Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 55 (2002)

Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds
(Preliminary Version)

Valentine Kabanets*
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093-0114
kabanets@cs.ucsd.edu

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

February 25, 2003

Abstract

We show that derandomizing Polynomial Identity Testing is, essentially, equivalent to proving 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 formula 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, $\mathsf{coRP} \subseteq \cap_{\epsilon>0}\mathsf{NTIME}(2^{n^\epsilon})$, 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.

Our techniques allow us to prove an unconditional circuit lower bound for a language in $\mathsf{NEXP}^{\mathsf{RP}}$: we prove that either (i) Permanent is not computable by polynomial-size arithmetic circuits, or (ii) $\mathsf{NEXP}^{\mathsf{RP}} \not\subset \mathsf{P/poly}$.

Finally, we prove that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ if both $\mathsf{BPP} = \mathsf{P}$ and the low-degree testing is in P; here, the 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: derandomization, circuit lower bounds, BPP, NEXP, polynomial identity testing.

*Research supported by a Postdoctoral Fellowship from the Natural Sciences and Engineering Research Council of Canada

†Research supported by NSF Award CCR-0098197 and USA-Israel BSF Grant 97-00188

ISSN 1433-8092

Contents

1 Introduction **3**

1.1 Derandomization from circuit lower bounds . . . 3
1.2 Polynomial Identity Testing . . . 3
1.3 Extending hardness-randomness tradeoffs . . . 4
1.4 Our results . . . 5
1.5 Our techniques . . . 5
1.6 Why weren't these results obtained before? . . . 6

2 Preliminaries **7**

2.1 Complexity classes . . . 7
2.2 Arithmetic circuits . . . 8
2.3 Polynomial identity testing and self-correction . . . 8
2.4 The Nisan-Wigderson designs . . . 10

3 Testing an Arithmetic Circuit for Permanent **10**

3.1 Case of arithmetic circuits over $\mathbb{Z}$ without divisions . . . 10
3.2 Case of arithmetic circuits over $\mathbb{Q}$ with divisions . . . 12

4 Circuit Lower Bounds via Derandomization **13**

4.1 If ACIT can be derandomized . . . 13
4.2 If RP can be derandomized . . . 15
4.3 If AFIT or SDIT can be derandomized . . . 15
4.4 If RNC can be derandomized . . . 16

5 A Hard Language in $\mathsf{NEXP}^{\mathsf{RP}}$ **16**

6 Low-Degree Testing and Derandomization **17**

6.1 Testing a Boolean circuit for Permanent . . . 17
6.2 Derandomization of MA . . . 19
6.3 Circuit lower bounds . . . 20

7 Conditional Derandomization of Polynomial Identity Tests **20**

7.1 Finding roots of multivariate polynomials . . . 20
7.2 Generalized NW generator . . . 22
7.3 Algebraic hardness-randomness tradeoffs . . . 23
7.4 Derandomization from $\mathsf{EXP} \neq \mathsf{NP}^{\mathsf{RP}}$ . . . 25

8 Conclusions **26**

1 Introduction

1.1 Derandomization from circuit lower bounds

In the early 1980's, Yao [Yao82, BH89] 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 to Boolean functions by Nisan and Wigderson [NW94], and significantly strengthened in a sequence of papers [BFNW93, IW97, ACR98, STV01, ISW99, ISW00, SU01, Uma02], which replaced the assumption of high average-case circuit complexity with that of high *worst-case* circuit complexity. For instance, Impagliazzo and Wigderson [IW97] 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? It is well-known that derandomizing BPP using a pseudorandom generator (as in Yao's original approach) does indeed require proving that $\mathsf{EXP} \not\subset \mathsf{P/poly}$ [ISW99]. On the other hand, Impagliazzo, Kabanets, and Wigderson [IKW02] showed that derandomizing promise-BPP would require proving that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$; here, the derandomized algorithm for a promise-BPP problem is allowed to be nondeterministic subexponential-time.

These results may explain why no unconditional derandomization of promise-BPP has been achieved so far. However, they leave open the case of BPP. Presumably, it is possible to derandomize BPP without derandomizing promise-BPP. However, our results show that even derandomizing RP requires a circuit lower bound of some form.

1.2 Polynomial Identity Testing

Deriving general consequences from the assumption $\mathsf{BPP} = \mathsf{P}$ 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 [Sch80, Zip79, DL78], evaluating a degree d multivariate polynomial on an 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 [Lov79, MVV87, CRS95], equivalence testing of read-once branching programs [BCW80], multiset equality testing [BK95], primality testing [AB99, AKS02], a number of complexity-theoretic results on probabilistically checkable proofs [LFKN92, Sha92, BFL91, AS98, ALM$^+$98], as well as sparse multivariate polynomial interpolation [Zip79, GKS90, CDGK91, RB91].

Recently, a number of probabilistic algorithm for the polynomial identity testing were proposed that use fewer random bits than the standard Schwartz-Zippel algorithm [CK97, LV98, AB99, KS01]. In the first two papers, the identity testing problem is solved by probabilistically searching for non-zeros of a given multivariate polynomial. Chen and Kao [CK97] argue that a randomly chosen rational approximation of a certain irrational point is likely to be a non-zero of any polynomial of specified total degree; generalizing this idea, Lewin and Vadhan [LV98] consider a randomly chosen polynomial approximation of a certain infinite power series.

While the algorithms in [CK97, LV98, AB99, KS01] achieve some saving in randomness compared with the original Schwartz-Zippel algorithm, they do not imply that Polynomial Identity Testing can be derandomized in the strong sense. That is, it is still a big open problem to come up with a deterministic polynomial-time (or, even, nondeterministic subexponential-time) algorithm for Polynomial Identity Testing.

The deterministic polynomial-time algorithm for Primality Testing discovered by Agrawal, Kayal, and Saxena [AKS02] 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 and Zuckerman [GZ97] showed that the same hardness assumptions imply the derandomization of the class MA introduced by Babai [Bab85, BM88]. Klivans and van Melkebeek [KM99] 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 hashing technique [VV86].

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 esthetic. 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 should 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, [LMN93] give a learning algorithm for constant-depth circuits based on lower bounds for such circuits; [PSZ97, PPZ99, PPSZ98] 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 [Zan98] 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 [RR97] ("a natural lower bound yields a cryptanalysis tool") or hardness-randomness tradeoffs. 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.

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) $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ 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 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 formula lower bounds. Since this problem belongs to the class coRNC, we conclude that derandomizing RNC is as hard as proving formula lower bounds.

For the converse direction, we extend the known hardness-randomness tradeoffs to the algebraic-complexity setting by showing the following. The 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.

Our techniques allow us to obtain an unconditional circuit lower bound for the complexity class $\mathsf{NEXP}^{\mathsf{RP}}$: we show that either $\mathsf{NEXP}^{\mathsf{RP}} \not\subset \mathsf{P/poly}$ or Permanent is not computable by polynomial-size arithmetic circuits. 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, Fortnow, Nisan, and Wigderson [BFNW93] saying that BPP can be derandomized under the assumption that $\mathsf{EXP} \neq \mathsf{MA}$. Our assumption is weaker since, by a straightforward argument, $\mathsf{NP}^{\mathsf{RP}} \subseteq \mathsf{MA}$.

Finally, we point out the relevance of the Low-Degree Testing (i.e., testing whether a given function is sufficiently close to some low-degree polynomial) to the problem of showing implications such as "$\mathsf{BPP} = \mathsf{P} \Rightarrow \mathsf{NEXP} \not\subset \mathsf{P/poly}$". Namely, we prove that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$, provided that both $\mathsf{BPP} = \mathsf{P}$ and the Low-Degree Testing can be done deterministically in polynomial time.

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 $\mathsf{NEXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{NEXP} = \mathsf{MA}$ [IKW02] as well as the downward self-reducibility of the Permanent.

Here is a brief outline of our argument. The main idea is 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. On the other hand, 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, any nontrivial (i.e., subexponential-time) derandomization of coRP yields an algorithm for testing whether an arithmetic circuit computes the Permanent. Assuming that the Permanent is computable by polynomial-size arithmetic circuits, we get a nondeterministic subexponential-time algorithm for the Permanent: we simply guess a polynomial-size arithmetic circuit for the Permanent and check the correctness of our guess.

Finally, the assumption that $\mathsf{NEXP} \subset \mathsf{P/poly}$ yields that $\mathsf{NEXP} = \mathsf{MA}$ [IKW02] and so, by the results of Valiant [Val79b] and Toda [Tod91], we conclude that the Permanent is NEXP-complete, contradicting 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 [NW94] with the straight-line factorization algorithm for multivariate polynomials by Kaltofen [Kal89].

As in [CK97, LV98], we consider the search problem: Given a multivariate polynomial f over a field F, find its non-zero 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 and Wigderson [NW94]. 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 [Kal89].

1.6 Why weren't these results obtained before?

As the reader might gather from the outlines in the preceding subsection, the proofs of our main results are rather simple. One may wonder why it took the pseudorandomness community so long to find them. For example, all the necessary ingredients for the conditional derandomization of Polynomial Identity Testing, the Nisan-Wigderson generator [NW94] and the Kaltofen factoring algorithm [Kal89], have been known for more than a decade.

In the case of derandomizing Polynomial Identity Tests, the main reason it was not done before seems that people tried to get a "full" pseudorandom generator for the algebraic complexity setting, and were bogged down trying to come up with a good definition of such a generator.

In the case of proving circuit lower bounds from the assumption $\mathsf{BPP} = \mathsf{P}$, the stumbling block (at least for the authors of the present paper) was in focusing only on Boolean (and ignoring algebraic) circuit complexity. In turn, that was because of the lack of known algebraic-complexity hardness-randomness tradeoffs.

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

2.1 Complexity classes

We use the standard definitions of complexity classes RP, BPP, RNC, PH, EXP, NEXP, $\#\mathsf{P}$, and $\mathsf{P/poly}$ [Pap94]; we define $\mathsf{SUBEXP} = \cap_{\epsilon>0}\mathsf{TIME}(2^{n^\epsilon})$ and, similarly, $\mathsf{NSUBEXP} = \cap_{\epsilon>0}\mathsf{NTIME}(2^{n^\epsilon})$. The class MA [Bab85, BM88] 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 $x \in \{0,1\}^n$,

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

where $y, z \in \{0,1\}^{n^c}$. For a complexity class $\mathcal{C}$, its "infinitely-often" version, io-$\mathcal{C}$, 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 $n \in \mathbb{N}$.

Recall that 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 $\{1,\ldots,n\}$. We need the following results by Valiant and Toda that together show that Perm over $\mathbb{Z}$ is PH-hard.

Theorem 1 ([Val79b]). Perm *is* $\#\mathsf{P}$*-complete.*

Theorem 2 ([Tod91]). $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

We say that Perm is computable in NP if the following language is in NP:

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

The definition for other nondeterministic complexity classes is similar.

Babai, Fortnow, and Lund obtained the following.

Theorem 3 ([BFL91]). $\mathsf{EXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{EXP} = \mathsf{MA}$.

Theorem 3 was extended to the class NEXP by Impagliazzo, Kabanets, and Wigderson.

Theorem 4 ([IKW02]). $\mathsf{NEXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{NEXP} = \mathsf{MA}$.

Since $\mathsf{MA} \subseteq \mathsf{PH}$, by combining Theorems 1, 2, and 4 we obtain the following.

Corollary 5. *If* $\mathsf{NEXP} \subset \mathsf{P/poly}$*, then* Perm *is* NEXP*-complete.*

We shall also need the following result of Babai, Fortnow, Nisan, and Wigderson.

Theorem 6 ([BFNW93]). *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 Arithmetic circuits

We consider arithmetic circuits whose gates can be labeled by $+$, $-$, $\times$, and $\div$; 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.

We will consider arithmetic circuits both over the set $\mathbb{Z}$ of integers and over a (finite or infinite) field $\mathbb{F}$. When the ground (finite) field is not sufficiently large, we shall need to evaluate the circuit on an extension field. For instance, given an arithmetic circuit computing a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ over some field $\mathbb{F} = \mathrm{GF}(q)$ for a prime power q, one can efficiently evaluate the polynomial f at points from an extension field $\mathrm{GF}(q^k)$ of $\mathrm{GF}(q)$, for some $k \in \mathbb{N}$, provided that this extension field can be efficiently constructed from the ground field $\mathbb{F}$. Also, since every elementary field operation in $\mathrm{GF}(q^k)$ can be efficiently simulated using the operations from the ground field $\mathrm{GF}(q)$, an arithmetic circuit computing f over $\mathrm{GF}(q^k)$ of size s gives rise to an arithmetic circuit for f over $\mathrm{GF}(q)$ of size $\mathsf{poly}(s, k)$. In other words, the arithmetic circuit complexity of f over a ground field $\mathbb{F}$ and over a "small" extension field of $\mathbb{F}$ are polynomially related.

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

Lemma 7. *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; hence, the total degree of the resulting polynomial is bounded by s. $\square$

2.3 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.

We shall be interested in the following versions of Polynomial Identity Testing Problem.

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 division-free 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 arithmetic circuit, using Gaussian elimination; the circuit can be made division-free by [Str73] (see also [Kal92] 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.

The following result is due to Schwartz and Zippel.

Lemma 8 ([Sch80, Zip79, DL78]). *Let $p(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ be any non-zero polynomial of total degree d over an integral domain $\mathbb{F}$. Let $S \subseteq \mathbb{F}$ be any finite subset. Then*

$$\mathbf{Pr}_{a \in S^n}[p(a) = 0] \leqslant \frac{d}{|S|}.$$

Building upon [Sch80], Ibarra and Moran obtained the following result, whose proof is included for completeness.

Lemma 9 ([IM83]). *ACIT over $\mathbb{Z}$ is in* coRP.

Proof. By Lemma 7, 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 8, 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 $p(a) \equiv 0 \mod m$ for every m. To analyze the probability that $p(a) \equiv 0 \mod m$ 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 $p(a) \equiv 0 \mod m$ for a random $m \in [2^{s^2}, 2^{s^3}]$ when $p(a) \neq 0$ is at most $(1 - s^{-4}) + 2^{-s^2} \leqslant 1 - s^{-5}$.

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

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

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 $1/2$. $\square$

By Lemma 8 and the well-known fact that the determinant of an integer matrix is computable in NC^2 [Chi85], we immediately obtain the following.

Corollary 10. *SDIT is 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 11 ([BF90, Lip91]). *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. □

2.4 The Nisan-Wigderson designs

Our derandomization procedure for ACIT will use a generalization of the Nisan-Wigderson generator [NW94] that is based on the following construction of combinatorial designs.

Lemma 12 ([NW94]). *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*

1. *$l \in O(m^2/\log n)$,*
2. *for all $1 \leqslant i \leqslant n$, $|S_i| = m$, and*
3. *for all $1 \leqslant i < j \leqslant n$, $|S_i \cap S_j| \leqslant \log n$.*

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

3 Testing an Arithmetic Circuit for Permanent

3.1 Case of arithmetic circuits over $\mathbb{Z}$ without divisions

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 $1 \leqslant i \leqslant n$: 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 $1 \leqslant i \leqslant n$. It follows immediately from the definition of Perm that

$$p_1(x) \equiv x, \tag{1}$$

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

$$p_i(X) \equiv \sum_{j=1}^{i} x_{1,j} p_{i-1}(X_j), \tag{2}$$

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 (1) and (2) above, then each p_i computes Perm of $i \times i$ matrices over $\mathbb{Z}$, for $1 \leqslant i \leqslant n$.

This reasoning leads us to the following.

Lemma 13. *The language*

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

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 $p_n \equiv \text{Perm}$, then each p_i computes Perm on $i \times i$ matrices.

Testing equation (1) and equations (2), for each $1 < i \leqslant n$, is equivalent to testing whether

$$h_1(x) \overset{\text{def}}{=} p_1(x) - x \equiv 0$$

and

$$h_i(X) \overset{\text{def}}{=} p_i(X) - \sum_{j=1}^{i} x_{1,j} p_{i-1}(X_j) \equiv 0.$$

Equivalently, we need to test whether

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

where X^i is a set of i^2 variables (new for each $1 \leqslant i \leqslant n$), 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. $\square$

Corollary 14. *If ACIT over $\mathbb{Z}$ is in* NSUBEXP*, then the language ACP of Lemma 13 is also in* NSUBEXP.

Proof. This is immediate from Lemma 13. $\square$

Corollary 15. *Suppose that ACIT over $\mathbb{Z}$ is in* NSUBEXP. *If* Perm *over $\mathbb{Z}$ is computable by polynomial-size arithmetic circuits (over $\mathbb{Z}$, without divisions), then* Perm $\in$ NSUBEXP.

Proof. If Perm is computable by polynomial-size arithmetic circuits, then, for each $n \in \mathbb{N}$, we can nondeterministically guess a $\mathsf{poly}(n)$-size arithmetic circuit C computing Perm on $n \times n$ integer matrices. Since, by our assumption, testing whether C is indeed computing Perm can be done in NSUBEXP by Corollary 14, we can verify in nondeterministic subexponential time that the guessed arithmetic circuit indeed computes Perm over $\mathbb{Z}$. Once we have such a circuit, we can deterministically evaluate it at a given 0-1 $n \times n$ matrix in polynomial time, by doing all operations modulo a sufficiently large number, e.g., modulo $2^{n \log n + 1}$, since the value of Perm on a 0-1 $n \times n$ matrix is at most $2^{n \log n}$. Hence, we conclude that Perm can be computed in nondeterministic subexponential time. $\square$

By modifying the proof of Corollary 15, one can easily show the following.

Corollary 16. *Suppose that ACIT over $\mathbb{Z}$ is in* NP. *If* Perm *over $\mathbb{Z}$ is computable by polynomial-size arithmetic circuits, then* Perm $\in$ NP.

Finally, we observe that the proof of Lemma 13 immediately yields the following.

Lemma 17. *The language*

$$AFP \overset{\text{def}}{=} \{F \mid F \textit{ is an arithmetic formula computing } \text{Perm} \textit{ of integer matrices}\}$$

is polynomial-time many-one reducible to AFIT.

Thus, we obtain the following version of Corollary 15.

Corollary 18. *Suppose that AFIT over $\mathbb{Z}$ is in* NSUBEXP. *If* Perm *over $\mathbb{Z}$ is computable by polynomial-size division-free arithmetic formulas, then* Perm $\in$ NSUBEXP.

3.2 Case of arithmetic circuits over $\mathbb{Q}$ with divisions

Here we will argue that if ACIT over $\mathbb{Z}$ can be derandomized, and if Perm over $\mathbb{Q}$ has polynomial-size arithmetic circuits (possibly using divisions), then we still get the conclusion that Perm of 0-1 matrices is in nondeterministic subexponential time.

Note that, in general, a given arithmetic circuit C over $\mathbb{Q}$ with divisions computes a rational function f/g, where f and g are polynomials over $\mathbb{Z}$. The assumption that C computes Perm means that $f \equiv g\mathrm{Perm}$. This implies, for any input integer matrix M such that $g(M) \neq 0$, we have $f(M)/g(M) = \mathrm{Perm}(M)$. The problem is that C may be undefined on a particular matrix M whose permanent we want to compute.

Fortunately, we can use the following result by Strassen [Str73]; Kaltofen [Kal88, Section 7] provides an alternative proof.

Theorem 19 ([Str73]). *Given an arithmetic circuit C (over $\mathbb{Q}$ with divisions) of size s computing a degree d polynomial $f(x_1, \dots, x_n)$, and a point $\vec{a} = (a_1, \dots, a_n) \in \mathbb{Z}^n$ such that C is defined at $\vec{a}$, one can construct, in time $\mathsf{poly}(s, d, |\vec{a}|)$, a new circuit C' such that*

1. *C' also computes f,*
2. *the only divisions in C' are by constants (independent of the input to C').*

Corollary 20. *If there is a family of polynomial-size arithmetic circuits, over $\mathbb{Q}$ with divisions, computing* Perm, *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 $\mathbb{Z}$, we get that both f and g are also computable by arithmetic circuits of size $\mathsf{poly}(s)$: 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 7. It follows by Lemma 8 that there is a tuple of integers $(a_1, \dots, a_n)$ at which g is non-zero, and such that the bit complexity of each a_i is polynomial in s. We conclude that the size of a "good" point $\vec{a}$ at which a given arithmetic circuit C of size s (over $\mathbb{Q}$, with divisions) is defined is bounded by a polynomial in s.

Assuming that a $\mathsf{poly}(n)$-size circuit C computes Perm of $n \times n$ matrices, we can obtain from C by Theorem 19 a new $\mathsf{poly}(n)$-size circuit C' for Perm such that the numerator of C' computes an integer polynomial $h(x_1, \dots, x_n)$ 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, suppose that we have such a pair of $\mathsf{poly}(n)$-size division-free arithmetic circuits C_1 and C_2 over $\mathbb{Z}$, where C_1 computes an n^2-variable polynomial and C_2 computes some constant c. We want to check $C_1(X) \equiv c\mathrm{Perm}(X)$ over integers, where X is an $n \times n$ matrix of variables.

Let us define (as in the previous subsection) restrictions C^i, $1 \leqslant i \leqslant n$, of C_1 so that if C_1 computes $c\mathrm{Perm}$ over $\mathbb{Z}$, then each C^i computes $c\mathrm{Perm}$ of $i \times i$ integer matrices. Now, equations (1) and (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 $2 \leqslant i \leqslant n$. Note that all these are polynomial identity tests for polynomial-size division-free arithmetic circuits over $\mathbb{Z}$. Thus, if ACIT over $\mathbb{Z}$ can be derandomized, then we can test whether $C_1(X) \equiv c\mathrm{Perm}(X)$ over $\mathbb{Z}$.

Finally, once we successfully verified that a given circuit C_1 computes $c\mathrm{Perm}$, we would like to be able to compute $\mathrm{Perm}(M)$ for every particular 0-1 matrix M efficiently. The obvious problem is that, as we evaluate $C_1(M)$ and compute c using C_2, we may get intermediate integer values whose bit complexity is exponential in the sizes of C_1 and C_2. If we use modular arithmetic modulo some m, we need to make sure that m does not divide c (so that we can compute $C_1(M)/c \mod m$). If we have a prime $m \geqslant 2^{n^2}$ that does not divide c, then we can recover $\mathrm{Perm}(M)$. It is not clear how to find such a prime efficiently deterministically, but it is easy to find m nondeterministically: guess an $\Omega(n^2)$-bit prime m together with its $\mathsf{polylog}(m)$-size certificate of primality (which exists by Pratt's result [Pra75]), simulate the computation of the circuit C_2 modulo m, accepting iff the result is non-zero.

These arguments yield the following strengthening of Corollary 15.

Theorem 21. *Suppose that ACIT over $\mathbb{Z}$ is in* NSUBEXP*, and that* Perm *of $n \times n$ matrices over $\mathbb{Q}$ is computable by polynomial-size arithmetic circuits over $\mathbb{Q}$ with divisions. Then* Perm *is in* NSUBEXP.

Proof. First, for any given n, 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 also guess a prime $2^{n^2} \leqslant m \leqslant 2^{|C_2|^2+n^2}$ together with its primality certificate, and test that $c \not\equiv 0 \mod m$; 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 $c \leqslant |C_2|^{2^{|C_2|}}$). If the primality certificate for m is correct, and if $c \not\equiv 0 \mod m$, then we continue; otherwise we reject.

By Corollary 20, we know that if Perm is computable by a polynomial-size arithmetic circuit (over $\mathbb{Q}$, with divisions), then 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 $C_1 \equiv c\mathrm{Perm}$. We can test whether $C_1(X) \equiv c\mathrm{Perm}(X)$ for $n \times n$ integer matrices X, by verifying identities (1) and (2) (modified as described in our discussion above), using the assumed nondeterministic subexponential-time algorithm for ACIT over $\mathbb{Z}$. If all these identities hold, then we know that $C_1(M) = c\mathrm{Perm}(M)$ for every 0-1 matrix M. Moreover, we can compute $\mathrm{Perm}(M)$ by simulating the computation of $C_1(M)/c$ modulo our previously guessed prime m; remember that m was chosen so that $c \not\equiv 0 \mod m$ and that $\mathrm{Perm}(M) < m$, for any 0-1 matrix M. □

Similarly, we can strengthen Corollary 16.

Theorem 22. *Suppose that ACIT over $\mathbb{Z}$ is in* NP *and that* Perm *is computable by polynomial-size arithmetic circuits over $\mathbb{Q}$ with divisions. Then* $\mathrm{Perm} \in \mathsf{NP}$.

4 Circuit Lower Bounds via Derandomization

4.1 If ACIT can be derandomized

Definition 23. We say that NEXP is *computable by polynomial-size arithmetic circuits* if the following two conditions hold:

1. $\mathsf{NEXP} \subset \mathsf{P/poly}$, and

2. Perm over $\mathbb{Q}$ is computable by polynomial-size arithmetic circuits (possibly with divisions).

If, in the definition above, condition (1) holds for $\mathsf{NEXP} \cap \mathsf{coNEXP}$ rather than NEXP, we will say that $\mathsf{NEXP} \cap \mathsf{coNEXP}$ is computable by polynomial-size arithmetic circuits.

Theorem 24. *If ACIT over $\mathbb{Z}$ is in $\mathsf{NTIME}(2^{n^\epsilon})$ for every $\epsilon > 0$, then NEXP is* not *computable by polynomial-size arithmetic circuits.*

Proof. Suppose, for the sake of contradiction, that ACIT $\in \cap_{\epsilon>0}\mathsf{NTIME}(2^{n^\epsilon})$, and NEXP is computable by polynomial-size arithmetic circuits. It follows by Theorem 4 and Corollary 5 that $\mathsf{NEXP} = \mathsf{coNEXP}$ and Perm is NEXP-complete. On the other hand, by Theorem 21, we have that Perm $\in \mathsf{NTIME}(2^{n^\epsilon})$, for every $\epsilon > 0$. Hence, we obtain that $\mathsf{coNEXP} \subseteq \mathsf{NTIME}(2^n)$, which is impossible by the following simple diagonalization argument. 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 $n \in \mathbb{N}$, we obtain our result. $\square$

Assuming stronger derandomization of ACIT, we can strengthen the conclusion of Theorem 24.

Theorem 25. *If ACIT over $\mathbb{Z}$ is in NP, then $\mathsf{NEXP} \cap \mathsf{coNEXP}$ is* not *computable by polynomial-size arithmetic circuits.*

Proof. If Perm is not computable by polynomial-size arithmetic circuits, then we are done. Thus, let us suppose that Perm is computable by polynomial size arithmetic circuits.

We know by Theorems 1 and 2 that Perm is PH-hard. On the other hand, our assumptions imply, by Theorem 22, that Perm $\in \mathsf{NP}$. This means that $\mathsf{PH} = \mathsf{NP} = \mathsf{coNP}$. Hence, by padding, we have that $\mathsf{NEXP} = \mathsf{coNEXP} = \mathsf{NEXP} \cap \mathsf{coNEXP}$. Appealing to Theorem 24 concludes the proof. $\square$

With extra care, we also obtain the "infinitely often" version of Theorem 24.

Theorem 26. *If ACIT over $\mathbb{Z}$ is in $\mathsf{io\text{-}NTIME}(2^{n^\epsilon})$ for every $\epsilon > 0$, then NEXP is* not *computable by polynomial-size arithmetic circuits.*

Proof. Suppose that NEXP is computable by polynomial-size arithmetic circuits. Then, by Theorem 4 and Corollary 5, we have $\mathsf{NEXP} = \mathsf{EXP}$ and Perm is EXP-complete.

A closer look at the proof of Theorem 21 reveals that if ACIT $\in \cap_{\epsilon>0}\mathsf{io\text{-}NTIME}(2^{n^\epsilon})$, then Perm $\in \mathsf{io\text{-}NTIME}(2^{n^\epsilon})/O(\log n)$, for every $\epsilon > 0$. Indeed, for a given input size n, the output of the many-one reduction of Lemma 13 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 Perm in nondeterministic subexponential time, for infinitely many input sizes, using logarithmic advice.

Next we will argue that

$$\mathsf{EXP} \subseteq \mathsf{io\text{-}NTIME}(2^{n^\epsilon})/\log^2 n \tag{3}$$

for every $\epsilon > 0$. Recall that, by our assumption, Perm is EXP-complete. This means that every language $L \in \mathsf{EXP}$ is reducible to Perm of 0-1 matrices in time n^{c_L}. We can 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 Perm is correct. Then we can pad up to the "good" size all the queries to Perm made by the reduction from L to Perm. By combining the advice strings of the reduction and the algorithm for Perm, we conclude that

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

for some constant c'_L dependent on L. Since $\log^2 n \geqslant c \log n$ for every constant c whenever n gets large, the inclusion (3) follows.

Finally, to derive a contradiction, we employ an argument from [IKW02]. Note that the existence of a universal Turing machine for $\mathsf{NTIME}(2^n)$ and the assumption that $\mathsf{NEXP} \subset \mathsf{P/poly}$ imply that there is a fixed constant c_0 such that

$$\mathsf{NTIME}(2^{n^\epsilon})/\log^2 n \subset \mathsf{SIZE}(n^{c_0}).$$

It follows that $\mathsf{EXP} \subset \text{io-}\mathsf{SIZE}(n^{c_0})$, which is impossible by a simple diagonalization argument. $\square$

4.2 If RP can be derandomized

Since ACIT over $\mathbb{Z}$ is in coRP by Lemma 9, we immediately obtain the following results.

Corollary 27. *If* $\mathsf{coRP} \subseteq \text{io-}\mathsf{NTIME}(2^{n^\epsilon})$ *for every* $\epsilon > 0$*, then* NEXP *is* not *computable by polynomial-size arithmetic circuits.*

Proof. This is immediate from Lemma 9 and Theorem 26. $\square$

Corollary 28. *If* $\mathsf{coRP} \subseteq \mathsf{NP}$*, then* $\mathsf{NEXP} \cap \mathsf{coNEXP}$ *is* not *computable by polynomial-size arithmetic circuits.*

Proof. This is immediate from Lemma 9 and Theorem 25. $\square$

Corollary 29. *If* $\mathsf{BPP} = \mathsf{P}$ *or if* $\mathsf{RP} = \mathsf{ZPP}$*, then* $\mathsf{NEXP} \cap \mathsf{coNEXP}$ *is* not *computable by polynomial-size arithmetic circuits.*

Proof. If $\mathsf{BPP} = \mathsf{P}$ or if $\mathsf{RP} = \mathsf{ZPP}$, then $\mathsf{coRP} \subseteq \mathsf{NP}$, and the result follows by Corollary 28. $\square$

4.3 If AFIT or SDIT can be derandomized

Here we show that the existence of nontrivial algorithms for solving even a special case of Polynomial Identity Testing (e.g., AFIT or SDIT) would also yield (slightly weaker) circuit lower bounds for NEXP. We need the following definition.

Definition 30. We say that NEXP is *computable by polynomial-size arithmetic formulas* if the following two conditions hold:

1. $\mathsf{NEXP} \subset \mathsf{P/poly}$, and
2. Perm over $\mathbb{Z}$ is computable by division-free polynomial-size arithmetic formulas.

First, we prove the following.

Theorem 31. *If AFIT over* $\mathbb{Z}$ *is in* $\text{io-}\mathsf{NTIME}(2^{n^\epsilon})$ *for every* $\epsilon > 0$*, then* NEXP *is* not *computable by polynomial-size arithmetic formulas.*

Proof. The proof is analogous to that of Theorem 26, except for using Lemma 17 instead of Lemma 13. $\square$

Next, we shall argue that any nontrivial derandomization of SDIT also leads to circuit lower bounds for NEXP. To this end, we prove that AFIT is many-one reducible to SDIT.

Theorem 32. *AFIT is polynomial-time many-one reducible to SDIT.*

The proof of Theorem 32 relies on the following result of Valiant [Val79a] (see also [Gat87]).

Theorem 33 ([Val79a]). *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* $\mathrm{Det}(A) \equiv f$.

Proof of Theorem 32. The requisite polynomial-time many-one reduction is the algorithm from Theorem 33. □

Thus, if SDIT can be done in nondeterministic subexponential time, then, by Theorem 32, so can AFIT, and hence, by Theorem 31, we conclude that NEXP is not computable by polynomial-size arithmetic formulas. Actually, using the ideas from the proof of Theorem 26 (introducing appropriate advice strings), we can prove the following, slightly stronger, statement.

Theorem 34. *If SDIT is in* io-NTIME$(2^{n^{\epsilon}})$ *for every $\epsilon > 0$, then* NEXP *is* not *computable by polynomial-size arithmetic formulas.*

4.4 If RNC can be derandomized

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

Corollary 35. *If* coRNC $\subseteq$ io-NTIME$(2^{n^{\epsilon}})$ *for every $\epsilon > 0$, then* NEXP *is* not *computable by polynomial-size arithmetic formulas.*

Proof. The proof is immediate from Corollary 10 and Theorem 34. □

5 A Hard Language in 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. The following result is due to Buhrman, Fortnow, and Thierauf.

Theorem 36 ([BFT98]). MA-EXP $\not\subseteq$ P/poly.

While we cannot strengthen Theorem 36, we can prove that a (seemingly) smaller complexity class than MA-EXP contains a language of superpolynomial *arithmetic* circuit complexity. Namely, we show the following.

Theorem 37. *At least one of the following holds:*

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

2. NEXP$^{\mathsf{RP}}$ $\not\subseteq$ P/poly.

First, observe that NP$^{\mathsf{BPP}}$ $\subseteq$ MA by a fairly straightforward argument. Hence, by padding, we have that NEXP$^{\mathsf{BPP}}$ $\subseteq$ MA-EXP, and so Theorem 37 shows the existence of a hard language in a smaller complexity class than MA-EXP. As a side remark, we want to point out that MA $\subseteq$ NP$^{\text{promise-RP}}$ (the proof is implicit in [BF99]); hence, MA-EXP $\subseteq$ NEXP$^{\text{promise-RP}}$.

For the proof of Theorem 37, we shall need the following.

Lemma 38. *If* Perm *is computable by polynomial-size arithmetic circuits, then* Perm $\in \mathsf{NP}^{\mathsf{RP}}$.

Proof. By Lemmas 9 and 13, 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 Perm $\in \mathsf{NP}^{\mathsf{RP}}$. □

Theorem 39. *If* NEXP *is computable by polynomial-size arithmetic circuits, then* $\mathsf{NEXP} \subseteq \mathsf{NP}^{\mathsf{RP}}$.

Proof. Recall that our assumption that NEXP is computable by polynomial-size arithmetic circuits means that both $\mathsf{NEXP} \subset \mathsf{P/poly}$ and Perm is computable by polynomial-size arithmetic circuits. The former yields, by Corollary 5, that Perm is NEXP-complete. By Lemma 38, the latter implies that Perm $\in \mathsf{NP}^{\mathsf{RP}}$, which concludes our proof. □

We are now ready to give a proof of Theorem 37.

Proof of Theorem 37. Assume NEXP is computable by polynomial-size arithmetic circuits. By Theorem 39, $\mathsf{NEXP} \subseteq \mathsf{NP}^{\mathsf{RP}}$. By padding, $\mathsf{NEE} \subseteq \mathsf{NE}^{\mathsf{RP}}$, where $\mathsf{NEE} = \mathsf{NTIME}(2^{2^{O(n)}})$. Since $\mathsf{EE} \not\subset \mathsf{P/poly}$ [Kan82], our claim follows. □

Later we shall need the following result that is similar to Theorem 39.

Theorem 40. *If* EXP *is computable by polynomial-size arithmetic circuits, then* $\mathsf{EXP} \subseteq \mathsf{NP}^{\mathsf{RP}}$.

Proof. The proof is analogous to that of Theorem 39. If $\mathsf{EXP} \subset \mathsf{P/poly}$, then $\mathsf{EXP} = \mathsf{MA}$ by Theorem 3; hence, by Theorems 1 and 2, we conclude that Perm is EXP-complete. Now, if Perm is computable by polynomial-size arithmetic circuits, then Perm $\in \mathsf{NP}^{\mathsf{RP}}$ by Lemma 38. □

6 Low-Degree Testing and Derandomization

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 we could deterministically test whether a given *Boolean* circuit computes a polynomial of low degree, then we would be able to adapt our arguments from Section 3 in the Boolean (rather than arithmetic) setting. We formalize this idea below.

The problem of *Low-Degree Testing* consists in checking 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 $p(x_1, \dots, x_n)$ 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., [RS96] and the references therein). We state the Low-Degree Testing as the following promise problem.

Low-Degree Testing Problem (LDT)
Positive inputs: A 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 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 [BFL91, FGL+96, RS96, AS98, ALM+98, RS97, AS97] imply that LDT is in promise-BPP. It is not known if LDT is hard for promise-BPP. We will argue below that if LDT can be solved in deterministic polynomial time, and if $\mathsf{BPP} = \mathsf{P}$, then $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

We say that LDT can be done in polynomial time 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 $d, k \in \mathbb{N}$ and a Boolean circuit C of size s computing a function $f : \mathbb{F}^n \to \mathbb{F}$, where $\mathbb{F}$ is a finite field of size at least $(d + k + s)^{c_0}$, the machine T runs in time $\mathsf{poly}(s, d, k, n, \log |\mathbb{F}|)$, accepting if C is a positive input, and rejecting if C is a negative input to LDT.

Next we describe algorithm TEST (see Algorithm 1 below) whose properties will be useful in deriving the main results of this section.

```
INPUT: $(q, C, M, b)$, where $q \in \mathbb{N}$ is given in unary, $C$ 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 $q$ is not a prime, then REJECT.

2. Viewing $C$ as a circuit on input $n \times n$ matrices over GF$(q)$, define $C_i$, $1 \leqslant i \leqslant n$, to be the restriction of $C$ to $i \times i$ matrices over GF$(q)$ satisfying the property: If $C$ computes Perm of $n \times n$ matrices over GF$(q)$, then each $C_i$ computes Perm of $i \times i$ matrices over GF$(q)$. For every $1 \leqslant i \leqslant n$, run the deterministic LDT algorithm on $C_i$ for $\mathbb{F} =$ 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 (1) and (2) hold for the functions $f_i$ computed by $C_i$, $1 \leqslant i \leqslant n$, when matrix elements are chosen uniformly at random from GF$(q)$. If any $C_i$ fails the test, then REJECT.

4. Apply Lemma 11 to $C$, getting a randomized circuit $\hat{C}$ that computes a degree $n$ polynomial over 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 41. 1. *Algorithm* TEST *is a* BPP *algorithm.*

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

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

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 $1 \leqslant i \leqslant n$, $p_i \not\equiv$ Perm of $i \times i$ matrices over GF(q),

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

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

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 $(1-1/n)^2 \geqslant 1-2/n$.

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 $\mathrm{Perm}(M) \equiv b \mod q$ correctly with probability close to one.

Thus, we have proved claim 1 of the lemma. Claim 2 is obvious. Claim 3 follows since TEST accepts with high probability only in case II. □

By analogy with Corollary 15, we obtain the following.

Corollary 42. *Suppose that* $\mathsf{BPP} \subseteq \mathsf{NSUBEXP}$ *and that LDT can be done in polynomial time. If* $\mathrm{Perm} \in \mathsf{P/poly}$*, then* $\mathrm{Perm} \in \mathsf{NSUBEXP}$.

Proof. Observe that our assumptions imply that the language $L(\textsc{Test})$ of the BPP algorithm TEST of Lemma 41 is in NSUBEXP. The following is a nondeterministic subexponential-time algorithm for Perm.

Let M be a given $n \times n$ integer matrix; for simplicity, let us assume that each entry of M can be described using at most n bits, and so the value $\mathrm{Perm}(M)$ can be described using at most n^3 bits. 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 $1 \leqslant i \leqslant n^4$; 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.

Let $M_i \equiv M \mod q_i$, for $1 \leqslant i \leqslant n^4$. We nondeterministically guess $b = \mathrm{Perm}(M)$, and, for each $1 \leqslant i \leqslant n^4$, we set $b_i \equiv b \mod q_i$. Then we 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 we output b; otherwise, we reject.

Note that by claim 3 of Lemma 41, any output value b will be such that $b_i \equiv \mathrm{Perm}(M) \mod q_i$, for all i. Hence, by the Chinese Remainder Theorem, we have that $b = \mathrm{Perm}(M)$.

By claim 2 of Lemma 41, 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 $b = \mathrm{Perm}(M)$. Thus, our nondeterministic algorithm will always output $b = \mathrm{Perm}(M)$. □

6.2 Derandomization of MA

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

Theorem 43. *Suppose that LDT can be done in polynomial time. If* $\mathsf{BPP} \subseteq \mathsf{NSUBEXP}$*, then* $\mathsf{MA} \subseteq \text{io-}\mathsf{NTIME}(2^{n^\epsilon})$*, for every* $\epsilon > 0$.

Proof. Suppose, for the sake of contradiction, that $\mathsf{MA} \not\subseteq \text{io-}\mathsf{NTIME}(2^{n^\epsilon})$ for some $\epsilon > 0$. Then, by Theorem 6, we have $\mathsf{EXP} = \mathsf{MA} = \mathsf{PH} \subset \mathsf{P/poly}$. Since Perm $\in \mathsf{EXP}$, we get that Perm over $\mathbb{Z}$ has polynomial-size Boolean circuits. It now follows by Corollary 42 that Perm $\in \mathsf{NSUBEXP}$.

Since Perm $\in \mathsf{EXP}$ and Perm is PH-hard by Theorems 1 and 2, we have that Perm is EXP-complete. Consequently, we get that $\mathsf{MA} = \mathsf{EXP} \subseteq \mathsf{NSUBEXP}$. A contradiction. $\square$

Remark 44. It is possible to prove a version of Theorem 43 with the class MA being replaced by promise-BPP (or the class APP introduced in [KRC00]). That is, the assumptions of Theorem 43 imply that promise-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 promise-BPP under the assumptions that both $\mathsf{BPP} = \mathsf{P}$ and LDT is in P.

6.3 Circuit lower bounds

As a corollary of Theorem 43, we obtain the following result: if both LDT and BPP can be derandomized, then NEXP does not have polynomial-size Boolean circuits.

Theorem 45. *Suppose that LDT can be done in polynomial time. If* $\mathsf{BPP} \subseteq \mathsf{NSUBEXP}$*, then* $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

Proof. By Theorem 43, we get from our assumptions that $\mathsf{MA} \subseteq \text{io-}\mathsf{NTIME}(2^n)$. This implies that $\mathsf{NEXP} \neq \mathsf{MA}$.

Indeed, suppose that $\mathsf{NEXP} = \mathsf{MA}$. Then $\mathsf{NEXP} = \mathsf{EXP} = \mathsf{coNEXP} \subseteq \text{io-}\mathsf{NTIME}(2^n)$. But a simple diagonalization argument shows that $\mathsf{coNEXP} \not\subseteq \text{io-}\mathsf{NTIME}(2^n)$.

Finally, since $\mathsf{NEXP} \neq \mathsf{MA}$, the result follows by Theorem 4. $\square$

Theorem 45 strengthens one of the results in [IKW02] saying that if promise-BPP can be derandomized, then $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$. This is because the assumption that promise-BPP can be done in deterministic polynomial time implies that both $\mathsf{BPP} = \mathsf{P}$ and that LDT is in deterministic polynomial time.

7 Conditional Derandomization of Polynomial Identity Tests

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.

Root Finding
GIVEN: An arithmetic circuit computing a non-zero polynomial $g(x_1, \dots, x_n, y)$ of total degree d over a field $\mathbb{F}$.
FIND: A list of arithmetic circuits which contains a circuit for every polynomial $p(x_1, \dots, x_n)$ such that

$$g(x_1, \dots, x_n, p(x_1, \dots, x_n)) \equiv 0.$$

We need the following result of Kaltofen.

Theorem 46 ([Kal89]). *There is a probabilistic polynomial-time algorithm that, given an arithmetic circuit of size s computing a polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$ of total degree at most d, with probability at least $3/4$ outputs the numbers $e_i \geqslant 1$ and irreducible polynomials $h_i \in \mathbb{F}[x_1, \dots, x_n]$, $1 \leqslant i \leqslant r$, 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 size of a coefficient of f.

We also use the following basic fact.

Lemma 47 (Gauss). *For a field $\mathbb{F}$, let $f(x_1, \dots, x_n, y) \in \mathbb{F}[x_1, \dots, x_n, y]$ and $p(x_1, \dots, x_n) \in \mathbb{F}[x_1, \dots, x_n]$ be any polynomials such that*

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

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

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

Corollary 48. *The root finding problem for a polynomial $g \in \mathbb{F}[x_1, \dots, x_n, y]$ 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 $\mathsf{poly}(n, d, a)$, where a is the maximum size of a coefficient of g.*

Proof. For the case of $\mathbb{F} = \mathbb{Q}$, this follows immediately from Lemma 47 and Theorem 46. Indeed, with high probability, the algorithm of Theorem 46 will produce a small arithmetic circuit computing $y - p(x_1, \dots, x_n)$. Substituting 0 for y, and multiplying by -1, we obtain an arithmetic circuit for $p(x_1, \dots, x_n)$.

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 $p(x_1, \dots, x_n)$, provided that the multiplicity e of the linear factor $y - p(x_1, \dots, x_n)$ 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 $\hat{p}(x_1, \dots, x_n) = p(x_1, \dots, x_n)^{q^k}$. 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 the entire field $\mathbb{F}$; since we do not know k, we output a circuit for every $0 \leqslant k < t$. The size of the new arithmetic circuit increases by at most $\mathsf{polylog}(|\mathbb{F}|)$. ☐

Remark 49. In the case of a finite field $\mathbb{F} = \mathrm{GF}(q^t)$ of characteristic q, the proof of Corollary 48 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, whenever we talk about the arithmetic circuit complexity of polynomials over finite fields, we shall adopt the *functional* viewpoint: the complexity of a polynomial p over a finite field $\mathbb{F}$ is defined as the size of a smallest arithmetic circuit that agrees with p over the entire domain $\mathbb{F}^n$.

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). We should point out that such a generalization of the NW generator was used earlier by Raz, Reingold, and Vadhan [RRV99] in the context of randomness extractors.

Below, for an l-tuple $a = (a_1, \ldots, a_l)$ and a subset $S \subseteq \{1, \ldots, l\}$ of size m, we denote by $a|_S$ the m-tuple of the elements of a indexed by the set S.

```
PARAMETERS: $l, m, n \in \mathbb{N}$.
INPUT: $a = (a_1, \ldots, a_l) \in F^l$, where $\mathbb{F}$ is a field.
ORACLE ACCESS: $p(x_1, \ldots, x_m) \in \mathbb{F}[x_1, \ldots, x_m]$.

  1. For given $m, n \in \mathbb{N}$, construct the set system $S_1, \ldots, S_n$ as given by Lemma 12.

  2. Output $(p(a|_{S_1}), \ldots, p(a|_{S_n})) \in \mathbb{F}^n$.
```

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 12, we enumerate all l-tuples $a = (a_1, \ldots, a_l)$, 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 zero of f among the outputs of the NW generator based on a polynomial p. We shall argue that p can then be computed by a "small" arithmetic circuit. This is made precise in the following lemma.

Lemma 50. *Let $f \in \mathbb{F}[y_1, \ldots, y_n]$ and $p \in \mathbb{F}[x_1, \ldots, x_m]$ 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 12. Suppose that $f(\mathrm{NW}^p(a)) = 0$ for all $a \in S^l$.*

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

Proof. Our proof is in two parts. First, by a "hybrid" argument, we obtain from f a non-zero polynomial $g(x_1, \ldots, x_m, y)$ such that $p(x_1, \ldots, x_m)$ is a y-root of this polynomial g. Then we appeal to Corollary 48 to conclude that p is computable by an arithmetic circuit with required parameters.

I. HYBRID ARGUMENT. We define the following collection of polynomials:

- $g_0(x_1, \ldots, x_l, y_1, \ldots, y_n) = f(y_1, \ldots, y_n)$,
- for $1 \leqslant i \leqslant n$, $g_i(x_1, \ldots, x_l, y_{i+1}, \ldots, y_n) = g_{i-1}(x_1, \ldots, x_l, p((x_1, \ldots, x_l)|_{S_i}), y_{i+1}, \ldots, y_n)$.

Observe that $g_i(x_1, \ldots, x_l, y_{i+1}, \ldots, y_n)$ is obtained from f by replacing the first i variables in f by the polynomials $p((x_1, \ldots, x_l)|_{S_j})$ for $1 \leqslant j \leqslant i$. Thus, $g_n(x_1, \ldots, x_l) = f(\mathrm{NW}^p(x_1, \ldots, x_l))$.

The l-variate polynomial g_n is of total degree at most $D = d_f d_p$. Since g_n vanishes on S^l where $|S| > D$, we have, by Lemma 8, $g_n \equiv 0$.

If $g_0 \not\equiv 0$, but $g_n \equiv 0$, then there must be the smallest $0 \leqslant i \leqslant n$ 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 have that $g(x_1, \dots, x_m, y) \not\equiv 0$, but $g(x_1, \dots, x_m, p(x_1, \dots, x_m)) \equiv 0$. Hence, by Corollary 48, the polynomial p is computable by an arithmetic circuit of size 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 condition (3) of Lemma 12). 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 shall state our conditional derandomization result for 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 7.

Theorem 51. *Let $p = \{p_m\}_{m \geqslant 0}$ be a family of exponential-time computable multilinear non-zero polynomials $p_m \in \mathbb{Z}[x_1, \dots, x_m]$ such that the maximum coefficient size of p_m is in $\mathsf{poly}(m)$. Suppose that the arithmetic circuit complexity of p over $\mathbb{Q}$ is $s_p(m)$ for some function $s_p : \mathbb{N} \to \mathbb{N}$.*

Let C be a $\mathsf{poly}(n)$-size division-free arithmetic circuit over $\mathbb{Z}$ computing an n-variate polynomial $f_n \in \mathbb{Z}[y_1, \dots, y_n]$ of total degree $d_f(n) \in \mathsf{poly}(n)$ and maximum coefficient size in $\mathsf{poly}(n)$. Then, for all sufficiently large n, testing whether $f_n \equiv 0$ can be done deterministically in time

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

Proof. (1) For $m = n^\epsilon$, Lemma 12 gives $l \leqslant n^{2\epsilon}$. Let S be a subset of $\mathbb{Z}$ of size at least $nd_f \in \mathsf{poly}(n)$.

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, computes 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 50, this testing algorithm must succeed for f_n, since otherwise the arithmetic circuit complexity of p would be polynomial.

(2) We can choose $m = (\log n)^d$, for some sufficiently large d to be specified later. This choice yields the NW generator on $l \leqslant (\log n)^{2d}$ 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 50, 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^ϵ} for any $\epsilon > 0$, contrary to our assumption on the hardness of p. □

A version of Theorem 51 for the case of finite fields $\mathbb{F}$ also holds.

Theorem 52. *Let $p = \{p_m\}_{m \geqslant 0}$ be a family of exponential-time computable multilinear non-zero polynomials $p_m \in \mathbb{F}[x_1, \dots, x_m]$ for some finite field $\mathbb{F}$. Suppose that the arithmetic circuit complexity of p over $\mathbb{F}$ is $s_p(m)$ for some function $s_p : \mathbb{N} \to \mathbb{N}$.*

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

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

Proof Sketch. The proof is similar to that of Theorem 51. If the field $\mathbb{F}$ is small (less than a polynomial in n), then we need to go to an extension field of $\mathbb{F}$ in order to have a polynomial-size subset S of field elements. Since we only need a $O(\log n)$-degree extension of $\mathbb{F}$, we can find such an extension in deterministic time $\mathsf{poly}(n)$ by exhaustive search. □

For AFIT, we obtain the following.

Corollary 53. *Let $p = \{p_m\}_{m \geqslant 0}$ be a family of exponential-time computable multilinear non-zero polynomials $p_m \in \mathbb{Z}[x_1, \dots, x_m]$ such that the maximum coefficient size of p_m is in $\mathsf{poly}(m)$. Suppose that the arithmetic circuit complexity of p over $\mathbb{Q}$ is $s_p(m)$ for some function $s_p : \mathbb{N} \to \mathbb{N}$.*

Let F be a $\mathsf{poly}(n)$-size division-free arithmetic formula over $\mathbb{Z}$ computing an n-variate polynomial $f_n \in \mathbb{Z}[y_1, \dots, y_n]$. Then, for all sufficiently large n, testing whether $f_n \equiv 0$ can be done deterministically in time

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

Proof. Note that an arithmetic formula F of $\mathsf{poly}(n)$ size computes a polynomial f of $\mathsf{poly}(n)$ degree (by Lemma 7). The coefficient sizes of f are also bounded by $\mathsf{poly}(n)$ (by an argument similar to the proof of Lemma 7). Hence, we can apply Theorem 51 to F. □

Remark 54. Theorem 51 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.*

We can now establish a weak converse of Theorems 31 and 34.

Theorem 55. *If NEXP is* not *computable by polynomial-size arithmetic circuits, then both AFIT and SDIT are in $\text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$ for every $\epsilon > 0$.*

The proof of Theorem 55 will use the following result implicit in [IKW02].

Lemma 56 ([IKW02]). *If* $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$*, then* $\mathsf{coRP} \subseteq \text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$ *for every* $\epsilon > 0$.

Proof of Theorem 55. If Perm is not computable by polynomial-size arithmetic circuits, then both AFIT and SDIT are in $\text{io-}\mathsf{TIME}(2^{n^\epsilon})$ for every $\epsilon > 0$, by Theorem 51. On the other hand, if $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$, then both AFIT and SDIT are in $\text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$ for every $\epsilon > 0$, by Lemma 56 and the fact that both AFIT and SDIT are coRP problems. □

We should point out that, unlike in the Boolean circuit complexity case (cf. [IW97]), 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 also would like to remark that, as one might suspect, a "uniform" version of Theorem 51 can be proved, along the lines of [IW98]; the details are omitted.

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 57 ([BFNW93]). *At least one of the following holds:*

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

2. $\mathsf{EXP} = \mathsf{MA} \subset \mathsf{P/poly}$.

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

Theorem 58. *At least one of the following holds:*

1. *AFIT is in* $\text{io-}\mathsf{TIME}(2^{n^\epsilon})$ *for every* $\epsilon > 0$*, or*

2. $\mathsf{EXP} = \mathsf{NP}^{\mathsf{RP}}$.

Proof. If AFIT cannot be done in deterministic subexponential time, then it follows by Corollary 53 that Perm over $\mathbb{Z}$ is computable by polynomial-size arithmetic circuits. Also, since AFIT is in $\mathsf{coRP} \subseteq \mathsf{BPP}$, the lack of derandomization for AFIT means that BPP is not in $\text{io-}\mathsf{TIME}(2^{n^\epsilon})$ for some ϵ. Hence, by Theorem 57, we know that $\mathsf{EXP} \subset \mathsf{P/poly}$.

Thus we have that EXP is computable by polynomial-size arithmetic circuits, and the conclusion $\mathsf{EXP} = \mathsf{NP}^{\mathsf{RP}}$ follows from Theorem 40. □

Note the following difference between Theorems 57 and 58. Our Theorem 58 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 [Kab01] proved an unconditional derandomization result for RP in a certain "uniform" setting, the condition of "uniformity" makes the result in [Kab01] too weak for Corollary 27 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 "NEXP is computable by polynomial-size arithmetic circuits $\Rightarrow \mathsf{BPP} \not\subseteq \cap_{\epsilon>0}\text{io-}\mathsf{NTIME}(2^{n^\epsilon})$" 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 same assumption imply (possibly arithmetic) circuit lower bounds for EXP, rather than $\mathsf{NEXP} \cap \mathsf{coNEXP}$?

Regarding the low-degree testing, we ask the following questions. Does the assumption that there is a deterministic polynomial-time algorithm for LDT imply any circuit lower bounds for NEXP? The answer is positive if one could show that LDT is *hard* for promise-BPP. This motivates the question: Is LDT hard for promise-BPP? The related question is to decide whether the assumptions that both $\mathsf{BPP} = \mathsf{P}$ and LDT is in P should imply that $\text{promise-}\mathsf{BPP} \subseteq \text{promise-}\mathsf{SUBEXP}$.

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.

References

[AB99] M. Agrawal and S. Biswas. Primality and identity testing via Chinese remaindering. In *Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science*, pages 202–209, 1999.

[ACR98] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. A new general derandomization method. *Journal of the Association for Computing Machinery*, 45(1):179–213, 1998. (preliminary version in ICALP'96).

[AKS02] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Manuscript, 2002.

[ALM$^+$98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the Association for Computing Machinery*, 45(3):501–555, 1998. (preliminary version in FOCS'92).

[AS97] S. Arora and M. Sudan. Improved low-degree testing and its applications. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, 1997.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the Association for Computing Machinery*, 45(1):70–122, 1998. (preliminary version in FOCS'92).

[Bab85] L. Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 421–429, 1985.

[BCW80] M. Blum, A.K. Chandra, and M.N. Wegman. Equivalence of free Boolean graphs can be tested in polynomial time. *Information Processing Letters*, 10:80–82, 1980.

[BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of the Seventh Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *Lecture Notes in Computer Science*, pages 37–48, Berlin, 1990. Springer Verlag.

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In C. Meinel and S. Tison, editors, *Proceedings of the Sixteenth Annual Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 100–109. Springer Verlag, 1999.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Complexity*, 3:307–318, 1993.

[BFT98] H. Buhrman, L. Fortnow, and L. Thierauf. Nonrelativizing separations. In *Proceedings of the Thirteenth Annual IEEE Conference on Computational Complexity*, pages 8–12, 1998.

[BH89] R. Boppana and R. Hirschfeld. Pseudo-random generators and complexity classes. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 1–26. JAI Press, Greenwich, CT, 1989.

[BK95] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the Association for Computing Machinery*, 42:269–291, 1995.

[BM88] L. Babai and S. Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[CDGK91] M. Clausen, A. Dress, J. Grabmeier, and M. Karpinsky. On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theoretical Computer Science*, 84(2):151–164, 1991.

[Chi85] A.L. Chistov. Fast parallel evaluation of the rank of matrices over a field of arbitrary characteristic. In *Fundamentals of Computation Theory*, volume 199 of *Lecture Notes in Computer Science*, pages 63–79. Springer Verlag, 1985.

[CK97] Z. Chen and M. Kao. Reducing randomness via irrational numbers. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 200–209, 1997.

[CRS95] S. Chari, P. Rohatgi, and A. Srinivasan. Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM Journal on Computing*, 24(5):1036–1050, 1995.

[DL78] R.A. DeMillo and R.J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7:193–195, 1978.

[FGL$^+$96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the Association for Computing Machinery*, 43(2):268–292, 1996. (preliminary version in FOCS'91).

[Gat87] J. von zur Gathen. Feasible arithmetic computations: Valiant's hypothesis. *Journal of Symbolic Computation*, 4:137–172, 1987.

[GKS90] D.Yu. Grigoriev, M. Karpinsky, and M.F. Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM Journal on Computing*, 19(6):1059–1063, 1990.

[GZ97] O. Goldreich and D. Zuckerman. Another proof that BPP$\subseteq$PH (and more). *Electronic Colloquium on Computational Complexity*, TR97-045, 1997.

[IKW02] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002. (preliminary version in CCC'01).

[IM83] O.H. Ibarra and S. Moran. Probabilistic algorithms for deciding equivalence of straight-line programs. *Journal of the Association for Computing Machinery*, 30(1):217–228, 1983.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science*, pages 181–190, 1999.

[ISW00] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudorandom generators with optimal seed length. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 1–10, 2000.

[IW97] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[IW98] R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *Proceedings of the Thirty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, pages 734–743, 1998.

[Kab01] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *Journal of Computer and System Sciences*, 63(2):236–252, 2001. (preliminary version in CCC'00).

[Kal88] E. Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *Journal of the Association for Computing Machinery*, 35(1):231–264, 1988.

[Kal89] E. Kaltofen. Factorization of polynomials given by straight-line programs. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 375–412. JAI Press, Greenwich, CT, 1989.

[Kal92] E. Kaltofen. On computing determinants of matrices without divisions. In P.S. Wang, editor, *Proceedings of the 1992 International Symposium on Symbolic and Algebraic Computation (ISSAC'92)*, pages 342–349, 1992.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55:40–56, 1982.

[KM99] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial hierarchy collapses. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 659–667, 1999.

[KRC00] V. Kabanets, C. Rackoff, and S. Cook. Efficiently approximable real-valued functions. *Electronic Colloquium on Computational Complexity*, TR00-034, 2000.

[KS01] A. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, pages 216–223, 2001.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the Association for Computing Machinery*, 39(4):859–868, 1992.

[Lip91] R. Lipton. New directions in testing. In J. Feigenbaum and M. Merrit, editors, *Distributed Computing and Cryptography*, pages 191–202. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 2, AMS, 1991.

[LMN93] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the Association for Computing Machinery*, 40(3):607–620, 1993.

[Lov79] L. Lovasz. On determinants, matchings and random algorithms. In L. Budach, editor, *Fundamentals of Computing Theory*. Akademia-Verlag, Berlin, 1979.

[LV98] D. Lewin and S. Vadhan. Checking polynomial identities over any field: Towards a derandomization? In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 438–447, 1998.

[MVV87] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[Pap94] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[PPSZ98] R. Paturi, P. Pudlák, M.E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the Thirty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, pages 628–637, 1998.

[PPZ99] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. *Chicago Journal of Theoretical Computer Science*, 1999. (preliminary version in FOCS'97).

[Pra75] V.R. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4:214–220, 1975.

[PSZ97] R. Paturi, M.E. Saks, and F. Zane. Exponential lower bounds for depth 3 Boolean circuits. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 86–91, 1997.

[RB91] R.M. Roth and G.M. Benedek. Interpolation and approximation of sparse multivariate polynomials. *SIAM Journal on Computing*, 20(2):291–314, 1991.

[RR97] A.A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55:24–35, 1997.

[RRV99] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 149–158, 1999.

[RS96] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[RS97] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 475–484, 1997.

[Sch80] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27(4):701–717, 1980.

[Sha92] A. Shamir. IP=PSPACE. *Journal of the Association for Computing Machinery*, 39(4):869–877, 1992.

[Str73] V. Strassen. Vermeidung von Divisionen. *Journal für die Reine und Angewandte Mathematik*, 264:182–202, 1973.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001. (preliminary version in STOC'99).

[SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the Forty-Second Annual IEEE Symposium on Foundations of Computer Science*, pages 648–657, 2001.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Uma02] C. Umans. Pseudo-random generators for all hardnesses. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, pages 127–134, 2002.

[Val79a] L. Valiant. Completeness classes in algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 249–261, 1979.

[Val79b] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[Yao82] A.C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[Zan98] F. Zane. *Circuits, CNFs, and Satisfiability.* PhD thesis, UCSD, 1998.

[Zip79] R.E. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of an International Symposium on Symbolic and Algebraic Manipulation (EUROSAM'79)*, Lecture Notes in Computer Science, pages 216–226, 1979.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'