CS221: Computational Complexity Prof. Salil Vadhan

Lecture 16: Recap, Overview, Relativization, and Randomized Computation

10/28 *Scribe: Alexander Sheh*

Contents

1 Announcements

- $\mathbf{TIME}(n) \stackrel{\text{def}}{=} \bigcup_c \mathbf{TIME}(cn)$, Papadimitrou's notation
- Schaefer-Umans catalogue of **PH**-complete problems on website

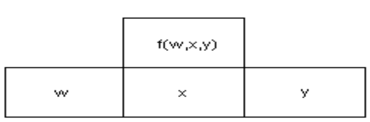
2 Recap

Theorem: REE is **EXPSPACE**-complete

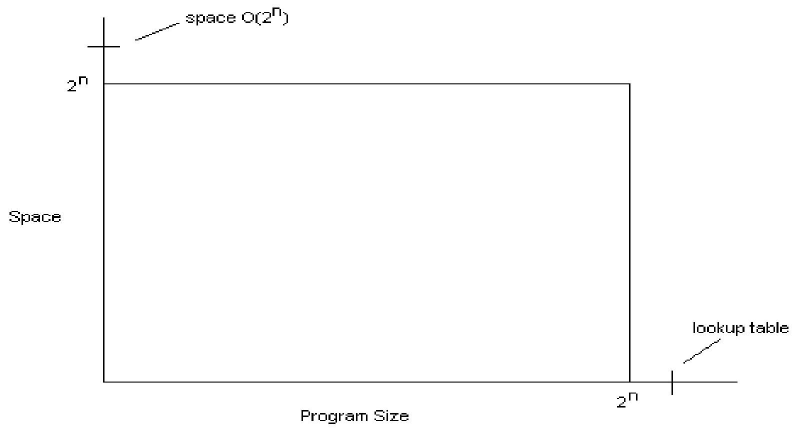
REE $\in$ **EXPSPACE** was shown in the previous lecture. REE is **EXPSPACE**-hard was also shown, involving the construction of a r.e.e. R such that TM M doesn't accept input x. Define $R =$ START $\cup$ MOVE $\cup$ FINISH, where START represents all strings violating the first configuration, MOVE represents all strings violating the transition rules, and FINISH represents all strings not containing an accepting state. Here, we define a simpler version of MOVE than last lecture,

MOVE $= \bigcup_{wxy} \sum^* wxy(\sum \uparrow (2^{n^k} - 1))(\sum - f(wxy)) \sum^*$

where $f(wxy) = z$ if z is the correct symbol of the configuration based on wxy. The idea for the simplification is taken from the Cook-Levin proof (i.e. upper box is a function of 3 boxes below it),



Proposition: $\exists$ language L $\in$ SPACE(O(2^n)) s.t. $\forall$ n, if M decides L correctly on length n then either $|M| \geq 2^n$ or M uses space $\geq 2^n$



We see that, for the language L, there is a tradeoff between resources and program size. For a program size larger than 2^n we can hardwire answers for all inputs of length n, i.e. a very efficient lookup table. For space larger than 2^n we can run a fixed size algorithm independent of n. But the problem cannot be solved if both resources are less than 2^n.

3 Overview

Diagram of Complexity Classes (only resources: space, time, nondeterminism, alternation)

"Positive" Results

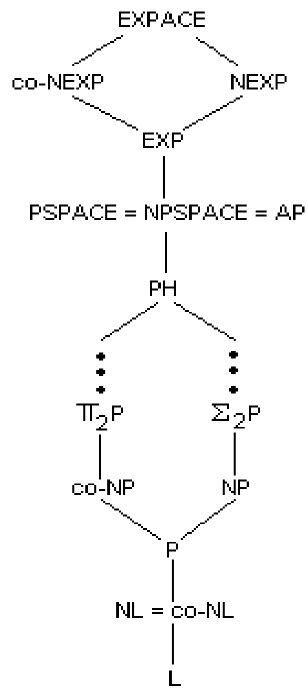
- (nontrivial) simulations of one resource by another (and alternation)
- completeness: relate natural problems to this picture

"Negative" Results

- i.e. "lower bounds", or separations
- Hierarchy Theorems: relate one resource to itself (Ex. L $\neq$ PSPACE, P $\neq$ EXP)
- combine nontrivial simulation with hierarchy theorem (Ex. NTIME(n) $\neq$ TIME(n), NL $\neq$ PSPACE)
- done comparatively little, don't even know P $\neq$ PSPACE!

4 Relativization

This material is in Papadamitriou 14.3. Noting that all our lower bounds so far have ultimately relied on the Hierarchy Theorems, which were proved by diagonalization, it is natural to ask "Can Diagonalization resolve P $\stackrel{?}{=}$ NP?"



- Observation: Diagonalization proofs (and most proofs derived from techniques in classical recursion theory) tend to hold in the presence of an oracle
- Why? Can enumerate algorithms that have oracle access, and can design a universal oracle algorithm (also with oracle access) that simulates other algorithms with access to the same oracle .

Theorem: $\exists$ oracle A s.t. $\mathrm{P}^A = \mathrm{NP}^A$

Proof: A = QBF (or any PSPACE-complete problem)

PSPACE $\subseteq \mathrm{P}^A \subseteq \mathrm{NP}^A \subseteq$ NPSPACE $\subseteq$ PSPACE

PSPACE $\subseteq \mathrm{P}^A$ holds because A is PSPACE-complete. Therefore, any language L $\in$ PSPACE can be decided by a polytime DTM that reduces L to A and then uses the oracle once. $\mathrm{P}^A \subseteq \mathrm{NP}^A$ is obvious. $\mathrm{NP}^A \subseteq$ NPSPACE holds because any NTM with an oracle A can be simulated by a NTM that is polyspace-bounded by computing the queries to A by itself in polyspace. NPSPACE $\subseteq$ PSPACE follows from Savitch's theorem.

Fact: $\exists$ oracle B s.t. $\mathrm{P}^B \neq \mathrm{NP}^B$

Proof in Papadimitrou, p. 340

Interpretation

Any theorem proved using only diagonalization should also hold if both machines are given oracles. So, if we could prove that P and NP were different by diagonalizing, we should be able to conclude that they are different relative to any oracle as well (but this conclusion is false). From above, we see that relativizing techniques alone cannot resolve P $\stackrel{?}{=}$ NP. However, we shouldn't be too discouraged. We have nonrelativizing techniques, and if any step doesn't hold in the presence of an oracle then we're okay. In general, as proofs become more combinatorial, they don't relativize. For example, with graph-theoretic analysis of computation in the PPST result, we used the fact that computation is "local". But oracles are applied to the entire tape at once, violating locality.

But still knowing the limits of relativizing proofs is useful. If we want to prove a result that doesn't relativize, we must make sure that some step in the proof doesn't relativize.

5 Randomized Computation

This material is in Papadimitriou 11.1.

- Is P the right notion of efficient computation? What about randomized algorithms?
- The algorithm can make "random choices" in its computation and we allow it to err with small probability.
- Are randomized algorithms physically implementable? This is more of a physics/EE question than a CS question. Randomized algorithms are used successfully in practice with various physical sources of "randomness", e.g. system clock, mouse movements, thermal noise from resistors, ... These sources are don't provide perfect randomness, though. CS 225 discusses theoretical approaches to dealing with this.
- Randomization has been central in complexity since the 80's.
- Also, randomized generalizations of classes have helped us even understand nonrandomized classes better.

5.1 Example: Identity Testing

Given: two algebraic expressions $p(x_1...x_n)$ and $q(x_1...x_n)$
e.g. $p(x_1...x_n) = (x_1 + x_2)(x_3 - x_4)x_6 + x_7(x_4 - x_5) + \cdots$
Decide: whether p $\stackrel{?}{\equiv}$ q

Here $p \equiv q$ refers to whether p and q define the same polynomial, i.e. have the same coefficients when written in canonical form

$$\sum_{i_1,\ldots,i_n \in \mathbb{N}} c_{i_1,\ldots,i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \tag{1}$$

Naive algorithm: Expand p and q into canonical form and compare their coefficients.

Problem: exponential blowup in number of terms

Better algorithm: Given $p(x_1...x_n)$ and $q(x_1...x_n)$,

1. Let m = $\max\{|p|, |q|\}$ and M = 2^m.
2. Choose randomly integers $\alpha_1...\alpha_n$ from S = {1...M},
3. Evaluate $p(\alpha_1...\alpha_n)$ and $q(\alpha_1...\alpha_n)$.

4. Accept if equal, reject otherwise.

We can evaluate directly in polynomial time. Clearly, if p $\equiv$ q then algorithm always accepts. The aim is to reject with high probability, if p $\not\equiv$ q.

Lemma: If p is an algebraic expression, then its *total degree* is at most its length $|p|$. The total degree is the maximum over the sum of the degree for that monomial, i.e. $\max_{c_1 \dots c_n \neq 0} \{i_1 + \dots + i_n\}$ in Equation (1).

Lemma: If $r(x_1 ... x_n)$ is nonzero polynomial of total degree d and we evaluate at random point asking whether it is a root, then

$$\Pr_{(\alpha_1 \dots \alpha_n) \in S^n}[r(\alpha_1 \dots \alpha_n) = 0] \leq \frac{\deg(r)}{|S|}$$

The above two lemmas imply the aim. To analyze the algorithm, apply the second lemma with $r = p - q$ and M $= 2^{\max |p|,|q|}$. By the lemmas, the error probability is at most

$$\frac{|\deg(r)|}{M} \leq \frac{\max\{|p|, |q|\}}{2^{\max\{|p|,|q|\}}},$$

which is exponentially small. We'll prove these lemmas and formalize the notion of randomized computation in the next lecture.