comput. complex. **13** (2004), 1 – 46
1016-3328/04/010001–46
DOI 10.1007/s00037-004-0182-6

computational complexity

DERANDOMIZING POLYNOMIAL IDENTITY TESTS MEANS PROVING CIRCUIT LOWER BOUNDS

Valentine Kabanets and Russell Impagliazzo

Abstract. We show that derandomizing Polynomial Identity Testing is essentially equivalent to proving arithmetic circuit lower bounds for NEXP. More precisely, we prove that if one can test in polynomial time (or even nondeterministic subexponential time, infinitely often) whether a given arithmetic circuit over integers computes an identically zero polynomial, then either (i) $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ or (ii) Permanent is not computable by polynomial-size arithmetic circuits. We also prove a (partial) converse: If Permanent requires superpolynomial-size arithmetic circuits, then one can test in subexponential time whether a given arithmetic circuit of polynomially bounded degree computes an identically zero polynomial.

Since Polynomial Identity Testing is a coRP problem, we obtain the following corollary: If $\mathsf{RP} = \mathsf{P}$ (or even $\mathsf{coRP} \subseteq \bigcap_{\epsilon>0} \mathsf{NTIME}(2^{n^\epsilon})$, infinitely often), then NEXP is not computable by polynomial-size arithmetic circuits. Thus establishing that $\mathsf{RP} = \mathsf{coRP}$ or $\mathsf{BPP} = \mathsf{P}$ would require proving superpolynomial lower bounds for Boolean or arithmetic circuits. We also show that any derandomization of RNC would yield new circuit lower bounds for a language in NEXP.

We also prove unconditionally that $\mathsf{NEXP}^{\mathsf{RP}}$ does not have polynomial-size Boolean or arithmetic circuits. Finally, we show that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ if both $\mathsf{BPP} = \mathsf{P}$ and low-degree testing is in P; here low-degree testing is the problem of checking whether a given Boolean circuit computes a function that is close to some low-degree polynomial over a finite field.

Keywords. circuit lower bounds, derandomization, polynomial identity testing, hardness-randomness tradeoffs.

Subject classification. 68Q10, 68Q15, 68Q17.

1. Introduction

1.1. Derandomization from circuit lower bounds.

In the early 1980's, Yao (Boppana & Hirschfeld 1989; Yao 1982) showed that one-way functions whose inverses have high average-case circuit complexity can be used to construct pseudorandom generators, which suffice for the derandomization of such probabilistic complexity classes as RP and BPP. Yao's approach to derandomization was extended by Nisan & Wigderson (1994) to arbitrary hard functions which are not necessarily one-way (in particular, to hard Boolean functions). The sequence of papers (Andreev *et al.* 1998; Babai *et al.* 1993; Impagliazzo *et al.* 1999, 2000; Impagliazzo & Wigderson 1997; Shaltiel & Umans 2001; Sudan *et al.* 2001; Umans 2003) significantly strengthened this result by replacing the assumption of high average-case circuit complexity with that of high *worst-case* circuit complexity. For instance, Impagliazzo & Wigderson (1997) showed that $\mathsf{BPP} = \mathsf{P}$, provided that some language in $\mathsf{E} = \mathsf{DTIME}(2^{O(n)})$ requires Boolean circuits of size $2^{\Omega(n)}$.

These results showing that computational hardness can be used as a source of computational pseudorandomness, termed *hardness-randomness tradeoffs*, are considered as evidence that BPP can be derandomized. However, in order to derandomize BPP using such an approach, one would need to prove superpolynomial circuit lower bounds for some language in EXP. Establishing superpolynomial lower bounds for general models of computation (such as Boolean circuits) is one of the biggest challenges in complexity theory that has withstood several decades of sustained effort by many researchers. If proving superpolynomial circuit lower bounds is indeed necessary for derandomizing BPP, then it seems unlikely that such a derandomization result will be obtained in the near future.

This raises an obvious question: Can we derandomize BPP *without* proving superpolynomial circuit lower bounds for EXP?

It is well known that derandomizing BPP using a pseudorandom generator (as in Yao's original approach) does indeed require proving that $\mathsf{EXP} \not\subset \mathsf{P/poly}$ (see, e.g., Impagliazzo *et al.* (1999) for a proof). Moreover, there is a very tight connection between the quality of a pseudorandom generator and the strength of circuit lower bounds for EXP. Roughly, the existence of a pseudorandom generator that stretches n-bit inputs to $s(n)$-bit outputs, for $n < s(n) < 2^n$, is equivalent to the existence of a language in E that requires Boolean circuits of size $s(n)$ (Shaltiel & Umans 2001; Umans 2003).

On the other hand, Impagliazzo *et al.* (2002) showed that derandomizing $\mathsf{promise\text{-}BPP}$ would require proving that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$; here the derandomized algorithm for a $\mathsf{promise\text{-}BPP}$ problem is allowed to be nondeterministic subexponential-time.

These results may explain why no unconditional derandomization of the class $\mathsf{promise\text{-}BPP}$ has been achieved so far. But they leave open the case of

BPP. Presumably it is possible to derandomize BPP without derandomizing promise-BPP. However, the results in this paper show that even derandomizing RP requires a circuit lower bound of some form.

1.2. Polynomial Identity Testing. Deriving general consequences from the assumption BPP $=$ P seems difficult due to the apparent lack of BPP-complete problems. However, we focus on a particular BPP problem, and argue that derandomizing this problem implies a circuit lower bound.

One of the most natural problems in BPP (in fact, in coRP) is Polynomial Identity Testing: Given an arithmetic circuit, decide if it computes the identically zero polynomial. By the well known Schwartz–Zippel lemma (DeMillo & Lipton 1978; Schwartz 1980; Zippel 1979), evaluating a degree d multivariate polynomial on a tuple of random elements from a finite subset S yields a probabilistic algorithm whose error probability is at most $d/|S|$. The importance of this problem is witnessed by a plethora of its applications to perfect matching (Chari *et al.* 1995; Lovász 1979; Mulmuley *et al.* 1987), equivalence testing of read-once branching programs (Blum *et al.* 1980), multiset equality testing (Blum & Kannan 1995), primality testing (Agrawal & Biswas 2003; Agrawal *et al.* 2002), a number of complexity-theoretic results on probabilistically checkable proofs (Arora *et al.* 1998; Arora & Safra 1998; Babai *et al.* 1991; Lund *et al.* 1992; Shamir 1992), as well as sparse multivariate polynomial interpolation (Clausen *et al.* 1991; Grigoriev *et al.* 1990; Roth & Benedek 1991; Zippel 1979).

Recently, a number of probabilistic algorithms for the polynomial identity testing were proposed that use fewer random bits than the standard Schwartz–Zippel algorithm (Agrawal & Biswas 2003; Chen & Kao 2000; Klivans & Spielman 2001; Lewin & Vadhan 1998). However, these algorithms do not derandomize Polynomial Identity Testing in the strong sense. It is still a big open problem to come up with a deterministic polynomial-time (or even nondeterministic subexponential-time) algorithm for that problem.

The deterministic polynomial-time algorithm for Primality Testing discovered by Agrawal *et al.* (2002) achieves derandomization of a very special case of Univariate Polynomial Identity Testing. One may hope that similar techniques will be useful to obtain derandomization of the general problem of Polynomial Identity Testing. However, our results show that this seemingly innocuous problem is, in fact, basically equivalent to the classic, notoriously difficult problem of proving arithmetic circuit lower bounds. For instance, we show that designing a (nondeterministic) subexponential-time algorithm to test whether a given symbolic determinant is identically zero is as hard as proving superpolynomial arithmetic formula lower bounds.

1.3. Extending hardness-randomness tradeoffs. Originally, hardness-randomness tradeoffs were aimed at derandomizing BPP algorithms. Goldreich & Zuckerman (1997) showed that the same hardness assumptions imply the derandomization of the class MA introduced by Babai (Babai 1985; Babai & Moran 1988). Klivans & Melkebeek (2002) extended the tradeoffs to another class introduced by Babai, the class AM, as well as to some other randomized algorithms, e.g., the Valiant & Vazirani (1986) hashing technique.

On the other hand, no hardness-randomness tradeoffs were known for the algebraic (rather than Boolean) complexity setting. There are at least two reasons why one might be interested in such algebraic tradeoffs. The first reason is purely aesthetic. Showing that hardness-randomness tradeoffs hold in another complexity setting would be another indication of the fundamental nature of the idea of converting computational hardness into computational pseudorandomness.

The second reason is a bit more practical. Suppose that it is eventually shown that Permanent requires superpolynomial-size arithmetic circuits. It would be important to know whether such lower bounds could be used to derandomize some algebraic algorithms. Of course, it is doubtful that derandomized algorithms derived from circuit lower bounds could be useful in practice, but even just proving their existence is interesting.

A final motivation is that it shows another example of duality between meta-algorithms for a model and lower bounds for that same model. Often, the techniques used to prove lower bounds for some class of circuits also yield positive results for algorithms taking such circuits as inputs. For example, Linial *et al.* (1993) give a learning algorithm for constant-depth circuits based on lower bounds for such circuits; Paturi *et al.* (1998, 1999, 2000) develop a new algorithm for k-SAT and a new lower bound for depth-3 circuits, using the same technique analyzing the solution space of CNF's. In his thesis, Zane (1998) made the interesting empirical point that progress on meta-algorithms is linked to progress in lower bounds. A few formalizations of this principle are known, e.g., natural proofs (Razborov & Rudich 1997) ("a natural lower bound yields a cryptanalysis tool") or hardness-randomness tradeoffs ("a pseudorandom generator can be constructed if and only if Boolean circuit lower bounds for E can be proved"). Here, we get a formal statement of such a connection for arithmetic circuits: Identity testing for arithmetic circuits can be derandomized if and only if lower bounds can be proved.

We should point out that Linial *et al.* (1993) and Paturi *et al.* (1998, 1999, 2000) do not just show the connection between meta-algorithms for a model and lower bounds for that model, but actually use that connection to obtain new algorithms and new lower bounds. On the other hand, our result just demonstrates such a connection for the arithmetic circuit model. Similarly to the known Boolean hardness-randomness tradeoffs, our result shows that a nontrivial (nondeterministic subexponential-time) algorithm for Polynomial Identity Testing exists if and only if NEXP contains functions of superpolynomial Boolean or arithmetic circuit complexity. Thus, in general, progress in designing efficient deterministic algorithms is linked to progress in proving superpolynomial circuit lower bounds of some sort.

1.4. Our results. In this paper, we show that derandomizing Polynomial Identity Testing is essentially *equivalent* to proving superpolynomial circuit lower bounds for NEXP. More precisely, we prove that if one can test in polynomial time (or even nondeterministic subexponential time, infinitely often) whether a given arithmetic circuit over integers computes an identically zero polynomial, then either (i) $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ or (ii) Permanent is not computable by polynomial-size arithmetic circuits. This implies that proving that $\mathsf{RP} = \mathsf{ZPP}$ or $\mathsf{BPP} = \mathsf{P}$ is *as hard as* proving superpolynomial (Boolean or arithmetic) circuit lower bounds for NEXP.

We also consider a special case of Polynomial Identity Testing, *Symbolic Determinant Identity Testing*: Given a matrix A of constants and variables, decide whether the determinant of A is an identically zero polynomial. We show that any nontrivial derandomization of this problem would also yield new arithmetic formula lower bounds. Since this problem belongs to the class coRNC, we conclude that derandomizing RNC is as hard as proving arithmetic formula lower bounds.

For the converse direction, we extend the known hardness-randomness tradeoffs to the algebraic-complexity setting by showing the following. Polynomial Identity Testing of n-variate $\mathsf{poly}(n)$-degree polynomials computed by $\mathsf{poly}(n)$-size arithmetic circuits can be done deterministically in subexponential time, provided that Permanent (or some other family of exponential-time computable multivariate polynomials) has superpolynomial arithmetic circuit complexity.

While almost all of our results are reductions among certain complexity-theoretic assumptions, we do prove one unconditional circuit lower bound, albeit for a somewhat unnatural complexity class $\mathsf{NEXP}^{\mathsf{RP}}$. We show that either $\mathsf{NEXP}^{\mathsf{RP}} \not\subseteq \mathsf{P/poly}$ or Permanent is not computable by polynomial-size arithmetic circuits. It is well known that $\mathsf{MA\text{-}EXP} \not\subseteq \mathsf{P/poly}$ (Buhrman *et al.* 1998). A simple argument shows that $\mathsf{NP}^{\mathsf{RP}} \subseteq \mathsf{MA}$ and hence, by padding, $\mathsf{NEXP}^{\mathsf{RP}} \subseteq \mathsf{MA\text{-}EXP}$. Thus $\mathsf{NEXP}^{\mathsf{RP}}$ is the smallest known uniform complexity class that is proved to contain a language of superpolynomial (Boolean or arithmetic) circuit complexity.

We also prove that a certain version of Polynomial Identity Testing can be derandomized under the assumption that $\mathsf{EXP} \neq \mathsf{NP}^{\mathsf{RP}}$. This is similar to the result of Babai *et al.* (1993) saying that BPP can be derandomized under the assumption that $\mathsf{EXP} \neq \mathsf{MA}$. Our assumption is weaker since, as mentioned above, $\mathsf{NP}^{\mathsf{RP}} \subseteq \mathsf{MA}$.

Our last result gives a connection between Low-Degree Testing (i.e., testing whether a given function is sufficiently close to some low-degree polynomial) and the problem of showing implications such as "$\mathsf{BPP} = \mathsf{P} \Rightarrow \mathsf{NEXP} \not\subset \mathsf{P/poly}$". We prove that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$, provided that both $\mathsf{BPP} = \mathsf{P}$ and Low-Degree Testing can be done deterministically in polynomial time; note that Low-Degree Testing can be placed in deterministic polynomial time, assuming the existence of a pseudorandom generator from $O(\log n)$ to n bits.

Before describing our techniques, we would like to say a few words about the assumption that NEXP contains a function of superpolynomial Boolean or arithmetic circuit complexity. Proving such lower bounds for NEXP is certainly a difficult open problem. On the other hand, $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ is technically a weaker statement than $\mathsf{NP} \not\subset \mathsf{P/poly}$, and so may be easier to prove. Moreover, a superpolynomial circuit lower bound is already known for the probabilistic version of NEXP, the class $\mathsf{MA\text{-}EXP}$ (Buhrman *et al.* 1998), and we prove in our paper that the smaller class $\mathsf{NEXP}^{\mathsf{RP}} \subseteq \mathsf{MA\text{-}EXP}$ contains a function hard for Boolean or arithmetic circuits. So we seem to be getting closer to proving circuit lower bounds for NEXP. Being optimistic, one may even view the main result of this paper as pointing to another approach to proving such lower bounds—by designing a nondeterministic subexponential-time algorithm for Polynomial Identity Testing.

1.5. Our techniques

Circuit lower bounds from $\mathsf{RP} = \mathsf{P}$. The proof that the assumption $\mathsf{RP} = \mathsf{P}$ implies circuit lower bounds is fairly simple. The main ingredients are the implication $\mathsf{NEXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{NEXP} = \mathsf{MA}$ (Impagliazzo *et al.* 2002) as well as the downward self-reducibility of the Permanent.

We now outline our argument that shows how the assumption $\mathsf{RP} = \mathsf{P}$ implies circuit lower bounds. Later in the paper we give a formal proof that works also for the case where $\mathsf{RP} \subseteq \mathsf{coNSUBEXP}$ infinitely often. That proof will differ from the present outline in some technical details, but the main idea will be the same.

First we show that testing whether a given arithmetic circuit is a correct circuit for the Permanent of $n \times n$ integer matrices can be reduced to n polynomial identity tests for multivariate polynomials represented by arithmetic circuits. In turn, testing whether a given arithmetic circuit computes an identically zero polynomial is a coRP problem by the Schwartz–Zippel lemma. Hence, testing whether a given arithmetic circuit computes the Permanent is a coRP problem.

Thus the assumption $\mathsf{RP} = \mathsf{P}$ yields a deterministic polynomial-time algorithm to test if an arithmetic circuit computes the Permanent. Assuming that the Permanent is computable by polynomial-size arithmetic circuits, we get a nondeterministic polynomial-time algorithm for the Permanent: we simply guess a polynomial-size arithmetic circuit for the Permanent, check the correctness of our guess, and then evaluate the guessed arithmetic circuit on an input integer matrix.

Finally, if $\mathsf{NEXP} \subset \mathsf{P/poly}$, then $\mathsf{NEXP} = \mathsf{MA} = \Sigma_2^p$ (Impagliazzo *et al.* 2002) and so, by the results of Valiant (1979b) and Toda (Toda 1991), computing the Permanent is NEXP-hard. But this means that $\mathsf{NEXP} \subseteq \mathsf{NP}$, which is impossible by diagonalization (the Nondeterministic Time Hierarchy theorem).

Derandomization of Polynomial Identity Testing from circuit lower bounds. The conditional derandomization result for the Polynomial Identity Testing is proved by combining the Nisan–Wigderson generator (Nisan & Wigderson 1994) with the straight-line factorization algorithm for multivariate polynomials by Kaltofen (1989).

As in Chen & Kao (2000) and Lewin & Vadhan (1998), we consider the search problem: Given a multivariate polynomial f over a field F, find its nonzero if it exists. The points at which we evaluate f will be chosen with the help of a "hard" function, a multivariate polynomial p of high arithmetic circuit complexity, using the combinatorial designs of Nisan & Wigderson (1994). As a result, we convert the polynomial f into a new polynomial g on *significantly fewer* variables and of total degree polynomial in the total degrees of f and p. The polynomial g will have the property that $g \equiv 0$ iff $f \equiv 0$. Then we look for a non-zero of g by a "brute-force" deterministic algorithm; the saving in running time is achieved since g has few variables.

The proof of correctness of our construction involves showing that any polynomial f for which our derandomization procedure fails to produce a non-zero can be used in designing a small arithmetic circuit for the supposedly hard polynomial p. Roughly speaking, if our NW generator based on p fails for a polynomial f, then p is a *root* of a certain polynomial $\hat{f}$ derived from f.

Thus, an arithmetic circuit for p can be found by *factoring* the polynomial $\hat{f}$, using Kaltofen (1989).

We should point out that the Nisan–Wigderson generator has already been used in the setting of arithmetic circuits by Raz *et al.* (2002). Also, the proof by Sudan *et al.* (2001) of Boolean hardness-randomness tradeoffs based on Reed–Muller error-correcting codes made an essential use of efficient algorithms for factoring multivariate polynomials; an efficient procedure for polynomial factorization was one of the main ingredients in the beautiful algorithm of Sudan (1997) for list-decoding Reed–Solomon error-correcting codes. The correctness proof of our algebraic pseudorandom generator was inspired by Sudan's list-decoding algorithm. An important difference in our case is the use of Kaltofen's (1989) algorithm that takes as input an arithmetic circuit computing some multivariate polynomial and produces a list of *arithmetic* circuits computing the factors of this polynomial; this should be contrasted with the more common setting where the factors are computable by efficient algorithms which need not be purely arithmetic (i.e., may use operations other than $+$ and $*$). For our analysis, it is crucial that factors of any multivariate polynomial which is computable by a small arithmetic circuit are themselves computable by small arithmetic circuits. It is rather surprising that such a factorization is possible, and, even more so, is constructible in randomized polynomial time.

Organization of the paper. We give the necessary background in Section 2. Section 3 contains the results about testing correctness of arithmetic circuits for the Permanent. In Section 4, we derive circuit lower bounds for NEXP from the assumption that variants of Polynomial Identity Testing can be derandomized. In Section 5, we obtain an unconditional circuit lower bound for a function in $\mathsf{NEXP}^{\mathsf{RP}}$. In Section 6, we look at the problem of Low-Degree Testing (LDT), and establish some implications of the assumption that LDT can be done in deterministic polynomial time. In Section 7, we present conditional derandomization of Polynomial Identity Testing. Some concluding remarks are contained in Section 8.

2. Preliminaries

In this section we recall some basic definitions of computational complexity, define several variants of the problem of Polynomial Identity Testing to be considered later in the paper, and state the complexity results that we will use.

2.1. Complexity classes. We use the standard definitions of complexity classes RP, BPP, RNC, PH, EXP, NEXP, $\#\mathsf{P}$, and $\mathsf{P/poly}$ (Papadimitriou 1994); we define $\mathsf{SUBEXP} = \bigcap_{\epsilon>0} \mathsf{TIME}(2^{n^\epsilon})$ and $\mathsf{NSUBEXP} = \bigcap_{\epsilon>0} \mathsf{NTIME}(2^{n^\epsilon})$. The class MA (Babai 1985; Babai & Moran 1988) contains exactly those languages L that satisfy the property: there is a polynomial-time computable predicate $R(x, y, z)$ and a constant $c \in \mathbb{N}$ such that, for every $x \in \{0, 1\}^n$,

$$x \in L \Rightarrow \exists y : \mathbf{Pr}_z[R(x, y, z) = 1] \geq 2/3, \text{ and}$$
$$x \notin L \Rightarrow \forall y : \mathbf{Pr}_z[R(x, y, z) = 1] \leq 1/3,$$

where $y, z \in \{0, 1\}^{n^c}$. For a complexity class $\mathcal{C}$, its "infinitely-often" version, $\mathsf{io}\text{-}\mathcal{C}$, is defined as the set of all languages L over an alphabet Σ for which there is a language $M \in \mathcal{C}$ over Σ such that $L \cap \Sigma^n = M \cap \Sigma^n$ for infinitely many $n \in \mathbb{N}$.

Definition 2.1. The *Permanent* of an $n \times n$ matrix $A = (a_{i,j})$ of integers is defined as $\mathrm{Perm}(A) = \sum_\sigma \prod_{i=1}^n a_{i,\sigma(i)}$, where the summation is over all permutations σ of $\{1, \ldots, n\}$. Sometimes we will be interested in computing the permanent of integer matrices whose entries are 0 and 1 only. We denote this restriction of Permanent to 0-1 matrices by *0-1 Permanent.*

It is well known that the complexity of computing the permanent of 0-1 matrices is the same as that of general integer matrices: the arey both $\#\mathsf{P}$-complete (Valiant 1979b). Toda (1991) shows that every language in the polynomial-time hierarchy can be decided in polynomial time with oracle access to some $\#\mathsf{P}$-complete problem. Together these results show that computing 0-1 Permanent is PH-hard. We formally state these two theorems below.

Theorem 2.2 (Valiant 1979b). *0-1 Permanent is* $\#\mathsf{P}$*-complete.*

Theorem 2.3 (Toda 1991). $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

Since 0-1 Permanent is an integer-valued (rather than Boolean-valued) function, it is not a language, and so we cannot talk, for example, about 0-1 Permanent being in the class NP. Instead, we will consider the graph of 0-1 Permanent, which is the language defined as follows.

Definition 2.4. The *graph of 0-1 Permanent* is the set

$$\{(M, v) \mid \; M \text{ is a 0-1 matrix and } v = \mathrm{Perm}(M)\}.$$

Lemma 2.5. *If the graph of 0-1 Permanent is in* NP *(respectively,* $\mathsf{NSUBEXP}$*), then* $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{NP}$ *(respectively,* $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{NSUBEXP}$*).*

Proof. We will only prove the lemma for the case of polynomial time; the case of subexponential time is completely analogous.

Let G be the assumed nondeterministic polynomial-time Turing machine that decides the graph of 0-1 Permanent. Let $L \in \mathsf{P}^{\text{0-1Perm}}$ be any language, and let A be an oracle Turing machine deciding L. On a given input x of size n, A may ask for up to a polynomial number of queries to 0-1 Permanent. Let $M_1, \dots, M_k$, for $k \in \mathsf{poly}(n)$, denote these queries.

On input x, our nondeterministic polynomial-time Turing machine for L will nondeterministically guess the queries $M_1, \dots, M_k$ and $\mathsf{poly}(n)$-bit integers $v_1, \dots, v_k$. Then it will simulate the nondeterministic Turing machine G on each of the pairs (M_i, v_i), for $i = 1, \dots, k$. If G rejects on any of these pairs, we also reject. If G accepts on all of these pairs, we know that $\text{Perm}(M_i) = v_i$ for all $1 \leq i \leq k$, and so we deterministically simulate the machine A on x, answering the query M_i of A with the value v_i. If, for any i, the ith query asked by A is different from M_i, we reject. Otherwise, we accept iff the simulation of A on x leads to acceptance. □

Remark 2.6. We could say that 0-1 Permanent is in *functional* NP if there is a nondeterministic polynomial-time Turing machine such that, for every input 0-1 matrix M, (i) there is at least one accepting computation of the Turing machine, and (ii) every accepting computation of the Turing machine halts with the value $\text{Perm}(M)$ written on the output tape. It is easy to show that 0-1 Permanent is in functional NP iff the graph of 0-1 Permanent is in NP.

We shall also need the following complexity results.

Theorem 2.7 (Babai *et al.* 1991). $\mathsf{EXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{EXP} = \mathsf{MA}$.

Corollary 2.8. *If* $\mathsf{EXP} \subset \mathsf{P/poly}$, *then* $\mathsf{EXP} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

Proof. If $\mathsf{EXP} \subset \mathsf{P/poly}$, then $\mathsf{EXP} = \mathsf{MA}$ by Theorem 2.7. Since $\mathsf{MA} \subseteq \mathsf{PH}$, the result follows by Theorem 2.3. □

Theorem 2.9 (Impagliazzo *et al.* 2002). $\mathsf{NEXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{NEXP} = \mathsf{MA}$.

Corollary 2.10. *If* $\mathsf{NEXP} \subset \mathsf{P/poly}$, *then* $\mathsf{NEXP} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

Proof. The proof is identical to that of Corollary 2.8, except that we use Theorem 2.9 instead of Theorem 2.7. □

Theorem 2.11 (Babai *et al.* 1993). *If* $\mathsf{MA} \not\subseteq \text{io-}\mathsf{NTIME}(2^{n^\epsilon})$ *for some* $\epsilon > 0$, *then* $\mathsf{EXP} = \mathsf{MA} \subset \mathsf{P/poly}$.

2.2. Diagonalization lemmas. Here we state some useful diagonalization results.

Lemma 2.12. $\mathsf{coNEXP} \not\subseteq \text{io-}\mathsf{NTIME}(2^n)$.

Proof. The following coNEXP machine diagonalizes against all $\mathsf{NTIME}(2^n)$ machines. On an input x of length n, simulate the xth nondeterministic Turing machine M_x on x for 2^n steps, and reject iff M_x accepts; note that a co-nondeterministic Turing machine can easily "flip" the decision of the nondeterministic machine M_x. Since we can assume that each nondeterministic Turing machine has descriptions of size n for all sufficiently large $n \in \mathbb{N}$, the result follows. □

Lemma 2.13. *For any constant* c, $\mathsf{EXP} \not\subset \text{io-}\mathsf{SIZE}(n^c)$.

Proof. Consider the following exponential-time algorithm. For a given input x of size n, enumerate all circuits of size at most n^c. Let S be the set of all such circuits; note that $|S| \leq 2^{n^{c+1}}$, for all sufficiently large n.

Let $\alpha_1, \dots, \alpha_{2^n}$ be an enumeration of all n-bit strings, say in the lexicographical order. For $i = 1, \dots, n^{c+1}$, do the following. Evaluate each circuit in S on the input α_i, noting the value b_i computed by the majority of these circuits. Remove from the set S all the circuits that compute b_i on input α_i; this removes at least half of the elements of S. Continue with the next i. Once S becomes empty at some stage $i' \leq n^{c+1}$, assign the value 0 to all $b_{i'}, \dots, b_{2^n}$.

By construction, the complement of the binary string $b_1 \dots b_{2^n}$ is the truth table of an n-variable Boolean function that cannot be computed by any circuit of size at most n^c. Once our algorithm has obtained this truth table, it finds the input string x on the list of α_i's, and outputs the complement of the corresponding bit b_i. Clearly, the language decided by this algorithm is not in $\text{io-}\mathsf{SIZE}(n^c)$. □

Lemma 2.14. *Suppose* $\mathsf{NEXP} \subset \mathsf{P/poly}$. *Then* $\mathsf{EXP} \not\subseteq \text{io-}\mathsf{NTIME}(2^n)/\log^2 n$.

Proof. Let U be a universal language for the class $\mathsf{NTIME}(2^n)$, i.e., $U = \{(M, x) \mid \text{nondeterministic TM } M \text{ accepts } x \text{ in time } 2^n\}$. Clearly, $U \in \mathsf{NEXP}$. Hence, there is a fixed constant c such that $U \in \mathsf{SIZE}(n^c)$. It follows that, for every language $L \in \mathsf{NTIME}(2^n)/\log^2 n$, we have $L \in \mathsf{SIZE}(n^{c+1})$ for all sufficiently large input lengths n.

If $\mathsf{EXP} \subseteq \mathsf{io\text{-}NTIME}(2^n)/\log^2 n$, then we would get $\mathsf{EXP} \subset \mathsf{io\text{-}SIZE}(n^{c+1})$. But this contradicts Lemma 2.13. □

Lemma 2.15. *Let O be any language, and suppose $\mathsf{NEXP}^O \subset \mathsf{P/poly}$. Then $\mathsf{EXP} \not\subseteq \mathsf{io\text{-}NTIME}^O(2^n)/\log^2 n$.*

Proof. The proof is the same as that of Lemma 2.14, except that we use a universal language $U \in \mathsf{NEXP}^O$ for the class $\mathsf{NTIME}^O(2^n)$. □

2.3. Arithmetic circuits. We consider arithmetic circuits whose gates can be labeled by $+$, $-$, and $\times$; multiplication by constants is also allowed. The size of a circuit is determined by the number of its gates together with the sizes of all constants used by the circuit. An arithmetic circuit where each gate has fan-out at most one is called an arithmetic formula. These are the definitions of so-called *division-free* circuits and formulas. Later in this subsection, we shall also discuss the case of more general circuits where divisions are also allowed.

Every division-free arithmetic circuit C on n inputs computes the n-variate polynomial $p_C(x_1, \dots, x_n)$ defined inductively as follows. The ith input gate of C computes the identity polynomial x_i; a "+" gate that receives its inputs from gates u and v computes the sum of the two polynomials corresponding to u and v; etc. The polynomial p_C is the polynomial computed by the output gate of C.

Let $q(x_1, \dots, x_n)$ be an arbitrary polynomial over some integral domain I. There are two ways of defining what it means for an arithmetic circuit C to *compute* q. The first option is to say that an arithmetic circuit C computes q if q and p_C are *identical* polynomials, denoted $q \equiv p_C$. That is, if we write each polynomial as a linear combination of monomials with coefficients from I, we obtain exactly the same expression (up to the order of monomials).

The second option is to say that an arithmetic circuit C computes q if q and p_C agree at every point in I^n, i.e., $p_C(\vec{a}) = q(\vec{a})$ for all $\vec{a} \in I^n$.

We should remark that these two options are equivalent in the case of *infinite* integral domains, since any two distinct polynomials may agree on only finitely many points.

In this paper, we use both of these possible definitions. When we talk about *Polynomial Identity Testing*, we use the first definition: given an arithmetic circuit C, decide if the polynomial p_C is identically zero. When we talk about the *arithmetic circuit complexity* of a polynomial q over an integral domain I, we use the second definition: the arithmetic circuit complexity of a polynomial $q(x_1, \dots, x_n)$ over I is defined as the size of a smallest arithmetic circuit C such that p_C and q agree over I^n. This distinction means that some results that we prove for infinite fields have only weak analogues for finite fields.

The following simple lemma bounds the degree of a polynomial computed by an arithmetic circuit (or formula) of a given size.

LEMMA 2.16. *An arithmetic circuit (respectively, formula) of size s on input variables $x_1, \ldots, x_n$ computes a polynomial of total degree at most 2^s (respectively, s).*

PROOF. Every multiplication gate in an arithmetic circuit can at most double the degree of the resulting polynomial, and so the total degree is bounded by 2^s. In the case of arithmetic formulas, each multiplication gate can at most add the degrees of its two subformulas, and so the total degree of the resulting polynomial is bounded by s. □

If we allow divisions in arithmetic circuits over the field $\mathbb{Q}$ of rationals, then these circuits will compute *rational* functions of the form f/g, where f and g are polynomials over the ring $\mathbb{Z}$ of integers. An arithmetic circuit C over $\mathbb{Q}$ with divisions is said to compute an integer polynomial h if the rational function f/g computed by C satisfies the equality $f \equiv g \times h$. That is, for every integer vector $\vec{a}$ such that $g(\vec{a}) \neq 0$ (and hence C is defined on this $\vec{a}$), we have $f(\vec{a})/g(\vec{a}) = h(\vec{a})$.

Later in the paper, we will talk about arithmetic circuits computing Permanent of integer matrices. If a division-free circuit over integers computes Permanent, then obviously it also computes 0-1 Permanent. However, according to the definition just given for circuits with divisions, an arithmetic circuit with divisions over $\mathbb{Q}$ that computes Permanent may be undefined on some inputs, and in particular, it may be undefined on input 0-1 matrices. One way to deal with this problem is to use Strassen's (1973) result.

THEOREM 2.17 (Strassen 1973). *Let C be an arithmetic circuit of size s over $\mathbb{Q}$ with divisions such that C computes a degree d polynomial $f(x_1, \ldots, x_n)$. Let $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ be such that C is defined at $\vec{a}$. Then, given C and $\vec{a}$, one can construct in time $\mathsf{poly}(s, d, |\vec{a}|)$ a new circuit C' of size $\mathsf{poly}(s, d, |\vec{a}|)$ such that*

(i) C' also computes f,

(ii) the only divisions in C' are by non-zero constants (independent of the input to C').

Proof. The upper bound on the size of C' will follow immediately from the upper bound on the running time of the algorithm constructing C' from C. We describe this algorithm, following (Kaltofen 1988, Section 7).

Define the polynomial $g(z, y_1, \ldots, y_n) = f(y_1 z + a_1, \ldots, y_n z + a_n)$. Note that g is computable by the new arithmetic circuit $\tilde{C}(z, y_1, \ldots, y_n) = C(y_1 \times z + a_1, \ldots, y_n \times z + a_n)$ of size only slightly bigger than s. Also note that $\tilde{C}$ at $z = 0$ computes $C(a_1, \ldots, a_n)$, and hence is defined.

The Taylor series of g about $z = 0$ is $g(z, y_1, \ldots, y_n) = \sum_{i=0}^{d} c_i(y_1, \ldots, y_n) z^i$, where c_i's are polynomials in the variables $y_1, \ldots, y_n$. The series is truncated at d because the degree of f is d, and hence so is the degree of g in z. Thus, $f(y_1 + a_1, \ldots, y_n + a_n) = g(1, y_1, \ldots, y_n) = \sum_{i=0}^{d} c_i(y_1, \ldots, y_n)$, and we can compute $f(x_1, \ldots, x_n) = \sum_{i=0}^{d} c_i(x_1 - a_1, \ldots, x_n - a_n)$.

So we can compute f if we can compute the c_i's. The crucial step of the proof is an efficient construction of $d+1$ new arithmetic circuits $C_0, C_1, \ldots, C_d$ from the circuit $\tilde{C}$ such that each C_i computes $c_i(y_1, \ldots, y_n)$ for $0 \leq i \leq d$, and the only divisions in C_i's are by non-zero constants.

The construction proceeds by associating with each gate u of the circuit $\tilde{C}$ the $d + 1$ gates $u_0, \ldots, u_d$ of the new circuit. Each such u_i computes the coefficient at z^i in the Taylor expansion of the rational function computed by gate u. Suppose that gate u in $\tilde{C}$ receives inputs from some gates v and w. Then gates $u_0, \ldots, u_d$ will receive its inputs from the corresponding gates $v_0, \ldots, v_d$ and $w_0, \ldots, w_d$. The actual circuitry involved in computing u_i's will depend on the type of the operation computed by gate u.

If u is $+$, then $u_i = v_i + w_i$. If u is $\times$, then $u_i = \sum_{j,k\,:\,j+k=i} v_j \times w_k$ is the convolution. The most difficult case is when u is $\div$. We first invert $\sum_{i=0}^{d} w_i z^i$ modulo z^{d+1} by computing $t_0, \ldots, t_d$ such that $(\sum_{i=0}^{d} w_i z^i)^{-1} = \sum_{i=0}^{d} t_i z^i \bmod z^{d+1}$. This can be done using the Newton iteration as follows. Compute $\alpha_0 = 1/w_0$, and for $i = 1, 2, \ldots, \lceil \log(d+1) \rceil$, $\alpha_i = 2\alpha_{i-1} - \alpha_{i-1}^2 (\sum_{j=0}^{d} w_j z^j) \bmod z^{\min\{d+1, 2^i\}}$; the last computed $\alpha_{\lceil \log(d+1) \rceil}$ will equal the inverse $(\sum_{i=0}^{d} w_i z^i)^{-1} \bmod z^{d+1}$. Note that the only division incurred in this process is by w_0. But w_0 equals the value at $z = 0$ of the rational function computed by gate w in $\tilde{C}$, which is a non-zero constant by the definition of $\tilde{C}$. Once we have computed $\sum_{i=0}^{d} t_i z^i$, we compute u_i as the convolution $u_i = \sum_{j,k\,:\,j+k=i} v_j \times t_k$. □

2.4. Polynomial identity testing and self-correction. We will consider multivariate polynomials over some integral domain, e.g., the ring $\mathbb{Z}$ of integers. The *degree* of a monomial $x_1^{d_1} \ldots x_k^{d_k}$ is defined as $\sum_{i=1}^{k} d_i$; the total degree of a polynomial is defined to be the maximum degree over all its monomials.

For a polynomial $p = \sum_m c_m * m$, where the summation ranges over all monomials m of p, we define the *maximum coefficient size* of p as the maximum bit complexity over the coefficients c_m, i.e., $\lceil \log_2 \max\{|c_m|\} \rceil$.

We say that the polynomial $p(x_1, \ldots, x_n)$ is *identically 0*, denoted $p \equiv 0$, if all coefficients of p are 0. Clearly, if $p \equiv 0$, then $p(\vec{a}) = 0$ at every point $\vec{a}$. On the other hand, a polynomial may vanish over a large domain and yet not be identically zero. For example, the polynomial $\sum_{i=1}^n x_i(1 - x_i)$ vanishes at each point of the Boolean cube $\{0, 1\}^n$, but obviously is not an identically zero polynomial. However, if $p(x_1, \ldots, x_n)$ is defined over an infinite field, say $\mathbb{Q}$, and vanishes at each point of $\mathbb{Q}^n$, then $p \equiv 0$; this is because any non-zero polynomial may have only a finite number of roots. The same implication holds also for finite fields $\mathbf{F}$, provided that the degree of the given polynomial p is significantly smaller than the size of $\mathbf{F}$, and so the number of possible roots of p is smaller than the size of $\mathbf{F}^n$. The following lemma provides a more precise statement of this.

Lemma 2.18 (DeMillo & Lipton 1978; Schwartz 1980; Zippel 1979). *For an arbitrary non-zero polynomial $p(x_1, \ldots, x_n)$ of total degree d over an integral domain $\mathbb{F}$, and any finite set $S \subseteq \mathbb{F}$, we have $\mathbf{Pr}_{a \in S^n}[p(a) = 0] \leq d/|S|$.*

In general, the problem of testing whether a given polynomial vanishes over a given domain is harder than the problem of testing if a given polynomial is identically zero. This paper is concerned only with the latter, easier problem. We define the following versions of Polynomial Identity Testing Problem.

Definition 2.19. [Polynomial Identity Testing]
Arithmetic Circuit Identity Testing Problem (ACIT)
Given: An arithmetic circuit C computing a polynomial $p(x_1, \ldots, x_n)$.
Decide: Is $p \equiv 0$?

Arithmetic Formula Identity Testing Problem (AFIT)
Given: An arithmetic formula F computing a polynomial $p(x_1, \ldots, x_n)$.
Decide: Is $p \equiv 0$?

Symbolic Determinant Identity Testing Problem (SDIT)
Given: An $n \times n$ matrix A over $\mathbb{Z} \cup \{x_1, \ldots, x_n\}$.
Decide: Is the determinant $\mathrm{Det}(A) \equiv 0$?

Note that, for an $n \times n$ matrix A of indeterminates, $\mathrm{Det}(A)$ is a degree n polynomial. This polynomial is computable by a polynomial-size division-free arithmetic circuit (Strassen 1973) (see also Kaltofen (1992) for a more efficient division-free algorithm). Thus, the problem of testing whether the determinant of a given symbolic matrix is identically zero is a very natural special case of Polynomial Identity Testing.

Lemma 2.20 (Ibarra & Moran 1983). *ACIT over $\mathbb{Z}$ is in* coRP.

Proof. By Lemma 2.16, a given arithmetic circuit C computes an n-variate polynomial p of total degree at most 2^s, where $n, s \in O(|C|)$. Let $S = \{1, \ldots, 2^{s^2}\}$. We would like to test whether $C(a) = 0$ for a randomly chosen n-tuple $a \in S^n$, and accept iff the equality holds. Obviously, if $p \equiv 0$, then we would accept with probability one. On the other hand, if $p \not\equiv 0$, then, by Lemma 2.18, we would accept with probability at most $2^s/2^{s^2} < 2^{-s}$.

The only problem is that, since p can have degree 2^s, the value of p on a given n-tuple $a \in S^n$ can be as big as $2^{s^2 2^s}$, double-exponential in s; obviously, such a value cannot be computed in time $\mathsf{poly}(s)$. The way out is to use modular arithmetic: carry out the computation of $p(a)$ modulo a *random* number $m \in [2^{s^2}, 2^{s^3}]$.

Clearly, if $p(a) = 0$, then $p(a) \equiv 0 \bmod m$ for every m. To analyze the probability that $p(a) \equiv 0 \bmod m$ for a random m when $p(a) \neq 0$, we consider two cases: (i) m is composite, and (ii) m is prime. By the Prime Number Theorem, the fraction of primes in the given interval $[2^{s^2}, 2^{s^3}]$ is at least s^{-4}, and so the probability that case (i) occurs is at most $1 - s^{-4}$. On the other hand, at most 2^s of the primes in our interval can divide $p(a) \neq 0$; therefore, the probability that $p(a) = 0$ modulo a random prime from our interval is at most 2^{-s^2}. Consequently, the probability that $p(a) \equiv 0 \bmod m$ for a random $m \in [2^{s^2}, 2^{s^3}]$ when $p(a) \neq 0$ is at most $(1 - s^{-4}) + 2^{-s^2} \leq 1 - s^{-5}$.

Thus, if $p \equiv 0$, then $p(a) \equiv 0 \bmod m$ for every a and m. On the other hand, if $p \not\equiv 0$, then

$$\mathbf{Pr}_{a,m}[p(a) \equiv 0 \bmod m] \leq 2^{-s} + (1 - s^{-5}) \leq 1 - s^{-6}.$$

By repeating the calculations for s^7 independently chosen m's and accepting iff $p(a) \equiv 0$ modulo every m, this error probability can be made less than 1/2. □

By Lemma 2.18 and the well known facts that the evaluation of arithmetic formulas is in NC (Brent 1974) (see also Buss *et al.* (1992)) and that the determinant of an integer matrix is computable in NC^2 (Chistov 1985), we get the following.

Corollary 2.21. *Both AFIT and SDIT are in* coRNC.

Another important property of polynomials is their robustness, as witnessed by the following result due to Beaver, Feigenbaum, and Lipton; for completeness, we include the proof. Recall that two functions $f, g : \mathbb{F}^n \to \mathbb{F}$, over a finite field $\mathbb{F}$, are said to be *ϵ-close*, for some $\epsilon > 0$, if $f(a) = g(a)$ for all but an ϵ fraction of points $a \in \mathbb{F}^n$.

Lemma 2.22 (Beaver & Feigenbaum 1990; Lipton 1991). *For a finite field $\mathbb{F}$, let $f : \mathbb{F}^n \to \mathbb{F}$ be a function that is ϵ-close to some n-variate polynomial p of total degree d. Then there is a probabilistic $\mathsf{poly}(n, d)$-time algorithm that, given oracle access to f, computes $p(a)$ on every point $a \in \mathbb{F}^n$ with high probability, provided that $\epsilon < 1/(4(d + 1))$ and $|\mathbb{F}| > d + 1$.*

Proof. Consider the following randomized algorithm. Given an input $a \in \mathbb{F}^n$, choose a point $b \in \mathbb{F}^n$ uniformly at random, and evaluate $f(a+tb)$ for $d+1$ distinct values of $t \in \mathbb{F} \setminus \{0\}$. Using these values, interpolate the univariate polynomial $q(t)$, and return the value $q(0)$.

For the analysis, note that the point $a + tb$, for every fixed $t \in \mathbb{F} \setminus \{0\}$, is uniformly distributed in $\mathbb{F}^n$. Hence, the probability that $f(a + tb) = p(a + tb)$ for all $d + 1$ values of t is at least $1 - (d + 1)\epsilon > 3/4$. Since the restriction of p to the line $\{a + tb \mid t \in \mathbb{F}\}$ is a univariate polynomial in t of degree at most d, it follows that $q(t) \equiv p(a + tb)$ with probability greater than 3/4. Thus, with probability greater than 3/4, the described algorithm will output $q(0) = p(a)$, as required. □

Using a more complicated algorithm, one can tolerate higher values of ϵ. In particular, ϵ can be arbitrarily close to 1/2 (Gemmell & Sudan 1992).

2.5. The Nisan–Wigderson designs. Our derandomization procedure for ACIT will use a generalization of the Nisan–Wigderson generator (Nisan & Wigderson 1994) that is based on the following construction of combinatorial designs.

Lemma 2.23 (Nisan & Wigderson 1994). *For every $m, n \in \mathbb{N}$, $n < 2^m$, there exists a family of sets $S_1, \dots, S_n \subseteq \{1, \dots, l\}$ such that*

(i) $l \in O(m^2/\log n)$,

(ii) *for all* $1 \leq i \leq n$, $|S_i| = m$,

(iii) *for all* $1 \leq i < j \leq n$, $|S_i \cap S_j| \leq \log n$.

Such a family can be constructed deterministically in time $\mathsf{poly}(n, 2^l)$.

3. Testing an arithmetic circuit for Permanent

In this section, we show that testing whether a given arithmetic circuit computes Permanent can be efficiently reduced to the problem of Arithmetic Circuit Identity Testing (ACIT). This will allow us to prove that, if ACIT is in NP and if Permanent is computable by polynomial-size arithmetic circuits, then the graph of 0-1 Permanent is in NP.

3.1. Case of division-free arithmetic circuits over $\mathbb{Z}$. Let p_n be a polynomial on n^2 variables $\{x_{i,j}\}_{i,j=1}^n$ over $\mathbb{Z}$. If p_n computes Perm of $n \times n$ integer matrices, then appropriate restrictions of p_n will compute Perm on $i \times i$ integer matrices, for $1 \leq i \leq n$: we can just place an $i \times i$ matrix A in the lower right corner, assigning 1 to the diagonal variables above A and 0 to the rest of variables. Let p_i denote such a restriction of p_n to $i \times i$ matrices, for $1 \leq i \leq n$. It follows immediately from the definition of Perm that

$$p_1(x) \equiv x, \tag{3.1}$$

and, for all $1 < i \leq n$,

$$p_i(X) \equiv \sum_{j=1}^{i} x_{1,j} p_{i-1}(X_j), \tag{3.2}$$

where X is a matrix of i^2 variables $x_{k,l}$, and X_j is the jth minor of the matrix X along the first row.

Conversely, by induction on i, if arbitrary polynomials $p_1, \dots, p_n$ satisfy all the identities given by (3.1) and (3.2) above, then each p_i computes Perm of $i \times i$ matrices over $\mathbb{Z}$, for $1 \leq i \leq n$.

Given oracle access to ACIT, we can easily test in polynomial time that all equations (3.1) and (3.2) hold. This would give us a polynomial-time Turing reduction from the language

$$ACP := \{C \mid C \text{ is an arithmetic circuit computing Perm of integer matrices}\}$$

to ACIT. With just a bit of extra work, we can achieve a many-one reduction. This will be useful since NP and $\mathsf{NSUBEXP}$ are not known to be closed under Turing reductions, but they are closed under many-one reductions.

Lemma 3.3. *The language ACP defined above is polynomial-time many-one reducible to ACIT.*

Proof. Let p_n be a polynomial on n^2 variables $\{x_{i,j}\}_{i,j=1}^n$ computed by a given arithmetic circuit C. Let p_i be the restrictions of p_n to $i \times i$ matrices of variables, defined so that if $p_n \equiv \mathrm{Perm}$, then each p_i computes Perm on $i \times i$ matrices.

Note that testing equation (3.1) and equations (3.2), for each $1 < i \leq n$, is equivalent to testing whether $h_1(x) := p_1(x) - x \equiv 0$ and $h_i(X) := p_i(X) - \sum_{j=1}^{i} x_{1,j} p_{i-1}(X_j) \equiv 0$. Equivalently, we need to test whether

$$h(X^1, X^2, \ldots, X^n, y) := h_1(X^1) \times y^{n-1} + h_2(X^2) \times y^{n-2} + \cdots + h_n(X^n) \equiv 0,$$

where X^i is a set of i^2 variables, and y is a new variable.

Clearly, C computes Perm of $n \times n$ integer matrices iff $h \equiv 0$. Also note that the polynomial h is computable by an arithmetic circuit of size $\mathsf{poly}(|C|)$, since every h_i is; the size of the circuit for h is also polynomial in n, but this dependence is taken into account because the size of C must be at least the number n^2 of its inputs. □

Corollary 3.4. *If ACIT over $\mathbb{Z}$ is in* NP *(respectively,* NSUBEXP*), then the language ACP of Lemma* 3.3 *is also in* NP *(respectively,* NSUBEXP*).*

Proof. This is immediate from Lemma 3.3. □

Corollary 3.5. *Suppose Permanent over $\mathbb{Z}$ is computable by polynomial-size arithmetic circuits. If ACIT is in* NP *(respectively,* NSUBEXP*), then the graph of 0-1 Permanent is in* NP *(respectively,* NSUBEXP*).*

Proof. We only prove the NP part of the claim; the proof of the NSUBEXP part is analogous.

Let (M, v) be an input, where M is a 0-1 $n \times n$ matrix, and v is an $O(n \log n)$-bit integer. Our goal is to design a nondeterministic polynomial-time Turing machine that accepts the input (M, v) iff $\mathrm{Perm}(M) = v$.

Since Permanent over $\mathbb{Z}$ is computable by polynomial-size arithmetic circuits, we can nondeterministically guess a $\mathsf{poly}(n)$-size arithmetic circuit C computing Perm on $n \times n$ integer matrices. By Corollary 3.4, we can test in NP whether the guessed circuit C is a correct circuit for Perm. If it is wrong, we reject. If it is correct, we evaluate C on the 0-1 matrix M in deterministic polynomial time, by performing all arithmetic operations modulo $2^{n \log n + 1}$. Since $\mathrm{Perm}(M) \leq 2^{n \log n}$, the result of our evaluation actually gives us $\mathrm{Perm}(M)$. Finally, we compare this value with the input v, accepting iff they are equal. □

Finally, we state analogous results for arithmetic formulas.

Lemma 3.6. *The language*

$$AFP := \{F \mid F \text{ is an arithmetic formula computing } \mathrm{Perm} \text{ of integer matrices}\}$$

is polynomial-time many-one reducible to Arithmetic Formula Identity Testing (AFIT).

Proof. The proof is exactly the same as that of Lemma 3.3. Just observe that, if we start with an arithmetic *formula* F that supposedly computes Perm, then the polynomial h defined in the proof of Lemma 3.3 will also be computable by an arithmetic *formula* of size $\mathsf{poly}(|F|)$. □

Corollary 3.7. *If AFIT over $\mathbb{Z}$ is in NP (respectively, $\mathsf{NSUBEXP}$), then the language AFP of Lemma 3.6 is also in NP (respectively, $\mathsf{NSUBEXP}$).*

Proof. This is immediate from Lemma 3.6. □

Corollary 3.8. *Suppose Permanent over $\mathbb{Z}$ is computable by polynomial-size arithmetic formulas. If AFIT is in NP (respectively, $\mathsf{NSUBEXP}$), then the graph of 0-1 Permanent is in NP (respectively, $\mathsf{NSUBEXP}$).*

Proof. The proof is the same as that of Corollary 3.5. Simply replace "circuits" with "formulas", and "Corollary 3.4" with "Corollary 3.7". □

3.2. Case of arithmetic circuits over $\mathbb{Q}$ with divisions. Here we will argue that if ACIT over $\mathbb{Z}$ is in $\mathsf{NSUBEXP}$ and if Perm over $\mathbb{Z}$ has polynomial-size arithmetic circuits (possibly using divisions), then we still get the conclusion that the graph of 0-1 Permanent is in $\mathsf{NSUBEXP}$.

First we use Theorem 2.17 to show that if Perm has polynomial-size arithmetic circuits over $\mathbb{Q}$ using divisions, then it has only slightly bigger arithmetic circuits that are defined everywhere.

Corollary 3.9. *If there is a family of polynomial-size arithmetic circuits over $\mathbb{Q}$ with divisions that compute* Perm *of integer matrices, then there is a family of pairs of polynomial-size division-free circuits (C_1^n, C_2^n) over $\mathbb{Z}$ such that C_2^n computes an integer constant $c \neq 0$, and $C_1^n \equiv c\mathrm{Perm}$ over all $n \times n$ integer matrices.*

Proof. For any arithmetic circuit C of size s computing a rational function f/g, where f, g are polynomials over $\mathbb{Z}$, both f and g are also computable by arithmetic circuits of size $\mathsf{poly}(s)$. This is because we can associate with each gate of C a pair of new gates, one for the numerator and the other for the denominator, and then simulate C using these pairs of gates.

Hence, the denominator g has bounded degree $2^{\mathsf{poly}(s)}$ by Lemma 2.16. It follows by Lemma 2.18 that there is a tuple of integers $(a_1, \ldots, a_n)$ at which g is non-zero, and such that the bit size of each a_i is polynomial in s. We conclude that the size of a "good" point $\vec{a}$ at which the given arithmetic circuit C of size s (over $\mathbb{Q}$, with divisions) is defined is bounded by a polynomial in s.

Finally, suppose a $\mathsf{poly}(n)$-size circuit C computes Perm of $n \times n$ matrices. Since the degree of Perm is n, we obtain from C by Theorem 2.17 a new $\mathsf{poly}(n)$-size circuit C' for Perm such that the numerator of C' computes an integer polynomial $h(x_1, \ldots, x_n)$ and the denominator computes an integer constant $c \neq 0$. Both the numerator and the denominator of C' are computable by division-free arithmetic circuits over $\mathbb{Z}$ of size $\mathsf{poly}(|C'|)$. □

Now we can prove an analogue of Corollary 3.5 for the case of arithmetic circuits with divisions.

THEOREM 3.10. *Suppose Permanent is computable by polynomial-size arithmetic circuits over $\mathbb{Q}$ with divisions. If ACIT over $\mathbb{Z}$ is in* NP *(respectively,* $\mathsf{NSUBEXP}$*), then the graph of 0-1 Permanent is in* NP *(respectively,* $\mathsf{NSUBEXP}$*).*

PROOF. We prove the NP part only; the $\mathsf{NSUBEXP}$ part has an analogous proof.

We need to describe a nondeterministic polynomial-time Turing machine that, on input (M, v), where M is a 0-1 $n \times n$ matrix and v is an $O(n \log n)$-bit integer, accepts iff $\mathrm{Perm}(M) = v$.

By Corollary 3.9, we know that there exist two division-free polynomial-size circuits C_1 and C_2 over $\mathbb{Z}$ such that C_2 computes a non-zero integer constant c and $C_1 \equiv c\mathrm{Perm}$. So, we nondeterministically guess two polynomial-size division-free arithmetic circuits C_1 and C_2 over $\mathbb{Z}$, where C_1 depends on n^2 variables, and C_2 has no input variables (and so C_2 just computes some integer constant c). We want to test whether $C_1(X) \equiv c\mathrm{Perm}(X)$ for $n \times n$ integer matrices X.

As in the previous subsection, let us define the restrictions C^i, $1 \leq i \leq n$, of C_1 so that if C_1 computes cPerm over $\mathbb{Z}$, then each C^i computes cPerm of $i \times i$ integer matrices. Then equations (3.1) and (3.2) become $C^1(x) \equiv cx$, and $C^i(X) \equiv \sum_{j=1}^{i} x_{1,j} C^{i-1}(X_j)$ for $2 \leq i \leq n$. Note that all these are polynomial identity tests for polynomial-size division-free arithmetic circuits over $\mathbb{Z}$. Similarly to the proof of Lemma 3.3, we use our assumption that ACIT is in NP to conclude that testing whether $C_1(X) \equiv c\mathrm{Perm}(X)$ for $n \times n$

integer matrices X is also in NP. If $C_1(X) \not\equiv c\mathrm{Perm}(X)$, we reject; otherwise, we continue.

We nondeterministically guess a prime $2^{n^2} \leq m \leq 2^{|C_2|^2+n^2}$, testing whether the guessed number is prime by the deterministic polynomial-time algorithm of Agrawal *et al.* (2002) [1]. If m is not prime, we reject. Otherwise, we test if $c \not\equiv 0 \bmod m$; the latter test can be done efficiently by evaluating the circuit C_2 modulo m. If $c \neq 0$, such a prime m always exists since $c \leq |C_2|^{2^{|C_2|}}$. If $c \equiv 0 \bmod m$, then we reject. Otherwise, we compute $\mathrm{Perm}(M)$ by simulating the computation of $C_1(M)/c$ modulo the prime m; remember that m was chosen so that both $c \not\equiv 0 \bmod m$ and $\mathrm{Perm}(M) < m$ for every 0-1 matrix M. Finally, we compare the computed value with the input v, accepting iff they are equal. □

4. Circuit lower bounds via derandomization

Here we prove that the existence of a nondeterministic subexponential-time algorithm for Polynomial Identity Testing would yield Boolean or arithmetic circuit lower bounds for some problem in NEXP. We prove this by showing the inconsistency of the assumptions that both Polynomial Identity Testing is easy and every problem in NEXP is computable by small circuits.

4.1. If ACIT can be derandomized. In this subsection we consider what happens if Arithmetic Circuit Identity Testing (ACIT) has a nontrivial algorithm.

Theorem 4.1. *The following three assumptions cannot be simultaneously true.*

(i) $\mathsf{NEXP} \subset \mathsf{P/poly}$,

(ii) Perm *is computable by polynomial-size arithmetic circuits over* $\mathbb{Q}$ *with divisions,*

(iii) ACIT over $\mathbb{Z}$ *is in* $\mathsf{NSUBEXP}$.

Proof. Assuming that all three assumptions of the theorem hold, we will derive a contradiction. First, if assumption (i) holds, then $\mathsf{NEXP} = \mathsf{MA}$ by Theorem 2.9. Since $\mathsf{MA} \subseteq \mathsf{EXP}$, we also get $\mathsf{NEXP} = \mathsf{EXP} = \mathsf{coNEXP}$. By Corollary 2.10, we conclude that

$$\mathsf{coNEXP} \subseteq \mathsf{P}^{0\text{-}1\mathrm{Perm}} \tag{4.2}$$

[1] Here the use of a deterministic polynomial-time primality testing algorithm is not crucial. Instead we can nondeterministically guess Pratt's certificates of primality (Pratt 1975).

Assumptions (ii) and (iii) imply by Theorem 3.10 that the graph of 0-1 Permanent is in $\mathsf{NSUBEXP}$. Hence, by Lemma 2.5, we have

$$\mathsf{P}^{\text{0-1Perm}} \subseteq \mathsf{NSUBEXP}. \tag{4.3}$$

Combining inclusions (4.2) and (4.3), we conclude that $\mathsf{coNEXP} \subseteq \mathsf{NSUBEXP}$. But this contradicts Lemma 2.12. □

Remark 4.4. It is possible to prove a version of Theorem 4.1 where assumption (ii) is that Perm is computable by 2^{n^ϵ}-size arithmetic circuits for every $\epsilon > 0$, and assumption (iii) is that ACIT over $\mathbb{Z}$ is in NP.

With extra care, we also obtain the "infinitely often" version of Theorem 4.1.

Theorem 4.5. *The following three assumptions cannot be simultaneously true.*

(i) $\mathsf{NEXP} \subset \mathsf{P/poly}$,

(ii) Perm *is computable by polynomial-size arithmetic circuits over* $\mathbb{Q}$ *with divisions,*

(iii) ACIT over $\mathbb{Z}$ *is in* $\bigcap_{\epsilon>0} \text{io-}\mathsf{NTIME}(2^{n^\epsilon})$.

Proof. By modifying the proof of Theorem 3.10, we show that assumptions (ii) and (iii) imply that, for every $\epsilon > 0$, the graph of 0-1 Permanent is in $\text{io-}\mathsf{NTIME}(2^{n^\epsilon})/O(\log n)$. Indeed, for a given input size n, the output of the many-one reduction of Lemma 3.3 has size n^d for some fixed constant d. We can use an advice string of size $(d+1)\log n$ to encode the "good" input size between n^d and $(n+1)^d$ at which a given $\mathsf{NTIME}(2^{n^\epsilon})$ algorithm for ACIT is correct; if there is no good input size in this interval, then the advice string can be arbitrary. By padding up the output of our many-one reduction so that it is of the "good" size, we can construct a correct polynomial-size arithmetic circuit for 0-1 Permanent in nondeterministic subexponential time, for infinitely many input sizes, using logarithmic advice.

Next we appropriately modify the proof of Lemma 2.5 to argue that, for every $\epsilon > 0$,

$$\mathsf{P}^{\text{0-1Perm}} \subseteq \text{io-}\mathsf{NTIME}(2^{n^\epsilon})/\log^2 n. \tag{4.6}$$

Indeed, consider an arbitrary language L that is Turing-reducible to Perm of 0-1 matrices in time n^{c_L}. We use $O(\log n)$-size advice string to encode the "good" input size in the interval between n^{c_L} and $(n+1)^{c_L}$ at which our

$\mathsf{NTIME}(2^{n^\epsilon})/O(\log n)$-time algorithm for the graph of 0-1 Permanent is correct. Then we pad up to the "good" size all the queries to Perm made by the reduction from L to 0-1 Perm. By combining the advice strings of the reduction and the algorithm for the graph of 0-1 Perm, we conclude that

$$L \in \text{io-}\mathsf{NTIME}(2^{n^\epsilon})/c'_L \log n$$

for some constant c'_L dependent on L. Since $\log^2 n \geq c \log n$ for every constant c whenever n gets large, the inclusion (4.6) follows.

Finally, as in the proof of Theorem 4.1, assumption (i) yields $\mathsf{NEXP} = \mathsf{EXP} \subseteq \mathsf{P}^{\text{0-1Perm}}$. Combined with inclusion (4.6), this means that, for every $\epsilon > 0$, $\mathsf{EXP} \subseteq \text{io-}\mathsf{NTIME}(2^{n^\epsilon})/\log^2 n$. But this, combined with our assumption that $\mathsf{NEXP} \subset \mathsf{P/poly}$, contradicts Lemma 2.14. □

We end this subsection with the following variation on Theorem 4.1, where we weaken assumption (i) at the expense of strengthening assumption (iii).

Theorem 4.7. *The following three assumptions cannot be simultaneously true.*

(i) $\mathsf{NEXP} \cap \mathsf{coNEXP} \subset \mathsf{P/poly}$,

(ii) Perm *is computable by polynomial-size arithmetic circuits over* $\mathbb{Q}$ *with divisions,*

(iii) ACIT over $\mathbb{Z}$ *is in* NP.

Proof. Assumptions (ii) and (iii) imply by Theorem 3.10 that the graph of 0-1 Permanent is in NP. Hence, by Lemma 2.5, $\mathsf{P}^{\text{0-1Perm}} \subseteq \mathsf{NP}$.

Using Theorems 2.2 and 2.3, we conclude that $\mathsf{PH} = \mathsf{NP} = \mathsf{coNP}$. Thus, by padding, we have $\mathsf{NEXP} = \mathsf{coNEXP} = \mathsf{NEXP} \cap \mathsf{coNEXP}$. Appealing to Theorem 4.1 concludes the proof. □

4.2. If RP can be derandomized. Since ACIT is in coRP by Lemma 2.20, we immediately see that Theorems 4.5 and 4.7 remain true if we replace their assumptions (iii) by the assumptions $\mathsf{coRP} \subseteq \bigcap_{\epsilon>0} \text{io-}\mathsf{NTIME}(2^{n^\epsilon})$ and $\mathsf{coRP} \subseteq \mathsf{NP}$, respectively.

As a consequence, we obtain the following.

Corollary 4.8. *If* $\mathsf{BPP} = \mathsf{P}$ *or if* $\mathsf{RP} = \mathsf{ZPP}$, *then one of the following must be true.*

(i) $\mathsf{NEXP} \cap \mathsf{coNEXP} \not\subset \mathsf{P/poly}$, *or*

(ii) *Permanent is not computable by polynomial-size arithmetic circuits over $\mathbb{Q}$ with divisions.*

Proof. If $\mathsf{BPP} = \mathsf{P}$ or if $\mathsf{RP} = \mathsf{ZPP}$, then $\mathsf{coRP} \subseteq \mathsf{NP}$. Hence, ACIT is in NP, and the result follows by Theorem 4.7. □

4.3. If AFIT or SDIT can be derandomized. Here we show that the existence of nontrivial algorithms for special cases of Polynomial Identity Testing, Arithmetic Formula Identity Testing (AFIT) or Symbolic Determinant Identity Testing (SDIT) would also yield Boolean or (somewhat weaker) arithmetic circuit lower bounds for some problem in NEXP.

Theorem 4.9. *The following three assumptions cannot be simultaneously true.*

(i) $\mathsf{NEXP} \subset \mathsf{P/poly}$,

(ii) Perm *is computable by polynomial-size division-free arithmetic formulas over $\mathbb{Z}$,*

(iii) *AFIT over $\mathbb{Z}$ is in* $\bigcap_{\epsilon>0} \mathsf{io}\text{-}\mathsf{NTIME}(2^{n^{\epsilon}})$.

Proof. The proof is analogous to that of Theorem 4.5, except for using Lemma 3.6 instead of Lemma 3.3. □

We need the following result of Valiant (1979a) (see also von zur Gathen (1987)).

Theorem 4.10 (Valiant 1979a). *There is a deterministic polynomial time algorithm that, given an arithmetic formula of size s computing a polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$, outputs an $(s+2) \times (s+2)$ matrix A over $\mathbb{Z} \cup \{x_1, \ldots, x_n\}$ such that* $\mathrm{Det}(A) \equiv f$.

An immediate corollary of this result is the following.

Corollary 4.11. *AFIT is polynomial-time many-one reducible to SDIT.*

Proof. The requisite polynomial-time many-one reduction is the algorithm from Theorem 4.10. □

Now we can prove an analogue of Theorem 4.5 for Symbolic Determinant Identity Testing (SDIT).

Theorem 4.12. *The following three assumptions cannot be simultaneously true.*

(i) $\mathsf{NEXP} \subset \mathsf{P/poly}$,

(ii) Perm *is computable by polynomial-size division-free arithmetic formulas over* $\mathbb{Z}$,

(iii) SDIT over $\mathbb{Z}$ *is in* $\bigcap_{\epsilon>0} \mathsf{io\text{-}NTIME}(2^{n^\epsilon})$.

Proof. First, we observe that assumption (iii) implies by Corollary 4.11 that AFIT is in $\mathsf{io\text{-}NTIME}(2^{n^\epsilon})/O(\log n)$. Here, similarly to the proof of Theorem 4.5, we use $O(\log n)$-size advice to identify the "good" input lengths where our subexponential-time algorithm for SDIT is correct. The rest of the proof is the same as that of Theorem 4.9, except we add our $O(\log n)$-size advice to the advice introduced there. □

4.4. If RNC can be derandomized. Usually, the question whether the class RNC can be derandomized is stated as whether $\mathsf{RNC} \overset{?}{=} \mathsf{NC}$. However, it is not yet known whether an even much weaker derandomization of RNC is possible, e.g., $\mathsf{RNC} \overset{?}{\subseteq} \mathsf{SUBEXP}$. The next result shows that resolving this question would require a proof of new circuit lower bounds.

Corollary 4.13. *If* $\mathsf{coRNC} \subseteq \bigcap_{\epsilon>0} \mathsf{io\text{-}NTIME}(2^{n^\epsilon})$, *then one of the following must be true:*

(i) $\mathsf{NEXP} \not\subset \mathsf{P/poly}$, *or*

(ii) Perm *is not computable by polynomial-size division-free arithmetic formulas over* $\mathbb{Z}$.

Proof. Since AFIT is in coRNC by Corollary 2.21, the claim follows from Theorem 4.9. □

5. A hard language in $\mathsf{NEXP}^{\mathsf{RP}}$

Currently, the smallest complexity class known to contain a language of superpolynomial Boolean circuit complexity is MA-EXP, the exponential-time analogue of the class MA.

Theorem 5.1 (Buhrman *et al.* 1998). $\mathsf{MA\text{-}EXP} \not\subset \mathsf{P/poly}$.

While we cannot strengthen Theorem 5.1, we can prove that a (seemingly) smaller complexity class than $\mathsf{MA\text{-}EXP}$ contains a language of superpolynomial Boolean or arithmetic circuit complexity.

THEOREM 5.2. *At least one of the following holds:*

(i) Perm *over* $\mathbb{Z}$ *is not computable by polynomial-size arithmetic circuits, or*

(ii) $\mathsf{NEXP}^{\mathsf{RP}} \not\subset \mathsf{P/poly}$.

By a fairly straightforward argument, $\mathsf{NP}^{\mathsf{BPP}} \subseteq \mathsf{MA}$. Hence, by padding, we have $\mathsf{NEXP}^{\mathsf{BPP}} \subseteq \mathsf{MA\text{-}EXP}$, and so Theorem 5.2 shows the existence of a hard language in a smaller complexity class than $\mathsf{MA\text{-}EXP}$. As a side remark, we want to point out that $\mathsf{MA} \subseteq \mathsf{NP}^{\mathsf{promise\text{-}RP}}$ (the proof is implicit in Buhrman & Fortnow (1999)); hence, $\mathsf{MA\text{-}EXP} \subseteq \mathsf{NEXP}^{\mathsf{promise\text{-}RP}}$. This means that the main difference between the previous result and our result is that we get rid of "promise" at the expense of using the arithmetic circuit model.

For the proof of Theorem 5.2, we shall need the following.

LEMMA 5.3. *If* Perm *is computable by polynomial-size arithmetic circuits, then* $\mathsf{P}^{\#\mathsf{P}} \in \mathsf{NP}^{\mathsf{RP}}$.

PROOF. By Lemmas 2.20 and 3.3, we know that the problem of testing whether a given arithmetic circuit computes Perm of integer matrices is many-one reducible to the coRP problem ACIT. Hence, we can nondeterministically guess a small arithmetic circuit for Perm and test its correctness with access to the RP oracle. It follows that the graph of 0-1 Permanent is in $\mathsf{NP}^{\mathsf{RP}}$. Proceeding similarly to the proof of Lemma 2.5 yields the required inclusion. □

PROOF OF THEOREM 5.2. Suppose that both $\mathsf{NEXP}^{\mathsf{RP}} \subset \mathsf{P/poly}$ and Permanent is computable by polynomial-size arithmetic circuits. The first assumption implies that $\mathsf{NEXP} \subset \mathsf{P/poly}$ and hence, by Corollary 2.10, $\mathsf{NEXP} \subseteq \mathsf{P}^{\text{0-1Perm}}$. By Lemma 5.3, we get $\mathsf{NEXP} \subseteq \mathsf{NP}^{\mathsf{RP}}$. But, combined with the assumption $\mathsf{NEXP}^{\mathsf{RP}} \subset \mathsf{P/poly}$, this contradicts Lemma 2.15 (for ACIT used as the oracle language O). □

Later we shall need the following result.

THEOREM 5.4. *If* $\mathsf{EXP} \subset \mathsf{P/poly}$ *and Permanent is computable by polynomial-size arithmetic circuits, then* $\mathsf{EXP} \subseteq \mathsf{NP}^{\mathsf{RP}}$.

PROOF. If $\mathsf{EXP} \subset \mathsf{P/poly}$, then $\mathsf{EXP} \subseteq \mathsf{P}^{\text{0-1Perm}}$ by Corollary 2.8. Applying Lemma 5.3 concludes the proof. □

6. Low-degree testing and derandomization

Here we argue that the assumption $\mathsf{BPP} = \mathsf{P}$ would imply *Boolean* circuit lower bounds for NEXP, provided one could solve in deterministic polynomial time the problem of Low-Degree Testing (see Definition 6.1 below).

6.1. Testing a Boolean circuit for Permanent. In Section 3, we showed how to test in probabilistic polynomial time whether a given arithmetic circuit computes the Permanent. The main reason our probabilistic algorithm is a coRP-style algorithm is that we are dealing with an *arithmetic* circuit, and hence, we know that the given circuit computes some polynomial of total degree bounded by the size of the circuit. If testing whether a given *Boolean* circuit computes a polynomial of low degree were in P, then we would be able to adapt our arguments from Section 3 to the *Boolean* setting. We formalize this idea below.

The problem of *Low-Degree Testing* is to check whether a given function $f : \mathbb{F}^n \to \mathbb{F}$ (e.g., represented by a Boolean circuit) is close to some "low-degree" polynomial $p(x_1, \ldots, x_n)$ over a finite field $\mathbb{F}$. This problem has been extensively studied in the context of program testing and probabilistically checkable proofs (see, e.g., Rubinfeld & Sudan (1996) and the references therein). We define Low-Degree Testing as the following promise problem.

Definition 6.1. **Low-Degree Testing Problem (LDT)**
Parameters: $s, d, k, n \in \mathbb{N}$; a prime q defining the field $\mathbb{F} = \mathrm{GF}(q)$.
Positive inputs: A size s Boolean circuit computing a function $f : \mathbb{F}^n \to \mathbb{F}$ that agrees with some degree d polynomial over a finite field $\mathbb{F}$.
Negative inputs: A size s Boolean circuit computing a function $f : \mathbb{F}^n \to \mathbb{F}$ that is not $1/k$-close to any degree d polynomial over $\mathbb{F}$.

The known randomized algorithms for low-degree testing (Arora *et al.* 1998; Arora & Safra 1998; Arora & Sudan 1997; Babai *et al.* 1991; Feige *et al.* 1996; Raz & Safra 1997; Rubinfeld & Sudan 1996) imply that LDT is in $\mathsf{promise\text{-}BPP}$. That is, there is a probabilistic polynomial-time algorithm that accepts each positive input with high probability, and rejects each negative input with high probability, but it may have arbitrary acceptance probability on an input that is neither positive nor negative.

It is not known if LDT is hard for $\mathsf{promise\text{-}BPP}$. We will argue below that if LDT can be solved in deterministic polynomial time, and if $\mathsf{BPP} = \mathsf{P}$, then $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

Definition 6.2. We say that *LDT is in* promise-P if there is a deterministic polynomial-time algorithm accepting all positive inputs and rejecting all negative inputs. More precisely, there must exist a constant $c_0 \in \mathbb{N}$ and a deterministic Turing machine T such that, given $d, k \in \mathbb{N}$, a prime q, and a Boolean circuit C of size s computing a function $f : \mathbb{F}^n \to \mathbb{F}$, where $\mathbb{F} = \mathrm{GF}(q)$ is a finite field of size at least $(d+k+s)^{c_0}$, the machine T runs in time $\mathsf{poly}(s, d, k, n, \log q)$, accepting if C is a positive input, and rejecting if C is a negative input to LDT.

Let us assume that LDT is in promise-P. Consider the following algorithm Test (see Algorithm 1 below).

Input: (q, C, M, b), where $q \in \mathbb{N}$, C is a Boolean circuit with $n^2 \log q$ inputs, M is an $n \times n$ matrix of elements from $\mathbb{Z}/q\mathbb{Z}$, and $b \in \mathbb{Z}/q\mathbb{Z}$.

1. Deterministically test if q is a prime, and that $q > (n + n^2 + |C|)^{c_0}$. If either test fails, then Reject.

2. Viewing C as a circuit on input $n \times n$ matrices over $\mathrm{GF}(q)$, define C_i, $1 \le i \le n$, to be the restriction of C to $i \times i$ matrices over $\mathrm{GF}(q)$ satisfying the property: If C computes Perm of $n \times n$ matrices over $\mathrm{GF}(q)$, then each C_i computes Perm of $i \times i$ matrices over $\mathrm{GF}(q)$. For every $1 \le i \le n$, run the deterministic LDT algorithm on C_i for $\mathbb{F} = \mathrm{GF}(q)$, $d = i$, and $k = n^2$. If any C_i is rejected by the LDT algorithm, then Reject.

3. Probabilistically test that equations (3.1) and (3.2) hold for the functions f_i computed by C_i, $1 \le i \le n$, when matrix elements are chosen uniformly at random from $\mathrm{GF}(q)$. If any C_i fails the test, then Reject.

4. Apply Lemma 2.22 to C, getting a randomized circuit $\hat{C}$ that computes a degree n polynomial over $\mathrm{GF}(q)$. Probabilistically test if $\hat{C}(M) = b$. If the equality holds, then Accept; otherwise, Reject.

Algorithm 1: Test

The properties of Algorithm Test are summarized in the following lemma.

Lemma 6.3. *Suppose that LDT is in* promise-P. *Then the following conditions hold.*

(i) Algorithm Test *is a* BPP *algorithm.*

(ii) If C is a Boolean circuit correctly computing Perm *of $n \times n$ matrices over* GF(q)*, for prime q, then the tuple $(q, C, M, C(M))$ is accepted by* Test *with probability one.*

(iii) If a tuple (q, C, M, b) is accepted by Test *with probability at least* 3/4*, then* $\mathrm{Perm}(M) \equiv b \bmod q$.

Proof. Clearly, the running time of Test is polynomial in $q, n, |C|$. We need to argue that every input to Test is accepted or rejected with high probability.

Observe that Steps 1 and 2 of Test are deterministic. If an input is not rejected after these first two steps, then we know that C_i's compute functions f_i that are $1/n^2$-close to some (uniquely determined) degree i polynomials p_i over the finite field GF(q). At this point there are three possibilities:

I. for some $1 \leq i \leq n$, $p_i \not\equiv$ Perm of $i \times i$ matrices over GF(q),

II. all p_i's compute Perm, and $\mathrm{Perm}(M) \equiv b \bmod q$, and

III. all p_i's compute Perm, and $\mathrm{Perm}(M) \not\equiv b \bmod q$.

Below we shall argue that in cases I and III, Test rejects with probability close to one, and in case II, Test accepts with probability close to one.

In Step 3, n polynomial identity tests are run on f_i's. Since each f_i is $1/n^2$-close to a polynomial p_i, we can assume, with probability at least $1 - 1/n$, that all these tests are run on the polynomials p_i. If at least one of p_i's is different from Perm, then this will be detected with probability at least $1 - 1/n$. Hence, if all p_i's compute Perm, then Step 3 will pass with probability at least $1 - 1/n$; if, on the other hand, at least one of the p_i's is not equal to Perm, then Step 3 will fail with probability at least $(1 - 1/n)^2 \geq 1 - 2/n$.

If Step 3 did not fail with high probability, then the circuit $\hat{C}$ correctly computes $\mathrm{Perm}(M)$ with probability close to one. Thus, Step 4 decides if $\mathrm{Perm}(M) \equiv b \bmod q$ correctly with probability close to one.

Thus, claim (i) of the lemma is proved. Claim (ii) is obvious. Claim (iii) follows since Test accepts with high probability only in case II. □

By analogy with Corollary 3.5, we obtain the following.

Corollary 6.4. *Suppose that the Permanent of integer $n \times n$ matrices with n-bit entries is computable by polynomial-size* Boolean *circuits. If both* BPP $\subseteq$ NSUBEXP *and LDT is in* promise-P*, then the graph of 0-1 Permanent is in* NSUBEXP.

PROOF. Observe that our assumptions imply that the language $L(\textsc{Test})$ of the BPP algorithm $\textsc{Test}$ of Lemma 6.3 is in $\mathsf{NSUBEXP}$. The following is a nondeterministic subexponential-time algorithm for computing the graph of 0-1 Permanent.

On input (M, b), where M is a 0-1 $n \times n$ matrix, and b is an $O(n \log n)$-bit integer, we nondeterministically guess a polynomial-size Boolean circuit C for Perm of $n \times n$ matrices with n-bit integer entries. Let C^i be the Boolean circuit computing the value of C modulo the ith prime q_i in the interval $[|C|^5, |C|^6]$, for $1 \leq i \leq n^2$; the Prime Number Theorem guarantees that there are sufficiently many primes in the chosen interval. Note that, if C is indeed a correct circuit computing Perm over integers, then each C^i is a correct circuit computing Perm modulo q_i.

For $1 \leq i \leq n^2$, set $M_i \equiv M \bmod q_i$, and $b_i \equiv b \bmod q_i$. Then nondeterministically check in subexponential time whether $(q_i, C^i, M_i, b_i) \in L(\textsc{Test})$. If these tests pass for every i, then accept; otherwise, reject.

For correctness, we see from Lemma 6.3(ii) that all tests $(q_i, C^i, M_i, b_i) \in L(\textsc{Test})$ will pass for the correct circuit C computing Perm and the value $b = \mathrm{Perm}(M)$. On the other hand, by Lemma 6.3(iii), if these tests pass, then $b_i \equiv \mathrm{Perm}(M) \bmod q_i$ for each i, and so, by the Chinese Remainder Theorem, $b = \mathrm{Perm}(M)$. □

6.2. Derandomization of MA. In general, it is not known whether the assumption that $\mathsf{BPP} = \mathsf{P}$ should imply any derandomization of the class MA. However, under the additional assumption that the LDT is easy, we get the following.

THEOREM 6.5. *Suppose that LDT is in* $\mathsf{promise\text{-}P}$. *If* $\mathsf{BPP} \subseteq \mathsf{NSUBEXP}$*, then* $\mathsf{MA} \subseteq \bigcap_{\epsilon>0} \mathsf{io\text{-}NTIME}(2^{n^\epsilon})$.

PROOF. Suppose, for the sake of contradiction, that for some $\epsilon > 0$,

$$\mathsf{MA} \not\subseteq \mathsf{io\text{-}NTIME}(2^{n^\epsilon}). \tag{6.6}$$

By Theorem 2.11, we then have $\mathsf{EXP} = \mathsf{MA} \subset \mathsf{P/poly}$. Hence, $\mathsf{EXP} \subseteq \mathsf{P}^{\text{0-1Perm}}$ by Corollary 2.8.

Observe that, given an $n \times n$ matrix M with n-bit integer entries, and a value i, $1 \leq i \leq n^3$, one can compute the ith bit of $\mathrm{Perm}(M)$ in time exponential in n. The inclusion $\mathsf{EXP} \subset \mathsf{P/poly}$ then implies that the Permanent of integer $n \times n$ matrices with n-bit entries is computable by polynomial-size Boolean circuits. It now follows by Corollary 6.4 that the graph of 0-1 Permanent is in $\mathsf{NSUBEXP}$. Hence, by Lemma 2.5, we get $\mathsf{EXP} = \mathsf{MA} \subseteq \mathsf{NSUBEXP}$, which contradicts (6.6). □

REMARK 6.7. It is possible to prove a version of Theorem 6.5 with the class MA being replaced by $\mathsf{promise\text{-}BPP}$ (or the class APP introduced in Kabanets *et al.* (2000)). That is, the assumptions of Theorem 6.5 imply that $\mathsf{promise\text{-}BPP}$ can be computed in nondeterministic subexponential time, infinitely often. It is an interesting open question whether one can achieve a *deterministic* subexponential-time simulation of $\mathsf{promise\text{-}BPP}$ under the assumptions that both $\mathsf{BPP} = \mathsf{P}$ and LDT is in $\mathsf{promise\text{-}P}$.

6.3. Circuit lower bounds. As a corollary of Theorem 6.5, we obtain the following result: if both LDT and BPP can be derandomized, then NEXP does not have polynomial-size Boolean circuits.

THEOREM 6.8. *Suppose that LDT is in* $\mathsf{promise\text{-}P}$. *If* $\mathsf{BPP} \subseteq \mathsf{NSUBEXP}$, *then* $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

PROOF. By Theorem 6.5, we get $\mathsf{MA} \subseteq \mathsf{io\text{-}NTIME}(2^n)$.

This implies that $\mathsf{NEXP} \neq \mathsf{MA}$. Indeed, if $\mathsf{NEXP} = \mathsf{MA}$, then $\mathsf{NEXP} = \mathsf{EXP} = \mathsf{coNEXP}$, but we know by Lemma 2.12 that $\mathsf{coNEXP} \not\subseteq \mathsf{io\text{-}NTIME}(2^n)$.

Finally, since $\mathsf{NEXP} \neq \mathsf{MA}$, the result follows by Theorem 2.9. □

Theorem 6.8 strengthens one of the results in Impagliazzo *et al.* (2002) saying that if $\mathsf{promise\text{-}BPP}$ can be derandomized, then $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$. This is because the assumption that $\mathsf{promise\text{-}BPP}$ can be done in deterministic polynomial time implies that both $\mathsf{BPP} = \mathsf{P}$ and LDT is in $\mathsf{promise\text{-}P}$.

7. Conditional derandomization of polynomial identity tests

Here we explain how arithmetic circuit lower bounds imply derandomization of certain versions of Polynomial Identity Testing.

7.1. Finding roots of multivariate polynomials. Our derandomization procedure will use the existence of an efficient algorithm for the following problem of finding roots of multivariate polynomials over a field $\mathbb{F}$, where $\mathbb{F}$ is either a finite field $\mathrm{GF}(q^t)$ of prime characteristic q, or the field $\mathbb{Q}$ of rationals; we assume that for $\mathrm{GF}(q^t)$ we are given a prime q and an irreducible degree t polynomial over $\mathrm{GF}(q)$.

Definition 7.1. **Root Finding Problem**
Given: An arithmetic circuit computing a non-zero polynomial $g(x_1, \ldots, x_n, y)$ of total degree d over a field $\mathbb{F}$.
Find: A list of arithmetic circuits such that, for every polynomial $p(x_1, \ldots, x_n)$ satisfying the identity

$$g(x_1, \ldots, x_n, p(x_1, \ldots, x_n)) \equiv 0,$$

there is a circuit on the list that agrees with p over $\mathbb{F}^n$.

Recall that the *maximum coefficient size* of a polynomial is the maximum bit complexity of its coefficients (when the polynomial is written as a linear combination of monomials). The following result of Kaltofen will be essential for us.

Theorem 7.2 (Kaltofen 1989). *There is a probabilistic polynomial-time algorithm that, given an arithmetic circuit of size s computing a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ of total degree at most d, with probability at least $3/4$ outputs the numbers $e_i \geq 1$ and irreducible polynomials $h_i \in \mathbb{F}[x_1, \ldots, x_n]$, $1 \leq i \leq r$, given by arithmetic circuits of size $\mathsf{poly}(s, d, \log|\mathbb{F}|)$ such that*

$$f = \prod_{i=1}^{r} h_i^{e_i}.$$

In case the characteristic q of $\mathbb{F}$ divides any e_i, i.e., $e_i = q^{k_i} e_i'$ with e_i' not divisible by q, the algorithm returns e_i' instead of e_i, and the corresponding arithmetic circuit computes $h_i^{q^{k_i}}$ instead of h_i. For $\mathbb{F} = \mathbb{Q}$, the sizes of produced arithmetic circuits are $\mathsf{poly}(s, d, a)$, where a is the maximum coefficient size of f.

We also use the following basic fact.

Lemma 7.3 (Gauss). *For a field $\mathbb{F}$, let $f(x_1, \ldots, x_n, y) \in \mathbb{F}[x_1, \ldots, x_n, y]$ and $p(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ be any polynomials such that*

$$f(x_1, \ldots, x_n, p(x_1, \ldots, x_n)) \equiv 0.$$

Then $y - p(x_1, \ldots, x_n)$ is an irreducible factor of $f(x_1, \ldots, x_n, y)$ in the ring $\mathbb{F}[x_1, \ldots, x_n, y]$.

Now we can prove that the Root Finding Problem is efficiently solvable.

Corollary 7.4. *The Root Finding Problem for a polynomial $g \in \mathbb{F}[x_1, \ldots, x_n, y]$ of total degree d computable by an arithmetic circuit of size s can be solved probabilistically in time $\mathsf{poly}(s, d, \log|\mathbb{F}|)$; for $\mathbb{F} = \mathbb{Q}$, the running time is $\mathsf{poly}(s, d, a)$, where a is the maximum coefficient size of g.*

Proof. For the case of $\mathbb{F} = \mathbb{Q}$, this follows immediately from Lemma 7.3 and Theorem 7.2. Indeed, with high probability, the algorithm of Theorem 7.2 will produce a small arithmetic circuit computing $y - p(x_1, \ldots, x_n)$. Substituting 0 for y, and multiplying by -1, we obtain an arithmetic circuit for $p(x_1, \ldots, x_n)$.

For the case of a finite field $\mathbb{F} = \mathrm{GF}(q^t)$ for some prime q, the same reasoning as above gives us a small arithmetic circuit for $p(x_1, \ldots, x_n)$, provided that the multiplicity e of the linear factor $y - p(x_1, \ldots, x_n)$ of g is not divisible by the characteristic q of the field $\mathbb{F}$. If q does divide e, i.e., if $e = q^k e'$ where e' is not divisible by q, we will get an arithmetic circuit computing $\hat{p}(x_1, \ldots, x_n) = p(x_1, \ldots, x_n)^{q^k}$. By raising the function $\hat{p}$ to the power q^{t-k}, we obtain an arithmetic circuit computing a function that agrees with the polynomial p over $\mathbb{F}^n$. Since we do not know k, we output a circuit for every $0 \leq k < t$. The size of the new arithmetic circuit increases by at most $\mathsf{polylog}(|\mathbb{F}|)$. □

Remark 7.5. In the case of a finite field $\mathbb{F} = \mathrm{GF}(q^t)$ of characteristic q, the proof of Corollary 7.4 does not guarantee the existence of a small arithmetic circuit that is *identically equal* to the requisite n-variate polynomial p. Instead, we are guaranteed to have a small arithmetic circuit computing a function that *coincides* with the polynomial p at all tuples from $\mathbb{F}^n$. For precisely this reason, we defined the arithmetic circuit complexity of a polynomial p over a finite field $\mathbb{F}$ as the size of a smallest arithmetic circuit that agrees with p over $\mathbb{F}^n$.

7.2. Generalized NW generator. First, we define a generalization of the NW generator to arbitrary fields. It is given oracle access to a supposedly hard polynomial p, and will be denoted NW^p. The algorithm for NW^p is described below (see Algorithm 2). Such a generalization of the NW generator was used earlier by Raz *et al.* (2002) in the context of randomness extractors.

```
Parameters: $l, m, n \in \mathbb{N}$.
Input: $a = (a_1, \ldots, a_l) \in \mathbb{F}^l$, where $\mathbb{F}$ is a field.
Oracle access: $p(x_1, \ldots, x_m) \in \mathbb{F}[x_1, \ldots, x_m]$.

    1. Construct the set system $S_1, \ldots, S_n$ as given in Lemma 2.23.
    2. Output $(p(a|_{S_1}), \ldots, p(a|_{S_n})) \in \mathbb{F}^n$, where $a|_S$ denotes the tuple of the
       elements of $a$ indexed by the set $S$.
```

Algorithm 2: Generator $\mathbf{NW}^p$

Given an n-variate polynomial f of total degree d_f over a field $\mathbb{F}$, we will search for non-zeros of f among the outputs of the NW generator. Let NW^p be the NW generator based on an m-variate polynomial p of total degree d_p, where $m < n$. For l given by Lemma 2.23, we enumerate all l-tuples $a = (a_1, \dots, a_l)$, where each $a_i \in S \subseteq \mathbb{F}$ for a subset S of the field $\mathbb{F}$ with $|S| > d_f d_p$, and check whether $f(\mathrm{NW}^p(a)) \neq 0$. The running time of this procedure is at most $\approx 2^{l \log |S|}$, not counting the time of oracle calls to p.

Suppose that the polynomial $f \not\equiv 0$, but we have not found any non-zero of f among the outputs of the NW generator based on the polynomial p. We shall argue that the arithmetic circuit complexity of p over $\mathbb{F}$ is "small".

Lemma 7.6. *Let $f \in \mathbb{F}[y_1, \dots, y_n]$ and $p \in \mathbb{F}[x_1, \dots, x_m]$ be any non-zero polynomials of total degrees d_f and d_p, respectively, where $|\mathbb{F}| > d_f d_p$. Let f be computable by an arithmetic circuit of size s, let $S \subseteq \mathbb{F}$ be any set of size $|S| > d_f d_p$, and let $l \in \mathbb{N}$ be given by Lemma 2.23. Suppose that $f(\mathrm{NW}^p(a)) = 0$ for all $a \in S^l$.*

Then the arithmetic circuit complexity of the polynomial p over $\mathbb{F}$ is at most $\mathsf{poly}(m, n, d_f, d_p, s, \log |\mathbb{F}|, M)$, where $M \leq (d_p + 1)^{\log n}$; when p is a multilinear polynomial, we have $M \leq n$. For the case $\mathbb{F} = \mathbb{Q}$, the circuit complexity of p is at most $\mathsf{poly}(m, n, d_f, d_p, s, a, M)$, where a is the maximum size of coefficients in f and p.

Proof. Our proof is in two parts. First, by a "hybrid" argument, we obtain from f a non-zero polynomial $g(x_1, \dots, x_m, y)$ such that $p(x_1, \dots, x_m)$ is a y-root of this polynomial g. Then we appeal to Corollary 7.4 to conclude that p is computable by an arithmetic circuit with required parameters.

I. Hybrid argument. We define the following collection of polynomials:

- ○ $g_0(x_1, \dots, x_l, y_1, \dots, y_n) = f(y_1, \dots, y_n)$,
- ○ for $1 \leq i \leq n$, $g_i(x_1, \dots, x_l, y_{i+1}, \dots, y_n)$ is obtained from f by replacing the first i variables in f by the polynomials $p((x_1, \dots, x_l)|_{S_j})$ for $1 \leq j \leq i$.

Note that $g_n(x_1, \dots, x_l) = f(\mathrm{NW}^p(x_1, \dots, x_l))$, and so the total degree of g_n is at most $D = d_f d_p$. Since g_n vanishes on S^l where $|S| > D$, by Lemma 2.18, that $g_n \equiv 0$.

If $g_0 \not\equiv 0$, but $g_n \equiv 0$, then there must be a smallest $0 \leq i \leq n$ such that $g_i \not\equiv 0$ but $g_{i+1} \equiv 0$. Since $g_i(x_1, \dots, x_l, y_{i+1}, \dots, y_n) \not\equiv 0$, we can fix the variables $y_{i+2}, \dots, y_n$ as well as the variables x_j for $j \notin S_{i+1}$ to some field elements from the set $S \subseteq F$ so that the restricted polynomial $\bar{g}_i(x_{j_1}, \dots, x_{j_m}, y_{i+1})$ remains a non-zero polynomial. For notational convenience, let us denote this new polynomial by $g(x_1, \dots, x_m, y)$.

II. Factoring. Thus, we know that g is a non-zero polynomial, but $g(x_1, \ldots, x_m, p(x_1, \ldots, x_m)) \equiv 0$. By Corollary 7.4, the polynomial p has circuit complexity at most polynomial in the degree of g, the size of an arithmetic circuit computing g, and either $\log |\mathbb{F}|$, for a finite field $\mathbb{F}$, or the maximum size of a coefficient of g, for $\mathbb{F} = \mathbb{Q}$.

The degree of g is at most D. The arithmetic circuit for g can be obtained from that of f together with at most n circuits computing the restrictions of p, where each restriction is a polynomial of degree at most d_p on at most $\log n$ variables (by Lemma 2.23(iii)). Every such polynomial contains at most $M = (d_p + 1)^{\log n}$ distinct monomials, and so can be computed by an arithmetic circuit of size $\mathsf{poly}(M)$. In the case where p is a multilinear polynomial, its restrictions to $\log n$ variables will have at most $2^{\log n} = n$ distinct monomials. Thus, the size of an arithmetic circuit computing g is at most $s + n\mathsf{poly}(M)$, and the conclusion of the lemma follows. □

7.3. Algebraic hardness-randomness tradeoffs. We give our conditional derandomization result for Arithmetic Circuit Identity Testing (ACIT) where the given arithmetic circuit C computes an n-variate polynomial of total degree $\mathsf{poly}(n)$. Clearly, this condition is satisfied in the case of polynomial-size arithmetic formulas by Lemma 2.16. The theorem below is stated for the case of polynomials both over $\mathbb{Z}$ and over a finite field $\mathbb{F}$. In the case of a finite field $\mathbb{F} = \mathrm{GF}(q^t)$, we assume to be given a prime q and an irreducible degree t polynomial over $\mathrm{GF}(q)$ as part of the input.

Theorem 7.7. *Let $p = \{p_m\}_{m \geq 0}$ be a family of exponential-time computable multilinear non-zero m-variate polynomials over $\mathbb{Z}$ (respectively, finite field $\mathbb{F}$). Suppose that the arithmetic circuit complexity of p over $\mathbb{Q}$ (respectively, $\mathbb{F}$) is $s_p(m)$ for some function $s_p : \mathbb{N} \to \mathbb{N}$, and that the maximum coefficient size of p_m over $\mathbb{Z}$ is at most $\mathsf{poly}(m)$.*

Let C be a $\mathsf{poly}(n)$-size arithmetic circuit over $\mathbb{Z}$ (respectively, $\mathbb{F}$) computing an n-variate polynomial f_n of total degree $d_f(n) \in \mathsf{poly}(n)$ and maximum coefficient size at most $\mathsf{poly}(n)$. Then, for all sufficiently large n, testing whether $f_n \equiv 0$ can be done deterministically in time

(i) 2^{n^ϵ} *for any $\epsilon > 0$, if $s_p(m) \in m^{\omega(1)}$;*

(ii) $2^{\mathsf{polylog}(n)}$, *if $s_p(m) \in 2^{m^{\Omega(1)}}$.*

Proof. We first give the proof for the case of the polynomial p and the circuit C over $\mathbb{Z}$. Then we explain how this proof should be modified for the case of finite fields.

(i) For $m = n^{\epsilon}$, Lemma 2.23 gives $l \leq n^{2\epsilon}$. Let S be a subset of $\mathbb{Z}$ of size at least $nd_f \in \mathsf{poly}(n)$. Then the size of the set S^l is at most $(nd_f)^l < 2^{n^{3\epsilon}}$. If p is computable on an input of size w in time 2^{w^c}, for some fixed constant c, then running the generator NW^p on the set S^l takes time $|S^l|2^{n^{3c\epsilon}} < 2^{n^{4c\epsilon}}$. We enumerate all elements of S^l, compute the output $\vec{r}$ of NW^p on each, and then evaluate the circuit C at $\vec{r}$. We output "f_n is non-zero" iff $C(\vec{r}) \neq 0$ for some $\vec{r}$.

Note that by our assumption about the degree and the coefficient sizes of p_m, the outputs $\vec{r}$ of NW^p will have bit size at most $\mathsf{poly}(n)$. Also, by our assumption about the degree and the coefficient sizes of f_n, the values $f_n(\vec{r})$ will have bit sizes bounded by $\mathsf{poly}(n)$, and hence they can be computed by simulating C on $\vec{r}$ modulo 2^{n^d}, for some sufficiently large constant d. Thus, we can test whether $f_n(\vec{r}) = 0$ in deterministic time $\mathsf{poly}(n)$, for every output $\vec{r}$ of the generator.

Since ϵ can be arbitrarily small, this leads to a subexponential-time deterministic algorithm for testing whether $f_n \equiv 0$. By Lemma 7.6, this testing algorithm must succeed for f_n, since otherwise the arithmetic circuit complexity of p would be polynomial.

(ii) We can choose $m = (\log n)^d$, for some sufficiently large d to be specified later. This choice yields the NW generator on $l \leq (\log n)^{2d}$ variables, where, as above, each variable assumes values in a set $S \subseteq F$ of size at least $nd_f \in \mathsf{poly}(n)$. Thus, running the NW generator takes time $2^{(\log n)^{c'd}}$, for some constant c'. If this generator fails for f, then, by Lemma 7.6, p_m is computable by an arithmetic circuit of size n^k, for some constant k independent of d. Since d can be arbitrarily large, the failure of the NW generator on f would imply that p_m is computable by arithmetic circuits of size $2^{m^{\epsilon}}$ for any $\epsilon > 0$, contrary to our assumption on the hardness of p.

Now we show how to deal with finite fields. The only difference in this case is that the field $\mathbb{F}$ may be too small, and so we cannot choose a polynomial-size subset S of $\mathbb{F}$. However, since we test whether the polynomial f_n computed by C is identically zero, we may consider f_n over a sufficiently large extension field of $\mathbb{F}$. Since we only need an $O(\log n)$-degree extension of $\mathbb{F}$, we can find such an extension in deterministic time $\mathsf{poly}(n)$ by exhaustive search. Once we have found such an extension field $\mathbb{F}'$, we can evaluate f_n on elements from $\mathbb{F}'$ by performing the operations of the circuit C over $\mathbb{F}'$. Finally, we take a polynomial-size subset $S \subseteq \mathbb{F}'$, and proceed as in the case of polynomials over $\mathbb{Z}$ considered above. □

Remark 7.8. Recall that an arithmetic formula F of $\mathsf{poly}(n)$ size computes a polynomial f of $\mathsf{poly}(n)$ degree, by Lemma 2.16. The coefficient sizes of f are also bounded by $\mathsf{poly}(n)$, by an argument similar to the proof of Lemma 2.16. Hence, by Theorem 7.7, arithmetic circuit lower bounds imply derandomization of Arithmetic Formula Identity Testing (AFIT).

Remark 7.9. Theorem 7.7 is stated for the assumption of almost everywhere high circuit complexity of a given polynomial p. The infinitely often versions of the tradeoffs also hold. That is, if p has high arithmetic circuit complexity *infinitely often*, then the polynomial identity testing problem is easy also *infinitely often*. Also, as one might suspect, a "uniform" version of Theorem 7.7 can be proved, along the lines of Impagliazzo & Wigderson (2001); the details are omitted.

We should point out that, unlike in the Boolean circuit complexity case (cf. Impagliazzo & Wigderson (1997)), the assumption of $2^{\Omega(m)}$ arithmetic circuit complexity for polynomials p_m does not seem to imply a polynomial-time derandomization procedure. The reason is that even though we can get an NW generator from $O(\log n)$ to n variables, each variable assumes values from some set of size $\mathsf{poly}(n)$, and we need to enumerate all $O(\log n)$-tuples of field elements of bit-size $O(\log n)$ each. Thus, we still get only quasipolynomial-time algorithm in this case.

We conclude this subsection by stating a weak converse of Theorem 4.1. Recall that Theorem 4.1 states that, if ACIT is in $\mathsf{NSUBEXP}$ infinitely often, then either $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ or Permanent does not have polynomial-size arithmetic circuits. We show the following.

Theorem 7.10. *If either $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ or Permanent is not computable by polynomial-size arithmetic circuits over $\mathbb{Q}$, then ACIT for size n circuits computing polynomials of degree and the maximum coefficient size at most $\mathsf{poly}(n)$ is in $\text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$ for every $\epsilon > 0$.*

Before giving the proof of Theorem 7.10, we would like to explain why it is only a *weak* converse of Theorem 4.1. First, we can test for zero only those arithmetic circuits C that compute polynomials of degree *at most* $\mathsf{poly}(|C|)$, whereas in general, a circuit C may compute a polynomial of degree $2^{|C|}$. Secondly, our derandomization (of this restricted version of ACIT) also needs *advice*, albeit the size of that advice is sublinear.

The proof of Theorem 7.10 will use the following result implicit in Impagliazzo *et al.* (2002).

Lemma 7.11 (Impagliazzo *et al.* 2002). *If $\mathsf{NEXP} \not\subset \mathsf{P/poly}$, then, for every $\epsilon > 0$, $\mathsf{coRP} \subseteq \text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$.*

Proof of Theorem 7.10. If Perm is not computable by polynomial-size arithmetic circuits, then, for every $\epsilon > 0$, our restricted version of ACIT is in $\mathsf{io\text{-}TIME}(2^{n^\epsilon})$ by Theorem 7.7. On the other hand, if $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$, then the coRP language ACIT is in $\bigcap_{\epsilon>0} \mathsf{io\text{-}}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$ by Lemma 7.11. □

7.4. Derandomization from $\mathsf{EXP} \neq \mathsf{NP}^{\mathsf{RP}}$. Babai, Fortnow, Nisan, and Wigderson show that BPP can be derandomized if $\mathsf{EXP} \neq \mathsf{MA}$.

Theorem 7.12 (Babai *et al.* 1993). *At least one of the following holds:*

(i) $\mathsf{BPP} \subseteq \mathsf{io\text{-}TIME}(2^{n^\epsilon})$ *for every* $\epsilon > 0$, *or*

(ii) $\mathsf{EXP} = \mathsf{MA} \subset \mathsf{P/poly}$.

Here we show that a certain version of Polynomial Identity Testing can be derandomized if $\mathsf{EXP} \neq \mathsf{NP}^{\mathsf{RP}}$.

Theorem 7.13. *At least one of the following holds:*

(i) For circuits C computing degree $\mathsf{poly}(|C|)$ *polynomials, testing whether C computes an identically zero polynomial is in* $\bigcap_{\epsilon>0} \mathsf{io\text{-}TIME}(2^{n^\epsilon})$, *or*

(ii) $\mathsf{EXP} = \mathsf{NP}^{\mathsf{RP}}$.

Proof. Suppose that the restricted version of ACIT in item (i) *cannot* be derandomized. Then, by Theorem 7.7, Perm over $\mathbb{Z}$ is computable by polynomial-size arithmetic circuits. Also, since ACIT is in $\mathsf{coRP} \subseteq \mathsf{BPP}$, the lack of derandomization for ACIT means that BPP is not in $\mathsf{io\text{-}TIME}(2^{n^\epsilon})$ for some ϵ. Hence, by Theorem 7.12, we know that $\mathsf{EXP} \subset \mathsf{P/poly}$.

Thus $\mathsf{EXP} \subset \mathsf{P/poly}$ and Perm is computable by polynomial-size arithmetic circuits. The conclusion $\mathsf{EXP} = \mathsf{NP}^{\mathsf{RP}}$ follows by Theorem 5.4. □

Note the following difference between Theorems 7.12 and 7.13. Our Theorem 7.13 says that if a *particular* BPP problem cannot be derandomized, then we get a *deeper* collapse for EXP; the collapse is deeper since, as we mentioned earlier, $\mathsf{NP}^{\mathsf{BPP}} \subseteq \mathsf{MA}$. Thus, we get weaker derandomization, or a stronger collapse.

8. Conclusions

We proved the necessity of circuit lower bounds for achieving even weak derandomization of RP and BPP. Thus any general derandomization results for RP would need to be preceded by a proof of a superpolynomial circuit lower bound for some language in NEXP. This relation between derandomizing RP and proving circuit lower bounds for NEXP may explain the lack of unconditional derandomization results for RP.

It is worth pointing out that although Kabanets (2001) proved an unconditional derandomization result for RP in a certain "uniform" setting, the condition of "uniformity" makes the result in Kabanets (2001) too weak for Theorem 4.5 to be applicable.

The results in the present paper do not rule out that $\mathsf{ZPP} = \mathsf{P}$ can be proved without having to prove any circuit lower bounds first. This leaves some hope that unconditional derandomization of ZPP could be achieved.

Also, on the positive side, one can view our results as providing another approach towards establishing circuit lower bounds—through derandomization. As we have seen, finding an (even nondeterministic) subexponential-time algorithm for Polynomial Identity Testing would yield nontrivial circuit lower bounds.

We conclude with some open problems. Can our result "$\mathsf{NEXP} \subset \mathsf{P/poly}$ and Permanent is computable by polynomial-size arithmetic circuits $\Rightarrow \mathsf{BPP} \not\subseteq \bigcap_{\epsilon>0} \text{io-}\mathsf{NTIME}(2^{n^\epsilon})$" be strengthened to get "$\mathsf{BPP} = \mathsf{NEXP}$" in the conclusion? If so, then this would say that even proving that $\mathsf{NEXP} \neq \mathsf{BPP}$ is impossible without proving superpolynomial circuit lower bounds.

Does the assumption $\mathsf{BPP} = \mathsf{P}$ imply *Boolean* circuit lower bounds for NEXP? Does the assumption that LDT is in promise-P imply any circuit lower bounds for NEXP? The related question is to decide whether the assumptions that both $\mathsf{BPP} = \mathsf{P}$ and LDT is in promise-P should imply that promise-$\mathsf{BPP} \subseteq$ promise-SUBEXP.

Our final question concerns the conditional derandomization of the Polynomial Identity Testing. Assuming the existence of polynomials of high arithmetic circuit complexity, can one test whether a univariate polynomial of degree d is identically zero in deterministic time *sublinear* (e.g., polylogarithmic) in d?

Acknowledgements

We want to thank Allan Borodin for answering our questions about algebraic complexity, as well as for suggesting that we consider the problem of zero testing of symbolic determinants. We are especially thankful to Erich Kaltofen for answering our questions about his results. We would like to thank Dieter van Melkebeek for commenting on an early version of this paper. We also wish to thank Avi Wigderson for his encouragement. Finally, we thank the anonymous referees for their detailed comments and suggestions.

References

M. Agrawal & S. Biswas (2003). Primality and identity testing via Chinese remaindering. *J. ACM* **50**, 429–443 (preliminary version in FOCS'99).

M. Agrawal, N. Kayal & N. Saxena (2002). PRIMES is in P. Manuscript.

A. E. Andreev, A. E. F. Clementi & J. D. P. Rolim (1998). A new general derandomization method. *J. ACM* **45**, 179–213 (preliminary version in ICALP'96).

S. Arora, C. Lund, R. Motwani, M. Sudan & M. Szegedy (1998). Proof verification and the hardness of approximation problems. *J. ACM* **45**, 501–555 (preliminary version in FOCS'92).

S. Arora & S. Safra (1998). Probabilistic checking of proofs: A new characterization of NP. *J. ACM* **45**, 70–122 (preliminary version in FOCS'92).

S. Arora & M. Sudan (1997). Improved low-degree testing and its applications. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, 485–495.

L. Babai (1985). Trading group theory for randomness. In *Proc. 17th Annual ACM Symposium on Theory of Computing*, 421–429.

L. Babai, L. Fortnow & C. Lund (1991). Non-deterministic exponential time has two-prover interactive protocols. *Comput. Complexity* **1**, 3–40.

L. Babai, L. Fortnow, N. Nisan & A. Wigderson (1993). BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Comput. Complexity* **3**, 307–318.

L. Babai & S. Moran (1988). Arthur–Merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. System Sci.* **36**, 254–276.

D. Beaver & J. Feigenbaum (1990). Hiding instances in multioracle queries. In *Proc. 7th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Comput. Sci. 415, Springer, 37–48.

M. Blum, A. K. Chandra & M. N. Wegman (1980). Equivalence of free Boolean graphs can be tested in polynomial time. *Inform. Process. Lett.* **10**, 80–82.

M. Blum & S. Kannan (1995). Designing programs that check their work. *J. ACM* **42**, 269–291.

R. Boppana & R. Hirschfeld (1989). Pseudo-random generators and complexity classes. In *Randomness and Computation*, S. Micali (ed.), Adv. Comput. Research 5, JAI Press, Greenwich, CT, 1–26

R.P. Brent (1974). The parallel evaluation of general arithmetic expressions. *J. ACM* **21**, 201–206.

H. Buhrman & L. Fortnow (1999). One-sided versus two-sided error in probabilistic computation. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science*, C. Meinel & S. Tison (eds.), Lecture Notes in Comput. Sci. 1563, Springer, 100–109.

H. Buhrman, L. Fortnow & L. Thierauf (1998). Nonrelativizing separations. In *Proc. 13th Annual IEEE Conference on Computational Complexity*, 8–12.

S. R. Buss, S. A. Cook, A. Gupta & V. Ramachandran. (1992). An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.* **21**, 755–780.

S. Chari, P. Rohatgi & A. Srinivasan (1995). Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM J. Comput.* **24**, 1036–1050.

Z. Chen & M. Kao (2000). Reducing randomness via irrational numbers. *SIAM J. Comput.* **29**, 1247–1256 (preliminary version in STOC'97).

A. L. Chistov (1985). Fast parallel evaluation of the rank of matrices over a field of arbitrary characteristic. In *Fundamentals of Computation Theory*, Lecture Notes in Comput. Sci. 199, Springer, 63–79.

M. Clausen, A. Dress, J. Grabmeier & M. Karpinsky (1991). On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theoret. Comput. Sci.* **84**, 151–164.

R. A. DeMillo & R. J. Lipton (1978). A probabilistic remark on algebraic program testing. *Inform. Process. Lett.* **7**, 193–195.

U. Feige, S. Goldwasser, L. Lovász, S. Safra & M. Szegedy (1996). Interactive proofs and the hardness of approximating cliques. *J. ACM* **43**, 268–292 (preliminary version in FOCS'91).

J. von zur Gathen (1987). Feasible arithmetic computations: Valiant's hypothesis. *J. Symbolic Comput.* **4**, 137–172.

P. Gemmell & M. Sudan (1992). Highly resilient correctors for multivariate polynomials. *Inform. Process. Lett.* **43**, 169–174.

O. Goldreich & D. Zuckerman (1997). Another proof that BPP$\subseteq$PH (and more). *Electronic Colloq. Comput. Complexity* **TR97-045**.

D. Yu. Grigoriev, M. Karpinsky & M. F. Singer (1990). Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput.* **19**, 1059–1063.

O. H. Ibarra & S. Moran (1983). Probabilistic algorithms for deciding equivalence of straight-line programs. *J. ACM* **30**, 217–228.

R. Impagliazzo, V. Kabanets & A. Wigderson (2002). In search of an easy witness: Exponential time vs. probabilistic polynomial time. *J. Comput. System Sci.* **65**, 672–694 (preliminary version in CCC'01).

R. Impagliazzo, R. Shaltiel & A. Wigderson (1999). Near-optimal conversion of hardness into pseudo-randomness. In *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science*, 181–190.

R. Impagliazzo, R. Shaltiel & A. Wigderson (2000). Extractors and pseudo-random generators with optimal seed length. In *Proc. 32nd Annual ACM Symposium on Theory of Computing*, 1–10.

R. Impagliazzo & A. Wigderson (1997). P = BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, 220–229.

R. Impagliazzo & A. Wigderson (2001). Randomness vs time: Derandomization under a uniform assumption. *J. Comput. System Sci.* **63**, 672–688 (preliminary version in FOCS'98).

V. Kabanets (2001). Easiness assumptions and hardness tests: trading time for zero error. *J. Comput. System Sci.* **63**, 236–252 (preliminary version in CCC'00).

V. Kabanets, C. Rackoff & S. Cook (2000). Efficiently approximable real-valued functions. *Electronic Colloq. Comput. Complexity* **TR00-034**.

E. Kaltofen (1988). Greatest common divisors of polynomials given by straight-line programs. *J. ACM* **35**, 231–264.

E. Kaltofen (1989). Factorization of polynomials given by straight-line programs. In *Randomness and Computation*, S. Micali (ed.), Adv. Comput. Research 5, JAI Press, Greenwich, CT, 375–412.

E. Kaltofen (1992). On computing determinants of matrices without divisions. In *Proc. 1992 International Symposium on Symbolic and Algebraic Computation (ISSAC'92)*, P. S. Wang (ed.), 342–349.

A. Klivans & D. van Melkebeek (2002). Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.* **31**, 1501–1526 (preliminary version in STOC'99).

A. Klivans & D. Spielman (2001). Randomness efficient identity testing of multivariate polynomials. In *Proc. 33rd Annual ACM Symposium on Theory of Computing*, 216–223.

D. Lewin & S. Vadhan (1998). Checking polynomial identities over any field: Towards a derandomization? In *Proc. 30th Annual ACM Symposium on Theory of Computing*, 438–447.

N. Linial, Y. Mansour & N. Nisan (1993). Constant depth circuits, Fourier transform and learnability. *J. ACM* **40**, 607–620.

R. Lipton (1991). New directions in testing. In *Distributed Computing and Cryptography*, J. Feigenbaum & M. Merrit (eds.), DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 2, Amer. Math. Soc., 191–202.

L. Lovász (1979). On determinants, matchings and random algorithms. In *Fundamentals of Comput. Theory*, L. Budach (ed.), Akademie-Verlag, Berlin, 565–574.

C. Lund, L. Fortnow, H. Karloff & N. Nisan (1992). Algebraic methods for interactive proof systems. *J. ACM* **39**, 859–868.

K. Mulmuley, U. Vazirani & V. Vazirani (1987). Matching is as easy as matrix inversion. *Combinatorica* **7**, 105–113.

N. Nisan & A. Wigderson (1994). Hardness vs. randomness. *J. Comput. System Sci.* **49**, 149–167.

C. H. Papadimitriou (1994). *Computational Complexity.* Addison-Wesley, Reading, MA.

R. Paturi, P. Pudlák, M.E. Saks & F. Zane (1998). An improved exponential-time algorithm for k-SAT. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science*, 628–637.

R. Paturi, P. Pudlák & F. Zane (1999). Satisfiability coding lemma. *Chicago J. Theoret. Comput. Sci.*, Article 11, 19 pp. (preliminary version in FOCS'97).

R. Paturi, M. E. Saks & F. Zane (2000). Exponential lower bounds for depth three Boolean circuits. *Comput. Complexity* **9**, 1–15 (preliminary version in STOC'97).

V. R. Pratt (1975). Every prime has a succinct certificate. *SIAM J. Comput.* **4**, 214–220.

R. Raz, O. Reingold & S. P. Vadhan (2002). Extracting all the randomness and reducing the error in Trevisan's extractors. *J. Comput. System Sci.* **65**, 97–128 (preliminary version in STOC'99).

R. Raz & S. Safra (1997). A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, 475–484.

A. A. Razborov & S. Rudich (1997). Natural proofs. *J. Comput. System Sci.* **55**, 24–35.

R. M. Roth & G. M. Benedek (1991). Interpolation and approximation of sparse multivariate polynomials. *SIAM J. Comput.* **20**, 291–314.

R. Rubinfeld & M. Sudan (1996). Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.* **25**, 252–271.

J. T. Schwartz (1980). Fast probabilistic algorithms for verification of polynomial identities. *J. ACM* **27**, 701–717.

R. Shaltiel & C. Umans (2001). Simple extractors for all min-entropies and a new pseudo-random generator. In *Proc. 42nd Annual IEEE Symposium on Foundations of Computer Science*, 648–657.

A. Shamir (1992). IP = PSPACE. *J. ACM* **39**, 869–877.

V. Strassen (1973). Vermeidung von Divisionen. *J. Reine Angew. Math.* **264**, 182–202.

M. Sudan (1997). Decoding of Reed Solomon codes beyond the error-correction bound. *J. Complexity* **13**, 180–193.

M. Sudan, L. Trevisan & S. Vadhan (2001). Pseudorandom generators without the XOR lemma. *J. Comput. System Sci.* **62**, 236–266 (preliminary version in STOC'99).

S. Toda (1991). PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.* **20**, 865–877.

C. Umans (2003). Pseudo-random generators for all hardnesses. *J. Comput. System Sci.* **67**, 419–440 (preliminary version in STOC'02).

L. Valiant (1979a). Completeness classes in algebra. In *Proc. 11th Annual ACM Symposium on Theory of Computing*, 249–261.

L. Valiant (1979b). The complexity of computing the permanent. *Theoret. Comput. Sci.* **8**, 189–201.

L. Valiant & V. Vazirani (1986). NP is as easy as detecting unique solutions. *Theoret. Comput. Sci.* **47**, 85–93.

A.C. Yao (1982). Theory and applications of trapdoor functions. In *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science*, 80–91.

F. Zane (1998). *Circuits, CNFs, and Satisfiability*. Ph.D. thesis, UCSD.

R.E. Zippel (1979). Probabilistic algorithms for sparse polynomials. In *Proc. International Symposium on Symbolic and Algebraic Manipulation (EUROSAM'79)*, Lecture Notes in Comput. Sci. 72, Springer 216–226.

Manuscript received 12 December 2003

Valentine Kabanets
School of Computing Science
Simon Fraser University
Vancouver, BC V5A 1S6
Canada
kabanets@cs.sfu.ca

Russell Impagliazzo
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093-0114
U.S.A.
russell@cs.ucsd.edu