CS 710: Complexity Theory 9/22/2011

Lecture 5: Time bounded Non-determinism

Instructor: Dieter van Melkebeek Scribe: Gautam Prakriya

In the last lecture we introduced the notion of NP-completeness. We showed that Satisfiability is NP-complete under polytime mapping reductions that run in time $n \cdot (\text{poly}(\log n)$ (where each bit in the reduction can be computed simultaneously in logspace and poly-log time.). The same holds for many other NP-complete problems such as VERTEX-COVER, 3-SAT, INDEPENDENT-SET etc. Infact, most of the natural problems in NP which are not known to be in P are NQLIN-complete. We begin this lecture by illustrating this fact - we show that 3-SAT and VC are NQLIN-complete.

In Sections 2 and 3, we answer two natural questions - (a) if P differs from NP, are there problems in NP that are not NP-complete?, (b) Are there any NP-complete problems which are not NQLIN-complete? We answer the question (a) in the affirmitive by proving the conditional existence of NP-intermediate languages and address (b) by proving a time hierarchy theorem for non-deterministic machines. We end the lecture by introducing the concept of Relativisation.

1 Completeness for NQLIN (Contd.)

Theorem 1. *3-SAT is complete for NQLIN under* $\leq_m^{QLIN}$.

Proof. Given a CNF, we need to convert it to a CNF with clauses of size ≤ 3. We do this by splitting each clause that has more than 3 literals into a number of clauses with atmost 3 literals each. While doing so, we introduce additional variables to preserve satisfiability. For instance we replace $(l_1 \vee l_2 \vee l_3 \vee l_4)$ by $(l_1 \vee l_2 \vee z) \wedge (l_3 \vee l_4 \vee \overline{z})$. Where z is the new-literal introduced. In general a clause with $m > 3$ literals can be replaced by $m - 2$ clauses each containing 3 literals.

$$(l_1 \vee l_2 \vee \dots l_m)$$

can be written as

$$(l_1 \vee l_2 \vee z_1) \wedge (\overline{z_1} \vee l_3 \vee z_2) \wedge (\overline{z_2} \vee l_4 \vee z_3) \wedge \dots \wedge (\overline{z_{m-2}} \vee l_{m-1} \vee l_m)$$

□

Theorem 2. *VC is complete for NQLIN under* $\leq_m^{QLIN}$.

Proof. Given a 3-SAT formula ϕ, we define a reduction that returns a graph G and a number k, s.t. G has a vertex cover of size k iff ϕ is satisfiable. For each boolean variable x, we produce an edge which connects nodes x and $\overline{x}$ in the graph. The gadget for clauses is as follows - For each clause, we introduce a node for each literal in the clause. We connect these nodes to each other and to the nodes in the variable gadgets with an identical label. Figure 1 contains the graph produced by the reduction for the formula $x_2 \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee x_4)$. k is defined to be $n + \sum(|C_j| - 1)$ where n is the number of variables and $|C_j|$ is the size of clause j. We now show that ϕ is satisfiable iff G has a vertex cover of size k_ϕ.

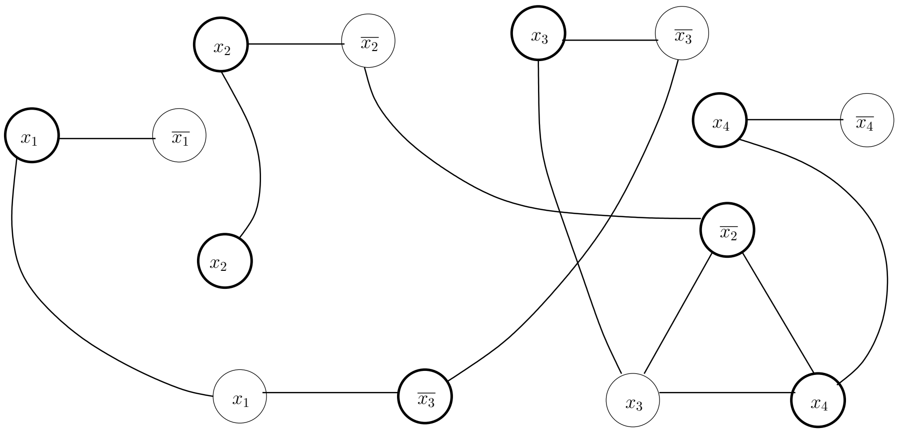

Figure 1: $x_2 \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee x_4)$ has a satisfying assignment iff this graph has a vertex cover of size $\leq 4 + 0 + 1 + 3$

Note that any vertex cover must contain atleast one literal from each of the vertex gadgets and atleast $|C_j| - 1$ from the gadget for clause j $(\forall j)$. Now, Suppose G has a vertex cover of size $n + \sum(|C_j| - 1)$, it is clear that exactly one vertex from each clause is not picked. This implies that atleast one outgoing edge from each clause is covered by a vertex in some vertex gadget (Each clause has $|C_j|$ outgoing edges.).

The literals from the vertex gadgets which are in the vertex cover define a assignment in an obvious way. The argument above shows that each clause must contain atleast one of these literals. Therefore, the assignment thus defined satisfies ϕ.

If ϕ has a satisfying assignment, pick the literals in the vertex gadgets that are set to true by the assignment. Since each clause must contain one of these literals, atleast one outgoing edge from each clause gadget is covered. Choosing the remaining vertices from each clause gives us a vertex cover of the required size.

□

2 Nondeterministic-Time Hierarchy Theorem

Recall that in an earlier lecture, we looked at the deterministic time-hierarchy. We now show a similar result for non-deterministic machines.

The argument we used in the deterministic case doesn't extend in an obvious way. This is because we don't know how to complement a NTIME($t(n)$)-language in NTIME($t'(n)$)[1]. However, we can complement a NTIME($t(n)$)-language in $2^{t(n)}$ deterministic steps. We use this fact along with a modified diagonalization argument to prove the following theorem.

Theorem 3. *Let $t, t' : \mathbb{N} \mapsto \mathbb{N}$ s.t, t' is time constructible and $t'(n) = \omega(t(n+1))$, then* NTIME($t(n)$) $\subsetneq$ NTIME($t'(n)$)

Proof. In the deterministic case, machine M(the universal machine we construct) diagonalizes the i^{th} input against Machine M_i. Here we associate an interval of inputs I_i with each machine M_i and

[1]The issue is that one might have to run through all non-deterministic choices of a machine on an input to to flip the answer of the machine.

ensure that for all i, M disagrees with M_i on atleast one element of I_i(The I_i's are all disjoint). The idea is that if the interval I_i is large enough then M when given one of the larger strings in I_i as input can deterministically complement M_i on one of the smaller elements of the interval. This technique is known as "Delayed Diagonalization". As in the deterministic case, we demand that each machine occurs infinitely often, so that the asymptotic behaviour of t and t' is captured.

The intervals are defined in such a way that the length of each element is one more than that of the previous one. Let the elements of I_i be $I_{i,1}, I_{i,2} \ldots I_{i,m}$, we require $|I_{i,k}| = |I_{i,k-1}| + 1$ for $2 \leq k \leq m$. For instance I_i could be the set $\{0^l, 0^{l+1}, \ldots 0^{l+m-1}\}$.

We now describe how M simulates M_i on I_i. The details of this simulation will also tell us what $m(= |I_i|)$ should be.

1. For input $I_{i,k}$, M simulates M_i on $I_{i,k+1}$ $(k < m)$. Note that $t'(|I_{i,k}|)$ is enough time to simulate $t(|I_{i,k+1}|)$ computation steps of M_i. This follows from the following facts: (a) $t'(|I_{i,k}|) = \omega(t(|I_{i,k+1}|))$ and (b) Non-deterministic simulation can be done with a constant factor overhead in time.

2. On input $I_{i,m}$, M simulates all runs of M_i on $I_{i,1}$ deterministically and flips the result. This tells us that the interval I_i has to be large enough, so that $t'(|I_{i,m}|) \geq 2^{\Omega(t'(|I_{i,1}|))}$ holds.

We won't get into details of how M computes these intervals. For a detailed illustration of how this can be done see page 70 of the book by Arora and Barak.

Now, suppose for the sake of contradiction that $M(x) = M_i(x)$ for all $x \in I_i$. This would imply by (1) that $M(I_{i,m}) = M_i(I_{i,1})$. This is a contradiction, since in (2) we set $M(I_{i,m}) = \overline{M_i(I_{i,1})}$. Therefore, we have shown that M disagrees with M_i on atleast one element of I_i.

□

3 NP-Intermediate Problems

Let NPC represent the set of NP complete problems under poly-time many-one reductions. We define the set of NP-intermediate problems as follows.

Definition 1. NPI = NP $\setminus$ (P $\cup$ NPC)

Theorem 4. *If* P $\neq$ NP *then* NPI $\neq \phi$.

Proof. To prove this theorem, we need to present a language $A \in$ NP $\setminus$ P s.t. SAT is not $\leq_m^p$ reducible to A. We will infact do more - we show that $\exists A \in$ NP $\setminus$ P s.t. SAT is not $\leq_o^p$ to A.

Firstly, let $M_1, M_2, M_3 \ldots$ be an enumeration of all deterministic Turing machines and $N_1, N_2, N_3 \ldots$ be an enumeration of all deterministic oracle Turing machines. We clock machines M_i and N_i by n^i $\forall i$. (Since we cannot enumerate the set of all poly-time Turing machines, we instead enumerate all Turing machines and clock them with polynomials.)

Our goal is to construct a set $A \in$ NP s.t $\forall i$:

1. $\mathbf{C_{2i-1}}$: $A \neq L(M_i)$.

2. $\mathbf{C_{2i}}$: $L({N_i}^A) \neq$ SAT .

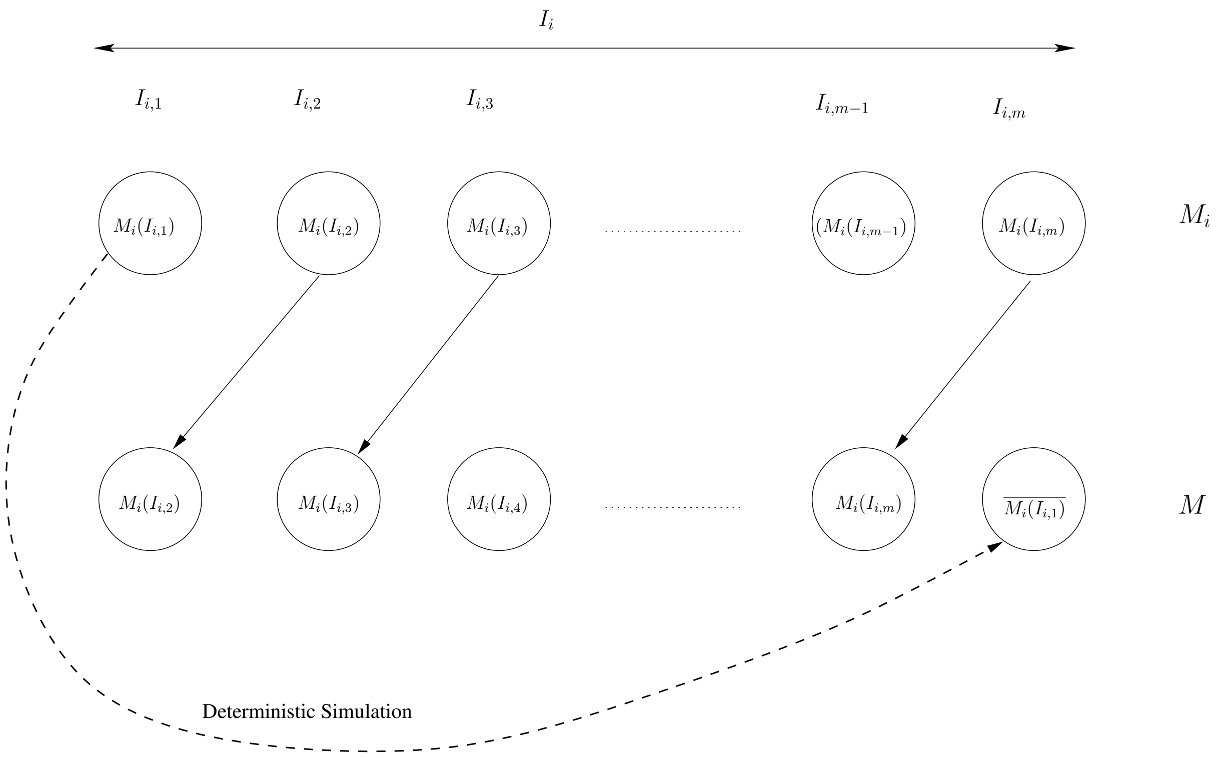

Figure 2: The Delayed Diagonalization of M against M_i on interval I_i

It is clear that by satisfying C_{2i-1}, $\forall i$, we realize $A \notin$ P, and by satisfying C_{2i}, $\forall i$, we ensure that there are no polynomial time oracle reductions from SAT to A, i.e, A is not NP-hard .

Before getting into the details of the construction, we make a simple observation which we use crucially in our construction. It follows from our hypothesis(P $\neq$ NP) that SAT differs from all polynomial time computable languages at infinitely many points. Therefore, given any language time $L \in P$ and $n \in \mathbb{N}$, $\exists N \geq n$ s.t. L disagrees with SAT on x_N (where by x_k we refer to k^{th} input in lexicographic order.).

As in the previous theorem, we associate disjoint intervals of inputs with the machines. (M_i's and N_i's). Suppose I is the interval we associate with some M_i. Then by forcing A to behave like SAT on I and picking a "large enough" I, we ensure that A disagrees with $L(M_i)$ on I, therefore satisfying C_{2i-1}. Similarly, on an interval J corresponding to some N_i, we force A to behave like $\emptyset$(the empty set).[2] . If J is large enough, $L(N_i^A)$ will disagree with SAT on some element of J, thus ensuring that A satisfies C_{2i}.

We now give a formal description of our construction.

A is constructed in phases. In phase i we satisfy conditions C_{2i-1} and C_{2i}.

[2]We could replace $\emptyset$ by any language in P.

```
(1)  Initially we set $A = \text{SAT}$ and $y = \epsilon$
(2)  foreach phase $i = 0, 1, 2, ...$
(3)      Note that except for a finite set of inputs, $A$ is identical to SAT. Therefore,
         $A$ is NP-hard. It follows that $A$ disagrees with $L(M_i)$ on infinitely many
         inputs. Let $\omega$ be the lexicographically smallest string that follows $y$ s.t,
         $M_i$ disagrees with $A$ on $\omega$.
(4)      $A \longleftarrow (A \cap \Sigma^{\leq|\omega|}) \cup (\emptyset \cap \Sigma^{>|\omega|})$, i.e, $A$ contains no strings of size greater
         than $|\omega|$. Since $A$ is now finite $A \in \text{P}$.
(5)      Since $A \in \text{P}$, $L(N_i^A)$ disagrees with SAT at infinitely many points. Let $u$
         be the lexicographically smallest that follows $\omega$ s.t, $L(N_i^A)$ disagrees with
         SAT at $u$. The size of the largest oracle query $N_i^A$ could have made to
         $A$ on input $u$ is $|u|^i$. (Since we clock $N_i^A$ with $n^i$.) We set $y$ to be the
         lexicographically smallest string of size $|u|^i + 1$. This ensures that when
         $A$ is modified in future iterations, the computation of $N_i^A$ on $u$ remains
         unaffected.
(6)      $A \longleftarrow (A \cap \Sigma^{<|y|}) \cup (\text{SAT} \cap \Sigma^{\geq|y|})$, i.e, $A$ agrees with SAT for all strings of
         size $\geq |y|$.
```

The A obtained clearly satisfies C_{2i-1} and C_{2i} $\forall i$. However, it is not clear that the A is in NP. The issue is given x we need to first find which of SAT and $\emptyset$ A behaves like at x. This can be achieved by running the above algorithm till we hit x, however, this may take exponential time. The problem lies in steps (3) and (5) - Checking whether M_i(or N_i^A) disagree with SAT at some input can potentially take time exponential in x.

We get around this by using the technique of delayed diagonalization to modify the above construction. Recall that the idea behind delayed diagonalization is to stretch the intervals out, so that the elements at the end of the interval are much larger than the ones in the beginning. This allows us to brute-force check an otherwise hard to compute condition on the initial elements.

In our setting, this translates to modifying steps (3) and (5) by "waiting long enough" so that it becomes easy to detect a disagreement with SAT for the earlier elements. We formalize this by defining a Polynomial-time computable function COND which on input x tells whether A behaves like SAT or $\emptyset$ at x.

If COND(x) is odd, A agrees with SAT on x, otherwise, A agrees with $\emptyset$.

```
     PROCEDURE COND($n \in \mathbb{N}$).
(1)  if $n = 0$
(2)      return 1
(3)      COMPUTE COND($n - 1$)
(4)      Check first $n$ strings in lexicographic order and see if they witness
         $C_{COND(n-1)}$. Clock each check for $n$ steps. If undecided in given time
         bound, conclude that the witness was not found.
(5)      if Witness found
(6)          return COND($n - 1$) + 1
(7)      else
(8)          return COND($n - 1$)
```

In Step 4, we run a deterministic algorithm for SAT on the first n strings.

Suppose that $c = \text{COND}(n-1)$ is even. Then in step 4 we check if $N^A_{c/2}$ disagrees with SAT on the first n strings. Since we clock our computation on each of these strings by n, any oracle query to the language A has size atmost n. This ensures that the computation of $N^A_{c/2}$ on the first n inputs remains unaffected regardless of future changes to A. This is because, any changes made to A in subequent iterations will effect only the membership of strings with size $> n$.

It is easy to check that this procedure runs in polynomial time. We have thus shown that $A \in \text{NP}$ and $A \notin \text{P} \cup \text{NPC}$.

□

Problems which are believed to be NP-intermediate include Graph Isomorphism and Factoring.

4 Relativisation

Definition 2. *A Statement is said to relativize if it continues to hold if all Turing Machines are given access to the same oracle.*

Example: The Non-deterministic Time hierarchy theorem relativises: $\forall$ oracles A $\text{NTIME}^A(t(n)) \subsetneq \text{NTIME}^A(t'(n))$ where, $t'(n) = \omega(t(n+1))$. ⊠

Example: Recall that $K_N = \{< M, x, 0^t > \mid \text{NTM } M \text{ accepts } x \text{ in time} \le t\}$ was shown to be complete for NP under $\le_m^{log}$. This statement relativises. $K^A_N = \{< M, x, 0^t > \mid \text{NTM } M^A \text{ accepts } x \text{ in time} \le t\}$ is complete for NP^A under $\le_m^{log}$, $\forall$ oracles A. ⊠

5 Next Time

In the next lecture, We continue studying the concept of relativisation. We show that the statements P = NP and P $\neq$ NP do not relativize. Which means any proof resolving P vs. NP must contain a statement which doesn't relativize. Note that most of the statements and Proofs we have studied so far do relativize. We also investigate non-determinism with bounds on space. We will study results which suggest that space bounded non-determinism is better understood, e.g. non-deterministic space is closed under complementation.

Acknowledgements

In writing the notes for this lecture, I perused the notes by Baris Aydinlioglu for lecture 3,4 from the Spring 2007 offering of CS 810, and the notes by Dmitri Svetlov for lectures 3,4 from the Spring 2010 offering of CS 710.

References

Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* 2009.