CS 710: Complexity Theory 9/28/2011

Lecture 6: Space Bounded Non-Determinism

Instructor: Dieter van Melkebeek Scribe: Prathmesh Prabhu

Last lecture, we discussed time bounded non-determinism. Parallel to the treatment of time bounded deterministic computation, we presented a hierarchy result showing that given more time, one can perform strictly more computation. We next explored an implication of the P = NP question being settled in the negative - if P $\neq$ NP, then there exists of a class of problems, NP-intermediate, that is neither in P, nor NP-Complete.

We ended the lecture introducing the concept of *Relativization*, the property that the verity of a statement remains unchanged if all the machines involved are given access to a common oracle. In this lecture, we develop the idea of relativiation further, and use it to advance an explanation of why the P = NP problem can be expected to be hard to settle in either direction.

In the second part of the lecture, we continue our study of non-determinism by looking at space bounded non-determinism. We present three main results in this setting, noting that they are in some sense stronger than the corresponding results in the time bounded case. We also state some important consequences of these theorems and their proofs - they provide a natural complete problem for the classes NL and PSPACE respectively; and a hierarchy theorem for NSPACE is obtained as a corollary of one of the theorems.

1 Relativization

We discussed Relativization in the last lecture. Here, we restate the idea of relativization, emphasizing the difference between the claims that "a statement relativizes" and "a proof (of some statement) relativizes". We say that a statement relativizes if, given an arbitrary oracle machine A, the statements remains true if each Turing machine involved in the statement is given access to A. A proof of a statement relativizes if every step of the proof when considered a statement in itself relativizes according to the definition above. Note that if a statement has a proof that relativizes, then that implies that the statement itself relativizes. On the other hand, it is possible that a given proof for a statement fails to relativize, while the statement still relativizes.

1.1 Relativization and the P vs NP problem

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 usually be rewritten as proofs with the use of the oracle. It is perhaps surprising just how many proof techniques relativize—so many that one tends to assume (and hopefully also verify) that a given statement relativizes unless proven otherwise.

This observation, and the following result that both the statements P = NP and P $\neq$ NP do not relativize, together hint at the hardness of settling the P vs NP question — Because the statement stated in either direction fails to relativize, a proof settling the P vs NP question also can not

relativize. This means that some step in the proof must use a result that does not relativize - and such results are very rare in complexity theory.

Theorem 1. *There exists oracles $\mathcal{A}$ and $\mathcal{B}$ such that* $\mathrm{P}^{\mathcal{A}} = \mathrm{NP}^{\mathcal{A}}$ *and* $\mathrm{P}^{\mathcal{B}} \neq \mathrm{NP}^{\mathcal{B}}$.

Existence of $\mathcal{A}$ implies that there is no relativizing proof of the statement $\mathrm{P} \neq \mathrm{NP}$, because the statement fails to hold relative to the oracle $\mathcal{A}$. Similarly, there is no relativizing proof for $\mathrm{P} = \mathrm{NP}$ because the statement does not hold relative to oracle $\mathcal{B}$.

Proof. We will prove both parts of the theorem by constructing the two oracles, $\mathcal{A}$ and $\mathcal{B}$.

Construction of $\mathcal{A}$

$\mathcal{A}$ is an oracle such that $\mathrm{P}^{\mathcal{A}} = \mathrm{NP}^{\mathcal{A}}$. One approach to constructing $\mathcal{A}$ relies on letting $\mathcal{A}$'s language be PSPACE-complete. The proof constructs such an $\mathcal{A}$ to expand both $\mathrm{P}^{\mathcal{A}}$ and $\mathrm{NP}^{\mathcal{A}}$ to PSPACE. In the following, we use an alternate proof technique, directly constructing an $\mathcal{A}$ that lets a specific machine in $\mathrm{P}^{\mathcal{A}}$ solve a problem that is complete in $\mathrm{NP}^{\mathcal{A}}$.

Recall from lecture 3 the NP-complete language K_N. Below, we define corresponding language $K_N^{\mathcal{A}}$ that has access to oracle $\mathcal{A}$. We call the NTM that accepts $K_N^{\mathcal{A}}$ $N^{\mathcal{A}}$.

$K_N^{\mathcal{A}} = \{< M, x, 0^t > \mid \mathrm{M}$ is an NTM that accepts x in $\leq t$ computation steps when given access to oracle $\mathcal{A}$ $\}$

The key observations for constructing $\mathcal{A}$ are

- $K_N^{\mathcal{A}}$ is complete for the class $\mathrm{NP}^{\mathcal{A}}$. Therefore, to show that $\mathrm{P}^{\mathcal{A}} = \mathrm{NP}^{\mathcal{A}}$, it suffices to show that there is a machine in the class $\mathrm{P}^{\mathcal{A}}$ that recognizes $K_N^{\mathcal{A}}$. Let this specific machine be $M^{\mathcal{A}}$.
- Membership of a string y in $K_N^{\mathcal{A}}$ is influenced by the response of $\mathcal{A}$ on queries of length less than $|y|$. This is the case because $N^{\mathcal{A}}$ simulates the NTM encoded by first part of y on the input (also encoded in y) for time no longer than the length of y. To query the oracle, $N^{\mathcal{A}}$ must write down the query on the oracle tape in no more than $n-1$ steps and then execute the query. Therefore, any query must be of length $< |y|$.

□

The idea is now to encode all responses of $N^{\mathcal{A}}$ inside $\mathcal{A}$ so that $M^{\mathcal{A}}$ can trivially query $\mathcal{A}$ for the correct response for any string x. As noted above, the response of $N^{\mathcal{A}}$ on strings of length n depends on the response of $\mathcal{A}$ on strings of length $< n$. Therefore, we can build $\mathcal{A}$ in stages, as follows:

```
(1)   A ⟵ ∅
(2)   foreach phase i = 0, 1, 2, ...
(3)      foreach x ∈ Σ^i
(4)         if N^{A^{so far}} accepts x
(5)            A ⟵ A ∪ x
```

Note that, $\mathcal{A}^{so\ far}$, the currently constructed $\mathcal{A}$ in any phase, has enough information to predict the response of $N^{\mathcal{A}}$ for the strings considered in this phase. Also, each phase constructs $\mathcal{A}$ for strings of increasing size. Therefore, no addition of an input to the membership of $\mathcal{A}$ affects the prior construction of $\mathcal{A}$.

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

Construction of $\mathcal{B}$

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

Let $M_1^{\mathcal{B}}, M_2^{\mathcal{B}}, M_3^{\mathcal{B}}, \ldots$ be an enumeration of all DTMs clocked at running times $n, n^2, n^3, ...,$ *i.e.* all polytime DTMs. Without loss of generality, let M_i have a running time of n^i. Every DTM M occurs infinitely often in this enumeration. Moreover, given any input x such that M halts on x, there are infinitely many instances $\{M_i\}$ of a given machine M in the enumeration such that M_i halts on x (and returns the same answer). Therefore, it suffices to show that $\forall i \quad L^{\mathcal{B}} \neq L(M_i^{\mathcal{B}})$.

We build $\mathcal{B}$ also in phases; in phase i we realize the condition $C_i : L^{\mathcal{B}} \neq L(M_i^{\mathcal{B}})$, and we fix the oracle on strings longer than those considered in the previous phase, to ensure that no prior computation results are affected. We first assume $\mathcal{B}$ to accept the empty language. Each phase computes the value of a function f, where $f(j)$ is greater than the length of the longest string whose memebership in $\mathcal{B}$ is affected by the j^{th} phase *and* not affected in any of the later phases. In the i^{th} phase, we test each string of length at least $f(i-1)$ (picking strings in a well-defined order, say, lexicographical order) to find a string such that $M_i^{\mathcal{B}}(0^{|x|})$ does not query $\mathcal{B}$ with x. (Note that such a string necessarily exists since $M_i^{\mathcal{B}}(x)$ is clocked to run in time $|x|^i$. Eventually, we will hit x such that $2^{|x|} > |x|^i$, *i.e.*, there are more strings of length $|x|$ than the time $M_i^{\mathcal{B}}$ is allowed to execute on x. Hence, there is some string of length $|x|$ that $M_i^{\mathcal{B}}$ does not query $\mathcal{B}$ with). For this x, we are free to choose $\mathcal{B}(x)$ without affecting the behaviour of $M_i^{\mathcal{B}}$ on strings of length $|x|$. We diagonalize for this string, *i.e.*, if $M_i^{\mathcal{B}}$ rejects $0^{|x|}$ ($M_i^{\mathcal{B}}$ claims that there is no string of length $|x|$ in $\mathcal{B}$) we add x to $\mathcal{B}$. Otherwise $M_i^{\mathcal{B}}$ accepts, claiming that there is a string of length $|x|$ in $\mathcal{B}$. We simply ensure that no string that can be queried on inputs of length $O(|x|)$ is added to $\mathcal{B}$ in any of the latter phases by by setting $f(i)$ greater than $|x|^i$. Note that phase i only changes membership in $\mathcal{B}$ of strings longer than $f(i-1)$, maintaining the diagonlization setup in the earlier phases. Hence, we must set $f(i)$ high enough keeping the last two points in mind.

$$f(i) = |x|^i + 1$$

The algorithm is given below.

(1) $\mathcal{B} \longleftarrow \emptyset$
(2) $f(0) \longleftarrow 0$
(3) **foreach** phase $i = 1, 2, 3, ...$
(4) let $\{y_i\}_I$ be a lexicographically ordered list of strings from Σ^* such that $\forall i \in I \quad |y_i| \geq f(i-1)$
(5) **foreach** $y \in \{y_i\}_I$
(6) **if** $M_i^{\mathcal{B}}(0^{|y|})$ does not query $\mathcal{B}$ with $|y|$
(7) **if** $M_i^{\mathcal{B}}(0^{|y|})$ rejects
(8) $\mathcal{B} \longleftarrow \mathcal{B} \cup \{y\}$
(9) $f(i) \longleftarrow |y|^i + 1$
(10) **break**

We note that no polytime machine $M_i^{\mathcal{B}}$ recognizes $\mathcal{B}$ (by construction). This concludes the proof.

2 Space Bounded Non-Determinism

We now present three theorems relating to space bounded non-deterministic computations, and some of their implications. We will see that the results here are stronger than their counterparts in the time bounded case. In the following, we assume that we have a function $s(n) : \mathbb{N} \to \mathbb{N}$ such that $s(n) = \Omega(\log(n))$. We require that the space bound (expressed via $s(n)$) is at least logarithmic because sub-logarithmic space classes are not very well behaved. Logarithmic space is required by random access machine to jump to any point in the input, and sub-logarithmic space computations severely restrict our chosen computational model.

Theorem 2. $\text{NSPACE}(s(n)) \subseteq \bigcup_{c>0} \text{DTIME}(2^{c \cdot s(n)})$

Theorem (2) strengthens the earlier result that $\text{NTIME}(s(n)) \subseteq \bigcup_{c>0} \text{DTIME}\left(2^{c \cdot s(n)}\right)$ by interjecting the class NSPACE, *i.e.*, $\text{NTIME}(s(n)) \subseteq \text{NSPACE}(s(n) \subseteq \bigcup_{c>0} \text{DTIME}(2^{c \cdot s(n)})$. Direct corollaries of the theorem are NL $\subseteq$ P and NPSPACE $\subseteq$ EXP.

Theorem 3. $\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s^2(n))$

Theorem (3) states that only quadratically more space is needed to perform nondeterministic computation deterministically. No similar result is known for time-bounded computation. This result is made possible due to the fact that space is in a sense more "flexible" than time, *i.e.*, we can reuse space on tapes whereas we cannot reuse time steps. It is in a qualitative sense "easier" to provide definite bounds and relationships between space complexity classes than it was with time bound setting.

A corollary of the theorem is PSPACE = NPSPACE: PSPACE $\subseteq$ NPSPACE from the definition of the computational models; and from theorem (3), $\text{NSPACE}(p(n)) \subseteq \text{DSPACE}(p^2(n)) = \text{DSPACE}(p(n^2)) = \text{DSPACE}(p'(n)) \Rightarrow \text{NPSPACE} \subseteq \text{PSPACE}$. Here p and p' are polynomial functions of n.

Theorem 4. $co\,\text{NSPACE}(s(n)) = \text{NSPACE}(s(n))$

We will actually prove that $co\,\text{NSPACE}(s(n)) \subseteq \text{NSPACE}(s(n))$. The equality is obtained by complementing twice.

Before proving these theorems, we present some direct corollaries of these theorems.

Corollary 1. $\text{L} \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP} \subset \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXP} \subseteq \text{NEXP}$.

Proof. We proved some of the inclusions above. The remaining inclusions are also easily proved. $\text{L} \subseteq \text{NL}$, $\text{P} \subseteq \text{NP}$ and $\text{EXP} \subseteq \text{NEXP}$ follow trivially from the definition of the computational models. $\text{NP} \subseteq \text{PSPACE}$ follows from a consideration of a PSPACE machine that iterates through every possible witness verifying membership, and thus emulates the computation of an NP machine. □

Whether these inclusions are strict is an open problem, the only separations that are known come from the hierarchy results which we discussed in previous lectures. For example, we know $\text{L} \subsetneq \text{PSPACE}$ but not whether $\text{L} \subsetneq \text{NP}$.

Corollary 2 (Hierarchy result for Non-Deterministic Space Bounded computations)**.** *If* $s, s' : \mathbb{N} \to \mathbb{N}$ *such that* $s(n)$ *is space constructable and* $s(n) = \omega(s'(n))$ *then* $\text{NSPACE}(s'(n)) \subsetneq \text{NSPACE}(s(n))$.

The proof of this corollary follows the same diagonalization strategy as we have seen before in the proof for the hierarchy result for deterministic space bounded computations. In the bounded time setting, we had to employ delayed diagonalization in extending the result from the deterministic model to non-deterministic one, because complementation is a hard problem in that setting. The proof carries over directly from the deterministic case for space bounded computations because complementation is easier (Theorem (4)).

The rest of this lecture is devoted to proving the first two theorems presented in the section. We defer the proof for theorem (4) to the next lecture.

2.1 $\text{NSPACE}(s(n)) \subseteq \bigcup_{c>0} \text{DTIME}(2^{c \cdot s(n)})$

Proof. We first assume that the function $s(n)$ is space constructible. The proof hinges on the interpretation of runs of an NTM N on the input x as a search problem on the infinite configuration graph defined as follows: The set of vertices of the graph is exactly the set of all possible configurations of N on input x; there is an edge from a vertex n to m iff there is a valid transition from the configuration n to m for N on the input x. For a space bounded computation, this graph is finite. The proof proceeds by showing that the size of the configuration graph is at worst exponential in the space bound for N, and that a deterministic machine can solve the search problem on this graph in polynomial time.

Formally, given NTM $N = (Q, \Sigma, \Gamma, \Delta, s, t, k)$ (WLOG, we assume that the machine has a unique start (s) and terminal (t) state respectively; k is the number of work tapes), the configuration graph corresponding to the runs of N on x is defined as -

Vertices Each vertex must correspond to a unique configuration of the space bounded machine. In particular, it must uniquely identify the state of the machine (Q), the position of the input tape-head ($|x|$), the position of the work-tape tape-heads ($s(|x|)^k$), and the contents of the work tapes ($(|\Gamma|^{s(|x|)})^k$). Note that x itself is not part of the configuration of the machine. The

space bounds on machines do not include the size of the input string. Putting the possibilities together, the total number of vertices is

$$|V| = |Q| \times n \times (|\Gamma|^{s(|x|)})^k \times (s|x|)^k$$

Note that $|V| = 2^{O(s(|x|))}$, provided $s^{|x|} > \log(|x|)$ (assumed to be true).

Edges $c \to c'$ iff there is a valid transition from c to c' for N on input x.

Now, x is accepted by N iff there is path from s to t in the graph defined above. This is an *s-t connectivity* problem on a graph of size $2^{O(s(|x|))}$. It can be solved by a DTM in time linear in $|V|$, or exponential in $s(|x|)$. □

An interesting consequence of the proof is a natural complete problem for the class NL. The problem of instance, *s-t connectivity* is defined below.

Definition 1 (ST-CON)**.** *Given a finite graph directed graph and two special vertices s and t, determine whether there is path from s to t.*

Fact 1 ST-CON is complete for NL under $<_m^{log}$

Proof. We show that (ST-CON) is (a) in-NL and (b) NL-hard.

in-NL Given an ST-CON problem instance, an NTM N can non-deterministically choose the next node from the possibilities and create a walk on the graph starting from s. If the walk ever hits t, N accepts. The space required is to store the current node. All nodes can be indexed with indices $\leq$ the size of the problem instance. Storing the current node only requires logarithmic space in the index bound.

NL-hard Given any problem in NL, the configuration graph for the problem is in the worst case exponential in the space bound. This can be encoded in space logarithmic in the size of the graph, *i.e.*, linear in the space bound of the original problem. So, any problem in NL can be converted to an ST-CON problem by generating the configuration graph for the problem completely - this conversion can be done by a log-space (deterministic) machine. We can assume, WLOG, that the configuration graph has a unique initial and accepting state. Now, the intial machine accepts iff the answer to the derived ST-CON problem is yes. Hence, there exist a $<_m^{log}$ reduction from NL to ST-CON.

□

Another natural complete problem for NL is 2-SAT.

Definition 2 (2-SAT)**.** *Given a formula in CNF form (conjunction of disjunction) such that no clause has more than 2 literals, decide whether there is an assignment of literals for which the formula evaluates to true.*

Fact 2 2-SAT is complete for NL under $<_m^{log}$.

Proof. Again, there are two parts to the proof.

in-NL Left as an exercise.

Initial configuration c_0
Next configuration
...
...
...
...
Final configuration c_1

Figure 1: An illustration of the configuration tableau used in the proof of Theorem 3. The tableau has $O(s(n))$ and height $t = 2^{O(s(n))}$.

NL-hard By theorem (4), the complement of ST-CON is also in NL. We show the following reduction:

$$\overline{ST - CON} <_m^{log} 2 - SAT$$

The reduction is by encoding a given $\overline{ST - CON}$ problem as a 2-SAT formula. For every node v of the graph, there is a variable x_v in the formula. x_v is true iff v is reachable from the source node s. The following 2-SAT clauses encode the problem instance (V, E, s, t)

- $(u, v) \in E \Rightarrow (\overline{x}_u \vee x_v) \in \Phi$
- $x_s \in \Phi$
- $\overline{x}_t \in \Phi$

Note that this 2-SAT instance can be encoded in log-space. Finally Φ is satisfiable iff $\overline{s \to t}$.

□

2.2 NSPACE$(s(n)) \subseteq$ DSPACE$(s^2(n))$

Proof. The main idea behind the proof here is enumerating all possible executions of the given NTM N in the form of computation tableaux in order to check if any valid tableau leads to acceptance. Figure 1 shows an example of a computation tableau. We would like to determine if there are a sequence of states that take us from c_0 to c_1 using no more than $s(n)$ space. As noted, the width of the tableau is $O(s(|x|))$ for input x, because any configuration of N can be encoded in $O(s(|x|))$ space, as noted in the last proof. On the other hand, the height of the tableau in the worst case can be $O(2^{O(s(n))})$, allowing for all possible configurations without repetition. Thus, the straightforward simulation where all possible tableaux are generated in order does not work, because it involves an exponential number of decisions (there is a decision point at each row of the tableau).

Again, a naive approach takes exponential space. To achieve the quadratic space blow up, we use a divide and conquer strategy. Instead of searching for a valid computation tableau of height $2^{s(|x|)}$ from c_0 to c_1, we guess an intermediate state $c_{\frac{1}{2}}$ and verify independently that there is a computation tableau of height $2^{s(|x|)/2}$ from c_0 to $c_{\frac{1}{2}}$ and a computation tableau of height $2^{s(|x|)/2}$ from $c_{\frac{1}{2}}$ to c_1. The benefit of breaking the problem into these subproblems lies in the fact that each of the subproblems is half the size of the original problem (we are searching for computation tableaux of half the height) and the two problems can be solved in sequence, and the space required to solve the first instance can be reused to solve the second instance.

Moreover, this problem reduction can be stated as a Boolean formula:

$$x \in L(M) \Leftrightarrow (\exists c_{1/2})(\forall b \in \{0,1\}) c_{\frac{b}{2}} \vdash_M^{\frac{t}{2}} c_{\frac{b+1}{2}}$$

This reduction can be applied recursively reusing the space that was taken up in the computation of the first part of the problem. The base case is reached after $\log(t)$ (here t is the height of the original tableau) recursions where we need only verify that one state transitions to another in one step. The fully unrolled recursion can be stated as the following full quantified Boolean formula:

$$\begin{aligned}
x \in & L(M) \Leftrightarrow \\
& c_0^0 = c_0 \wedge c_1^0 = c_1 \\
& \wedge (\exists c_{\frac{1}{2}}^1)(\forall b^1 \in \{0,1\}) \\
& \qquad [(b^1 = 0) => c_0^1 = c_0^0 \wedge c_1^1 = c_{\frac{1}{2}}^1] \\
& \qquad [(b^1 = 1) => c_0^1 = c_{\frac{1}{2}}^1 \wedge c_1^1 = c_1^0] \\
& \wedge (\exists c_{\frac{1}{2}}^2)(\forall b^2 \in \{0,1\}) \\
& \qquad [(b^2 = 0) => c_0^2 = c_0^1 \wedge c_1^2 = c_{\frac{1}{2}}^2] \\
& \qquad [(b^2 = 1) => c_0^2 = c_{\frac{1}{2}}^2 \wedge c_1^2 = c_1^1] \\
& \ldots \\
& \wedge (\exists c_{\frac{1}{2}}^k)(\forall b^k \in \{0,1\}) \\
& \qquad [(b^k = 0) => c_0^k = c_0^{k-1} \wedge c_1^k = c_{\frac{1}{2}}^k] \\
& \qquad [(b^k = 1) => c_0^k = c_{\frac{1}{2}}^k \wedge c_1^k = c_1^{k-1}] \\
& \wedge c_0^k \vdash_M^1 c_1^k
\end{aligned}$$

The depth of this full quantified Boolean formula is logarithmic in the height of the original tableau, *i.e.*, $O(s(|x|))$. Each guessed configuration takes $O(s(|x|))$ space. Therefore, the total space required to write this formula down on one of the tapes is $O(s(|x|))^2$.

The innermost predicate whether the configurations guessed to come immediately before and after $c_{\frac{x}{t}}$ have valid one step transitions to and from $c_{\frac{x}{t}}$. This check requires space no more than linear in the configurations being checked, *i.e.*, $O(s(|x|))$. Finally, iterating through all possible configuration at each guess point is trivial (similar to incrementing an $O(s(|x|))$ bit counter).

Hence, the total space requirement to write down the Boolean formula and check for validity is $O(s(|x|))^2$. □

In both proofs above (for theorems 2 and 3), we implicitly assumed that $s(n)$ is space constructable. This assumption is not a major handicap. If $s(n)$ is not space constructible, we can run the procedure several times while increasing a fixed space-bound until the computation has enough space to complete. More precisely, iterate over space bounds $s = 1, 2, 3, ...$ If the computation reaches an accepting state, then accept. If the computation does not accept but tries to exceed the space bound, repeat this procedure with a larger space bound. If the computation does not on any path try to exceed the space bound, then accept if there is a path to an accepting state, otherwise reject. Once again, we see the economy of being able to reuse space to attempt simulation through trial-and-error — a facility we did not have in the time bounded setting.

Similar to the proof of theorem 2, this proof is closely connected to an important problem - TQBF.

Definition 3 (TQBF)**.** *Given a fully quantified Boolean formula, determine whether the formula is true.*

Corollary 3. *TQBF is complete for* PSPACE *under* $<_m^{log}$.

Proof. The proof of Theorem 3 gives a polynomial mapping reduction for a general PSPACE problem transforming it into a TQBF in polynomial time. This is a polynomial time log-space reduction to TQBF. Thereofre, TQBF is PSPACE hard. It is easy to specify a machine to solve TQBF that runs in PSPACE. The machine simply iterates over all possible valuations to the variables in the Boolean formula. Both storing the current valuation of the variables, and evaluating the formula for this valuation take space only polynomial in the size of the formula. □

The PSPACE complete problem gives us another way of showing that there exists oracle A such that $P^A = NP^A$. Note that PSPACE $\subseteq$ $\mathrm{P}^{\mathrm{TQBF}}$ by the corollary and $\mathrm{P}^{\mathrm{TQBF}} \subseteq$ PSPACE because all queries to the oracle can be constructed in polynomial time. Therefore $\mathrm{P}^{\mathrm{TQBF}}$ = PSPACE. Taking this idea further we get the following containment:

$$\mathrm{P}^{\mathrm{TQBF}} = \mathrm{PSPACE} = \mathrm{NP}^{\mathrm{TQBF}} \subseteq (\mathrm{PSPACE}^{\mathrm{TQBF}} = \mathrm{PSPACE})$$

The last equality is model dependent because it matters whether we count the cells used on the oracle tape or not (if the oracle tape is not counted, a $\mathrm{PSPACE}^{\mathrm{TQBF}}$ machine could abuse the oracle tape to compute more than an unassisted PSPACE machine could).

Many PSPACE complete problems can be interpreted as adversarial game theory calculations. TQBF can be thought of as a game between two players, $\forall$ and $\exists$, who place quantifiers in a Boolean formula. $\exists$ wins if, after no player can move, the formula is true. Determining if there is a dominating strategy for $\exists$ given a formula is another PSPACE complete problem.

3 Next Time

In the next lecture, we will wrap up the discussion of space bounded non-deterministic models with the proof of Theorem 4. We will then introduce the polynomial-time hierarchy, look at some properties of these classes and present complete problems for each of the class.

Acknowledgements

In writing the notes for this lecture, I perused the notes by Baris Aydinlioglu and Matthew Anderson for lectures 4 and 5 from the Spring 2007 offering of CS 810, and the notes by Dmitri Svetlov and Michael Correll for lectures 4 and 5 from the Spring 2010 offering of CS 710.