PSEUDORANDOM GENERATORS, TYPICALLY-CORRECT DERANDOMIZATION, AND CIRCUIT LOWER BOUNDS

Jeff Kinne, Dieter van Melkebeek, and Ronen Shaltiel

Abstract. The area of derandomization attempts to provide efficient deterministic simulations of randomized algorithms in various algorithmic settings. Goldreich and Wigderson introduced a notion of "typically-correct" deterministic simulations, which are allowed to err on few inputs. In this paper we further the study of typically-correct derandomization in two ways.

First, we develop a generic approach for constructing typically-correct derandomizations based on seed-extending pseudorandom generators, which are pseudorandom generators that reveal their seed. We use our approach to obtain both conditional and unconditional typically-correct derandomization results in various algorithmic settings. We show that our technique strictly generalizes an earlier approach by Shaltiel based on randomness extractors, and simplifies the proofs of some known results. We also demonstrate that our approach is applicable in algorithmic settings where earlier work did not apply. For example, we present a typically-correct polynomial-time simulation for every language in BPP based on a hardness assumption that is (seemingly) weaker than the ones used in earlier work.

Second, we investigate whether typically-correct derandomization of BPP implies circuit lower bounds. Extending the work of Kabanets and Impagliazzo for the zero-error case, we establish a positive answer for error rates in the range considered by Goldreich and Wigderson. In doing so, we provide a simpler proof of the zero-error result. Our proof scales better than the original one and does not rely on the result by Impagliazzo, Kabanets, and Wigderson that NEXP having polynomial-size circuits implies that NEXP coincides with EXP.

Keywords. Typically-Correct Derandomization, Pseudorandom Generators, Circuit Lower Bounds, Randomized Algorithms

Subject classification. 68Q10, 68Q15, 68Q17, 68Q25, 03D15

1. Introduction

Randomized Algorithms and Derandomization One of the central topics in the theory of computing deals with the power of randomness – can randomized procedures be efficiently simulated by deterministic ones? In some settings exponential gaps have been established between randomized and deterministic complexity; in some settings efficient derandomizations[1] are known; in others the question remains wide open. The most famous open setting is that of time-bounded computations, i.e., whether BPP=P, or more modestly, whether BPP lies in deterministic subexponential time. A long line of research gives "hardness versus randomness tradeoffs" for this problem (see [Mil01] for an introduction). These are *conditional results* that give derandomizations assuming a hardness assumption (typically circuit lower bounds of some kind), where the efficiency of the derandomization depends on the strength of the hardness assumption. The latter is used to construct an efficient *pseudorandom generator*, which is a deterministic procedure G that stretches a short "seed" s into a longer "pseudorandom string" $G(s)$ with the property that the uniform distribution on pseudorandom strings is computationally indistinguishable from the uniform distribution on all strings. G allows us to derandomize a randomized procedure $A(x, r)$ that takes an input x and a string r of "coin tosses" as follows: We run the pseudorandom generator on all seeds to produce all pseudorandom strings of length $|r|$; for each such pseudorandom string we run A using that pseudorandom string as "coin tosses", and output the majority vote of the answers of A. Note that this derandomization procedure takes time that is exponential in the seed length of the pseudorandom generator. For example, efficient pseudorandom generators with logarithmic seed length imply that BPP=P, whereas subpolynomial seed length only yields simulations of BPP in deterministic subexponential time.

Typically-Correct Derandomization Weaker notions of derandomization have been studied, in which the deterministic simulation is allowed to err on some inputs. Impagliazzo and Wigderson were the first to consider derandomizations that succeed with high probability on any efficiently samplable distribution; related notions have subsequently been investigated in [Kab01, TV07, GSTS03, SU07]. Goldreich and Wigderson [GW02] introduced a weaker notion in which

[1] In this paper the term "derandomization" always refers to "full derandomization", i.e., obtaining equivalent deterministic procedures that do not involve randomness at all.

the deterministic simulation only needs to behave correctly on most inputs of any given length. We refer to such simulations as "typically-correct derandomizations". The hope is to construct typically-correct derandomizations that are more efficient than the best-known everywhere-correct derandomizations, or to construct them under weaker assumptions than the hypotheses needed for everywhere-correct derandomization.

Previous Work on Typically-Correct Derandomization Goldreich and Wigderson [GW02] had the key idea to obtain typically-correct derandomizations by "extracting randomness from the input": extract $r = E(x)$ in a deterministic way such that $B(x) = A(x, E(x))$ behaves correctly on most inputs. If this approach works (as such) and E is efficient, the resulting typically-correct derandomization B has essentially the same complexity as the original randomized procedure A. As no more than n bits of randomness can be extracted from an input of length n, this approach on its own is limited to algorithms A that use at most this many random bits; in combination with pseudorandom generators one can try to handle algorithms that use a larger number of random bits. Goldreich and Wigderson managed to get the approach to work *unconditionally* for logspace algorithms for undirected connectivity, a problem which has been fully derandomized by now [Rei08]. Under a *hardness assumption* that is not known to imply BPP=P, namely that there are functions that are *mildly* hard on average for small circuits with access to an oracle for satisfiability, they showed that BPP has polynomial-time typically-correct derandomizations that err on very few inputs, namely at most a subexponential number. Their construction uses well-known explicit constructions of extractors.

Van Melkebeek and Santhanam [vMS05] gave a different construction that yields the same derandomization conclusion as [GW02] from the (seemingly) weaker *uniform* hardness condition that there are functions in P that are mildly hard on average for efficient Merlin-Arthur protocols. Their argument is an application of the easy-witness method (c.f., [Kab01, IKW02]) combined with very efficient probabilistically checkable proofs for languages in P (c.f., [BFLS91]) and dispersers.

Zimand [Zim08] showed *unconditional* typically-correct derandomizations with polynomial overhead for sublinear-time algorithms, which can be viewed as randomized decision trees that use a sublinear number of random bits. Zimand's approach relies on a notion of randomness extractors called "exposure-resilient extractors" introduced in [Zim06].

Shaltiel [Sha09] described a generic approach to obtain typically-correct derandomization results. Loosely speaking he showed how to construct a typically-correct derandomization for any randomized procedure that uses a

sublinear amount of randomness when given an extractor with exponentially small error that extracts randomness from distributions that are "recognizable by the procedure." We elaborate on Shaltiel's approach in Section 5. Using this approach and "off the shelf" randomness extractors, Shaltiel managed to reproduce Zimand's result for decision trees as well as realize *unconditional* typically-correct derandomizations for 2-party communication protocols and streaming algorithms.

Shaltiel also combined his approach with pseudorandom generator constructions to handle procedures that require a polynomial number of random bits. He obtained typically-correct derandomizations with a polynomially small error rate for randomized algorithms computable by polynomial-sized constant-depth circuits, based on the known hardness of parity for such circuits. He also derived a *conditional* typically-correct derandomization result for BPP under a hardness hypothesis that is incomparable to the Goldreich-Wigderson hypothesis (and is also not known to imply BPP=P), namely that there are functions that are *very* hard on average for small circuits without access to an oracle for satisfiability. The resulting error rate is exponentially small. For both results Shaltiel applies the pseudorandom generators that follow from the hardness versus randomness tradeoffs twice: once to reduce the need for random bits to sublinear, and once to construct the required randomness extractor with exponentially small error. Whereas the first pseudorandom generator application can do with functions that are *mildly* hard on average, the second one requires functions that are *very* hard on average.

Our Approach In this paper we develop an alternative generic approach for constructing typically-correct derandomizations. The approach builds on "seed-extending pseudorandom generators" rather than "extractors". A seed-extending pseudorandom generator is a generator G which outputs the seed as part of the pseudorandom string, i.e., $G(s) = (s, E(s))$ for some function E.[2]

An immediate question is whether seed-extending pseudorandom generators exist. In the cryptographic setting they do *not*. This is because in that setting the adversary is allowed more computational resources than the generator itself and can therefore distinguish the pseudorandom distribution $(s, E(S))$ for uniform s from the true uniform distribution, namely by checking whether its input (s, r) satisfies $r = E(s)$. In the setting of derandomization, however, the generator can use more computational resources than the adversary, and the

[2]Borrowing from the similar notion of "strong extractors" in the extractor literature, such pseudorandom generators have been termed "strong" in earlier papers. In coding-theoretic terms, they could also be called "systematic". However, we find the term "seed-extending" more informative.

adversary cannot simply execute the generator. In fact, several pseudorandom generators aimed at derandomization can be made seed-extending, in particular, the well-known Nisan-Wigderson pseudorandom generator construction [NW94].

We show that whenever a seed-extending pseudorandom generator passes certain statistical tests defined by the randomized procedure $A(x, r)$, the deterministic procedure $B(x) = A(x, E(x))$ forms a typically-correct derandomization of A, where the error rate depends on the error probability of the original randomized algorithm and on the error of the pseudorandom generator.

Note that this approach differs from the typical use of pseudorandom generators in derandomization, where the pseudorandom generator G is run on every seed. As the latter induces a time overhead that is exponential in the seed length, one aims for pseudorandom generators that are computable in time exponential in the seed length. A polynomial-time simulation is achieved only in the case of logarithmic seed lengths. In contrast, we run G *only once*, namely with the input x of the randomized algorithm as the seed. We use the pseudorandom generator to *select* one "coin toss sequence" $r = E(x)$ on which we run the randomized algorithm. As opposed to the traditional derandomization setting, our approach benefits from pseudorandom generators that are computable in time less than exponential in the seed length. With a pseudorandom generator computable in time polynomial in the output length, we obtain nontrivial polynomial-time typically-correct derandomizations even when the seed length is just subpolynomial.

Our approach has the advantage of being more direct than the one of [Sha09], in the sense that it derandomizes the algorithm A in "one shot". More importantly, it obviates the second use of pseudorandom generators in Shaltiel's approach and allows us to start from the (seemingly) *weaker assumption* that there are functions which are *mildly* hard on average for small circuits without access to an oracle for satisfiability.

While our assumption is (seemingly) weaker than both the one in [GW02] and the one in [Sha09], the error rate of our typically-correct derandomizations is only polynomially small. We can decrease the error rate by strengthening the hardness assumption. Under the same hardness assumption as [Sha09] our approach matches the exponentially small error rate in that paper. [vMS05] achieves a smaller error rate from a hardness assumption that is incomparable to ours – although it is plausible that P being hard on average for fixed-polynomial time Merlin-Arthur protocols implies being hard on average for fixed-polynomial size deterministic circuits, this implication remains open.

We can similarly relax the hardness assumption in a host of other settings.

In some cases this allows us to establish new *unconditional* typically-correct derandomizations, namely for models where functions that are *very* hard on average are not known but functions which are only *mildly* hard on average are known unconditionally.

We also determine the precise relationship between our approach and Shaltiel's. We show that in the range of exponentially small error rates, "extractors for recognizable distributions" are equivalent to seed-extending pseudorandom generators that pass the statistical tests we need. This means that all the aforementioned results of [Sha09] can also be obtained by interpreting the extractors used in [Sha09] as seed-extending pseudorandom generators and then using our new approach. We can also handle situations where [Sha09] does not apply, and therefore our approach is more generic.

Typically-Correct Derandomization and Circuit Lower Bounds Kabanets and Impagliazzo [KI04] showed that subexponential-time derandomizations of BPP imply circuit lower bounds that seem beyond the scope of current techniques. We ask whether subexponential-time typically-correct derandomizations imply such lower bounds. Another contribution of our paper is an affirmative answer in the case of the error rates considered by Goldreich and Wigderson.

Our result is a strengthening of [KI04] from the everywhere-correct setting to the typically-correct setting. In developing it, we also obtain a simpler proof for the everywhere-correct setting. Our proof scales better than the one in [KI04], yields the same lower bound for a smaller class, and does not rely on the result from [IKW02] that NEXP having polynomial-size circuits implies that NEXP coincides with EXP.

The fact that typically-correct derandomization of BPP with very low error rates implies circuit lower bounds can be viewed as evidence that the former will be difficult to establish. It remains open whether typically-correct derandomization of BPP with higher error rates implies circuit lower bounds. However, we provide a different type of indication that establishing such weaker derandomization of BPP may also be difficult, namely that it cannot be established through an argument that algebrizes – a notion developed by Aaronson and Wigderson [AW09] that includes both relativizing proof techniques as well as techniques based on arithmetization.

Organization We start Section 2 with the formal definitions of the notions used throughout the rest of the paper, and the key lemma that shows how seed-extending pseudorandom generators yield typically-correct derandomizations. In Sections 3 and 4 we present our conditional and unconditional results obtained by applying our approach using the Nisan-Wigderson pseudorandom

generator construction. In Section 5 we compare our approach with Shaltiel's extractor-based approach. In Section 6 we develop our results on circuit lower bounds that follow from typically-correct and everywhere-correct derandomization of BPP, and in Section 7 we show that typically-correct derandomization of BPP cannot be established through an algebrizing argument.

2. Typically-Correct Derandomization And the PRG Approach

In this section we introduce notation and terminology used throughout the paper, state and prove the key lemma showing that seed-extending pseudorandom generators yield typically-correct derandomization, and introduce the seed-extending pseudorandom generator construction used for most of our results.

2.1. Notation and Concepts. We view a randomized algorithm as defined by a deterministic algorithm $A(x, r)$ where x denotes the input and r the string of "coin tosses". We typically restrict our attention to one input length n, in which case A becomes a function $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ where m represents the number of random bits that A uses on inputs of length n. We say that $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ *computes* a function $L : \{0,1\}^n \to \{0,1\}$ with error ρ if for every $x \in \{0,1\}^n$, $\Pr_{R \leftarrow U_m}[A(x,R) \neq L(x)] \leq \rho$, where U_m denotes the uniform distribution over $\{0,1\}^m$ and $R \leftarrow U_m$ denotes that R is a random variable with distribution U_m. We say that the randomized algorithm A *computes* a language L with error $\rho(\cdot)$, if for every input length n, the function induced by A computes the function induced by L with error $\rho(n)$.

Given a randomized algorithm A for L, our goal is to construct a deterministic algorithm B of complexity comparable to A that is typically correct for L. By the latter we mean that B and L agree on most inputs of any given length, or equivalently, that for any input length the relative Hamming distance between the functions induced by B and L is small.

Definition 2.1 (typically-correct behavior). *Let $L : \{0,1\}^n \to \{0,1\}$ be a function. We say that a function $B : \{0,1\}^n \to \{0,1\}$ is* within distance δ *of L if $\Pr_{X \leftarrow U_n}[B(X) \neq L(X)] \leq \delta$. We say that an algorithm* computes *a language L* to within $\delta(\cdot)$ *if for every input length n, the function computed by the algorithm is within distance $\delta(n)$ of the function defined by the language at length n. For two classes of languages $\mathcal{C}_1$ and $\mathcal{C}_2$, we say that $\mathcal{C}_1$ is within $\delta(\cdot)$*

of $\mathcal{C}_2$ if for every language $L_1 \in \mathcal{C}_1$ there is a language $L_2 \in \mathcal{C}_2$ that is within $\delta(\cdot)$ of L_1.

In general, a function $G : \{0,1\}^n \to \{0,1\}^\ell$ is ϵ-pseudorandom for a test $T : \{0,1\}^\ell \to \{0,1\}$ if $|\Pr_{R \leftarrow U_\ell}[T(R) = 1] - \Pr_{S \leftarrow U_n}[T(G(S)) = 1]| \leq \epsilon$. In this paper we are dealing with tests $T(x, r)$ that receive two inputs, namely x of length n and r of length m, and with corresponding pseudorandom functions G of the form $G(x) = (x, E(x))$, where x is of length n and $E(x)$ of length m. We call such functions "seed-extending".

DEFINITION 2.2 (seed-extending function). *A function $G : \{0,1\}^n \to \{0,1\}^{n+m}$ is* seed-extending *if it is of the form $G(x) = (x, E(x))$ for some function $E : \{0,1\}^n \to \{0,1\}^m$. We refer to the function E as the extending part of G.*

Note that a seed-extending function G with extending part E is ϵ-pseudorandom for a test $T : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ if

$$|\Pr_{X \leftarrow U_n, R \leftarrow U_m}[T(X, R) = 1] - \Pr_{X \leftarrow U_n}[T(X, E(X)) = 1]| \leq \epsilon. \tag{2.3}$$

A seed-extending $\epsilon(\cdot)$-pseudorandom generator for a family of tests $\mathcal{T}$ is a deterministic algorithm G such that for every input length n, G is a seed-extending $\epsilon(n)$-pseudorandom function for the tests in $\mathcal{T}$ corresponding to input length n.

2.2. The Seed-Extending Pseudorandom Generator Approach. Our key observation is that good seed-extending pseudorandom generators G for certain simple tests based on the algorithm A yield good typically-correct derandomizations of the form $B(x) = A(x, E(x))$. The following lemma states the quantitative relationship.

LEMMA 2.4 (Main Lemma). *Let $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ and $L : \{0,1\}^n \to \{0,1\}$ be functions such that*

$$\Pr_{X \leftarrow U_n, R \leftarrow U_m}[A(X, R) \neq L(X)] \leq \rho. \tag{2.5}$$

Let $G : \{0,1\}^n \to \{0,1\}^{n+m}$ be a seed-extending function with extending part E, and let $B(x) = A(G(x)) = A(x, E(x))$.

(i) If G is ϵ-pseudorandom for the test $T(x, r) = A(x, r) \oplus L(x)$, then B is within distance $\rho + \epsilon$ of L.

(ii) If G is ϵ-pseudorandom for tests of the form $T_{r'}(x,r) = A(x,r) \oplus A(x,r')$ for all $r' \in \{0,1\}^m$, then B is within distance $3\rho + \epsilon$ of L.

Note that if A computes L with error ρ then condition (2.5) of the lemma is met. The two parts of the lemma differ in the complexity of the tests and in the error bound. The complexity of the tests plays a critical role for the existence of pseudorandom generators. In the first item the tests use the language L as an oracle, which may result in too high a complexity. In the second item we reduce the complexity of the tests at the cost of introducing non-uniformity and increasing the error bound. The increase in the error bound is often not an issue as we can easily reduce ρ by slightly amplifying the success probability of the original algorithm A before applying the lemma.

Proof (of Lemma 2.4). For the first item, notice that a test of the form $T(x,r) = A(x,r) \oplus L(x)$ passes iff $A(x,r) \neq L(x)$. If G is ϵ-pseudorandom for T then $|\Pr_{X \leftarrow U_n}[A(X,E(X)) \neq L(X)] - \Pr_{X \leftarrow U_n, R \leftarrow U_m}[A(X,R) \neq L(X)]| \leq \epsilon$. By assumption the latter probability is at most ρ, so $\Pr_{X \leftarrow U_n}[A(X,E(X)) \neq L(X)] \leq \rho + \epsilon$.

For the second item, pick a string r' that minimizes $\Pr_{X \leftarrow U_n}[A(X,r') \neq L(X)]$. An averaging argument shows that the latter probability is at most ρ. By the pseudorandomness of G, we have

$$|\Pr_{X \leftarrow U_n}[A(X,E(X)) \neq A(X,r')] - \Pr_{X \leftarrow U_n, R \leftarrow U_m}[A(X,R) \neq A(X,r')]| \leq \epsilon. \quad (2.6)$$

As $\Pr_{X \leftarrow U_n, R \leftarrow U_m}[A(X,R) \neq L(X)] \leq \rho$ and $\Pr_{X \leftarrow U_n}[A(X,r') \neq L(X)] \leq \rho$, the second term of (2.6) is at most 2ρ, so $\Pr_{X \leftarrow U_n}[A(X,E(X)) \neq A(X,r')] \leq 2\rho + \epsilon$. Using again that $\Pr_{X \leftarrow U_n}[A(X,r') \neq L(X)] \leq \rho$, we conclude that $\Pr_{X \leftarrow U_n}[A(X,E(X)) \neq L(X)] \leq 3\rho + \epsilon$. □

2.3. Hardness-Based Constructions of Seed-Extending Generators. Some of the constructions of pseudorandom generators in the literature that are geared towards derandomization are seed-extending or can be easily modified to become seed-extending. The generators that we consider are hardness-based, i.e., they are procedures G with access to an oracle for a language H such that the function G_H they compute is pseudorandom for a given class of tests as long as the language H is hard for a related class of algorithms. We first define the notion of hardness we need and then discuss the hardness-based pseudorandom generator we use, namely a seed-extending variant of the Nisan-Wigderson construction.

DEFINITION 2.7 (hardness on average). *A language L is $\delta(\cdot)$-hard for a class of algorithms $\mathcal{A}$ if no $A \in \mathcal{A}$ is within distance $\delta(n)$ of L for infinitely many input lengths n.*

Notice that worst-case hardness corresponds to setting $\delta(n) = \frac{1}{2^n}$. We use the term "mild hardness" when $\delta(n) = \frac{1}{n^c}$ for some constant $c > 0$, and the term "very high hardness" when $\delta(n) = \frac{1}{2} - \frac{1}{2^{n^\epsilon}}$ for some constant $\epsilon > 0$.

For many of our results the relevant class of algorithms $\mathcal{A}$ are circuits or branching programs of a certain size. We measure the size of a circuit or branching program by the string length of its standard description as a labeled directed acyclic graph; up to a logarithmic factor, this measure corresponds to the number of connections in the circuit or branching program. We use the notation $\text{SIZE}(s)$ to refer to Boolean circuits of size s, $\text{SIZE}^O(s)$ to refer to Boolean circuits of size s that have access to oracle gates for the language O, and $\text{BP-SIZE}(s)$ to refer to branching programs of size s.

For circuits and branching programs, hardness can be amplified using the XOR lemma. Several versions of the XOR lemma exist (see [GNW95] for an overview); the following instantiation for circuits suffices for our purposes.

LEMMA 2.8 (XOR Lemma for circuits [Imp95]). *Let $H : \{0,1\}^n \to \{0,1\}$ be a language and define $H' : \{0,1\}^{k \cdot n} \to \{0,1\}$ by $H'(x_1, ..., x_k) = H(x_1) \oplus H(x_2) \oplus ... \oplus H(x_k)$. For any $\gamma > 0$, if H is δ-hard for size s circuits at input length n, then H' is δ'-hard for size s' circuits at input length $k \cdot n$, where $\delta' = \frac{1}{2} - (1-\delta)^k - \gamma$ and $s' = \Omega(\frac{\gamma^2}{\log(1/(\delta\gamma))}) \cdot s$.*

Nisan and Wigderson [NW94] described a hardness-based pseudorandom generator construction that can be applied in a wide variety of algorithmic settings. We use a seed-extending variant of the Nisan-Wigderson construction for all of our results in Sections 3 and 4. We state the properties that we need for the algorithmic setting of circuits in the following lemma. For completeness, we review the Nisan-Wigderson construction in the appendix and in particular verify that it can be made *seed-extending* in the way stated next. We also refer to the appendix for remarks on algorithmic settings other than circuits.

LEMMA 2.9 (seed-extending NW-generator for circuits [NW94]). *Let n and m be positive integers and $H : \{0,1\}^{\lfloor\sqrt{n}/2\rfloor} \to \{0,1\}$ a function. There is a seed-extending function $\text{NW}_{H;n,m} : \{0,1\}^n \to \{0,1\}^{n+m}$ with the following properties.*

(i) If H is $(\frac{1}{2} - \frac{\epsilon}{m})$-hard at input length $\lfloor\sqrt{n}/2\rfloor$ for circuits of size $s + m \cdot 2^{O(\log m/\log n)}$ and depth $d+1$ then $\text{NW}_{H;n,m}$ is ϵ-pseudorandom for tests

$T : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ *computable by circuits of size* s *and depth* d.

(ii) For each $1 \leq j \leq m$, *the* j^{th} *bit in the extending portion of* $\mathrm{NW}_{H;n,m}(x)$ *is equal to* $H(y_j)$ *for some* y_j *of length* $\lfloor \sqrt{n}/2 \rfloor$; *there is a Turing machine that outputs* y_j *on input* (x, n, m, j) *and that runs in* $O(\log(m+n))$ *space as long as* $m(\cdot)$ *is constructible in that amount of space.*

Some of our typically-correct derandomization results are unconditional because languages of the required hardness to use for H have been proven to exist. Others are conditioned on reasonable but unproven hypotheses regarding the existence of languages H that are hard on average. For the conditional results, we can assume a mildly hard function and use Lemma 2.8 to amplify the hardness to the level required in Lemma 2.9.

We point out that the construction in Lemma 2.9 is almost optimal in the following sense: The existence of an ϵ-pseudorandom generator for circuits of size s implies the existence of a language H that is $(\frac{1}{2} - \epsilon)$-hard at length n for circuits of size $s - O(1)$, namely for H the function that outputs the first bit in the extending portion of G.

Remark Our applications do not benefit from seed-extending pseudorandom generator constructions that recover in a blackbox fashion and are based on *worst-case* rather than average-case hardness. By definition, whenever such a pseudorandom generator $G = G_H$ based on $H : \{0,1\}^\ell \to \{0,1\}$ fails a test (2.3), there exists a small oracle circuit C, say of size s, such that $C^T = H$. This property implies that G_H has to query H in at least $(\frac{1}{2} - \epsilon)2^\ell/s$ positions, as can be argued directly and also follows from [Vio05]. The latter condition rules out the combination of mild hardness levels (say $s = n^{O(1)}$ and $\ell = n^{\Omega(1)}$) and a polynomial running time for G, which we need for our applications.

3. Conditional Results

In this section we obtain a number of typically-correct derandomization results that are conditioned on unproven but reasonable hardness hypotheses. These results are summarized in Figure 3.1.

3.1. Bounded-Error Polynomial Time. The first setting we consider is that of BPP. We use a modest hardness assumption to show that any language in BPP has a polynomial-time deterministic algorithm that errs on a polynomially small fraction of the inputs.

Theorem	Setting	Hardness Assumption	Conclusion
Thm 3.1	BPP=BP.P	P $\frac{1}{n^c}$-hard for SIZE(n^d)	BPP within $\frac{1}{n^c}$ of P
Thm 3.3	BP.$\oplus$P	$\oplus$P $\frac{1}{n^c}$-hard for SIZE$^{\oplus\text{SAT}}(n^d)$	BP.$\oplus$P within $\frac{1}{n^c}$ of $\oplus$P
Thm 3.4	AM=BP.NP	NP $\cap$ coNP $\frac{1}{n^c}$-hard for SIZE$^{\text{SAT}}(n^d)$	AM within $\frac{1}{n^c}$ of NP
Thm 3.6	BP.L	L $\frac{1}{n^c}$-hard for BP-SIZE(n^d)	BP.L within $\frac{1}{n^c}$ of L

Figure 3.1: Our conditional typically-correct derandomization results. Each row states that if the hardness assumption holds for every constant d then the conclusion follows.

THEOREM 3.1 (typically-correct derandomization of BPP). *Let L be a language that is computed by a randomized bounded-error polynomial-time algorithm A. For any positive constant c, there is a positive constant d (depending on c and the running time of A) such that the following holds. If there is a language H in P that is $\frac{1}{n^c}$-hard for circuits of size n^d, then there is a deterministic polynomial-time algorithm B that computes L to within $\frac{1}{n^c}$.*

We compare Theorem 3.1 to previous conditional typically-correct derandomization results for BPP after the proof.

PROOF (of Theorem 3.1). Let A be a polynomial-time randomized bounded-error algorithm computing a language L, and let $c > 0$ be a constant. We obtain the typically-correct deterministic algorithm B by using Item (ii) of Lemma 2.4 with the Nisan-Wigderson construction as the generator. More specifically, we set

$$B(x) = A'(\text{NW}_{H';n,n^b}(x))$$

where A' is an error-reduced version of A that uses n^b random bits for a constant b depending on the running time of A and where H' is the result of applying a certain amount of hardness amplification to H. We now analyze how to set the parameters of the various ingredients and establish the stated properties.

1. Error Reduction.
   To keep the error term 3ρ from invoking Item (ii) of Lemma 2.4 less than $\frac{1}{2n^c}$, we let A' take the majority vote of $O(\log n)$ independent trials of A so that A' has error at most $\frac{1}{6n^c}$.

2. Nisan-Wigderson construction.
   Setting $\rho = \frac{1}{6n^c}$ in Item (ii) of Lemma 2.4, B computes L to within distance $\frac{1}{n^c}$ if $\text{NW}_{H';n,n^b}$ is $\frac{1}{2n^c}$-pseudorandom against tests $T_{r'}$ of the form $T_{r'}(x,r) = A'(x,r) \oplus A'(x,r')$ for r' an arbitrary string of length n^b. Using

the standard reduction from Turing machines with advice to circuits, the tests $T_{r'}$ are circuits of size $O(n^{2b})$ for some constant b depending on the running time of A. By Lemma 2.9, $\mathrm{NW}_{H';n,n^b}$ is $\frac{1}{2n^c}$-pseudorandom against the tests $T_{r'}$ if H' is $(\frac{1}{2} - \frac{1}{2n^{c+b}})$-hard for circuits of size $O(n^{2b})$ on inputs of length $\lfloor\sqrt{n}/2\rfloor$. Thus a sufficient hardness condition for H' is to be $(\frac{1}{2} - \frac{1}{n^a})$-hard for circuits of size n^a on inputs of length n, for $a = 2\max(c+b, 2b) + 1$.

3. XOR Lemma.
Let $H : \{0,1\}^n \to \{0,1\}$ be $\frac{1}{n^c}$-hard for circuits of size n^d and define $H' : \{0,1\}^{k \cdot n} \to \{0,1\}$ by $H'(x_1, ..., x_k) = H(x_1) \oplus H(x_1) \oplus ... \oplus H(x_k)$. By Lemma 2.8, H' is $(\frac{1}{2} - \frac{1}{n^a})$-hard for circuits of size n^a if we can choose k and γ such that (1) $(1 - \frac{1}{n^c})^k + \gamma \leq \frac{1}{(nk)^a}$ and (2) $n^d \cdot (\frac{\gamma^2}{\log(n^c/\gamma)}) \geq (nk)^a$. To satisfy (1), we choose $\gamma = \frac{1}{2(nk)^a}$ and set $k = n^{c+1}$ to ensure that for sufficiently large n, $(1 - \frac{1}{n^c})^k \leq e^{-k/n^c} = e^{-n} \leq \frac{1}{2(nk)^a}$. With these choices, (2) simplifies to $n^d \geq 8n^{3(c+2)a} \log(2n^{c+(c+2)a})$ which can be satisfied by choosing $d = 3(c+2)a + 1$.

This establishes the correctness of B, i.e., B computes L to within $\frac{1}{n^c}$ provided H is $\frac{1}{n^c}$-hard for circuits of size n^d. Now consider the complexity of B. By Item (ii) of Lemma 2.9, $\mathrm{NW}_{H';n,n^b}$ is computable in time polynomial in n provided H' is, which in turn is computable in time polynomial in n provided H is. □

We now compare Theorem 3.1 to previous conditional typically-correct derandomization results for BPP. Figure 3.2 lists the hardness assumptions and error rates of Theorem 3.1 and previous works for the setting of polynomial-time simulations of BPP. We claim that the hardness assumption in Theorem 3.1 is implied by all the other ones in Figure 3.2, and is in this sense (seemingly) weaker .

Claim 3.2. *The hardness assumption in Theorem 3.1 is implied by the hardness assumptions of each of the rows in Figure 3.2.*

Proof. *Second and Fourth Row* Our hardness assumption is implied by the ones used by [GW02] and [Sha09] for obvious reasons.

First Row The [IW97] assumption implies a positive constant ϵ' such that E is $(\frac{1}{2} - 2^{-\epsilon' n})$-hard for $\mathrm{SIZE}(2^{\epsilon' n})$ by the known worst-case to average-case reductions for E [IW97]. For any constants c and d, a padding argument shows that this in turn implies that P is $(\frac{1}{2} - n^{-c})$-hard for $\mathrm{SIZE}(n^d)$, which is in

	Hardness Assumption	# Mistakes
[IW97]	E is $1/2^n$-hard for SIZE($2^{\Omega(n)}$)	0
[GW02]	P is 1/3-hard for $\text{SIZE}^{\text{SAT}}(n^d)$	2^{n^ϵ}
[vMS05]	P is 1/3-hard for promise-MATIME(n^d)	2^{n^ϵ}
[Sha09]	P is $\frac{1}{2} - 2^{-n^{\Omega(1)}}$-hard for SIZE($n^d$)	$\frac{2^n}{2^{n^{\Omega(1)}}}$
Thm 3.1	P is $1/n^c$-hard for SIZE(n^d)	$\frac{2^n}{n^c}$

Figure 3.2: Comparison of hardness assumptions that give polynomial-time deterministic simulations of BPP languages. Each row states that if the hardness condition holds for every constant d then every language in BPP has a poly-time deterministic simulation that agrees with it on all except the number of inputs given in the last column for each input length n.

fact stronger than the assumption in the last row. More precisely, we argue the contrapositive: If every language in P can be computed to within distance $(\frac{1}{2} - n^{-c})$ by a circuit of size n^d for infinitely many input lengths n, then every language in E can be computed to within distance $(\frac{1}{2} - 2^{-\epsilon' n})$ by circuits of size $2^{\epsilon' n}$ for infinitely many input lengths n.

Let $L \in \text{DTIME}(2^{an})$ for some positive constant a, and define for any positive integer k a padded version $L_{pad,k}$ of L decidable in $\text{DTIME}(N^k)$ by setting $L_{pad,k} = \{(x, y) \mid x \in L \text{ and } \lceil 2^{a|x|/k} \rceil \leq |x| + |y| < \lceil 2^{a(|x|+1)/k} \rceil\}$, where we use N to denote the length of the input to $L_{pad,k}$. Then, by assumption, there exists a family of circuits C_N of size N^d that infinitely often computes $L_{pad,k}$ to within distance $\frac{1}{2} - N^{-c}$. An averaging argument shows that for each such input length N, there exists a setting for y such that this circuit $C_N(\cdot, y)$ computes L correctly on inputs of length $n \approx k(\log N)/a$ with the same advantage. Setting k a large enough constant so that $2^{\epsilon' n} \geq \max(N^c, N^d)$ and fixing y as above yields circuits of size at most $2^{\epsilon' n}$ computing L to within distance $\frac{1}{2} - 2^{-\epsilon' n}$ on infinitely many input lengths n.

Third Row We show that if P is 1/3-hard for promise-Merlin-Arthur protocols running in time $n^{d'}$ for all constants d', then P is $\frac{1}{n^c}$-hard for circuits of size n^d for all constants c and d. We argue the contrapositive, demonstrating efficient promise-Merlin-Arthur protocols for languages in P that are within 1/3 infinitely often assuming P can be approximated well by small circuits infinitely often.

Let $L \in \text{P}$. We use the probabilistically checkable proofs of [BFLS91] that can be verified in polylogarithmic time (and are a key component of the ar-

gument of [vMS05]). Because two slightly different inputs may be indistinguishable to a sublinear-time verifier, [BFLS91] first encodes the input by an efficiently computable error-correcting code E. We consider the language $E(L) = \{E(x) \mid x \in L\}$ and the PCP for $E(L)$ with the promise that the input is a valid encoding under E. The proof system of [BFLS91] for this promise problem has the following properties, where n denotes the original input length for L. (i) The correct proof is computable in time polynomial in n. (ii) The verifier runs in polylog(n) time, querying a polylogarithmic number of locations in the input and the proof. (iii) The proof system is tolerant to errors in the proof itself: given a proof within distance 15% of a correct proof, the proof system still accepts with high probability. For our purposes, it is more convenient to consider a padded version of the [BFLS91] proof system in which we concatenate $2^{\Theta(n/\log n)}$ copies of the original proof such that the new proofs are of length exactly 2^n, and the verifier picks one of the copies uniformly at random to run the original proof system on. (If the verifier picks the last – possibly incomplete – copy and makes a query outside of the proof, the verifier accepts.) This modified proof system has the following properties on inputs of length n. (i') Each bit of a correct proof can be computed in time polynomial in n. (ii') The verifier runs in time $O(n)$. (iii') Given a proof that is within distance, say 1%, of a correct proof, the proof system accepts with high probability.

The basic idea is to construct an efficient promise-Merlin-Arthur protocol that approximates L to within 1/3 by running the modified [BFLS91] PCP for $E(L)$ on the encoded input. The prover sends a small circuit that, for many inputs $x \in L$, approximates the correct PCP proof for the membership of $E(x)$ to the promise problem $E(L)$, and the verifier uses that circuit to compute the bits of the proof it needs. The soundness property, completeness property, and property (iii') guarantee that even though the circuit may make a small fraction of mistakes, the protocol behaves correctly on most inputs.

We now provide the details. Let L_{proof} be a language encoding correct proofs in the modified [BFLS91] protocol for $E(L)$, namely $L_{proof} = \{(x, i, b) \mid x \in L, |x| = |i|, |b| \leq 1$, and the i-th bit of the correct proof for $E(x)$ being in $E(L)$ is 1 $\}$, where i is written in binary but may have leading zeroes. The role of b in the definition of L_{proof} is to make sure that all input lengths N for L_{proof} are useful for deciding some related input length n for L, namely $n = \lfloor N/2 \rfloor$. Note that $L_{proof} \in$ P, so by assumption there is a family of circuits of size N^d that computes L_{proof} to within distance $\frac{1}{N^c}$ on infinitely many input lengths N. For each such input length N, we claim that the following protocol behaves like a Merlin-Arthur protocol for L on most inputs of length $n = \lfloor N/2 \rfloor$. The verifier computes the error-correcting encoding $E(x)$ of the input x and runs

the modified [BFLS91] proof system for the promise problem $E(L)$ to check that $E(x) \in E(L)$; the prover sends a circuit of size N^d and the parity of N, and the verifier evaluates that circuit on input (x, i, b) when it needs the i-th bit of the proof for $E(x)$ belonging to the promise problem $E(L)$. (If N is odd, the verifier runs the proof system once with $b = 0$ and once with $b = 1$ and accepts if at least one of the runs accepts; if N is even, b is set to the empty string.)

For $x \notin L$, the protocol outputs the correct answer by the soundness of the PCP system for the promise problem. For $x \in L$, the protocol outputs a correct value if the circuit sent by the prover is correct on at least 99% of the locations in the proof of membership for $E(x)$ to the promise problem $E(L)$, by property (iii') of the modified [BFLS91] proof system. For input lengths where the prover's circuit is correct on all but $\frac{1}{N^c}$ of the proof bits, an averaging argument shows the circuit is correct on 99% of the locations for the proofs of all but a $O(1/N^c)$ fraction of inputs x of length n and at least one value of b. Thus, the protocol behaves like a Merlin-Arthur protocol for L on all but a fraction $O(1/n^c) < 1/3$ of the inputs of length n.

The running time of the protocol is the time to compute $E(x)$ (which is a fixed polynomial) plus the time to run the modified PCP for the promise problem, answering each query by evaluating a circuit of size $O(n^d)$. Overall, this gives a running time of $O(n^{d+1}\text{polylog}(n))$ for d sufficiently large. □

We also point out that plugging our assumption into the hardness versus randomness tradeoffs of [NW94] (on which [IW97] is based) gives the incomparable result that BPP is in deterministic subexponential time, i.e., in time 2^{n^ϵ} for every positive constant ϵ. Note that the latter statement already follows if we replace P in the hardness assumption by E=DTIME($2^{O(n)}$). In contrast, the approaches to typically-correct derandomization in [GW02], [vMS05], and [Sha09] do not yield any typically-correct derandomization when starting from the modest hardness assumption that we use. Under their respective stronger assumptions, these papers do yield typically-correct algorithms that are closer to L. We remark that we can match the distance in [Sha09] if we are allowed to assume the same hardness hypothesis.

3.2. Extensions to Other Algorithmic Settings. [KvM02] observed that the Nisan-Wigderson generator can be used to give hardness versus randomness tradeoff results in a number of different algorithmic settings. This approach also works within our typically-correct derandomization framework. In this section we discuss the last three applications listed in Figure 3.1.

3.2.1. BP.⊕P Algorithms. Our conditional results for BP.⊕P algorithms and Arthur-Merlin protocols rely on the fact that all the ingredients in the proof of Theorem 3.1 relativize: error reduction using majority voting, the Nisan-Wigderson construction relativizes (see the remark after the proof of Lemma 2.9 in the appendix), the XOR Lemma, and our main lemma. Thus, we have the following as a corollary to the proof of Theorem 3.1.

THEOREM 3.3 (relativized version of Theorem 3.1). *Let O be any language, and let L be a language that is computed by a randomized bounded-error polynomial-time algorithm A that has oracle access to O. For any positive constant c, there is a positive constant d (depending on c and the running time of A) such that the following holds. If H is a language that is $\frac{1}{n^c}$-hard for circuits of size n^d that have access to O oracle gates, then there is a polynomial-time algorithm B that uses oracle access to both H and O that computes L to within $\frac{1}{n^c}$.*

Theorem 3.3 immediately yields a typically-correct derandomization result for the class BP.⊕P, a class that is of interest as a key step in the result that any language within the polynomial hierarchy can be solved with an oracle for counting [Tod91]. Recall that a language L in BP.⊕P is defined by a deterministic procedure A that on input (x, R, z) with $|x| = n$ runs in time n^k for some constant k and such that

(i) for every $x \in L$, $\Pr_{R \leftarrow U_{n^k}}[|\{z \in \{0,1\}^{n^k} \text{ s.t. } A(x,R,z) = 1\}| \equiv 1 \pmod 2]$ is at least $\frac{2}{3}$, and

(ii) for every $x \notin L$, $\Pr_{R \leftarrow U_{n^k}}[|\{z \in \{0,1\}^{n^k} \text{ s.t. } A(x,R,z) = 1\}| \equiv 1 \pmod 2]$ is at most $\frac{1}{3}$.

⊕SAT is a natural ⊕P-complete language consisting of Boolean formulae that have an odd number of satisfying assignments. Applying Theorem 3.3 with the oracle O set to ⊕SAT, requiring the hard function H to lie within ⊕P, and using the facts that BP.⊕P$=$ BPP$^{\oplus\text{SAT}}$ and P$^{\oplus\text{SAT}} = \oplus$P, we obtain the typically-correct derandomization result for BP.⊕P algorithms listed in Figure 3.1.

3.2.2. Arthur-Merlin Protocols. Recall that a language L is decidable by a polynomial-time *Arthur-Merlin protocol* (AM) if there is a deterministic procedure V (the verification predicate) that on input (x, R, z) with $|x| = n$ runs in time n^k for some constant k such that

(i) for every $x \in L$, $\Pr_{R \leftarrow U_{n^k}}[\exists z \in \{0,1\}^{n^k} V(x,R,z) = 1] \geq \frac{2}{3}$, and

(ii) for every $x \notin L$, $\Pr_{R \leftarrow U_{n^k}}[\exists z \in \{0,1\}^{n^k} V(x,R,z) = 1] \leq \frac{1}{3}$.

Notice that we can view AM as BP.NP and that if we remove the randomness from the above definition we would be left with an NP predicate. Thus, derandomizing AM means obtaining simulations of AM on nondeterministic machines. Using the fact that AM = BP.NP $\subseteq$ $\mathrm{BPP}^{\mathrm{NP}}$, an immediate application of Theorem 3.3 with the oracle O set to SAT yields a conditional typically-correct derandomization of AM into $\mathrm{P}^{\mathrm{SAT}}$ under the assumption of a language $H \in \mathrm{P}^{\mathrm{NP}}$ that is mildly hard on average for polynomial-size circuits that have access to SAT oracle gates. By looking more closely at the proof of Theorem 3.3 and strengthening the assumption on the complexity of the hard function H, namely to NP $\cap$ coNP, we obtain a conditional typically-correct derandomization of AM into NP.

THEOREM 3.4 (typically-correct derandomization of AM). *Let L be a language computable by a polynomial-time Arthur-Merlin protocol. For every constant $c > 0$ there is a constant d such that if* NP $\cap$ coNP *contains a language H that is $\frac{1}{n^c}$-hard for circuits of size n^d that have access to* SAT *oracle gates, then there is a nondeterministic polynomial-time algorithm B that computes L to within $\frac{1}{n^c}$.*

PROOF. We follow the proofs of Theorems 3.1 and 3.3. We define B as the language of all inputs x for which $\exists z \in \{0,1\}^{n^b} V'(\mathrm{NW}_{H';n,n^b}(x), z)$, where V' is an error-reduced version of the verification predicate V that uses n^b random bits for a constant b depending on the running time of V and where H' is the result of applying some amount of hardness amplification to the assumed hard function H. We need to verify both the *correctness* and the *complexity* of B. Correctness follows by Theorem 3.3 and the fact that AM $\subseteq$ $\mathrm{BPP}^{\mathrm{SAT}}$, as discussed above.

As for the complexity of B, we first point out that error-reduction can be performed within AM using parallel repetition, so that an AM protocol with verification procedure V' and reduced error can be given. Now,

$$B(x) = 1 \Leftrightarrow \exists z \in \{0,1\}^{n^b} V'(x, H'(y_1), ..., H'(y_{n^b}), z), \tag{3.5}$$

where each y_1, ..., y_{n^b} is some efficiently computable substring of x of length $\lfloor \sqrt{n}/2 \rfloor$. By Item (ii) of Lemma 2.9 and the fact that $H \in$ NP $\cap$ coNP, $V'(x, H'(y_1), ..., H'(y_{n^b}), z)$ defines a predicate on (x,z) that is decidable in $\mathrm{P}^{\mathrm{NP} \cap \mathrm{coNP}} = \mathrm{NP} \cap \mathrm{coNP}$, which turns the right-hand side of (3.5) into an NP-predicate on x. Thus, B is in NP.

□

Remark In the context of everywhere-correct derandomization it is known that hardness against nondeterministic circuits (rather than circuits with access to a satisfiability oracle) is sufficient to derandomize Arthur-Merlin protocols [MV05, SU05]. In fact, [SU06] shows that the assumption that EXP contains a language that cannot be computed by small nondeterministic circuits implies that EXP contains a language that cannot be computed by small circuits that make non-adaptive calls to a satisfiability oracle. In the context of typically-correct derandomization we need hard languages that can be computed in P^{NP} or NP∩coNP and we do not know whether we can replace hardness for circuits with oracle access to satisfiability by hardness for nondeterministic circuits.

3.2.3. Space-Bounded Setting. We obtain the final result listed in Figure 3.1 by observing that the proof of Theorem 3.1 follows through in the setting of derandomizing BP.L algorithms – randomized algorithms that run in logarithmic space and are allowed two-way access to their random bits [Nis93].[3]

THEOREM 3.6 (typically-correct derandomization of BP.L). *Let L be a language that is computed by a randomized bounded-error log-space algorithm A that has two-way access to its random bits. For any positive constant c, there is a positive constant d (depending on c and the space usage of A) such that the following holds. If there is a language H computable in logarithmic space that is $\frac{1}{n^c}$-hard for branching programs of size n^d, then there is a deterministic log-space algorithm B that computes L to within $\frac{1}{n^c}$.*

PROOF. We follow the same outline as the proof of Theorem 3.1. That is, we define B by $B(x) = A'(\mathrm{NW}_{H';n,n^b}(x))$ where A' is an error-reduced version of A that uses n^b random bits for a constant b depending on the running time of A and where H' is the result of applying the XOR lemma to H. We need to verify the *correctness* and the *complexity* of B.

Correctness follows as in the proof of Theorem 3.1 with two modifications. First, we make use of the remark after the proof of Lemma 2.9 in the appendix to apply the Nisan-Wigderson construction to branching programs. Second, we use a version of the XOR lemma for branching programs, which reads the same as Lemma 2.8 except that we replace "circuits" by "branching programs", and set $\delta' = \frac{1}{2} - (1-\delta)^k - \gamma$ and $s' = \Omega(\frac{\gamma^4}{\log^2(1/(\delta\gamma))}) \cdot s$.

[3] Recall that BP.L algorithms are potentially much more powerful than randomized space-bounded algorithms that are given *one-way* access to their randomness – referred to as BPL algorithms. While it is known that BPL is contained in $\mathrm{DSPACE}(\log^{1.5} n)$ [SZ99], all that is known for BP.L is that BP.L ⊆ BPP ⊆ PSPACE.

As for the complexity of B, we first observe that $\mathrm{NW}_{H';n,n^b}$ is computable in logarithmic space by Item (ii) of Lemma 2.8 and the assumption that H is computable in logarithmic space. As A' is also computable in logarithmic space and B is the composition of A' and $\mathrm{NW}_{H';n,n^b}$, B is computable in logarithmic space. □

4. Unconditional Results

In this section we obtain *unconditional* typically-correct derandomization results in a number of algorithmic settings.

4.1. Constant-Depth Circuits. Nisan [Nis91] used the NW-construction together with the fact that the parity function is $(\frac{1}{2} - \frac{1}{2^{n^{\Omega(1)}}})$-hard for constant-depth circuits [Hås87] to obtain everywhere-correct derandomization of uniform randomized constant-depth circuits ($\mathrm{BP.AC}^0$) by uniform quasipolynomial-size constant-depth circuits. The transformation works for various notions of uniformity, including log-space and polynomial-time uniformity.

[Sha09] obtained a more efficient derandomization of uniform $\mathrm{BP.AC}^0$ in the typically-correct setting, replacing "quasipolynomial-size" by "polynomial-size". The approach of [Sha09] relies on certain extractors that have exponentially small error. We elaborate on the extractor-based approach of [Sha09] in Section 5 and point out that it can only handle randomized algorithms that use a sublinear number of random bits. In order to handle algorithms that use a polynomial number of random bits, [Sha09] first uses Nisan's generator to reduce the randomness of a uniform $\mathrm{BP.AC}^0$ circuit to sublinear and then uses the exponentially strong lower bounds for constant-depth circuits computing parity once more to construct the extractor that is needed.

By using a single application of Nisan's generator along with Lemma 2.4, our approach gives a simpler proof of the typically-correct derandomization results for uniform $\mathrm{BP.AC}^0$ of [Sha09]. As before, the result holds for either log-space or polynomial-time uniformity and shows that for any constant c, uniform $\mathrm{BP.AC}^0$ is within distance $\frac{1}{n^c}$ of uniform AC^0, the class of uniform polynomial-size constant-depth circuits. The error can be further reduced by allowing the deterministic algorithm parity gates: uniform $\mathrm{BP.AC}^0$ is within distance $\frac{1}{2^{n^{\Omega(1)}}}$ of uniform $\mathrm{AC}^0[\oplus]$.

4.2. Constant-Depth Circuits with Few Symmetric Gates. In contrast to the approach of [Sha09], our techniques also yield results in settings where the best-known lower bounds only yield moderate hardness on average.

One such model is that of constant-depth circuits that are allowed a small number of arbitrary symmetric gates, i.e., gates that compute functions which only depend on the Hamming weight of the input, such as parity and majority. In this setting Viola [Vio06] constructed a simple function computable by uniform constant-depth circuits that have access to parity gates that is $(\frac{1}{2} - \frac{1}{s})$-hard for circuits of size s that use $\log s$ symmetric gates, for a function $s = n^{\Omega(\log n)}$. As the approach of [Sha09] requires a hard function with exponentially strong hardness to build a seedless extractor with exponentially small error, that approach cannot make use of this hardness result to achieve derandomization of randomized circuits with few symmetric gates. Our approach *can* exploit these weaker hardness results and gives the following for both log-space and polynomial-time uniformity.

THEOREM 4.1. *Let L be a language and A a uniform randomized circuit of constant depth and polynomial size that uses $o(\log^2 n)$ symmetric gates such that A computes L with error at most ρ. Then there is a uniform deterministic circuit B of constant depth and polynomial size that uses exactly the same symmetric gates as A in addition to a polynomial number of parity gates such that B computes L to within $3\rho + \frac{1}{n^{\Omega(\log n)}}$.*

We point out that the error term 3ρ can be removed using standard error reduction provided A uses even fewer symmetric gates. For example, suppose A computes a language L using $o(\log n)$ symmetric gates and let A' be the randomized algorithm that takes the majority vote of $O(\log n)$ independent trials of A to reduce ρ to $\frac{1}{4n^c}$ for some constant c. Then A' uses $o(\log^2 n)$ symmetric gates and by Theorem 4.1 there is a uniform deterministic polynomial-size constant-depth circuit that uses $o(\log^2 n)$ symmetric gates in addition to a polynomial number of parity gates and computes L to within $\frac{1}{n^c}$.

PROOF (of Theorem 4.1). Let A be a uniform circuit of depth d and size n^b that uses $o(\log^2 n)$ symmetric gates and computes a language L with error at most ρ on every input, for some constants d and b. We obtain the typically-correct deterministic algorithm B by using Item (ii) of Lemma 2.4 with the Nisan-Wigderson construction as the generator, i.e., we set

$$B(x) = A(\mathrm{NW}_{H;n,n^b}(x))$$

for some H. We first explain how to set the parameters and choose the hard language H so as to verify the *correctness* of B – that B computes L to within distance $3\rho + \frac{1}{n^{\Omega(\log n)}}$.

1. Nisan-Wigderson construction.
   By Item (ii) of Lemma 2.4 B computes L to within distance $3\rho + \epsilon$ if $\mathrm{NW}_{H;n,n^b}$ is ϵ-pseudorandom against tests $T_{r'}$ of the form $T_{r'}(x,r) = A(x,r) \oplus A(x,r')$, which are circuits of size $O(n^b)$ and depth $d+1$ that use $o(\log^2 n)$ symmetric gates. In a remark following the proof of Lemma 2.9, we point out that the NW generator is secure with the same parameters given in Item (i) of Lemma 2.9 against circuits T that have access to a certain number of symmetric gates if the hard function H is hard with the same parameters stated in the lemma with respect to circuits that have access to the exact same symmetric gates. In particular, $\mathrm{NW}_{H;n,n^b}$ is ϵ-pseudorandom against the tests $T_{r'}$ if H is $(\frac{1}{2} - \frac{\epsilon}{n^b})$-hard on inputs of length $\lfloor \sqrt{n}/2 \rfloor$ against circuits of size $O(n^b)$ and depth $d+2$ that use $o(\log^2 n)$ symmetric gates.

2. Hard language H.
   [Vio06] exhibits a function H that is computable by log-space uniform linear-size constant-depth circuits that have access to parity gates such that H is $(\frac{1}{2} - \frac{1}{s})$-hard on inputs of length n for circuits of size s and depth $d+2$ that use at most $\log s$ symmetric gates, for $s = n^{\alpha \log n}$ where α is a constant depending on d. Then H has the required hardness provided $\frac{\epsilon}{n^b} \geq \frac{1}{\lfloor \sqrt{n}/2 \rfloor^{\alpha \log(\lfloor \sqrt{n}/2 \rfloor)}}$. We can choose ϵ of the form $\frac{1}{n^{\Omega(\log n)}}$ to satisfy this inequality.

This guarantees the correctness of B. Now consider the complexity of B.

By Item (ii) of Lemma 2.9 and the complexity of computing H stated above, $\mathrm{NW}_{H;n,n^b}$ is computable by a log-space uniform constant-depth polynomial-size circuit that has access to parity gates. Thus B is computable by a circuit as described in the statement of Theorem 4.1 and maintains the uniformity of A (either log-space or polynomial-time). □

4.3. Multi-Party Communication Complexity. Let us first recall the multi-party communication model. We use the number on the forehead model [BNS92], where the input consists of k strings x_1, ..., x_k each of length n such that the j^{th} player sees each string except x_j. For a randomized protocol all players also have read-only access to a publicly shared random string r. The players communicate by taking turns writing messages on a shared blackboard until one of the players stops the protocol and outputs an answer. A randomized protocol A using m bits of public randomness computes a language L with error ρ if for every instance $x = (x_1, ..., x_k)$, $\Pr_{R \leftarrow U_m}[A(x_1, ..., x_k; R) \neq L(x)] \leq \rho$. The communication cost of the protocol is the maximum number of bits written

on the blackboard over all possible inputs x of the above form and random bit sequences R. A protocol is polynomial-time uniform if whenever a player sends a message, that message can be computed in polynomial time as a function of the player's view. We similarly define the notion of log-space uniformity.

[Sha09] proves a typically-correct derandomization result for uniform two-party communication protocols. The proof of [Sha09] is tailored to the two-party case and does not extend to the general case of k-party communication. Using our approach we can handle $k > 2$. We show that every uniform randomized k-party communication protocol has a corresponding uniform deterministic k-party communication protocol that is typically correct and has a communication cost that is larger by a factor roughly equal to the amount of randomness of the original randomized protocol. The following statement holds for both log-space and poly-time uniformity.

THEOREM 4.2. *Let L be a language over k-tuples of n-bit strings and let A be a uniform randomized communication protocol that computes L with error at most ρ using k players, q bits of communication, and m bits of public randomness, with k, q, m, and $\log(1/\epsilon)$ functions computable within the uniformity bounds. There is a positive constant α such that for $q' = \alpha \cdot 4^k \cdot m \cdot (q + \log(m/\epsilon))$ there is a uniform deterministic communication protocol B using k players and q' bits of communication that computes L to within $3\rho + \epsilon$ if $q' \leq n$.*

For $k = 2$, Theorem 4.2 yields a weaker result than that of [Sha09] – which gives a deterministic protocol with communication complexity $O(q + m)$ rather than $O(q \cdot m + m \log m)$. As we explain in Section 5 it is possible to recast the argument of [Sha09] in the terminology of seed-extending pseudorandom generators, and therefore the approach of this paper can also produce the stronger result for $k = 2$.

We point out that the error term 3ρ can be removed by using error reduction. For example, by using randomness-efficient error reduction [CW89, IZ89], for any constant c the randomized protocol A can be replaced with a protocol A' that has error at most $\frac{1}{n^c}$ using $m + O(\log n)$ random bits and $O(q \cdot \log n)$ bits of communication.

PROOF (of Theorem 4.2). Let A be a uniform randomized communication protocol that computes a language L with error at most ρ on every input and uses k players, q bits of communication and m bits of public randomness. We obtain the typically-correct deterministic protocol B by using Item (ii) of Lemma 2.4 with the following seed-extending hardness-based pseudorandom generator $G_{H;n,\ell,m}$. The generator simply partitions its inputs into ℓ disjoint

blocks and applies a hard function H on each block in order to generate the m pseudorandom bits. More precisely, for any $\ell \leq \lfloor n/m \rfloor$ we define $G_{H;n,\ell,m}$ as

$$G_{H;n,\ell,m}(x_1,\ldots,x_k) = (x_1,\ldots,x_k; H(x_1|_{S_1},\ldots,x_k|_{S_1}),\ldots,H(x_1|_{S_m},\ldots,x_k|_{S_m})),$$

where $S_1,...,S_m$ are disjoint subsets of $[n]$ each of size ℓ and $x|_{S_i}$ is the substring of x of length ℓ formed by taking the bits of x indexed by S_i. We point out that $G_{H;n,\ell,m}$ is only well-defined when $\ell \cdot m \leq n$. G has the property that if H is $(\frac{1}{2} - \frac{\epsilon}{m})$-hard for non-uniform communication protocols operating on k-tuples of ℓ-bit inputs that use q bits of communication, then G is ϵ-pseudorandom against non-uniform randomized communication protocols that operate on k-tuples of n-bit inputs, use m bits of randomness, and use q bits of communication. This pseudorandomness guarantee can be argued directly; it also follows from the remark after the proof of Lemma 2.9 in the appendix, where we observe that G can be seen as a degenerate case of the Nisan-Wigderson construction.

We next set the parameters and the language H so as to ensure that the function

$$B(x_1,...,x_k) = A(G_{H;n,\ell,m}(x_1,...,x_k))$$

is within $3\rho + \epsilon$ from L (as long as $q' \leq n$).

1. Pseudorandom generator $G_{H;n,\ell,m}$.
   By Lemma 2.4, B computes L to within $3\rho + \epsilon$ if $G_{H;n,\ell,m}$ is a seed-extending ϵ-pseudorandom generator secure against tests $T_{r'}$ of the form $T_{r'}(x,r) = A(x_1,...,x_k;r) \oplus A(x_1,...,x_k;r')$, which are communication protocols that use at most $2q$ bits of communication.

2. Hard language H.

   By the pseudorandomness property stated above, $G_{H;n,\ell,m}$ is ϵ-pseudorandom for tests $T_{r'}$ if H is $(\frac{1}{2} - \frac{\epsilon}{m})$-hard on k-tuples of ℓ-bit inputs for protocols that use $2q$ bits of communication. [BNS92] demonstrate a function, the generalized inner product, which for some positive constant β and any $\epsilon' > 0$ is $(\frac{1}{2} - \epsilon')$-hard for non-uniform k-party communication protocols on k-tuples of ℓ-bit inputs that use at most $\beta \cdot (\frac{\ell}{4^k} - \log(1/\epsilon'))$ bits of communication. Letting H be this function, H has the hardness needed if $2q \leq \beta \cdot (\frac{\ell}{4^k} - \log(m/\epsilon))$. We choose $\ell = \lceil 4^k \cdot (\frac{2q}{\beta} + \log(m/\epsilon)) \rceil$ so that if $\ell \cdot m \leq n$ then $G_{H;n,\ell,m}$ is well-defined and H has the required hardness.

We conclude that for $\ell = \lceil 4^k \cdot (\frac{2q}{\beta} + \log(m/\epsilon)) \rceil$, if $\ell \cdot m \leq n$ then $G_{H;n,\ell,m}$ is an ϵ-pseudorandom generator against the tests $T_{r'}$ and thus B computes L to within $3\rho + \epsilon$.

We next exhibit a protocol of the prescribed form to evaluate the function

$$B(x_1, \ldots, x_k) = A(x_1, \ldots, x_k; H(x_1|_{S_1}, \ldots, x_k|_{S_1}), \ldots, H(x_1|_{S_m}, \ldots, x_k|_{S_m})).$$

Phase 0: All players calculate the value ℓ given above and terminate the protocol if $\ell \cdot m > n$.

Phase 1: Player 1 writes $x_2|_{S_1}, ..., x_2|_{S_m}$ on the public blackboard.

Phase 2: Player 2 evaluates each of $H(x_1|_{S_1}, ..., x_k|_{S_1}), ..., H(x_1|_{S_m}, ..., x_k|_{S_m})$ and writes the results on the public blackboard.

Phase 3: All players execute the protocol for A on input $(x_1, ..., x_k; r)$ using the bits written on the blackboard from Phase 2 as the random bits r.

Phase 1 requires $\ell \cdot m$ bits of communication and guarantees that player 2 has all inputs needed to evaluate H in Phase 2, Phase 2 requires m bits of communication, and Phase 3 requires q bits of communication. Altogether we can evaluate B using $\ell \cdot m + m + q$ bits of communication. Taking α a sufficiently large constant such that $q' = \alpha \cdot 4^k \cdot m \cdot (q + \log(m/\epsilon)) \geq \ell \cdot m + m + q$, the protocol requires at most q' bits of communication. Noting that $q' > \ell \cdot m$ we also have that $G_{H;n,\ell,m}$ is well-defined and B computes L to within $3\rho + \epsilon$ if $q' \leq n$.

We finally remark on the uniformity of the construction. Each player must determine the block size ℓ, execute the protocol A, and player 2 must compute H. The latter can be performed in logarithmic space for H the generalized inner product problem, and the remainder can be done within the uniformity bounds of A assuming each of the quantities k, q, m, and $\log(1/\epsilon)$ are constructible within the uniformity bounds. □

5. Comparison with the Extractor-Based Approach

We have seen several settings in which seed-extending pseudorandom generators allow us to prove typically-correct derandomization results that do not follow from the extractor-based approach of [Sha09]. We now show that the approach of [Sha09] is essentially equivalent to having seed-extending pseudorandom generators with *exponentially small error*. This reaffirms our claim that our approach is more general since we additionally obtain meaningful results from pseudorandom generators with larger error.

Overview of the Extractor-Based Approach We start with a high-level overview of the approach of [Sha09] that uses a notion of extractors for recognizable distributions, which we now explain. For any function $f : \{0,1\}^n \to \{0,1\}$, [Sha09] defines the distribution *recognized* by f as $U_n|f = 1$, i.e., the uniform distribution over $f^{-1}(1) = \{x \in \{0,1\}^n \mid f(x) = 1\}$. A function $E : \{0,1\}^n \to \{0,1\}^m$ is a (k,ϵ)-extractor for distributions recognizable by some collection of functions $f : \{0,1\}^n \to \{0,1\}$, if for every such function f with $|f^{-1}(1)| \geq 2^k$, the distribution $E(U_n|f = 1)$ has statistical distance at most ϵ from the uniform distribution on m bit strings, i.e.,

$$\sum_{r \in \{0,1\}^m} |\frac{1}{2^m} - \Pr_{X \leftarrow U_n}[E(X) = r | f(X) = 1]| \leq \epsilon.$$

[Sha09] shows the following general approach towards typically-correct derandomization. Let $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ be a randomized algorithm that computes some language L with error ρ at length n. Let $\Delta = 100m$ and let $E : \{0,1\}^n \to \{0,1\}^m$ be an $(n - \Delta, 2^{-\Delta})$-extractor for distributions recognizable by functions of the form $f_{r_1,r_2}(x) = A(x,r_1) \oplus A(x,r_2)$ where $r_1, r_2 \in \{0,1\}^m$ are arbitrary strings. Then $B(x) = A(x, E(x))$ is within $3\rho + 2^{-10m}$ of L at length n.

Comparison The above approach requires extractors with error that is exponentially small in m, and breaks down completely when the error is larger. We now observe that an extractor with exponentially small error yields a seed-extending pseudorandom generator with exponentially small error.

THEOREM 5.1. *Let $T : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ be a function. Let $\Delta = m + \log(1/\epsilon)$ and let $E : \{0,1\}^n \to \{0,1\}^m$ be an $(n - \Delta, 2^{-\Delta})$-extractor for distributions recognizable by functions of the form $f_r(x) = T(x,r)$ where $r \in \{0,1\}^m$ is an arbitrary string. Then, $G(x) = (x, E(x))$ is ϵ-pseudorandom for T.*

As a consequence the extractors used in [Sha09] can be viewed as seed-extending pseudorandom generators with exponentially small error. More precisely, given a randomized algorithm $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ the extractor-based approach sets $\Delta = 100m$ and requires an $(n - \Delta, 2^{-\Delta})$-extractor for distributions that are recognizable by functions of the form $f_{r_1,r_2}(x) = A(x,r_1) \oplus A(x,r_2)$. The pseudorandom generator approach of this paper requires a seed-extending generator $G(x) = (x, E(x))$ that fools tests of the form $T_{r_2}(x,r_1) = A(x,r_1) \oplus A(x,r_2) = f_{r_1,r_2}(x)$. By Theorem 5.1, an extractor E that can be used to obtain typically-correct derandomization following the extractor-based approach

gives rise to a seed-extending ϵ-pseudorandom generator $G(x) = (x, E(x))$ with $\epsilon = 2^{m-\Delta} = 2^{-99m} < 2^{-10m}$ which can be used to obtain typically-correct derandomization following the approach of this paper.

We remark that in some algorithmic settings, e.g., 2-party communication protocols, [Sha09] obtains typically-correct derandomizations that are more efficient than the ones that follow from applying our methodology directly based on the NW-construction and known hardness results. Nevertheless, by Theorem 5.1 the extractors used in [Sha09] define seed-extending pseudorandom generators that yield typically-correct derandomizations matching the efficiency of the extractor-based approach.

We now prove Theorem 5.1. The analysis below uses the same approach as the analysis of [Sha09] showing that extractors yield typically-correct derandomization.

Proof (of Theorem 5.1). Consider a probability space with two independent random variables $X \leftarrow U_n$ and $R \leftarrow U_m$. By conditioning on R we have that

$$\begin{aligned}
&|\Pr[T(X,R)=1] - \Pr[T(X,E(X))=1]| \\
&\quad = \Big|\sum_{r\in\{0,1\}^m} \Pr[T(X,r)=1 \wedge R=r] - \Pr[T(X,r)=1 \wedge E(X)=r]\Big| \\
&\quad = \Big|\sum_{r\in\{0,1\}^m} \Pr[T(X,r)=1] \cdot \\
&\qquad\qquad (\Pr[R=r \,|\, T(X,r)=1] - \Pr[E(X)=r \,|\, T(X,r)=1])\Big| \\
&\quad \le \sum_{r\in\{0,1\}^m} \Pr[T(X,r)=1] \cdot \qquad (5.2) \\
&\qquad\qquad |\Pr[R=r \,|\, T(X,r)=1] - \Pr[E(X)=r \,|\, T(X,r)=1]|.
\end{aligned}$$

We next argue that the contribution of each individual $r \in \{0,1\}^m$ to the right-hand side of (5.2) is at most $2^{-\Delta}$. This yields an upper bound of $2^m 2^{-\Delta} = \epsilon$ on the left-hand side of (5.2), which by definition means that $G(x) = (x, E(x))$ is ϵ-pseudorandom for T.

We consider two cases. If $\Pr[T(X,r)=1] < 2^{-\Delta}$ then the contribution of r to the right-hand side of (5.2) is less than $2^{-\Delta}$ because of the first factor. Otherwise, the set $f_r^{-1}(1)$ has size at least $2^{n-\Delta}$ and by the given extractor property of E, $|\frac{1}{2^m} - \Pr[E(X) = r \,|\, f_r(X) = 1]| \le 2^{-\Delta}$. Since $\Pr[R = r \,|\, T(X,r) = 1] - \Pr[E(X) = r \,|\, T(X,r) = 1] = \frac{1}{2^m} - \Pr[E(X) = r \,|\, f_r(X) = 1]$, the second factor on the right-hand side of (5.2) is at most $2^{-\Delta}$, and so is the entire term corresponding to r. □

Conversely, we observe that seed-extending pseudorandom generators with error that is exponentially small in m yield extractors for recognizable distributions.

THEOREM 5.3. *Let $f : \{0,1\}^n \to \{0,1\}$ be a function and let $E : \{0,1\}^n \to \{0,1\}^m$ be a function such that $G(x) = (x, E(x))$ is ϵ-pseudorandom for tests $T(x,r)$ of the form $T_z(x,r) = f(x) \wedge (r = z)$ where $z \in \{0,1\}^m$ is an arbitrary string. If $\epsilon \leq 2^{-(m+2\Delta)}$ then E is an $(n-\Delta, 2^{-\Delta})$-extractor for the distribution recognized by f.*

PROOF (of Theorem 5.3). Consider the test $T_z(x,r) = f(x) \wedge (r = z)$ for any $z \in \{0,1\}^m$. By the given pseudorandomness property we have that for independently chosen $X \leftarrow U_n$ and $R \leftarrow U_m$,

$$\begin{aligned} &|\Pr[T_z(X,R) = 1] - \Pr[T_z(X, E(X)) = 1]| \\ &= \; |\Pr[f(X) = 1] \cdot \Pr[R = z] - \Pr[f(X) = 1] \cdot \Pr[E(X) = z \,|\, f(X) = 1]| \\ &= \; \Pr[f(X) = 1] \cdot |\Pr[R = z] - \Pr[E(X) = z \,|\, f(X) = 1]| \leq \epsilon. \end{aligned}$$

Letting P denote the distribution recognized by f and setting $k = \log(|f^{-1}(1)|)$, we can rewrite the above inequality as $2^{k-n} \cdot |2^{-m} - \Pr[E(P) = z]| \leq \epsilon$, which implies that

$$\sum_{z \in \{0,1\}^m} |2^{-m} - \Pr[E(P) = z]| \leq 2^m \epsilon / 2^{k-n}. \tag{5.4}$$

We want to show that the right-hand side of (5.4) is at most $2^{-\Delta}$ for $k \geq n - \Delta$. This is the case since $\epsilon \leq 2^{-(m+2\Delta)}$. □

Together, Theorems 5.1 and 5.3 essentially say that in many algorithmic settings, $(n - cm, 2^{-cm})$-extractors for a sufficiently large constant $c > 1$ give seed-extending pseudorandom generators with error $\epsilon = 2^{-c'm}$ for a constant $c' > 1$ and vice versa. As a consequence the approach of [Sha09] is essentially equivalent to the special case of seed-extending pseudorandom generators with error that is exponentially small. The results we obtain using seed-extending pseudorandom generators with *larger* error, such as the conditional result of Theorem 3.1 and the unconditional result of Theorem 4.1, do *not* seem to follow from the [Sha09] approach.

Handling algorithms that toss a super-linear number of coins Another advantage of the approach of this paper is that we can directly handle randomized algorithms that toss a super-linear number of coins. This is because we can use *stretching* seed-extending pseudorandom generators, in which the length

of the extending part $E(x)$ is super-linear. In contrast an extractor $E(x)$ cannot have an output length that is super-linear as it is impossible to extract more random bits than are present in the input distribution. Indeed, this is why [Sha09] handles randomized algorithms that toss a super-linear number of coins by first applying a pseudorandom generator to reduce the number of coins to sub-linear and only then running an extractor.

In some algorithmic settings both the approach of this paper and [Sha09] can only handle sub-linear randomness. For example, consider the setting of communication protocols from Section 4.3. We cannot hope for unconditional stretching seed-extending pseudorandom generators that fool tests $A(x_1, \ldots, x_k; r)$ defined by randomized k-party communication protocols. This is because in such a protocol we only place limitations on communication complexity and allow the computation of an arbitrary function of r for free. Therefore, such a protocol can implement any statistical test at no cost and distinguish a uniformly chosen string r from one that is generated deterministically from fewer random bits. Even if we restrict our attention to polynomial-time uniform protocols, we are still allowing each party in the protocol to apply an arbitrary polynomial-time computable function to the public random coin sequence r. Thus, the existence of a pseudorandom generator for such protocols presumes the existence of pseudorandom generators for polynomial time, which we do not know to exist unconditionally.

More generally, what differentiates randomized communication protocols from say randomized algorithms corresponding to $\mathrm{BP.AC}^0$ is the way that they are charged for performing computations on the random coin sequence r. Communication protocols can compute any function of r for free, whereas algorithms for $\mathrm{BP.AC}^0$ are restricted to functions in AC^0. It remains open whether one can obtain typically-correct derandomizations of communication protocols that toss a super-linear number of coins.[4]

6. Circuit Lower Bounds

It is well-known that the existence of pseudorandom generators for polynomial-size circuits (which yields everywhere-correct derandomization of BPP) implies that EXP does not have polynomial-size circuits; this is the easy direction of the hardness versus randomness tradeoffs. Impagliazzo et al. [IKW02] showed that *any* everywhere-correct derandomization of promise-BPP into NSUBEXP

[4] [New91] shows that every randomized k-party communication protocol can be simulated by another randomized k-party protocol which tosses only $O(\log n)$ coins. However, the transformation does not preserve uniformity.

– using pseudorandom generators or otherwise – implies that NEXP does not have polynomial-size circuits. Building on [IKW02], Kabanets and Impagliazzo [KI04] showed that *any* everywhere-correct derandomization of BPP into NSUBEXP implies that NEXP does not have Boolean circuits of polynomial size or that the permanent over $\mathbb{Z}$ does not have arithmetic circuits of polynomial size. We present a simpler proof of the latter result and show how to extend it to the setting of typically-correct derandomization.

6.1. Results. We use the following terminology and notation in the statements and proofs of our lower bound results. We consider both Boolean and arithmetic circuits, where the latter have internal nodes representing addition, subtraction, and multiplication, and leaves representing variables and the constants 0 and 1. We measure the size of circuits by the string length of their description, and assume that the description mechanism is such that the description of a circuit of size s can easily be padded into the description of an equivalent circuit of size s' for any $s' > s$. For any function $s(n)$ we denote by $\mathrm{SIZE}(s(n))$ the class of languages L such that L at length n can be decided by a Boolean circuit of size $s(n)$ for all but finitely many input lengths n. We denote by $\mathrm{ASIZE}(a(n))$ the class of families $(p_n)_{n\in\mathbb{N}}$ of polynomials over $\mathbb{Z}$ where p_n has n variables and can be computed by an arithmetic circuit of size $a(n)$ for all but finitely many $n \in \mathbb{N}$. ACZ denotes the language of all arithmetic circuits that compute the zero polynomial over $\mathbb{Z}$. Perm denotes the permanent of matrices over $\mathbb{Z}$, and 0-1-Perm its restriction to matrices with all entries in $\{0, 1\}$.

Everywhere-Correct Derandomization Our approach yields the following parameterized version of the main result of [KI04], namely circuit lower bounds that follow from everywhere-correct derandomization of the specific BPP-language ACZ.

THEOREM 6.1. *Let $\gamma(n)$ denote the maximum circuit complexity of Boolean functions on n inputs. There exists a constant $c > 0$ such that the following holds for any functions $a(\cdot)$, $s(\cdot)$, and $t(\cdot)$ such that $a(\cdot)$ and $s(\cdot)$ are constructible, $a(\cdot)$ and $t(\cdot)$ are monotone, and $n \le s(n) < \gamma(n)$.*

If $\mathrm{ACZ} \in \mathrm{NTIME}(t(n))$ *then*

(i) $(\mathrm{N} \cap \mathrm{coN})\mathrm{TIME}\left(t((s(n))^c \cdot a((s(n))^c))\right) \not\subseteq \mathrm{SIZE}(s(n))$, *or*

(ii) $\mathrm{Perm} \notin \mathrm{ASIZE}(a(n))$.

In particular, we obtain the following instantiation for the exponential time bounds considered for part (i) in [KI04].

COROLLARY 6.2. *There exists a constant $c > 0$ such that the following holds for any functions $a(\cdot)$, $s(\cdot)$, and $t(\cdot)$ such that $a(\cdot)$ and $s(\cdot)$ are constructible, $a(\cdot)$ and $t(\cdot)$ are monotone, $s(n) \geq n$, and for sufficiently large n*

$$t\left((s(n))^c \cdot a((s(n)^c))\right) \leq 2^n. \tag{6.3}$$

If $\mathrm{ACZ} \in \mathrm{NTIME}(t(n))$ *then*

(i) $(\mathrm{N} \cap \mathrm{coN})\mathrm{TIME}(2^n) \not\subseteq \mathrm{SIZE}(s(n))$, *or*

(ii) $\mathrm{Perm} \notin \mathrm{ASIZE}(a(n))$.

Let us compare Theorem 6.1 and Corollary 6.2 to the corresponding results in [KI04]. First, we point out that part (i) states a lower bound for $(\mathrm{N} \cap \mathrm{coN})\mathrm{TIME}(\cdot)$ rather than for $\mathrm{NTIME}(\cdot)$, where we use $(\mathrm{N} \cap \mathrm{coN})\mathrm{TIME}(\cdot)$ as a shorthand for $\mathrm{NTIME}(\cdot) \cap \mathrm{coNTIME}(\cdot)$. Theorem 6.1 and Corollary 6.2 give such a lower bound for the entire range of the parameters; [KI04] only manages to do so in the case where all the parameters are polynomially bounded. More importantly, due to the use of the implication that EXP having polynomial-size circuits implies that EXP coincides with MA [BFNW93], the arguments in [KI04] can only give lower bounds for time bounds on the left-hand side of (i) that are exponential. This is true even when all of $a(n)$, $s(n)$, and $t(n)$ are polynomial, in which case our Theorem 6.1 only needs the time bound in the left-hand side of (i) to be superpolynomial. Finally, due to its dependence on the result from [IKW02] that NEXP having polynomial-size circuits implies that NEXP coincides with EXP, the proof in [KI04] only works when $s(n)$ is polynomially bounded; our proof gives nontrivial results for $s(n)$ ranging between linear and linear-exponential.[5]

Typically-Correct Derandomization We initiate the study of whether typically-correct derandomization of BPP implies circuit lower bounds. We show that it does in the case of typically-correct derandomizations that run in NSUBEXP and are of the quality considered by Goldreich and Wigderson [GW02].

THEOREM 6.4. *If for every positive constant ϵ there exists a nondeterministic Turing machine which runs in time 2^{n^ϵ} and correctly decides ACZ on all but at most 2^{n^ϵ} of the inputs of length n for all but finitely many n, then*

(i) NEXP does not have Boolean circuits of polynomial size, or

[5] Scott Aaronson and we independently came up with a number of alternate arguments that do not rely on [IKW02], including one that is even more elementary but does not scale as well as the one we describe here. See [AvM10] for more details.

(ii) Perm does not have arithmetic circuits of polynomial size.

Note that Theorem 6.4 strengthens the main result of [KI04], which establishes the theorem in the special case where the nondeterministic machines decide ACZ correctly on all inputs. We can parameterize Theorem 6.4 in the same way as Theorem 6.1. However, we only obtain nontrivial results for polynomially bounded $a(n)$ and $s(n)$, in which case $t(n)$ can be subexponential. For that reason, we only state the latter special case. The error rate considered in Theorem 6.4 is the largest one for which our argument gives nontrivial lower bounds.

We first prove Theorem 6.4 and then analyze how the argument parameterizes to Theorem 6.1 and Corollary 6.2 in the case of zero error rate. We end with some extensions and variations of both theorems.

6.2. Proof for the Typically-Correct Setting. The proof of Theorem 6.4 has two main ingredients. The first ingredient is an unconditional circuit lower bound for $\mathrm{P}^{\text{0-1-Perm}[1]}$, the class of languages that can be decided in polynomial time with one query to an oracle for 0-1-Perm.

Claim 6.5. *For every constant d,* $\mathrm{P}^{\text{0-1-Perm}[1]} \not\subseteq \mathrm{SIZE}(n^d)$.

The second ingredient gives a conditional simulation of that class in nondeterministic subexponential time with subpolynomial advice.

Claim 6.6. *If the hypothesis of Theorem 6.4 holds and* Perm *has arithmetic circuits of polynomial size, then*

$$\mathrm{P}^{\text{0-1-Perm}[1]} \subseteq \cap_{\epsilon>0}\mathrm{NTIME}(2^{n^\epsilon})/n^\epsilon.$$

By combining both claims we obtain that if the hypothesis of Theorem 6.4 holds and Perm has arithmetic circuits of polynomial size, then for every constant d, $\mathrm{NTIME}(2^n)/n \not\subseteq \mathrm{SIZE}(n^d)$. The latter implies that for every constant d, $\mathrm{NTIME}(2^n) \not\subseteq \mathrm{SIZE}(n^d)$. Otherwise, any language in $\mathrm{NTIME}(2^n)/n$ can be decided on inputs of length n by a circuit of size $(2n)^d$, namely a circuit simulating an $\mathrm{NTIME}(2^m)$-computation on an input of length $m = n + n$ with its second input hardwired to an advice string of length n. Since NEXP contains a language that is hard for $\mathrm{NTIME}(2^n)$ under linear-time reductions, the statement that $\mathrm{NTIME}(2^n) \not\subseteq \mathrm{SIZE}(n^d)$ for every constant d implies that NEXP does not have circuits of polynomial size. This finishes the proof of Theorem 6.4 modulo the proofs of both claims.

PROOF (of Claim 6.5). The claim follows because the polynomial-time hierarchy PH does not have circuits of fixed polynomial size [Kan82], PH is contained in $\mathrm{P}^{\#\mathrm{P}[1]}$ [Tod91], and 0-1-Perm is complete for #P under reductions that make a single query [Zan91]. □

In the rest of the proof we establish Claim 6.6.

PROOF (of Claim 6.6). It is enough to consider $\mathrm{P}^{\text{0-1-Perm}[1]}$-computations that run in time n. Consider such a computation, and let M denote the query it makes to its 0-1-Perm-oracle on a given input of length n. The dimension m of M cannot exceed $\sqrt{n}$ as the computation does not have enough time to generate larger square matrices. By the paddability of 0-1-Perm, we can assume without loss of generality that M has dimension $m = \sqrt{n}$ independent of the input of length n, and maintain a running time of $O(n)$.

It suffices to design, for every $\epsilon > 0$, a nondeterministic machine N_ϵ running in time 2^{n^ϵ} and an advice sequence $a(\cdot, \epsilon)$ where $a(n, \epsilon)$ has length at most n^ϵ such that the following holds: On input an m-by-m 0-1-matrix M, N_ϵ with advice $a(n, \epsilon)$ outputs $\mathrm{Perm}(M)$ on every accepting computation path, and has at least one such computation path. Our machine N_ϵ acts as follows.

1. Guess a polynomial-sized candidate arithmetic circuit C for Perm on matrices of dimension m.
2. Verify the correctness of C. Halt and reject if the test fails.
3. Use the circuit C to determine the permanent of M in deterministic polynomial time.

The circuit in step 1 exists by virtue of the hypothesis that Perm has polynomial-size arithmetic circuits. Say the circuit C we guess is of size $s \leq m^b$ and purportedly computes the permanent of m-by-m matrices over $\mathbb{Z}$. The constant b is chosen large enough so that such a circuit exists. The crux of the procedure is the second step, which is a nondeterministic test with small advice that has an accepting computation path on input C iff C does what it is purported to do. Once that test is passed, we evaluate C modulo $m! + 1$ on the given 0-1-matrix M. Evaluating C this way ensures that the intermediate results remain small so the computations can be done in polynomial time; since the permanent of M is a non-negative integer no larger than $m!$, the outcome of the computation gives the correct value of the permanent of M.

The test in the second step is based on the following well-known translation to ACZ exploiting the downward self-reducibility of the permanent. For completeness we include a proof in the Appendix.

LEMMA 6.7. *There exists a polynomial-time algorithm that takes an arithmetic circuit C and an integer m, and produces an arithmetic circuit $\tilde{C}$ such that C computes the permanent of m-by-m matrices over $\mathbb{Z}$ iff $\tilde{C} \in$ ACZ.*

We use Lemma 6.7 to transform the circuit C into the circuit $\tilde{C}$, and show how to test that $\tilde{C}$ is in ACZ. We will exploit the fact that ACZ is in coNP and that it is highly paddable to transform the almost-correct nondeterministic subexponential-time tests given by the hypothesis of Theorem 6.4 into perfect nondeterministic subexponential-time tests for ACZ with small advice. Let N'_ϵ denote the nondeterministic Turing machine from the hypothesis of Theorem 6.4 corresponding to ϵ. We will use $N'_{\epsilon'}$ for some ϵ' related to ϵ.

Note that the false positives $\tilde{C}$ of $N'_{\epsilon'}$ can be detected nondeterministically by guessing an accepting computation path of $N'_{\epsilon'}$ on input $\tilde{C}$, guessing an input x and a modulus μ, evaluating $\tilde{C}$ on input x modulo μ, and verifying that the result is nonzero. Since the modulus μ never needs to be larger than $2^{\tilde{s}}$, where $\tilde{s}$ denotes the size of the circuit $\tilde{C}$, the overhead of the test beyond running $N'_{\epsilon'}$ is only polynomial. Now, suppose that we are given the exact number $\mathrm{fp}(\tilde{s}, \epsilon')$ of false positives of $N'_{\epsilon'}$ at length $\tilde{s}$. Then the following nondeterministic test for membership to ACZ is sound for instances $\tilde{C}$ of length $\tilde{s}$, i.e., if the test accepts $\tilde{C}$ then $\tilde{C}$ is in ACZ for sure.

(a) Guess a list of $\mathrm{fp}(\tilde{s}, \epsilon')$ distinct instances of length $\tilde{s}$ and nondeterministically test that they are all false positives of $N'_{\epsilon'}$. If there is a test that fails, halt and reject.

(b) Accept iff $\tilde{C}$ is not on that list and $N'_{\epsilon'}$ accepts $\tilde{C}$.

Note that this test runs in time $\mathrm{fp}(\tilde{s}, \epsilon') \cdot 2^{\tilde{s}^{\epsilon'}} \cdot \mathrm{poly}(\tilde{s})$, which is $2^{O(\tilde{s}^{\epsilon'})}$. Note also that we can make sure that the size s of C as well as the size $\tilde{s}$ of $\tilde{C}$ only depend on m in an easily computable way, say $\tilde{s} = m^c$ for some constant c. This follows from the paddability of circuit descriptions. As a result, the information $\mathrm{fp}(\tilde{s}, \epsilon')$ really takes on the form of an advice.

The above test is sound but not necessarily complete – it may still have false negatives. In order to remedy that problem, we exploit a further paddability property of circuit descriptions, namely that we can obtain many different circuits equivalent to a given circuit by adding a little bit of circuitry that isn't used in the evaluation of the output gate. Consider the equivalents of $\tilde{C} \in$ ACZ of length ℓ that we can obtain using this type of padding. If the number of distinct pads exceeds the total number of errors $N'_{\epsilon'}$ makes at length ℓ, we can nondeterministically guess a pad that is accepted by $N'_{\epsilon'}$ and therefore also by the above test when provided with $\mathrm{fp}(\ell, \epsilon')$ as advice.

How large does ℓ need to be for this approach to work? There exists a positive constant α such that the number of padded versions of $\tilde{C}$ of length $\ell = \tilde{s} + \Delta$ is at least $2^{\alpha\Delta}$. We need $2^{\alpha\Delta} > 2^{\ell^\epsilon}$. The latter condition is satisfied for every $0 < \epsilon < 1$ and sufficiently large $\tilde{s}$ when we set $\Delta = \tilde{s}$, i.e., $\ell = 2\tilde{s}$.

The resulting nondeterministic test for $\tilde{C}$ runs in time

$$2^{O(\ell^{\epsilon'})} = 2^{O(\tilde{s}^{\epsilon'})} = 2^{O(m^{c\epsilon'})}, \tag{6.8}$$

and works correctly when provided $\mathrm{fp}(\ell, \epsilon')$ as advice. The bit length of the advice is bounded by the logarithm of (6.8). Plugging in this test as the second step in the three-step approach mentioned at the beginning of the proof, we obtain a machine N_ϵ with the properties we need for any constant ϵ with $\epsilon > c\epsilon'$ by setting $a(n, \epsilon) = \mathrm{fp}(2m^c, \epsilon')$. □

6.3. Proofs for the Everywhere-Correct Setting. We establish Theorem 6.1 by analyzing how the proof of Theorem 6.4 parameterizes in the case of zero error rate.

Proof (of Theorem 6.1). The two ingredients in the proof of Theorem 6.4 translate as follows given the parameters of Theorem 6.1.

Claim 6.9. *There exists a constant c such that for every time constructible function $s(\cdot)$ satisfying $n \le s(n) < \gamma(n)$, $\mathrm{DTIME}^{\text{0-1-Perm[1]}}((s(n)^c)) \not\subseteq \mathrm{SIZE}(s(n))$.*

Claim 6.10. *There exists a constant d such that the following holds for any functions $a(\cdot)$ and $t(\cdot)$ with $a(\cdot)$ constructible and $t(\cdot)$ monotone. If $\mathrm{ACZ} \in \mathrm{NTIME}(t(n))$ and $\mathrm{Perm} \in \mathrm{ASIZE}(a(n))$, then*

$$\mathrm{DTIME}^{\text{0-1-Perm[1]}}(n) \subseteq \mathrm{NTIME}(t(n \cdot \log^d n \cdot a(\sqrt{n}))).$$

Given those two claims, we obtain the following by padding Claim 6.10 to length $(s(n))^c$, exploiting the closure under complementation of deterministic computations, and combining it with Claim 6.9: If $\mathrm{ACZ} \in \mathrm{NTIME}(t(n))$ and $\mathrm{Perm} \in \mathrm{ASIZE}(a(n))$, then

$$(\mathrm{N} \cap \mathrm{coN})\mathrm{TIME}\left(t((s(n))^c \cdot \log^d((s(n))^c) \cdot a((s(n))^{c/2}))\right) \not\subseteq \mathrm{SIZE}(s(n)).$$

Theorem 6.1 follows by simplifying the last expression using the monotonicity of $a(\cdot)$ and $t(\cdot)$ and the fact that $s(n) \ge n$. All that remains are the proofs of the claims.

PROOF (of Claim 6.9). The argument of [Kan82] gives that $\Sigma_4\text{TIME}(s(n)\log^a(s(n))) \not\subseteq \text{SIZE}(s(n))$ for some constant a. [Tod91] shows that there exists a constant b and a problem $A \in \#\text{P}$ such that for any constructible function $t(\cdot)$ with $t(n) \geq n$, $\Sigma_4\text{TIME}(t(n)) \subseteq \text{DTIME}^{A[1]}((t(n))^b)$. The claim follows by combining the above as before with the completeness of 0-1-Perm for #P under reductions that make a single query [Zan91]. □

PROOF (of Claim 6.10). We follow the proof of Claim 6.6 and set $m = \sqrt{n}$.

The crux is the 3-step construction of a nondeterministic machine N that takes an m-by-m 0-1-matrix M and outputs $\text{Perm}(M)$ on every accepting computation path, and has at least one such computation path. In the first step N guesses an arithmetic circuit of size $a(m)$. By the constructibility of $a(\cdot)$, this step takes time $O(a(m))$. In the second step, we run the nondeterministic algorithm for ACZ from the hypothesis on the circuit $\tilde{C}$ given by Lemma 6.7. A careful reading of the proof of the lemma reveals that $\tilde{C}$ is of size $m^2 \cdot \log^d m \cdot a(m)$ for some constant d, so this step takes $t(m^2 \cdot \log^d m \cdot a(m))$ time. The third step takes time $O(m^2 \cdot \log^d m \cdot a(m))$. As we can assume without loss of generality that $t(n) \geq n$ and since $t(\cdot)$ is monotone, the three steps combined take time $O(t(m^2 \cdot \log^d m \cdot a(m)))$. The total running time of the nondeterministic simulation of the given $\text{DTIME}^{\text{0-1-Perm}[1]}(n)$-computation is of the same order. □

This finishes the proof of Theorem 6.1. □

The proof of Corollary 6.2 immediately follows from Theorem 6.1.

PROOF (of Corollary 6.2). Note that condition (6.3) gives an upper bound of 2^n on the time bound on the left-hand side of (ii) in the statement of Theorem 6.1. Also, we can assume without loss of generality that $t(n) \geq n$ for all but finitely many n; otherwise, the hypothesis of Corollary 6.2 fails as a nondeterministic machine deciding ACZ needs to be able to look at its entire input. Thus, condition (6.3) implies that $s(n)$ is upper bounded by $2^{n/c}$, which is less than $\gamma(n)$ for $c > 1$ and n sufficiently large. Corollary 6.2 then follows from Theorem 6.1 verbatim. □

We already discussed how Theorem 6.1 and Corollary 6.2 compare to the corresponding results in [KI04]. We now compare the argument of this paper to the one of [KI04]. The reader is also referred to [AvM10] for a more detailed discussion. For simplicity we consider the original statement of [KI04]. Namely, the goal is to obtain a contradiction from the hypotheses that ACZ is in NP,

NEXP has polynomial-size circuits, and Perm has polynomial-size arithmetic circuits. Both proofs start by using the first and the third hypothesis to collapse $P^{\#P}$ into NP. This step is captured by Claim 6.10. [KI04] then uses the result from [IKW02] that NEXP having polynomial-size circuits implies that NEXP coincides with EXP, and the result from [BFNW93] that EXP having polynomial-size circuits implies that EXP coincides with MA, to conclude that NEXP is in MA, which is unconditionally contained in $P^{\#P}$. This, in turn, collapses NEXP all the way down to NP, which contradicts the time hierarchy for nondeterministic machines.

Our proof does not attempt to collapse NEXP into NP. Instead we use the fact that NEXP having polynomial-size circuits immediately implies that NP has circuits of size n^c for some fixed constant c. Since by [Kan82] and [Tod91] (see Claim 6.9) we know unconditionally that $P^{\#P}$ does not have the latter property, we obtain a contradiction as we already derived that $P^{\#P}$ is in NP.

6.4. Extensions. We observe a few variations of Theorems 6.1 and 6.4. First, the theorems also hold when we simultaneously replace ACZ by AFZ (the restriction of ACZ to arithmetic formulas), and "arithmetic circuits" by "arithmetic formulas".

Second, we can play with the underlying i.o. and a.e. quantifiers. In fact, we can strengthen both theorems by either relaxing the hypothesis to hold only i.o. rather than a.e. or by improving one of the lower bound conclusions (i) or (ii) to hold a.e. rather than i.o. This follows because on the one hand the lower bounds in Claims 6.5 and 6.9 hold a.e. rather than just i.o. as stated. On the other hand, if one of the hypotheses of Claims 6.6 and 6.10 holds only i.o., the concluding simulation can be made to work i.o. when provided with a pointer to a nearby input length where the hypotheses hold. The latter can be handled with a logarithmic amount of advice, which the rest of the argument can handle.

As an example, in the case of Theorem 6.4 it suffices for the nondeterministic machines N_ϵ to correctly decide ACZ on all but at most 2^{n^ϵ} of the inputs of length n for *infinitely many* n. Related to the latter variation, we point out that by [IW01] EXP differs from BPP iff all of BPP has deterministic typically-correct derandomizations that run in subexponential time and err on no more than a polynomial fraction of the inputs of length n for infinitely many n. Thus, extending this i.o.-version of Theorem 6.4 to the setting with polynomial error rates would show that EXP$\neq$BPP implies circuit lower bounds.

7. Relativization and Algebrization

In Section 6, we showed that typically-correct derandomizations of BPP with the parameters considered by Goldreich and Wigderson [GW02] imply circuit lower bounds (Theorem 6.4). This can be seen as evidence that establishing such a typically-correct derandomization will be difficult. Although we do not know if typically-correct derandomizations of BPP with the weaker parameters of say Theorem 3.1 imply circuit lower bounds, we have other indications that establishing such weaker derandomizations of BPP will also be difficult – it would require non-relativizing, and indeed non-algebrizing, techniques.

Algebrization Let us recap the notion of algebrization [AW09], which generalizes the concept of relativization. A complexity class inclusion $\mathcal{C}_1 \subseteq \mathcal{C}_2$ is said to *algebrize* if for every oracle A and every low-degree extension $\tilde{A}$ of A, $\mathcal{C}_1^A \subseteq \mathcal{C}_2^{\tilde{A}}$. A complexity class separation $\mathcal{C}_1 \nsubseteq \mathcal{C}_2$ is said to *algebrize* if for every oracle A and low-degree extension $\tilde{A}$ of A, $\mathcal{C}_1^{\tilde{A}} \nsubseteq \mathcal{C}_2^A$. An inclusion or separation is said to *relativize* if the above holds with $\tilde{A}$ replaced by A.

Notice that any statement which relativizes also algebrizes. The converse does not hold. As an example, the inclusion PSPACE $\subseteq$ IP [Sha92] does not relativize but does algebrize. In fact, [AW09] observe that all known non-relativizing proofs that are based on arithmetization algebrize. At the same time [AW09] argues that several open questions in complexity theory require non-algebrizing techniques to be settled.

Typically-Correct Derandomization and Algebrization We show that the same is true of the question whether typically-correct derandomizations of BPP exist. On the one hand, a negative answer cannot algebrize, even for zero error. This is because ruling out typically-correct derandomization of BPP in particular implies BPP $\nsubseteq$ P, but for any PSPACE-complete language A and its multi-linear extension $\tilde{A}$, $\mathrm{BPP}^{\tilde{A}} \subseteq \mathrm{PSPACE}^{\tilde{A}} \subseteq \mathrm{P}^A$. On the other, we show that a positive answer cannot algebrize either, even for very large error rates and even if we only want simulations in nondeterministic subexponential time.

THEOREM 7.1. *There exists an oracle B and a multi-quadratic extension $\tilde{B}$ of B such that there is a language in* $\mathrm{BPTIME}^B(O(n))$ *that is* $(\frac{1}{2} - \frac{1}{2^{n/3}})$*-hard for* $\mathrm{NTIME}^{\tilde{B}}(2^n)$.

PROOF. The construction can be broken up into two main parts.

1. Construct B and a multi-quadratic extension $\tilde{B}$ of B such that any language computable in $\mathrm{NTIME}^{\tilde{B}}(2^n)$ can be computed in $\mathrm{BPTIME}^B(c \cdot n)$

for some constant c. This follows from the proof of a result of [AW09] showing that there exists an oracle B and a multi-quadratic extension $\tilde{B}$ of B such that $\mathrm{NTIME}^{\tilde{B}}(2^n) \subseteq \mathrm{SIZE}^B(c \cdot n)$. The proof closely follows the proof due to [Wil85] of the containment in the plain relativization setting (i.e., $\mathrm{NTIME}^B(2^n) \subseteq \mathrm{SIZE}^B(c \cdot n)$) and actually yields a construction such that for some constant c,

$$\Pr_{z \in \{0,1\}^{c|x|}}[(\forall x \in \{0,1\}^n) B(\langle x, z \rangle) = N^{\tilde{B}}(x)] \leq 2/3, \tag{7.2}$$

where $N^{\tilde{B}}$ is $\mathrm{NE}^{\tilde{B}}$-complete. [AW09] use the implication that $\mathrm{NTIME}^{\tilde{B}}(2^n) \subseteq \mathrm{SIZE}^B(c \cdot n)$, which follows by hardwiring a value of z for which the predicate on the left-hand side of (7.2) holds. Instead, by picking z uniformly at random we obtain that $\mathrm{NTIME}^{\tilde{B}}(2^n) \subseteq \mathrm{BPTIME}^B(c \cdot n)$.

2. Given B and $\tilde{B}$ construct a hard language L. We derive the hard language L using a relativizing hierarchy theorem of [GW00] for deterministic machines, which shows that for any constant c there is a language $L \in \mathrm{DTIME}^B(2^{O(n)})$ that is $(\frac{1}{2} - \frac{1}{2^{n/3}})$-hard for $\mathrm{DTIME}^B(2^{c \cdot n})$. By the first part $\mathrm{NTIME}^{\tilde{B}}(2^n) \subseteq \mathrm{BPTIME}^B(c \cdot n) \subseteq \mathrm{DTIME}^B(2^{c \cdot n})$, so the language L has the required hardness. Moreover, L is computable in $\mathrm{DTIME}^B(2^{O(n)}) \subseteq \mathrm{NTIME}^{\tilde{B}}(2^{O(n)}) \subseteq \mathrm{BPTIME}^B(O(n))$, where the latter inclusion follows from $\mathrm{NTIME}^{\tilde{B}}(2^n) \subseteq \mathrm{BPTIME}^B(O(n))$ by padding. □

We point out that weaker hierarchy theorems for deterministic time could have been used in place of the one from [GW00] in order to conclude that a positive answer cannot algebrize. We stated the result using the [GW00] hierarchy theorem because it holds for all but finitely many input lengths and achieves hardness very close to $\frac{1}{2}$.

Acknowledgements

We would like to thank Oded Goldreich for suggesting the term "typically-correct derandomization," and Matt Anderson, Valentine Kabanets, Salil Vadhan, and anonymous reviewers for helpful comments. The third author thanks Salil Vadhan for suggesting this research direction to him and for collaboration at an early stage of this research.

A preliminary version of this work appeared under the title "Pseudorandom Generators and Typically-Correct Derandomization" in the 13-th Annual International Workshop on Randomization and Computation, held in Berkeley, California 2009.

A significant portion of this work was completed while the first author was a graduate student at the University of Wisconsin-Madison and while the second author was visiting the University of Haifa, the Weizmann Institute of Science, and Humboldt University in Berlin. Portions of this work were completed while the first author was supported by NSF award CCF-0728809 and by a Cisco Systems Distinguished Graduate Fellowship, while the second author was supported by NSF award CCF-0728809 and by the Humboldt Foundation, and while the third author was supported by BSF grant 2004329 and ISF grant 686/07.

References

[AvM10] Scott Aaronson and Dieter van Melkebeek. A note on circuit lower bounds from derandomization. *Electronic Colloquium on Computational Complexity (ECCC)*, 17(105), 2010.

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory*, 1(1):1–54, 2009.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 21–31, 1991.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[BNS92] László Babai, Noam Nisan, and Márió Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space tradeoffs. *Journal of Computer and System Sciences*, 45(2):204–232, 1992.

[CW89] Aviad Cohen and Avi Wigderson. Dispersers, deterministic amplification, and weak random sources (extended abstract). In *Proceed-*

ings of the IEEE Symposium on Foundations of Computer Science (FOCS), pages 14–19, 1989.

[GNW95] Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR-lemma. *Electronic Colloquium on Computational Complexity (ECCC)*, 2(50), 1995.

[GSTS03] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. Uniform hardness versus randomness tradeoffs for Arthur-Merlin games. *Computational Complexity*, 12(3–4):85–130, 2003.

[GW00] Oded Goldreich and Avi Wigderson. On pseudorandomness with respect to deterministic observers. In Carleton Scientific, editor, *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 77–84, 2000.

[GW02] Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. In *Proceedings of the International Workshop on Randomization and Computation (RANDOM)*, pages 209–223, 2002.

[Hås87] Johan Håstad. *Computational limitations of small-depth circuits.* MIT Press, Cambridge, MA, USA, 1987.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.

[Imp95] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 538–545, 1995.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 220–229, 1997.

[IW01] Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63(4):672–688, 2001.

[IZ89] Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 248–253, 1989.

[Kab01] Valentine Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *Journal of Computer and System Sciences*, 63(2):236–252, 2001.

[Kan82] Ravi Kannan. Circuit-size lower bounds and nonreducibility to sparse sets. *Information and Control*, 55(1):40–56, 1982.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1/2):1–46, 2004.

[KvM02] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[Mil01] Peter Bro Miltersen. Derandomizing complexity classes. In *Handbook of Randomized Computing*, pages 843–941. Kluwer Academic Publishers, 2001.

[MV05] Peter Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. *Computational Complexity*, 14(3):256–279, 2005.

[New91] Ilan Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39(2):67–71, 1991.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

[Nis93] Noam Nisan. On read-once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4), 2008.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[Sha09] Ronen Shaltiel. Weak derandomization of weak algorithms: explicit versions of Yao's lemma. In *Proceedings of the IEEE Conference on Computational Complexity*, 2009.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM*, 52(2):172–216, 2005.

[SU06] Ronen Shaltiel and Christopher Umans. Pseudorandomness for approximate counting and sampling. *Computational Complexity*, 15(4):298–341, 2006.

[SU07] Ronen Shaltiel and Christopher Umans. Low-end uniform hardness vs. randomness tradeoffs for AM. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 430–439, 2007.

[SZ99] Michael E. Saks and Shiyu Zhou. $\mathrm{BP_HSPACE(S)} \subseteq \mathrm{DSPACE(S^{3/2})}$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

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

[TV07] Luca Trevisan and Salil P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

[Vio05] Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. *Computational Complexity*, 13(3-4):147–188, 2005.

[Vio06] Emanuele Viola. Pseudorandom bits for constant-depth circuits with few arbitrary symmetric gates. *SIAM Journal on Computing*, 36(5):1387–1403, 2006.

[vMS05] Dieter van Melkebeek and Rahul Santhanam. Holographic proofs and derandmization. *SIAM Journal on Computing*, 35(1):59–90, 2005.

[Wil85] Christopher B. Wilson. Relativized circuit complexity. *Journal of Computer and System Sciences*, 31(2):169–181, 1985.

[Zan91] Viktoria Zanko. #P-completeness via many-one reductions. *International Journal of Foundations of Computer Science*, 2(1):77–82, 1991.

[Zim06] Marius Zimand. Exposure-resilient extractors. In *Proceedings of the IEEE Conference on Computational Complexity*, pages 61–72, 2006.

[Zim08] Marius Zimand. Exposure-resilient extractors and the derandomization of probabilistic sublinear time. *Computational Complexity*, 17(2):220–253, 2008.

A. Analysis of the Nisan-Wigderson Construction

Our typically-correct derandomization results use the Nisan-Wigderson generator construction [NW94]. Lemma 2.9 states that given a sufficiently hard function, the construction gives a *seed-extending* pseudorandom generator. In this section we review this well-known construction to verify that the original analysis carries through when the generator outputs its seed.

Definition of NW*-Generator* When taking a seed of length n and outputting m bits, the generator makes use of the following combinatorial object.

DEFINITION A.1 (combinatorial design). *A (k, ℓ) design of size m over $[n]$ is a sequence $S_1, S_2, \ldots, S_m$ of subsets of $[n]$ such that (a) $|S_i \cap S_j| \leq k$ for $1 \leq i < j \leq m$, and (b) $|S_i| = \ell$ for $1 \leq i \leq m$.*

The following construction suffices for our results. It has been (re)derived and used in several contexts, including in [NW94]. We provide a proof for completeness.

LEMMA A.2. *For any positive integers n, k, ℓ, m, and n such that $\ell \leq \sqrt{n}/2$ and $k \geq \frac{\log m}{\log \ell}$ there is a (k, ℓ) design of size m over $[n]$. Further, there is a Turing machine that on input (k, ℓ, m, n, i) outputs the i^{th} set and uses $O(\log(m+n))$ space.*

PROOF. For q a positive integer, let $\mathrm{GF}(2^q)$ denote the finite field of size 2^q. The main idea is to view the elements of $[n]$ as points in $\mathrm{GF}(2^q) \times \mathrm{GF}(2^q)$, let the sets S_i correspond to the graphs of polynomials of degree at most k over $\mathrm{GF}(2^q)$, and use the fact that two distinct such polynomials can intersect in at most k points.

Now we provide the details. Let q be the integer such that $\sqrt{n}/2 < 2^q \leq \sqrt{n}$. We identify the elements of $\mathrm{GF}(2^q)$ with the bit strings of length q. Since under the given conditions $m \leq 2^{(k+1)q}$, we can view $i \in [m]$ as defining a sequence of $k+1$ strings of length q (by padding with 0's as needed), and thus as a sequence of $k+1$ elements over $\mathrm{GF}(2^q)$. We interpret this sequence as the successive coefficients of a polynomial p_i of degree at most k over $\mathrm{GF}(2^q)$. We take the first ℓ points $y_1, ..., y_\ell$ in $\mathrm{GF}(2^q)$, say in lexicographic order, and define S_i as

$$S_i = \{(y_1, p_i(y_1)), ..., (y_\ell, p_i(y_\ell))\}.$$

Note that $\mathrm{GF}(2^q)$ contains at least ℓ elements as $\ell \leq \sqrt{n}/2 < 2^q$, and that $|S_i| = \ell$. The intersection size $|S_i \cap S_j|$ equals the number of y's on which p_i and p_j agree. For distinct i and j, that number is upper bounded by the maximum degree k.

Finally, consider the complexity of generating the set S_i. We must (a) perform arithmetic of $O(\log(m+n))$ bit numbers to determine q, keep counters, etc., (b) determine an irreducible polynomial of degree q over $\mathrm{GF}(2)$, and (c) using the irreducible polynomial perform arithmetic over $\mathrm{GF}(2^q)$. (b) can be performed in $O(q) = O(\log n)$ space by exhaustive search, and both (a) and (b) can be performed in $O(\log(m+n))$ space as well. □

Given such a design, we define the NW-generator as follows based on a presumed hard Boolean function H. Our definition differs from the original one [NW94] only in that the generator additionally outputs its seed.

Definition A.3 (seed-extending NW generator [NW94]). *Let n and m be integers, and $S_1, S_2, \ldots, S_m$ the (k, ℓ)-design of size m over $[n]$ with $\ell = \lfloor \sqrt{n}/2 \rfloor$ and $k = \lceil \frac{\log m}{\log \ell} \rceil$ provided by Lemma A.2. Given a function $H : \{0,1\}^\ell \to \{0,1\}$ the Nisan-Wigderson generator $\mathrm{NW}_{H;n,m} : \{0,1\}^n \to \{0,1\}^{n+m}$ is defined as*

$$\mathrm{NW}_{H;n,m}(x) = (x, H(x|_{S_1}), ..., H(x|_{S_m})),$$

where $x|_{S_i}$ denotes the substring of x of length ℓ formed by taking the bits of x indexed by S_i.

The NW-construction has the property that if the function H is hard on average for a certain class of algorithms, then $\mathrm{NW}_{H;n,m}$ is pseudorandom for related tests. Lemma 2.9 formalizes this property in the case of circuits. We include a proof sketch for reasons of completeness, where we focus on verifying that the argument given in [NW94] goes through with our modification of the generator. The proof sketch also gives us an opportunity to point out how the

argument translates to other types of algorithms we consider; we provide these observations following the proof sketch.

PROOF (sketch of Lemma 2.9). The argument goes by contradiction: we assume a test T computable by a circuit of size s and depth d that ϵ-distinguishes the output of $\mathrm{NW}_{H;n,m}$ from uniform in the sense that

$$|\Pr_{X\leftarrow U_n, R\leftarrow U_m}[T(X,R)=1] - \Pr_{X\leftarrow U_n}[T(\mathrm{NW}_{H;n,m}(X))=1]| \geq \epsilon.$$

We use T to construct a circuit that is not much larger and that computes H well on average, contradicting the assumed hardness of H. There are two parts to the argument, namely the construction of a predictor $\tilde{T}$, and the construction of a circuit that uses $\tilde{T}$ to compute H well on average.

Construction of a predictor A circuit $\tilde{T}$ is an ϵ'-predictor for $\mathrm{NW}_{H;n,m}$ if there is an index j such that when given the first $j-1$ bits of a sample from $\mathrm{NW}_{H;n,m}$, $\tilde{T}$ predicts the jth bit with success at least $\frac{1}{2}+\epsilon'$. The transformation from an ϵ-distinguisher to an ϵ'-predictor with $\epsilon' = \frac{\epsilon}{m}$ is a standard step in hardness-based pseudorandom generators. The key observation for our purposes is that the first n bits of $\mathrm{NW}_{H;n,m}$ are uniform at random and so cannot be predicted with any advantage. Thus the bit j has to fall within the extending part of $\mathrm{NW}_{H;n,m}$, which means that the original analysis carries through without any change in the parameters. Let us go through the analysis in some detail.

We consider the behavior of T on hybrid distributions D_i that output their first $n+i$ bits according to $\mathrm{NW}_{H;n,m}$ and output their remaining $m-i$ bits uniformly, for $i=0,...,m$. Notice that $D_0 \equiv U_{n+m}$ and $D_m \equiv \mathrm{NW}_{H;n,m}$ so that we have by assumption $|\Pr_{Z\leftarrow D_0}[T(Z)=1] - \Pr_{Z\leftarrow D_m}[T(Z)=1]| \geq \epsilon$. Using this fact we have that

$$\begin{aligned}
\epsilon &\leq |\Pr_{Z\leftarrow D_0}[T(Z)=1] - \Pr_{Z\leftarrow D_m}[T(Z)=1]| \\
&= |\sum_{i=1}^{m} \Pr_{Z\leftarrow D_i}[T(Z)=1] - \Pr_{Z\leftarrow D_{i-1}}[T(Z)=1]| \\
&\leq \sum_{i=1}^{m} |\Pr_{Z\leftarrow D_i}[T(Z)=1] - \Pr_{Z\leftarrow D_{i-1}}[T(Z)=1]|,
\end{aligned}$$

so there must exist an index i for which $|\Pr_{Z\leftarrow D_i}[T(Z)=1] - \Pr_{Z\leftarrow D_{i-1}}[T(Z)=1]| \geq \frac{\epsilon}{m}$. From this point, an averaging argument shows that there is a way to fix the last $m-i+1$ bits so that either T or $\neg T$ with these bits fixed indeed

predicts the $(n+i)^{th}$ bit of $\mathrm{NW}_{H;n,m}$ with success $\frac{1}{2}+\frac{\epsilon}{m}$ when given the first $n+i-1$ bits. We let $\tilde{T}$ be this circuit, so we have that

$$\Pr_{X \leftarrow U_n}[\tilde{T}(X, H(X|_{S_1}), ..., H(X|_{S_{i-1}})) = H(X|_{S_i})] \geq \frac{1}{2} + \frac{\epsilon}{m}.$$

Using $\tilde{T}$ to compute H In this part of the argument, we use $\tilde{T}$ to construct a circuit not much larger than the circuit for T that computes H well on average. An averaging argument shows that there is a way to fix the bits in X that are outside of S_i to preserve the prediction probability of $\tilde{T}$. Let $\tilde{Y}$ denote a string of length n that has these positions of X fixed to these values and with $X|_{S_i} = Y$. Then we have that

$$\Pr_{Y \leftarrow U_\ell}[\tilde{T}(\tilde{Y}, H(\tilde{Y}|_{S_1}), ..., H(\tilde{Y}|_{S_{i-1}})) = H(Y)] \geq \frac{1}{2} + \frac{\epsilon}{m}. \tag{A.4}$$

Consider $H(\tilde{Y}|_{S_j})$ for some $1 \leq j \leq i-1$. Notice that $\tilde{Y}$ has all bits fixed except those indexed by S_i, so for each $1 \leq j \leq i-1$, the function $H(\tilde{Y}|_{S_j})$ is a function that depends on only $|S_j \cap S_i|$ many bits – which by construction is most $k = O(\log m / \log n)$. We plug in either a DNF or CNF into $\tilde{T}$ for each of these functions, and we are left with a circuit that computes H on inputs of length $\ell = \lfloor \sqrt{n}/2 \rfloor$ with success at least $\frac{1}{2} + \frac{\epsilon}{m}$.

Parameters Consider the size and depth of the circuit that we have created. $\tilde{T}$ has the same size and depth as T, and to this we have added at most m circuits for the functions $H(\tilde{Y}|_{S_j})$, each of which is a CNF or DNF of size $2^{O(k)} = 2^{O(\log m / \log n)}$. Choosing either a CNF or DNF for each to ensure the depth increases only by one, this yields the parameters stated in Item (i) of Lemma 2.9. The efficiency of constructing the generator, Item (ii), follows by the efficiency of the designs of Lemma A.2. □

Remark The argument in the proof of Lemma 2.9 can be adapted for (non-uniform) models of computation other than circuits. We point out the modifications and observations about the above proof we need for the models we consider.

- Relativized circuits.
  The above argument carries through when both the circuits underlying the hardness hypothesis and the circuits underlying the tests can have gates that compute some fixed oracle O. Such oracle gates contribute their number of inputs to the size of the circuit. In particular, if H has

the stated hardness against circuits that have oracle gates for an oracle O, then $\mathrm{NW}_{H;n,m}$ is ϵ-pseudorandom for tests T with the stated parameters that have access to O oracle gates.

- Circuits with a limited number of special gates.
  If the tests T of Item (i) of Lemma 2.9 are allowed a certain number of special gates (e.g., gates for arbitrary symmetric functions), then $\mathrm{NW}_{H;n,m}$ is ϵ-pseudorandom for T provided H has the stated hardness against circuits that have access to the same exact number and type of special gates as the tests T. This follows from the argument above because the circuit that approximates H consists of a single copy of the test circuit T or its negation, with some of its input bits fixed and others computed by small regular circuits without special gates.

- Branching programs.
  The correctness argument carries over as such for branching programs instead of circuits. The size parameter in Item (i) becomes slightly different. Each of the functions $H(\tilde{Y}|_{S_j})$ can be computed by a branching program of size $2^{O(k)}$. Incorporating those into the branching program for $\tilde{T}$ means replacing some edges of the branching program for $\tilde{T}$ with a branching program of size $2^{O(k)}$, resulting in an overall blowup in size of $2^{O(k)}$ for $k = O(\log m/\log n)$. Thus, if H is $(\frac{1}{2} - \frac{\epsilon}{m})$-hard at input length $\lfloor\sqrt{n}/2\rfloor$ for branching programs of size $s \cdot 2^{O(\log m/\log n)}$ then $\mathrm{NW}_{H;n,m}$ is ϵ-pseudorandom for tests $T : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ computable by branching programs of size s.

- Communication protocols.
  In the proof of Theorem 4.2 in Section 4.3 we use a hardness-based pseudorandom generator $G_{H;n,\ell,m}$ that can be seen as a degenerate form of the Nisan-Wigderson construction with the sets S_i pairwise disjoint. The above proof carries through for this generator as well. Namely, let T be a randomized communication protocol taking k-tuples of n bit inputs and using m bits of randomness and q bits of communication that ϵ-distinguishes the output of the generator. Then the approximation to H given in (A.4) is within $\frac{1}{2} + \frac{\epsilon}{m}$ of H on k-tuples of ℓ-bit strings. The approximation can be computed by running the protocol T or its negation with certain input bits fixed and others set to the outcome of $H(\tilde{Y}|_{S_j})$ for some $j < i$. As the S_j are chosen disjointly for the generator $G_{H;n,\ell,m}$, $H(\tilde{Y}|_{S_j})$ is a function with *all* input bits fixed and therefore does not require any additional communication between the players. Altogether, the

approximation given in (A.4) can be computed by a non-uniform protocol that uses q bits of communication.

We conclude that if H is $(\frac{1}{2}-\frac{\epsilon}{m})$-hard for non-uniform protocols operating on k-tuples of ℓ-bit inputs that use q bits of communication then $G_{H;n,\ell,m}$ is ϵ-pseudorandom for non-uniform randomized communication protocols that operate on k-tuples of n-bit inputs, use m random bits, and q bits of communication.

B. The Permanent and Arithmetic Circuit Zero Testing

For completeness we include a proof of Lemma 6.7 showing how to reduce the correctness of an arithmetic circuit for the permanent over $\mathbb{Z}$ to an instance of ACZ.

Proof (of Lemma 6.7). We use the following notation. Let M be an m-by-m matrix M, $0 \leq k \leq m$, and $1 \leq i, j \leq k$. We denote by $M^{(k)}$ the matrix obtained by taking the m-by-m identity matrix and replacing the top left k-by-k submatrix by the corresponding submatrix of M. By $M^{(k-1)}_{-i,-j}$ we denote the same for $k-1$ but starting from the matrix M with the i-th row and j-th column deleted.

We have that C correctly computes the permanent of m-by-m matrices over $\mathbb{Z}$ iff for each $1 \leq k \leq m$, the polynomial

$$\tilde{C}_k = C(X^{(k)}) - \sum_{j=1}^{k} C(X^{(k-1)}_{-k,-j}) \cdot x_{kj}$$

is identically zero, as well as the polynomial $\tilde{C}_0 = C(X^{(0)}) - 1$, where X denotes an m-by-m matrix of variables $(x_{ij})_{i,j=1}^{m}$. By introducing one more variable x_0, those conditions can be expressed equivalently as whether the following polynomial is identically zero: $\tilde{C} = \sum_{k=0}^{m} \tilde{C}_k \cdot x_0^k$. The straightforward implementation of $\tilde{C}$ given C yields an arithmetic circuit that consists of $O(m^2)$ copies of C and some simple additional circuitry. That arithmetic circuit is in ACZ iff C correctly computes the permanent on m-by-m matrices over $\mathbb{Z}$. □

Manuscript received August 16, 2010

Jeff Kinne
Dept. of Math and Computer Science
Indiana State University
Terre Haute, Indiana 47809
USA
jkinne@indstate.edu

Dieter van Melkebeek
Department of Computer Sciences
University of Wisconsin-Madison
1210 West Dayton Street
Madison, WI 53706-1685
USA
dieter@cs.wisc.edu

Ronen Shaltiel
Department of Computer Science
University of Haifa, 31905
Israel
ronen@cs.haifa.ac.il