Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds*
(Extended Abstract)

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

March 4, 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.

1 Introduction

1.1 Derandomization from circuit lower bounds

In the early 1980's, Yao [57, 12] 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 [38], and significantly strengthened in a sequence of papers [8, 26, 3, 51, 24, 25, 48, 53], which replaced the assumption

*The full version of the paper is available at www.eccc.uni-trier.de/eccc/ as TR02-055.

†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

of high average-case circuit complexity with that of high *worst-case* circuit complexity. For instance, Impagliazzo and Wigderson [26] 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 $\mathsf{EXP} \not\subset \mathsf{P/poly}$ [24]. On the other hand, Impagliazzo, Kabanets, and Wigderson [23] 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 [47, 59, 18], 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 [35, 37, 14], equivalence testing of read-once branching programs [10], multiset equality testing [11], primality testing [1, 2], a number of complexity-theoretic results on probabilistically checkable proofs [36, 49, 7, 5, 4], as well as sparse multivariate polynomial interpolation [59, 21, 17, 46].

Recently, a number of probabilistic algorithm for the polynomial identity testing were proposed that use fewer random bits than the standard Schwartz-Zippel algorithm [15, 33, 1, 32]. While these algorithms achieve some saving in randomness compared with the original Schwartz-Zippel algorithm, 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 [2] 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 [20] showed that the same hardness assumptions imply the derandomization of the class MA introduced by Babai [6, 9]. Klivans and van Melkebeek [31] 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 [56].

By extending hardness-randomness tradeoffs to the *algebraic* complexity setting, we have provided 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, [34] give a learning algorithm for constant-depth circuits based on lower bounds for such circuits; [42, 41, 40] 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 [58] 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 [45] ("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 the 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 the 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.

We also show that either $\mathsf{NEXP}^{\mathsf{RP}} \not\subset \mathsf{P/poly}$ or Permanent is not computable by polynomial-size arithmetic circuits. We 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 a result in [8]. 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

We use downward self-reducibility of the Permanent to argue that testing if a given arithmetic circuit computes Permanent is polynomial-time reducible to Polynomial Identity Testing. Thus, if the latter problem is solvable in P and if Permanent is computable by polynomial-size arithmetic circuits, then we can compute Permanent in NP. Finally, the assumption that $\mathsf{NEXP} \subset \mathsf{P/poly}$ yields that $\mathsf{NEXP} = \mathsf{MA}$ [23] and so, by the results of Valiant [55] and Toda [52], we conclude that the Permanent is NEXP-complete, contradicting the Nondeterministic Time Hierarchy theorem.

The conditional derandomization result for Polynomial Identity Testing is proved by combining the Nisan-Wigderson generator [38] with the straight-line factorization algorithm for multivariate polynomials by Kaltofen [30]. Roughly speaking, if our NW generator based on a "hard" polynomial p fails to provide a non-zero for a given 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 [30].

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 [38] and the Kaltofen factoring algorithm [30], 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 present conditional derandomization of Polynomial Identity Testing. Section 6 contains some further results that can be obtained using our techniques. Some concluding remarks are contained in Section 7.

2 Preliminaries

2.1 Complexity classes

We use the standard definitions of complexity classes RP, BPP, RNC, PH, EXP, NEXP, $\#\mathsf{P}$, and $\mathsf{P/poly}$ [39]; 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 [6, 9] 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 $x \in$

$\{0,1\}^n$, $x \in L \Rightarrow \exists y : \mathbf{Pr}_z[R(x,y,z) = 1] \geqslant 2/3$, 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 ([55]). Perm *is* #P*-complete.*

Theorem 2 ([52]). $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

We say that Perm is computable in $\mathsf{NTIME}(t)$ if the following language is in $\mathsf{NTIME}(t)$:

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

Note that if Perm $\in \mathsf{NTIME}(t)$, then Perm $\in \mathsf{coNTIME}(t)$, and so we obtain the following lemma.

Lemma 3. *If* Perm $\in \mathsf{NTIME}(t)$*, then*

$$\mathsf{P}^{\mathrm{Perm}} \subseteq \mathsf{NTIME}(\mathsf{poly}(n)t(\mathsf{poly}(n))).$$

We shall need the following.

Theorem 4 ([23]). $\mathsf{NEXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{NEXP} = \mathsf{MA}$.

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.*

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 need the following simple lemma that bounds the degree of a rational function (polynomial, if no division gates are used) computed by an arithmetic circuit of a given size (see, e.g., [22] for a proof).

Lemma 6. *An arithmetic circuit (respectively, formula) of size s on input variables $x_1, \ldots, x_n$ computes a rational function of total degree at most 2^s (respectively, s).*

2.3 Polynomial identity testing

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 $\mathrm{Det}(A) \equiv 0$?

We will need the following results.

Lemma 7 ([47, 59, 18]). *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 $\mathbf{Pr}_{a \in S^n}[p(a) = 0] \leqslant d/|S|$.*

Lemma 8 ([22]). *ACIT over $\mathbb{Z}$ is in* coRP.

By Lemma 7 and the well-known fact that the determinant of an integer matrix is computable in NC^2 [16], we immediately obtain the following.

Corollary 9. *SDIT is in* coRNC.

2.4 The Nisan-Wigderson designs

Our derandomization procedure for ACIT will use a generalization of the Nisan-Wigderson generator [38] that is based on the following construction of combinatorial designs.

Lemma 10 ([38]). *For every $m, n \in \mathbb{N}$, $n < 2^m$, there exists a family of sets $S_1, \ldots, S_n \subseteq \{1, \ldots, l\}$ 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

First, we prove the following.

Lemma 11. *The language ACP defined as*

$$\{C \mid C \text{ is an arithmetic circuit computing } \mathrm{Perm} \text{ over } \mathbb{Z}\}$$

is polynomial-time many-one reducible to ACIT.

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

$$h_1(x) \stackrel{\mathrm{def}}{=} p_1(x) - x \equiv 0 \tag{1}$$

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

$$h_i(X) \stackrel{\text{def}}{=} p_i(X) - \sum_{j=1}^{i} x_{1,j} p_{i-1}(X_j) \equiv 0, \tag{2}$$

where X_j is the jth minor of the matrix X along the first row.

Equivalently, in order to verify that $C \in \text{ACP}$, we need to test whether

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

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

We conclude the proof by observing that the polynomial h is computable by an arithmetic circuit of size $\mathsf{poly}(|C|)$, since every h_i is. □

Corollary 12. *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 Lemma 11, 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. □

We also obtain the following version of Corollary 12.

Corollary 13. *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 $f \equiv g\text{Perm}$. Thus, for any input integer matrix M such that $g(M) \neq 0$, we have $f(M)/g(M) = \text{Perm}(M)$. 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 [50]; Kaltofen [29, Section 7] provides an alternative proof.

Theorem 14 ([50]). *Given an arithmetic circuit C (over $\mathbb{Q}$ with divisions) of size s computing a degree d polynomial $f(x_1, \ldots, x_n)$, and a point $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ 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 15. *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 $C_1^n \equiv c\text{Perm}$ 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 6. It follows by Lemma 7 that there is a tuple of integers $(a_1, \ldots, a_n)$ 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 14 a new $\mathsf{poly}(n)$-size circuit C' for Perm such that the numerator of C' computes an integer polynomial $h(x_1, \ldots, x_n)$ and the denominator computes an integer constant $c \neq 0$. Both the numerator and the denominator of C' are computable by division-free arithmetic circuits over $\mathbb{Z}$ of size $\mathsf{poly}(|C'|)$. □

Now, 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 $C_1(X) \equiv c\text{Perm}(X)$ 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 cPerm over $\mathbb{Z}$, then each C^i computes cPerm 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 $C_1(X) \equiv c\text{Perm}(X)$ over $\mathbb{Z}$.

Finally, once we successfully verified that a given circuit C_1 computes cPerm, we would like to be able to compute $\text{Perm}(M)$ 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 $\text{Perm}(M)$. 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 [43]), 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 12.

Theorem 16. *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 15, 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 $C_1 \equiv c\text{Perm}$. We can test whether $C_1(X) \equiv c\text{Perm}(X)$ 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 $C_1(M) = c\text{Perm}(M)$ for every 0-1 matrix M. Moreover, we can compute $\text{Perm}(M)$ 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 $\text{Perm}(M) < m$, for any 0-1 matrix M. □

Similarly, we can prove the following.

Theorem 17. *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* Perm $\in$ NP.

4 Circuit Lower Bounds via Derandomization

4.1 If ACIT can be derandomized

Definition 18. We say that NEXP is *computable by polynomial-size arithmetic circuits* if the following two conditions hold:

1. NEXP $\subset$ 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 NEXP $\cap$ coNEXP rather than NEXP, we will say that NEXP $\cap$ coNEXP is computable by polynomial-size arithmetic circuits.

Theorem 19. *If ACIT over $\mathbb{Z}$ is in* NTIME(2^{n^ϵ}) *for every $\epsilon > 0$, then* NEXP *is* not *computable by polynomial-size arithmetic circuits.*

Proof. Suppose that ACIT $\in \cap_{\epsilon>0}$NTIME(2^{n^ϵ}), and NEXP is computable by polynomial-size arithmetic circuits. It follows by Theorem 4 and Corollary 5 that NEXP = coNEXP and Perm is NEXP-complete. On the other hand, by Theorem 16, we have that Perm $\in$ NTIME(2^{n^ϵ}), for every $\epsilon > 0$. Hence, we obtain by Lemma 3 that coNEXP $\subseteq$ NTIME(2^n). But this 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. □

Assuming stronger derandomization of ACIT, we get the following.

Theorem 20. *If ACIT over $\mathbb{Z}$ is in* NP*, then* NEXP$\cap$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 17, that Perm $\in$ NP. This means, by Lemma 3, that PH = NP = coNP. Hence, by padding, we have that NEXP = coNEXP = NEXP $\cap$ coNEXP. Appealing to Theorem 19 concludes the proof. $\square$

With extra care, we also obtain the "infinitely often" version of Theorem 19.

Theorem 21. *If ACIT over $\mathbb{Z}$ is in* io-NTIME(2^{n^ϵ}) *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 NEXP = EXP and Perm is EXP-complete.

A closer look at the proof of Theorem 16 reveals that if ACIT $\in \cap_{\epsilon>0}$io-NTIME(2^{n^ϵ}), then

$$\text{Perm} \in \text{io-NTIME}(2^{n^\epsilon})/O(\log n),$$

for every $\epsilon > 0$. Indeed, for a given input size n, the output of the many-one reduction of Lemma 11 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 NTIME(2^{n^ϵ}) 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

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

for every $\epsilon > 0$. Recall that, by our assumption, Perm is EXP-complete. This means that every language $L \in$ 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 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

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

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

Finally, to derive a contradiction, we employ an argument from [23]. Note that the existence of a universal Turing machine for NTIME(2^n) and the assumption that NEXP $\subset$ P/poly imply that there is a fixed constant c_0 such that

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

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

Since ACIT over $\mathbb{Z}$ is in coRP by Lemma 8, we immediately obtain the following.

Corollary 22. *If* coRP $\subseteq$ io-NTIME(2^{n^ϵ}) *for every $\epsilon > 0$, then* NEXP *is* not *computable by polynomial-size arithmetic circuits.*

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

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

First, similarly to the proof of Theorem 21, we can show the following.

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

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

The proof of Theorem 25 is immediate by the following result of Valiant [54] (see also [19]).

Theorem 26 ([54]). *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* $\mathrm{Det}(A) \equiv f$.

Thus, if SDIT can be done in nondeterministic subexponential time, then, by Theorem 25, so can AFIT, and hence, by Theorem 24, we conclude that NEXP is not computable by polynomial-size arithmetic formulas. Actually, using the ideas from the proof of Theorem 21 (introducing appropriate advice strings), we can prove the following, slightly stronger, statement.

Theorem 27. *If SDIT is in* io-NTIME(2^{n^ϵ}) *for every $\epsilon > 0$, then* NEXP *is* not *computable by polynomial-size arithmetic formulas.*

By using Corollary 9 and Theorem 27, we obtain the following.

Corollary 28. *If* coRNC $\subseteq$ io-NTIME(2^{n^ϵ}) *for every $\epsilon > 0$, then* NEXP *is* not *computable by polynomial-size arithmetic formulas.*

5 Conditional Derandomization of Polynomial Identity Tests

5.1 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 $g(x_1, \ldots, x_n, y)$ of total degree d over a field $\mathbb{F}$.

FIND: A list of arithmetic circuits which contains a circuit for every polynomial $p(x_1, \ldots, x_n)$ such that

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

We need the following result of Kaltofen.

Theorem 29 ([30]). *There is a probabilistic polynomial-time algorithm that, given an arithmetic circuit of size s computing a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ of total degree at most d, with probability at least $3/4$ outputs the numbers $e_i \geqslant 1$ and irreducible polynomials $h_i \in \mathbb{F}[x_1, \ldots, x_n]$, $1 \leqslant i \leqslant r$, 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 30 (Gauss). *For a field $\mathbb{F}$, let $f(x_1, \ldots, x_n, y) \in \mathbb{F}[x_1, \ldots, x_n, y]$ and $p(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ be any polynomials such that $f(x_1, \ldots, x_n, p(x_1, \ldots, x_n)) \equiv 0$. Then $y - p(x_1, \ldots, x_n)$ is an irreducible factor of $f(x_1, \ldots, x_n, y)$ in the ring $\mathbb{F}[x_1, \ldots, x_n, y]$.*

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

Corollary 31. *The root finding problem for a polynomial $g \in \mathbb{F}[x_1, \ldots, x_n, y]$ of total degree d computable by an arithmetic circuit of size s can be solved probabilistically in time $\mathsf{poly}(s, d, \log|\mathbb{F}|)$; for $\mathbb{F} = \mathbb{Q}$, the running time is $\mathsf{poly}(n, d, a)$, 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 30 and Theorem 29. □

5.2 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 1). We should point out that such a generalization of the NW generator was used earlier by Raz, Reingold, and Vadhan [44] 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 10.
2. Output $(p(a|_{S_1}), \ldots, p(a|_{S_n})) \in \mathbb{F}^n$.

Algorithm 1: 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 10, 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 32. *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 10. 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 31 to conclude that p is computable by an arithmetic circuit with required parameters.

I. Hybrid argument. For $0 \leqslant i \leqslant n$, we define

$$g_i(x_1, \ldots, x_l, y_{i+1}, \ldots, y_n)$$

to be f after the first i variables in f are replaced by the polynomials $p((x_1, \ldots, x_l)|_{S_j})$, for $1 \leqslant j \leqslant i$. Thus,

$$g_0(x_1, \ldots, x_l, y_1, \ldots, y_n) = f(y_1, \ldots, y_n)$$

and

$$g_n(x_1, \ldots, x_l) = f(\mathrm{NW}^p(x_1, \ldots, x_l)).$$

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 7, $g_n \equiv 0$.

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

$$g_i(x_1, \ldots, x_l, y_{i+1}, \ldots, y_n) \not\equiv 0,$$

we can fix the variables $y_{i+2}, \ldots, y_n$ 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 $\bar{g_i}(x_{j_1}, \ldots, x_{j_m}, y_{i+1})$ remains a non-zero polynomial. For notational convenience, let us denote this new polynomial by $g(x_1, \ldots, x_m, y)$.

II. Factoring. Thus, we have that $g(x_1, \ldots, x_m, y) \not\equiv 0$, but $g(x_1, \ldots, x_m, p(x_1, \ldots, x_m)) \equiv 0$. Hence, by Corollary 31, 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 10). 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. □

5.3 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 6.

Theorem 33. *Let $p = \{p_m\}_{m \geqslant 0}$ be a family of exponential-time computable multilinear non-zero polynomials*
$p_m \in \mathbb{Z}[x_1, \ldots, x_m]$ 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 $f_n \in \mathbb{Z}[y_1, \ldots, y_n]$ 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 $s_p(m) \in 2^{m^{\Omega(1)}}$.*

A version of Theorem 33 for the case of finite fields $\mathbb{F}$ also holds.

We can now establish a weak converse of Theorems 24 and 27.

Theorem 34. *If* NEXP *is* not *computable by polynomial-size arithmetic circuits, then both AFIT and SDIT are in* io-$[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$ *for every $\epsilon > 0$.*

The proof of Theorem 34 will use the following result implicit in [23].

Lemma 35 ([23]). *If* NEXP $\not\subset$ P/poly, *then* coRP $\subseteq$ io-$[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$ *for every $\epsilon > 0$.*

of Theorem 34. If Perm is not computable by polynomial-size arithmetic circuits, then both AFIT and SDIT are in io-$\mathsf{TIME}(2^{n^\epsilon})$ for every $\epsilon > 0$, by Theorem 33. On the other hand, if NEXP $\not\subset$ P/poly, then both AFIT and SDIT are in io-$[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$ for every $\epsilon > 0$, by Lemma 35 and the fact that both AFIT and SDIT are coRP problems. □

We should point out that, unlike in the Boolean circuit complexity case (cf. [26]), 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 33 can be proved, along the lines of [27]; the details are omitted.

6 Further Results

Here we state without proofs some other results that can be obtained using our techniques.

Theorem 36. *At least one of the following holds:*

1. Perm *over $\mathbb{Z}$ is not computable by polynomial-size arithmetic circuits, or*
2. $\mathsf{NEXP}^{\mathsf{RP}} \not\subset \mathsf{P/poly}$.

Theorem 36 should be compared with the result of [13] saying that MA-EXP $\not\subset$ P/poly. We show that a (seemingly) smaller class, $\mathsf{NEXP}^{\mathsf{RP}}$ contains a function of superpolynomial (Boolean or arithmetic) complexity.

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

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

Theorem 37 must be contrasted with the result in [8] saying that either BPP $\subseteq$ io-SUBEXP, or EXP $=$ MA. Our theorem achieves either a weaker derandomization, or a stronger collapse.

Finally, we also show that if Low Degree Testing can be done in P, and if BPP $\subseteq$ NSUBEXP, then NEXP $\not\subset$ P/poly.

7 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 the known unconditional derandomization results for RP in a certain "uniform" setting [28] are too weak for Corollary 22 to be applicable.

The results in the present paper do not rule out that ZPP $=$ 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 non-trivial circuit lower bounds.

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

Does the assumption BPP $=$ P imply *Boolean* circuit lower bounds for NEXP? Does the same assumption imply (possibly arithmetic) circuit lower bounds for EXP?

Finally, 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

[1] 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.

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

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

[4] 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).

[5] 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).

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

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

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

[9] 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.

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

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

[12] 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.

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

[14] 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.

[15] 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.

[16] A. 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.

[17] 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.

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

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

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

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

[22] O. 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.

[23] 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).

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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).

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

[30] 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.

[31] A. Klivans and D. 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

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

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

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

[40] R. Paturi, P. Pudlák, M. 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.

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

[42] R. Paturi, M. 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.

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

[44] 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.

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

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

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

[48] 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.

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

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

[51] 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).

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

[53] 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.

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

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

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

[57] A. 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.

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

[59] R. 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.