime DOTMs. 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 $C_i : L^B \neq L(N_i^B)$, and we fix the oracle on strings longer than those considered in the previous phase, to ensure that no prior computation results are affected. The construction proceeds as follows.

```
(1)  $B \longleftarrow \emptyset$ $f(0) \longleftarrow -1$ foreach phase $i = 1, 2, 3, ...$
(2)      pick an integer $k$ such that $k > f(i-1)$ and $k^i < |\Sigma|^k$ if $N_i^B(0^k)$ rejects
(3)          pick a string $y$ of length $k$ $N_i$ has not yet queried while deciding $0^k$
(4)          $B \longleftarrow B \cup y$ $f(i) \longleftarrow k^i$
```

In this algorithm, $f(i)$ keeps track of the maximum length of queries N_i could have made thus far. At each phase, we choose k large enough 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 add any string of length k to B, which realizes C_i. If N_i rejects, then we add to B some k-length string not yet queried by N_i, realizing C_i. We note that no polytime DOTM decides L^B, and the proof is complete. $\square$

4 Next lecture

In the next lecture we will move from the time-bounded setting to the space-bounded one, where we will investigate the effects of limits on memory space on the power of computation that can be performed. We will obtain some results indicating that the space-bounded setting is better understood, *e.g.* nondeterministic space is closed under complementation, whereas the closure of nondeterministic time under complementation is open (no pun intended).