CS 221: Computational Complexity **Prof. Salil Vadhan**

Lecture Notes 24

April 26, 2010 **Scribe: Michael Jemison**

1 Recap

A sequence of polynomials $(p_n) = (p_n(x_1, \ldots, x_n))$ is in **AlgP/poly** if for all n:

- $\deg(p_n) \leq \text{poly}(n)$.
- p_n is computable by an algebraic circuit of size $\text{poly}(n)$.

It also will be useful for us to consider a *quasipolynomial* analogue of the above. A *quasipolynomial* function is one of the form $2^{\text{polylog}(n)}$, which we'll abbreviate as $\widetilde{\text{poly}}(n)$. Let $\widetilde{\textbf{AlgP/poly}}$ denote the class obtained by replacing all the $\text{poly}(n)$'s above with $\widetilde{\text{poly}}(n)$.

2 AlgNP/poly

Recall that a sequence (p_n) is defined to be in **AlgNP/poly** if there is a sequence $(q_n) \in$ **AlgP/poly** and a polynomial $t(n)$ such that

$$p_n(x_1, \ldots, x_n) = \sum_{e_{n+1}, \ldots, e_{t(n)} \in \{0,1\}} q_{t(n)}(x_1, \ldots, x_n, e_{n+1}, \ldots, e_{t(n)}).$$

To get a better feel for the kinds of polynomials captured by this definition, we provide a sufficient condition for being in **AlgNP/poly**: that the *coefficients* of p_n are efficiently computable.

Lemma 1 *Suppose (p_n) has the following properties for every n:*

1. $\deg(p_n) \leq \text{poly}(n)$ *and*
2. *There exists a* $\text{poly}(n)$*-sized* boolean *circuit C_n such that $p_n(x_1, ..., x_n) = \sum_{i_1, \ldots, i_n} C_n(i_1, ..., i_n) x_1^{i_1} ... x_n^{i_n}$.*

(Here C_n takes as input the binary representations *of $i_1, \ldots, i_n$ and outputs the binary representation of a coefficient.) Then $(p_n) \in$* **AlgNP/poly**.

Corollary 2 $\text{Perm}_{n \times n}(X) = \sum_{\sigma \in S_n} X_{1\sigma(1)} ... X_{n\sigma(n)} \in$ **AlgNP/poly**.

Proof: Let $C_{n \times n}(i_1 1, \ldots, i_{nn})$ output 1 if its input is a permutation matrix and 0 otherwise. ■

Proof of Lemma 1: Separate C_n into circuits computing the different bits of the binary representation: $C_n(i_1, ..., i_n) = \sum_j C_{n,j}(i_1, ..., i_n) * 2^j$. (For simplicity we consider nonnegative coefficients only, but the argument easily generalizes to C_n that also output a sign bit.)

Produce via the Cook-Levin reduction, for every n, j a formula $\phi_{n,j}(i_1, ..., i_n, e_{n+1}, ..., e_m)$, for $m = \text{poly}(n)$, such that

$$C_{n,j} = \sum_{e_{n+1},...,e_m \in \{0,1\}} \phi_{n,j}(i_1, ..., i_n, e_{n+1}, ..., e_m).$$

(Recall that $\phi_{n,j}(i_1, \ldots, i_n, e_{n+1}, \ldots, e_m) = 1$ iff $e_{n+1}, \ldots, e_m$ are the values at the gates of $C_{n,j}$ in an accepting computation on input $i_1, \ldots, i_n$. So if $C_{n,j}(i_1, \ldots, i_n) = 1$ there exactly one way to set the e_j's to make $\phi_{n,j}$ accept and otherwise there is no way to make $\phi_{n,j}$ accept.)

Now we arithmetize $\phi_{n,j}$ in the same way as in the **IP** = **PSPACE** proof, replacing AND with multiplication and NOT with $1-x$. This yields an algebraic formula $\widehat{\phi_{n,j}}$ of size and degree $\text{poly}(n)$ that agrees with $\phi_{n,j}$ on boolean inputs.

By substitution, we have

$$p_n(x_1, ..., x_n) = \sum_{i_1,...,i_n,e_{n+1},...,e_m} \left(\sum_j \widehat{\phi_{n,j}}(i_1, ..., i_n, e_{n+1}, ..., e_m) 2^j x_1^{i_1} ... x_n^{i_n} \right).$$

We're almost done, except that $x_v^{i_v}$ is not a polynomial in the binary representation of i_v. To handle this, observe that for say $v = 1$, if we write $i_1 = i_{1,0} + i_{1,1}2 + i_{1,2}2^2 + ...$, we can replace

$$x_1^{i_1} = \prod_k (i_{1,k} x_1^{2^k} + 1 - i_{1,k}).$$

■

3 Projection Reductions

Definition 3 *For two polynomials $p(x_1, \ldots, x_n)$ and $q(y_1, \ldots, y_m)$ over $\mathbb{F}$, we write $p(x_1, ..., x_n) \leq_{\text{proj}} q(y_1, ..., y_m)$ if there is a substitution $\sigma : \{y_1, ..., y_m\} \to \{x_1, ..., x_n\} \cup \mathbb{F}$ such that $p(x_1, ..., x_n) = q(\sigma(y_1), ..., \sigma(y_m))$.*

For two sequences of polynomials (p_n) and (q_n), we write $(p_n) \leq_{\text{proj}} (q_n)$ if there is a $t(n) = \text{poly}(n)$ such that for all n, $p_n \leq_{\text{proj}} q_{t(n)}$.

We write $(p_n) \leq_{\widetilde{\text{proj}}} (q_n)$ if there is a $t(n) = \widetilde{\text{poly}}(n)$ such that for all n, $p_n \leq_{\text{proj}} q_{t(n)}$.

Projection reductions also occur in boolean complexity. On Problem Set 1, you showed that Linear Programming is complete for **P/poly** under (boolean) projection reductions. However, in algebraic complexity, they turn out to lead to a mathematically very clean formulation of the **P** vs. **NP** problem.

4 The Permanent vs. Determinant Problem

Theorem 4 *Over fields of characteristic other than 2,* Permanent *is complete for* **AlgNP/poly** *under* $\leq_{\text{proj}}$.

(The restriction to characteristic other than 2 is essential, for otherwise the Permanent is the same as the Determinant.)

Theorem 5 *Over every field,* DETERMINANT *is complete for* $\widetilde{\mathbf{AlgP/poly}}$ *under* $\leq_{\widetilde{\text{proj}}}$.

Thus the relationship between **AlgNP/poly** and $\widetilde{\mathbf{AlgP/poly}}$ is equivalent to asking whether the Permanent is a projection of a quasipolynomially larger Determinant:

Corollary 6 $\mathbf{AlgNP/poly} \subseteq \widetilde{\mathbf{AlgP/poly}}$ *iff* $\text{Perm} \leq_{\widetilde{\text{proj}}} \text{Det}$.

This latter question makes no direct reference to computation, and one might hope to approach it using methods from algebraic geometry. Indeed, Mulmuley's "Geometric Complexity Theory" program is one major effort at attacking this problem using algebraic geometry and representation theory.

We now prove the completeness of the determinant.

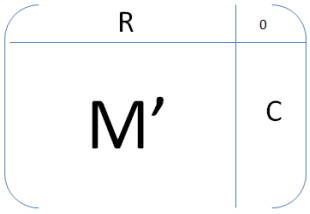
Lemma 7 (PS6) *If* $(p_n) \in \widetilde{\mathbf{AlgP/poly}}$ *then* (p_n) *has algebraic circuits of depth* $\text{polylog}(n)$*, and hence algebraic formulas of size* $2^{\text{polylog}(n)} = \widetilde{\text{poly}}(n)$

In fact, it is known how to do the depth reduction (but not necessarily the transformation to formulas) while preserving polynomial size. Thus, in the algebraic world, $\mathbf{P} = \mathbf{NC^2}$, something that we do not expect for boolean complexity.

The above lemma (transforming circuits to formulas) is the only reason we use quasipolynomial rather than polynomial bounds here. Given the lemma, to prove Theorem 5, it suffices to prove the following.

Lemma 8 *If* $p(x_1, ..., x_n)$ *is computable by algebraic formula of size* s*, then* $p \leq_{\text{proj}} \det_{(s+2)\times(s+2)}$.

Proof: We will give a recursive mapping from

formulas $F(x_1, ..., x_n) \mapsto$ matrices $M(x_1, ..., x_n) =$ 

,

where r and c denote the first row and column of M. The mapping will satisfy the following properties:

1. The entries of $M(x_1, \ldots, x_n)$ are from $\{x_1, \ldots, x_n\} \cup \mathbb{F}$,
2. $F(x_1, ..., x_n) = \det(M(x_1, ..., x_n))$,
3. $\det(M'(x_1, ..., x_n)) = 1$, and
4. If F has size s, then M is of size at most $(s + 2) \times (s + 2)$.

We recursively define the mapping as follows:

$$x_i \mapsto \begin{pmatrix} X_i & 0 \\ 1 & 1 \end{pmatrix}$$

$$c \in \mathbb{F} \mapsto \begin{pmatrix} C & 0 \\ 1 & 1 \end{pmatrix}$$

$$F_1 \cdot F_2 \mapsto M = \begin{pmatrix} M_1 & 0 \\ 0 & M_2 \end{pmatrix}$$

$$F_1 + F_2 \mapsto M = \begin{pmatrix} R_1 & R_2 & 0 \\ M_1' & 0 & C_1 \\ 0 & M_2' & C_2 \end{pmatrix}.$$

In the case of $F_1 * F_2$ we see that $\det(M) = \det(M_1)\det(M_2)$ and $\det(M') = \det(M_1')\det(M_2') = 1$. For $F_1 + F_2$, we see that $\det(M) = \det(M_1) \cdot \det(M_2') + \det(M_2) \cdot \det(M_1') = F_1 + F_2$ and we still have $\det(M') = \det(M_1') \cdot \det(M_2') = 1$. It is clear that this mapping gives the result. ■

5 Connection to Boolean Complexity

Proposition 9 *If* **AlgNP/poly** $\subseteq$ **AlgP/poly** *over* $\mathbb{Q}$*, then* $\mathbf{P}^{\#\mathbf{P}} \subseteq \mathbf{P/poly}$.

Proof: Implement the addition and multiplication gates using the grade-school algorithms. The only issue is that the bit-lengths of numbers may blow up exponentially during the computation of the circuit. This can be handled by working modulo a randomly chosen poly(n)-bit prime — with high probability such a prime will not divide any of the poly(n) numerators or denominators (each of magnitude at most $2^{\text{poly}(n)}$) occurring in the computation of the circuit, and being larger than $n!$ will not affect the computation of a 0-1 permanent. ■

Thus proving algebraic lower bounds might be *easier* than proving boolean lower bounds, and this is one motivation to work on them (in addition to the model being natural in its own right).