# DERANDOMIZING ARTHUR-MERLIN GAMES AND APPROXIMATE COUNTING IMPLIES EXPONENTIAL-SIZE LOWER BOUNDS

Barış Aydinlioğlu, Dan Gutfreund,
John M. Hitchcock, and Akinori Kawachi

**Abstract.** We show that if Arthur-Merlin protocols can be derandomized, then there is a language computable in deterministic exponential-time with access to an NP oracle, that requires circuits of *exponential* size. More formally, if every promise problem in prAM, the class of promise problems that have Arthur-Merlin protocols, can be computed by a deterministic polynomial-time algorithm with access to an NP oracle then there is a language in $\mathrm{E}^{\mathrm{NP}}$ that requires circuits of size $\Omega(2^n/n)$. The lower bound in the conclusion of our theorem suffices to construct pseudorandom generators with exponential stretch.
We also show that the same conclusion holds if the following two related problems can be computed in polynomial time with access to an NP-oracle: (i) approximately counting the number of accepted inputs of a circuit, up to multiplicative factors; and (ii) recognizing an approximate lower bound on the number of accepted inputs of a circuit, up to multiplicative factors.

**Keywords.** Approximate counting, Arthur-Merlin protocols, circuit complexity, derandomization.

**Subject classification.** 68Q15, 68Q17.

## 1. Introduction

**1.1. Background.** The fascinating connection between the existence of explicit functions that cannot be computed by small Boolean circuits and efficiently computable pseudorandom generators (PRGs) that suffice for derandomization, is one of the greatest achievements of complexity theory. The following two are equivalent (Impagliazzo and Wigderson (1997)):

1. There exists a language in the class $\mathrm{E} = \mathrm{TIME}(2^{O(n)})$ that requires Boolean circuits of size $2^{\Omega(n)}$ to be computed.
2. There exists a PRG $G : \{0,1\}^m \to \{0,1\}^n$ that is computable in time $\mathrm{poly}(n)$ and fools Boolean circuits of size $\mathrm{poly}(n)$, where $n = 2^{\Omega(m)}$.

It follows that both these items imply derandomization of probabilistic polynomial-time algorithms with only polynomial-time overhead in the running time (in the sequel we call such derandomizations *full*). Namely, BPP = P. The connection between computational hardness and derandomization, which was first suggested by Blum and Micali (1984) and Yao (1982) and was later coined *hardness vs. randomness*, supported the common belief (or maybe even is the origin of the belief) that in algorithmic settings, randomness does not enhance computational power in a significant way. Furthermore, it pointed out a tight relation between two central concepts in computational complexity: circuit lower bounds and pseudorandomness.

In fact, this connection is so deep and profound that it extends to many other settings. Klivans and van Melkebeek observed that the proof of Impagliazzo and Wigderson (1997) relativizes and thus extends to other complexity classes (Klivans and van Melkebeek (2002)). For example, one can add an NP oracle to all the machines and circuits involved in the foregoing equivalence and obtain the derandomization of the class $\mathrm{BPP}^{\mathrm{NP}}$ (of languages that can be computed in probabilistic polynomial-time with access to an NP oracle), assuming hardness against circuits having access to an NP-oracle. They also showed, under a similar assumption, a full derandomization of the class AM (of languages for which membership can be proven via a constant-round interactive proof), i.e. AM = NP. This result was later improved by Miltersen and Vinodchandran (2005); Shaltiel and Umans (2005); Umans (2003) who obtained an equivalence between hardness and pseudorandomness as above in the nondeterministic setting. That is, hardness against non-deterministic circuits (instead of NP-oracle circuits) is equivalent to PRGs that fool non-deterministic circuits and hence implies the derandomization of the class AM.

The foregoing equivalence also extends to other settings of parameters. For example, one can weaken the lower bound in Item 1 to hold against circuits of size $\mathrm{poly}(n)$, and then weaken the quality of the PRG in Item 2 so it only has a polynomial stretch, i.e., $n = \mathrm{poly}(m)$. This in turn implies a weaker derandomization of BPP placing it in the class SUBEXP (Babai, Fortnow, Nisan, and Wigderson (1993)). Furthermore, there is a smooth transition of tradeoffs between the hardness in Item 1 and the quality of the PRG in Item 2, where the exponential setting of parameters that we stated above is at the one extreme (called the *high-end*) and the polynomial setting is at the other

(the *low-end*) (Umans (2003)).

Unfortunately, it is a challenging task to prove lower bounds for circuit size in general, and the hardness vs. randomness paradigm has been useful in obtaining *unconditional* derandomizations only in very limited computational models (Nisan (1991); Viola (2007)). A natural question then arises: Do we really need to prove circuit lower bounds (or equivalently construct PRGs) in order to derandomize more general randomized complexity classes such as BPP or AM? Several works investigated this question and showed that in some settings the answer is yes, i.e., derandomization itself implies circuit lower bounds![1]

The first result of this flavor was given by Buhrman, Fortnow, and Thierauf (1998) who showed that if the class MA is equal to NP then NEXP $\not\subseteq$ P/poly.[2] Impagliazzo, Kabanets, and Wigderson (2002) significantly strengthened this result by showing that if MA is contained in subexponential nondeterministic time then NEXP $\not\subseteq$ P/poly. A similar conclusion follows from the derandomization of the class prBPP (of promise problems that can be solved in probabilistic polynomial-time) since it implies the derandomization of the class MA. Kabanets and Impagliazzo showed that if the problem of Polynomial Identity Testing, which is known to be in BPP, is in SUBEXP, then either NEXP $\not\subset$ P/poly or computing the Permanent cannot be done by polynomial-size arithmetic circuit (Kabanets and Impagliazzo (2004)). Results of a similar flavor were given in Santhanam (2007) Arvind and Mukhopadhyay (2008); Dvir, Shpilka, and Yehudanoff (2009); Kinne, van Melkebeek, and Shaltiel (2009).

While the lower bounds obtained from the derandomization assumptions in the aforementioned results are not strong enough to obtain PRGs that im-

[1]By 'derandomization' we mean showing that a randomized class is contained in a deterministic class in a non-trivial way. Note that PRGs are sufficient for derandomization but are not known to be necessary. Furthermore, while PRGs directly yield circuit lower bounds, it is not clear if derandomization does so in general. Very recently, Goldreich (2010) showed that in the context of *uniform* derandomization, PRGs and derandomization are equivalent. Informally, there exist PRGs that fool efficient (uniform) algorithms if and only if the class prBPP can be fully derandomized *on the average* (i.e., it is infeasible for efficient algorithms to generate instances on which the derandomization fails). Goldreich (2010) also shows an equivalence between the full derandomization of prBPP *on the worst case* and the existence of targeted PRGs. Roughly speaking, these are PRGs that are given as an auxiliary input the description of the statistical test that needs to be fooled. In the context of this work, it is important to point out that PRGs that fool uniform algorithms as well as targeted PRGs do not seem to directly imply circuit lower bounds.

[2]This result is actually a corollary of their main result and is not mentioned in their paper. It is explicitly stated in Impagliazzo, Kabanets, and Wigderson (2002) (see Remark 26).

ply back the derandomization assumptions, they still suggest that the two-way connections between hardness and pseudorandomness also extend to derandomization. A natural question is how general this phenomenon is? Is it as general as the equivalence between circuit lower bounds and pseudorandomness which holds in so many different settings? Can we extend it to other settings of parameters or models of computation?

Note that all of the aforementioned results start with the assumption that a weak derandomization is possible (placing some probabilistic class in a subexponential class that does not require probability) and conclude in a lower bound for superpolynomial-size (either Boolean or arithmetic) circuits. Thus the connections hold in the low-end setting of parameters, and in particular, they only imply PRGs with polynomial stretch. (We mention that some of the results do not imply PRGs at all since they obtain lower bounds which are seemingly too weak for the construction of PRGs). An exception is Kinne, van Melkebeek, and Shaltiel (2009) who gave an alternative proof to Kabanets and Impagliazzo (2004) for which the parameters scale better. Thus they obtain results also for parameter settings in between the low-end and the high-end. However, their proof still falls short from proving a connection for the high-end (namely an exponential-size lower bound from full derandomization), and furthermore, their lower bounds, just like Kabanets and Impagliazzo (2004), are with respect to arithmetic circuits and thus do not imply PRGs that fool Boolean circuits. Furthermore, inspecting the proofs of all other results mentioned above, one can see that they do not imply stronger lower bounds and PRGs even if full derandomization is assumed.

**1.2. Our Results.** In this paper we extend the connections among derandomization, circuit lower bounds and PRGs to the high-end setting, by showing that a full derandomization of a probabilistic class (and in fact a certain task) implies exponential-size circuit lower bounds and PRGs with exponential stretch. This is done in the context of derandomizing AM.

**Arthur-Merlin games.** The probabilistic class we consider is the promise version of the class of languages that can be accepted by an Arthur-Merlin game, denoted prAM.

THEOREM 1.1. *If every promise problem in* prAM *can be computed in deterministic polynomial-time with access to an* NP *oracle, then there exists a language in* $\mathrm{E}^{\mathrm{NP}}$ *that requires deterministic circuits of size* $\alpha 2^n/n$ *for some positive constant* $\alpha$ *and all but finitely many input lengths* $n$.

Recall that $P^{NP}$ denotes the class of languages that can be computed in deterministic polynomial-time with an NP oracle, and $E^{NP}$ its linear-exponential analogue. See Section 2 for definitions of promise problems and of prAM.

Note that a full derandomization of prAM would mean that every problem in it has a solution in the class NP. Our derandomization assumption is seemingly weaker since NP∪coNP $\subseteq P^{NP}$. Note also that the circuit lower bound achieved, albeit only for deterministic circuits, is close to the maximal one, which is $\Theta(2^n/n)$ (Shannon (1949)). (On the other hand, we note that in order to achieve a full derandomization of prAM we currently need lower bounds for the class NE ∩ coNE with respect to nondeterministic circuits.)

Combining the results of Klivans and van Melkebeek (2002); Nisan and Wigderson (1994) and Theorem 1.1 we get that the derandomization of the class prAM implies the existence of PRGs with *exponential* stretch that can be computed efficiently with an NP oracle, and fool deterministic Boolean circuits.

Theorem 1.1 is a step towards the converse of the hardness vs. randomness tradeoffs of Miltersen and Vinodchandran (2005); Shaltiel and Umans (2005); Umans (2003), namely that the existence of a language in the class $E^{NP}$ that requires *nondeterministic* circuits of size $2^{\Omega(n)}$ implies the existence of a PRG with exponential stretch that can be computed efficiently with an NP oracle, and fools *nondeterministic* Boolean circuits. This in turn implies that prAM $\subseteq P^{NP}$.[3] Finally, we note that the lower bound in the conclusion of our theorem does imply the inclusion prMA $\subseteq P^{NP}$.

**Approximate lower bound problem.** Consider the following promise problem, which we call the *approximate lower bound problem*[4]: Given a nondeterministic circuit, decide whether it accepts on a large fraction of its input settings or on a significantly smaller fraction, where "large" and "significantly smaller" are quantified by inputs to the problem, and where we are promised that one of the two cases hold. (A formal statement of this problem is implicit in Theorem 1.2 below.) This problem is complete for the class prAM (Goldwasser and Sipser (1989)); hence the circuit lower bound in Theorem 1.1 follows by derandomizing just this particular problem.

Now consider a special case of the approximate lower bound problem, where the given circuit is deterministic instead of nondeterministic. It turns out that derandomizing this seemingly easier problem suffices to yield the circuit lower

[3] We commit a common abuse of notation and mean by the inclusion pr$\mathcal{C}_1 \subseteq \mathcal{C}_2$ that for any promise problem $\Pi \in$ pr$\mathcal{C}_1$, there is some language in $\mathcal{C}_2$ that agrees with $\Pi$ on the promise. See Section 2 for details.

[4] also called "set lower bound protocol", "Goldwasser-Sipser protocol" in some texts.

bound in Theorem 1.1.

THEOREM 1.2. *Let $C$ denote a Boolean circuit, and $b$ an integer in binary. If for some positive constant $\delta$ there is a language in $\mathrm{P}^{\mathrm{NP}}$ that contains all instances $(C, b)$ where $|C^{-1}(1)| \geq b$ and does not contain any instances where $|C^{-1}(1)| < \delta \cdot b$, then there is a language in $\mathrm{E}^{\mathrm{NP}}$ that requires circuits of size $\alpha 2^n/n$ for some positive constant $\alpha$ and all but finitely many input lengths $n$.*

**Approximate counting.** A close relative of the approximate lower bound problem is the problem of computing an estimate on the number of inputs accepted by a (deterministic) circuit. Specifically, given $(C, \eta)$, the problem is to output a number $c$ such that $(1-\eta)|C^{-1}(1)| \leq c \leq (1+\eta)|C^{-1}(1)|$.

The result of Shaltiel and Umans (2006) implies that if $\mathrm{E}^{\mathrm{NP}}$ requires exponential-size nondeterministic circuits, then approximate counting can be done in time polynomial in $1/\eta$ and in the size of $C$, when given access to an NP oracle. Theorem 1.2 implies a complement to this, namely that if approximate counting can be computed as stated, then $\mathrm{E}^{\mathrm{NP}}$ requires exponential-size (albeit deterministic) circuits. This follows by setting $\delta < (1-\eta)/(1+\eta)$; in this case the approximation with parameter $\eta$ is enough to distinguish the two cases of Theorem 1.2 with parameter $\delta$.

**Parameterization.** In fact we show a yet stronger result, of which Theorem 1.2 is a special case. Namely, we show a parameterized version of Theorem 1.2 that reveals a randomness vs. hardness tradeoff for the approximate lower bound problem.

THEOREM 1.3. *Let $t$, $1/\delta$, and $s$ be functions such that $t$ and $1/\delta$ are monotone, $1/\delta$ and $s$ are constructible, and $m/\delta(m) = O(2^n)$ whenever $m(n) = O(s(n)\log s(n))$. Let $C$ denote a Boolean circuit on $m$ inputs, and $b$ an integer in binary. If there is a language in $\mathrm{DTIME}(t(n))^{\mathrm{NP}}$ that contains all instances $(C, b)$ where $|C^{-1}(1)| \geq b$ and does not contain any instances where $|C^{-1}(1)| < \delta(m) \cdot b$, then there is a language in $\mathrm{DTIME}(t'(n))^{\mathrm{NP}}$ that requires circuits of size $s(n)$ for all but finitely many input lengths $n$, where*

$$t'(n) = t\left((s(n) \cdot \frac{1}{\delta(O(s(n)\log s(n)))})^{O(1)}\right).$$

Theorem 1.3 gives the following interesting instantiations that yield a hard language in $\mathrm{E}^{\mathrm{NP}}$.

COROLLARY 1.4. *For each combination of the parameters $t$, $\delta$, and $s$ in the table below, under the hypothesis of Theorem 1.3, there is a language in* $\mathrm{E}^{\mathrm{NP}}$ *that requires circuits of size $s(n)$ for all but finitely many input lengths $n$.*

| $t(n)$ | $\delta(n)$ | $s(n)$ |
|---|---|---|
| $n^{O(1)}$ | $\Omega(1)$ | $\Omega(2^n/n)$ |
| $2^{(\log n)^{O(1)}}$ | $1/2^{(\log n)^{O(1)}}$ | $2^{n^{\Omega(1)}}$ |
| $2^{n^{o(1)}}$ | $1/n^{O(1)}$ | $n^{\omega(1)}$ |

Note that Theorem 1.2 follows from the first line of the table.

**1.3. Our Techniques.** We present two approaches to establish results like ours. The first one is rather elementary, yet it yields the full strength of Theorem 1.1, Theorem 1.2, and Theorem 1.3. The second one is more involved, as it builds on a number of earlier works, and yields quantitatively weaker results. We nevertheless include the second approach as it may be useful in obtaining circuit lower bounds in other contexts. The second approach was discovered earlier and appears in the preliminary version of this paper, Gutfreund and Kawachi (2010). The first approach was reported in Aaronson, Aydınlıoğlu, Buhrman, Hitchcock, and van Melkebeek (2010).

Underlying both of our approaches are ideas that date back to Kannan (1982) who used it to prove unconditional circuit lower bounds. Consider the problem of computing, within the polynomial-time hierarchy, a language $L$ that is hard for circuits of size $s := n^k$ for some $k > 0$ (for all but finitely many lengths $n$). By a counting argument, the number of size-$s$ circuits is less than the number of strings of length $\ell := s^{1+\Omega(1)}$. Hence, if we view each length-$\ell$ string as the truth table of a function on $\log \ell$ bits (assuming wlog that $\ell$ is a power of 2), we see that the task of computing a hard language $L$ on $n$ bits can be accomplished by first finding the truth table of a hard language $L'$ on $\log \ell = O(\log n)$ bits, then trivially extending this truth table (say, by padding with zeroes) so that it corresponds to a function $L$ on $n$ bits, and finally returning the entry in the truth table that corresponds to the given input. Now, observe that it is a coNP task to decide whether a given truth table of length $\ell$ is hard (i.e., whether the truth table corresponds to a language $L'$ that is hard for size-$s$ circuits). Observe further that it is a $\Sigma_2^{\mathrm{P}}$ task to decide whether a given string is the prefix of some length-$\ell$ truth table that is hard. It follows that by doing a binary search we can find the lexicographically least truth table of a hard $L'$, hence of a hard $L$, in $\mathrm{P}^{\Sigma_2^{\mathrm{P}}}$.

The argument in the preceding paragraph gives fixed-polynomial-size circuit lower bounds in the polynomial-time hierarchy. To prove exponential-size circuit lower bounds in the exponential-time hierarchy, one runs the above argument with $s := 2^{\Theta(n)}$ instead of $s := n^k$. Thus, one gets a function in $\mathrm{E}^{\Sigma_2^{\mathrm{P}}}$ that requires circuits of linear-exponential size. Though quite simple, this is currently (as it has been for almost 30 years) the best known exponential-size lower bound.

Both our proofs essentially show that in order to find exponentially hard truth-tables we can replace the $\Sigma_2^{\mathrm{P}}$ oracle in the above argument with a prAM oracle. The derandomization hypothesis then implies a function in $\mathrm{E}^{\mathrm{NP}}$ that requires circuits of linear-exponential size. Note that prAM $\subseteq \Pi_2^{\mathrm{P}}$ (Fürer, Goldreich, Mansour, Sipser, and Zachos (1989)) (and clearly oracle access to $\Sigma_2^{\mathrm{P}}$ is equivalent to oracle access to $\Pi_2^{\mathrm{P}}$), so our oracle is certainly not stronger than the oracle in Kannan's proof, and is widely believed to be weaker. Indeed our result implies the lower bound for $\mathrm{E}^{\Sigma_2^{\mathrm{P}}}$. In Section 5 we explain why we nevertheless *do not* prove a new explicit lower bound.

Our two proofs differ in the way they substitute the $\Sigma_2^{\mathrm{P}}$ oracle with a prAM oracle. In our first (i.e., elementary) proof, instead of using a $\Sigma_2^{\mathrm{P}}$ oracle for finding the lexicographically least truth table that is hard via binary search, we use a prAM oracle for finding *some* hard truth table via *approximate halving*. Specifically, we construct a truth table by successively setting its the next bit so as to eliminate a significant fraction of those circuits that agree with the truth table constructed thus far. We accomplish this by using the prAM oracle to get an approximate count on the number of circuits that would be eliminated in case the next bit in the truth table is set to zero or to one. Approximate counting ensures that at each iteration, although we may not eliminate half of the remaining circuits (as we could if we had an oracle that did exact counting), we do eliminate a significant fraction of them. Hence we build a hard truth table fairly quickly.

Our alternate proof builds on yet another strategy of Kannan (1982) who used it to improve his above-described lower bound on fixed-polynomial-size circuits. The idea is to use the well-known result of Karp and Lipton (1980) that if SAT is computable by polynomial size circuits (for almost all input lengths), then PH, the polynomial-time hierarchy, collapses to $\Sigma_2^{\mathrm{P}}$. Specifically, consider two cases: SAT has circuits of size $s := n^k$ for some $k > 0$, or not. In the latter case we can just take $L = \mathrm{SAT}$, since the language SAT itself has the desired hardness (albeit only for infinitely many input lengths $n$ rather than all but finitely many). In the former case, due to the collapse of PH, the binary search for finding a hard truth table is now computable in $\Sigma_2^{\mathrm{P}}$ rather

than $\mathrm{P}^{\Sigma_2^{\mathrm{P}}}$. It follows that for every $k > 0$ there is a language in $\Sigma_2^{\mathrm{P}}$ that is hard for size-$n^k$ circuits. Note that it does not seem possible with this approach to achieve lower bounds that hold on almost all input lengths.

This argument is fruitful in obtaining fixed-polynomial size circuit lower bounds further down in PH, because it can leverage any improvement to the Karp and Lipton (1980) result that gives a stronger collapse consequence of PH. For example, the current strongest consequence of SAT having polynomial size circuits, due to Cai (2007), is that PH collapses to $\mathrm{S}_2^{\mathrm{P}}$. Plugging this result into the preceding argument immediately yields a hard language in $\mathrm{S}_2^{\mathrm{P}}$ instead of in $\Sigma_2^{\mathrm{P}}$.

Despite its usefulness in obtaining fixed-polynomial size circuit lower bounds, it is not clear how this argument can be scaled up to give exponential size circuit lower bounds. Specifically, the problem lies in the case distinction on the hardness of SAT. For example, if we replace $s := n^k$ with $s := 2^{\Theta(n)}$, then the case that SAT is hard for size-$s$ circuits scales up fine, but then we do not know how to handle the first case that SAT has size-$s$ circuits. In particular, we do not know of any Karp-Lipton style collapse[5] with that setting of $s$. We may attempt to replace SAT with a seemingly harder language, say some language in a class $\mathcal{C}$, where $\mathcal{C}$ is contained in the class for which we want to show a lower bound. Unfortunately this issue cannot be fixed this way; for otherwise we would not have the current gap between the known super-polynomial size lower bounds (which hold with respect to classes that are contained in the second level, namely MA-EXP (Buhrman, Fortnow, and Thierauf (1998))), and the known exponential-size lower bounds (which only hold with respect to classes that are in the third level of the hierarchy, namely $\mathrm{E}^{\Sigma_2^{\mathrm{P}}}$ (Kannan (1982))). Indeed, it was argued by Miltersen, Vinodchandran, and Watanabe (1999) that Karp-Lipton style collapses that are needed for Kannan's strategy hold with respect to size functions up to *half-exponential* (a function $s$ is half-exponential if $s(s(n)) \in 2^{\Theta(n)}$) but do not seem to carry over to larger size bounds, to $2^{\Theta(n)}$ in particular.

The main contribution of our alternate proof is a way to scale up this easy/hard case analysis and obtain the desired exponential circuit lower bounds. For this we exploit a connection with learning theory. We refer to Section 4 for more intuition and details. We believe that this approach may be useful in scaling fixed-polynomial-size circuit lower bounds to exponential level.

[5]Results that show the containment of some uniform class in a non-uniform class imply a collapse of high uniform classes into lower classes are called Karp-Lipton style collapses (after Karp and Lipton (1980) who were the first to show such a result).

## 2. Basic Notions and Notation

For a Boolean function $f : \{0,1\}^* \to \{0,1\}$, we denote by $f^n$ the restriction of $f$ to instances of length $n$. We use "language" and "Boolean function" interchangeably. For a (possibly infinite) family of circuits $\mathcal{C}$ we denote by $\mathcal{C}^n$ the circuits in $\mathcal{C}$ with exactly $n$ input gates.

For an integer $n > 0$, we denote by $[n]$ the set $\{1, \ldots, n\}$. For a string $s \in \{0,1\}^*$ we denote by $|s|$ the length of $s$. For two strings $s, t \in \{0,1\}^*$ we denote by $s \circ t$ their concatenation.

**2.1. Complexity Classes.** We assume that the reader is familiar with standard complexity classes such as P, NP, E etc. For a class of (oracle-aided) algorithms $\mathcal{A}$ and a class of functions $\mathcal{F}$ we denote by $\mathcal{A}^{\mathcal{F}}$ the class of functions that are computable by some algorithm in $\mathcal{A}$ that is given oracle (i.e., unit cost) access to a function in $\mathcal{F}$.

Often when we describe algorithms that use as oracle some function in a class $\mathcal{F}$ it is convenient to actually assume that the algorithm has unit cost access to several (constant number of) functions $f_1, \ldots, f_c$ all in $\mathcal{F}$. We can then think of the algorithm having access to a single function in $\mathcal{F}$ by binding the functions to a single function $f(i, x) = f_i(x)$ for $1 \leq i \leq c$. This holds whenever the function class is closed under such composition, which is always the case in this paper.

For a size function $s : \mathbb{N} \to \mathbb{N}$, we denote by $\mathrm{SIZE}(s(n))$ the class of languages computable by $s(n)$-size $n$-input Boolean circuits, where we assume the standard bounded fan-in model; the circuits consist of 2-input AND and OR gates, and 1-input NOT gates. We measure the size of a circuit by the number of its gates.

We recall a standard fact regarding circuit descriptions that we use in subsequent sections. Every $s(n)$-size $n$-input Boolean circuit can be described by a string of length $\Theta(s(n) \log s(n))$. Precisely, such a circuit can be described by a sequence of $s(n)$ triples, where each triple describes a gate with $O(1)+2\log s(n)$ bits: a constant number of bits for the gate type, $\log s(n)$ bits for identifying the first input of the gate, and another $\log s(n)$ bits for the second input, where a gate is identified by the position of the triple describing it. It follows that $3s(n) \log s(n)$ bits suffice to describe all $n$-input Boolean circuits of $s(n)$-size. We interpret each string of length $3s(n) \log s(n)$ as the description of some such circuit. We note that from its description such a circuit can be simulated in time $\mathrm{poly}(s(n))$.

For more details on complexity classes, see textbooks of Goldreich (2008) and Arora and Barak (2009).

**2.2. Promise Problems.** The notion of promise problems were introduced by Even, Selman, and Yacobi (1984); see the survey by Goldreich (2005). Recall that a promise problem $\Pi$ is defined by two disjoint sets $\Pi^Y \subseteq \{0,1\}^*$ which we call the 'yes' instances of $\Pi$, and $\Pi^N \subseteq \{0,1\}^*$ which we call the 'no' instances of $\Pi$. A function $f : \{0,1\}^* \to \{0,1\}$ agrees with a promise problem $\Pi$, if $f(x) = 1$ for every $x \in \Pi^Y$, $f(x) = 0$ for every $x \in \Pi^N$ and $f(x)$ can take any value in $\{0,1\}$ if $x \notin \Pi^Y \cup \Pi^N$.

For a class of algorithms $\mathcal{A}$ and a class of promise problems $\mathcal{F}$, a function $g : \{0,1\}^* \to \{0,1\}$ is said to be in the class $\mathcal{A}^{\mathcal{F}}$, if there exists an algorithm $A \in \mathcal{A}$ and a promise problem $\Pi \in \mathcal{F}$, such that when $A$ is given oracle access to *any* function $f$ that agrees with $\Pi$, it computes the function $g$. In other words, while $A$ may ask queries which are not in $\Pi^Y \cup \Pi^N$ and hence receive arbitrary answers, it must compute the same function $g$ regardless of the values of these arbitrary answers.

We take a fairly standard, though formally inaccurate, point of view and say that a class of promise problems $\mathcal{F}$ is contained in a class of Boolean functions $\mathcal{C}$, if for every promise problem $\Pi \in \mathcal{F}$, there exists a function in $\mathcal{C}$ which agrees with $\Pi$.[6]

The class prAM contains all the promise problems for which there is an Arthur-Merlin protocol whose completeness holds with respect to all the 'yes' instances and the soundness holds with respect to all the 'no' instances. The protocol may behave arbitrarily on instances which are not in $\Pi^Y \cup \Pi^N$. Formally, prAM is defined as follows.

Definition 2.1. *We say that a promise problem $\Pi$ is in the class* prAM *if there is a polynomial-time computable relation $R(\cdot,\cdot,\cdot)$ such that the following holds:*

- ***Completeness:*** *For every $x \in \Pi^Y$, $\Pr_r[\exists y \text{ such that } R(x,y,r) = 1] \geq 2/3$*
- ***Soundness:*** *For every $x \in \Pi^N$, $\Pr_r[\exists y \text{ such that } R(x,y,r) = 1] \leq 1/3$,*

*where $|r| = |y| = \text{poly}(|x|)$.*

It is well known (Babai and Moran (1988); Goldwasser and Sipser (1989)) that the definition above is equivalent to the class of all the promise problems

[6]Strictly speaking, a promise problem is a partial function that is undefined on the out-of-promise instances. We view such a problem as a member of the language class $\mathcal{C}$ if it can be extended to a (total) function in $\mathcal{C}$.

that have interactive protocols (in the model of Goldwasser, Micali, and Rackoff (1989)) with a constant number of rounds between an all-powerful prover (Merlin) and a probabilistic polynomial-time verifier (Arthur).

## 3. Elementary Approach

In this section we present an elementary proof of our results. We first describe the intuition for the proof of Theorem 1.2 and then provide the formal proof for the fully parameterized version, Theorem 1.3.

**3.1. Proof Idea.** As mentioned in Section 1.3, the idea behind the main proof is to construct a language that requires large circuits by setting the prefix of its characteristic sequence so as to quickly diagonalize against all small circuits. More specifically, we mimic the process of successively setting the next bit of the characteristic sequence to the minority vote of the circuits of size $\alpha 2^n/n$ that are consistent with the sequence constructed thus far. This ideal process would reduce the number of consistent circuits by at least half in each step, implying that we'd be done after $O(\alpha 2^n)$ steps.

Now, let $A$ be a language in $\mathrm{P}^{\mathrm{NP}}$ that solves the promise problem described in the statement of Theorem 1.2. Using $A$ as an oracle, we can mimic the above ideal process and guarantee that we reduce the number of consistent circuits by some constant factor $\beta < 1$, where $\beta$ depends on $\delta$. To do so, it suffices to approximate the number of circuits consistent with the sequence thus far extended with a zero, do the same for the extension with a one, and select the extension that gives the smaller estimate. This approximation can be obtained by a binary search on the approximate number $b$, by calling $A$ with an input $C$ that embodies the characteristic sequence so far. The process ends after $O(\alpha 2^n/\log(1/\beta))$ steps, which is less than $2^n$ for sufficiently small $\alpha$ and sufficiently large $n$. Since $A$ lies in $\mathrm{P}^{\mathrm{NP}}$, the resulting process yields a language in $\mathrm{E}^{\mathrm{NP}}$ that has no circuits of size $\alpha 2^n/n$ for all but finitely many input lengths $n$.

**3.2. Formal Proof.** We now fill in the details and give a formal proof of Theorem 1.3.

Let $s(n) \geq n$ denote the circuit lower bound we are shooting for, i.e., we want to construct a language $L$ that requires circuits of size $s(n)$ for all but finitely many input lengths $n$.

Let $A$ denote a language that solves the promise problem given in the statement of Theorem 1.3. For a given Boolean circuit $C$ on $m$ inputs and an integer $b$ in binary, $A$ contains $(C, b)$ if $|C^{-1}(1)| \geq b$ and does not contain $(C, b)$

if $|C^{-1}(1)| < \delta(m) \cdot b$. In the middle case where $|C^{-1}(1)| \in [\delta(m) \cdot b, b)$, the membership of $(C, b)$ to $A$ can be arbitrary.

The circuits $C$ we supply to $A$ take as input the description of a circuit $D$ of size $s(n)$ on $n$ inputs. Thus, $C$ takes $m = O(s(n) \log s(n))$ inputs.

We construct $L$ iteratively, where in iteration $i = 0, 1, \ldots$, we determine $\chi_i$, the $i$th symbol of the characteristic string $\chi$ of $L$. To explain iteration $i$ we denote by $S_i$ the set of circuits of size $s(n)$ on $n$ inputs that agree with $\chi$ up to its $i$th symbol, i.e., a circuit $D$ is in $S_i$ iff for $j = 0, \ldots, i-1$, $D$ outputs $\chi_j$ when given as input the $n$-bit binary encoding of integer $j$. In iteration $i$ we first tentatively set $\chi_i$ to 0 and use $A$ to obtain an estimate $\sigma_0$ on the size of $S_{i+1}$. Then we set $\chi_i$ to 1 and obtain an estimate $\sigma_1$. We finalize $\chi_i$ to the value $c \in \{0, 1\}$ such that $\sigma_c = \min(\sigma_0, \sigma_1)$ (say we set $c = 0$ in case of a tie).

To estimate $|S_{i+1}|$, first we construct a circuit $C$ that recognizes $S_{i+1}$. The circuit $C$ takes as input a binary string of length $m$ that is the description of a size $s(n)$ circuit $D$ on $n$ inputs, and returns 1 iff $D \in S_{i+1}$. More precisely, $C$ contains as hardcode the characteristic string $\chi$ constructed thus far and simulates its input $D$ on inputs $j = 0, \ldots, i$, and accepts iff $D$ agrees with $\chi$ for all $j$. Next, we run a binary search for the largest integer $b^*$ such that $(C, b^*) \in A$. Note that $(C, 0) \in A$ so $b^*$ exists. The binary search returns a value $\tilde{b}$ such that (i) $(C, \tilde{b}) \in A$ and (ii) $(C, \tilde{b} + 1) \notin A$. As $A(C, \cdot)$ may not be perfectly monotone, $\tilde{b}$ may differ from $b^*$ but the specification of $A$ guarantees that $|C^{-1}(1)| \geq \delta(m) \cdot \tilde{b}$ (because of (i)) and $|C^{-1}(1)| < \tilde{b} + 1$ (because of (ii)), so our estimate $\tilde{b}$ satisfies

$$|S_{i+1}| \leq \tilde{b} \leq |S_{i+1}|/\delta(m). \tag{3.1}$$

At the end of iteration $i$, if the smaller estimate $\sigma_c$ in iteration $i$ was 0 then by (3.1) we conclude that the diagonalization is complete and we terminate the iterations. Finally, to decide whether $x \in \{0, 1\}^n$ belongs to $L$, we interpret $x$ as an integer and accept if $x$ is less than the length of $\chi$ and the $x$th symbol of $\chi$ is 1; we reject $x$ otherwise. This completes the construction of $L$.

For the construction to work, we need to make sure that the diagonalization is completed by the time we exhaust the $2^n$ inputs of length $n$. Consider the number of circuits eliminated in round $i$. According to our estimate this number is $\sigma_{\neg c}$, but by (3.1) it may actually be as little as $\delta(m) \cdot \sigma_{\neg c}$. Since the total number of circuits under consideration at round $i$ is, again by (3.1), at most $2\sigma_{\neg c}$, it follows that at least a $\delta(m)/2$ fraction of those circuits are eliminated during round $i$. Thus, $|S_i| \leq (1 - \delta(m)/2)^i \cdot |S_0| \leq \exp(-i\delta(m)/2) \cdot 2^m$. The latter quantity is less than 1 for $i > 2\ln(2)m/\delta(m)$. So, as long as $2\ln(2)m/\delta(m) < 2^n$, we

can complete the diagonalization process as required. Note that the condition is met if $m/\delta(m) = O(2^n)$ for every $m(n) = O(s(n) \log s(n))$.

Let us now analyze the complexity of the resulting language $L$. The circuits $C$ used in the $i$th step can be constructed in time $\text{poly}(i, s(n))$. The binary search in each step requires at most $m$ calls to $A$, as $b^*$ ranges up to $2^m$. Assuming $A$ can be decided in time $t(N)$ on inputs of length $N$ (when given access to an oracle for NP), the amount of time for the $i$th step is $O(m \cdot t(\text{poly}(i, s(n)))$ (when given access to NP). By the previous paragraph, the number of steps is $O(m/\delta(m))$. Hence, given access to NP, the amount of time over all steps is

$$\begin{aligned} & O\left(\frac{m^2}{\delta(m)} \cdot t(\text{poly}(\frac{m}{\delta(m)}, s(n)))\right) \\ = \; & O\left(t(\text{poly}(s(n), \frac{1}{\delta(O(s(n)\log s(n)))}))\right), \end{aligned} \tag{3.2}$$

where we assume that $t(N)$ and $1/\delta(m)$ are monotone and, without loss of generality, that $t(N) \geq N$. If $s(n)$ and $1/\delta(m)$ are constructible as well, (3.2) also bounds the overall time complexity of $L$. We have thus proved Theorem 1.3.

## 4. Learning-Based Approach

In this section we present an alternate, learning-based approach to results like Theorem 1.1, Theorem 1.2, and Theorem 1.3.

Using this approach we do not know how to achieve the same quantitative strength as using the approach from the previous section. Nevertheless, as explained in Section 1.3, we believe that the learning-based approach has future potential for scaling fixed-polynomial circuit lower bounds to exponential circuit lower bounds.

Given the above, for simplicity of exposition, we do not attempt to obtain the strongest quantitative results the learning-based approach can give, and we only focus on the setting of Theorem 1.1. In particular, we prove the following result:

THEOREM 4.1. *If every promise problem in* prAM *can be computed in deterministic polynomial-time with access to an* NP *oracle then there exists a language in* $\text{E}^{\text{NP}}$ *that requires circuits of size* $2^{\epsilon n}$ *for some positive constant* $\epsilon$ *and infinitely many input lengths* $n$.

Note that Theorem 4.1 is weaker than Theorem 1.1 because the circuit lower bound is only $2^{\epsilon n}$ for some positive $\epsilon$ rather than $\Theta(2^n/n)$, and because

the lower bound only holds infinitely often rather than almost everywhere. The approach in this section can be strengthened to yield a circuit lower bound of $2^{\epsilon n}$ for *every* positive constant $\epsilon < 1$ and infinitely many input lenghts; however, it does not seem to yield a circuit lower bound of $\Omega(2^n/n)$ as in Theorem 1.1, neither does it seem possible with this approach to achieve a lower bound that holds almost everywhere. These limitations seem to be inherent in Kannan's argument on which our learning-based approach is based (see Section 1.3).

We start by describing the intuition. We provide the formal proof in the subsequent subsections.

**4.1. Proof Idea.** As we described in the Section 1.3, the approach of Kannan (1982) for proving fixed-polynomial-size circuit lower bounds, based on Karp-Lipton style collapses, does not seem to scale for proving exponential-size lower bounds. In particular, it seems that one needs a Karp-Lipton style collapse from the assumption that some exponential-time class has exponential-size circuits. Such results are currently not known.

Thus in order to prove Theorem 4.1 we need a different strategy. Somewhat surprisingly, our proof also goes via an easy/hard case analysis but not with respect to the classes that we are interested in, namely $\mathrm{E}^{\mathrm{NP}}$ and exponential-size circuits, but rather NP and fixed polynomial-size circuits. Consider two cases, either SAT can be computed by circuits of size, say $n^{10}$, or not. In the former case we are in a good position because we can use a Karp-Lipton style collapse. The result of Chakaravarthy and Roy (2008) shows that if SAT has polynomial-size circuits then PH collapses to the class $\mathrm{P}^{\mathrm{prAM}}$, and therefore by the derandomization assumption to $\mathrm{P}^{\mathrm{NP}}$. From here we proceed as in Kannan (1982). Namely, we can find in deterministic polynomial-time with access to an NP oracle a truth-table of a function on $O(\log n)$ bits that is hard for circuits of size $\mathrm{poly}(n)$. The exponential-size lower bound now follows by translation.

The second case, in which SAT does not have circuits of size $n^{10}$, is more interesting. This is because at a first glance, the (fixed) polynomial-size lower bound for SAT seems to have nothing to do with exponential-size lower bounds. We show how to scale this fixed polynomial-size lower bound to the exponential level, and we do that via a connection to computational learning theory. We believe that this part of the proof is of independent interest.

Let us briefly discuss some notions from computational learning theory. Let $s(n), s'(n)$ be size functions (where $s'(n) \geq s(n)$). An algorithm $A$ exactly learns a Boolean function $f$ with respect to the concept class $\mathrm{SIZE}(s)$ and hypothesis class $\mathrm{SIZE}(s')$, if the following holds: if $f$ can be computed by circuits of size $s(n)$, then $A$, on input $1^n$, outputs a circuit of size $s'(n)$ that

computes $f$ at input length $n$ (i.e., $A$ has to learn a circuit for $f$ from the hypothesis class but not necessarily from the concept class). Clearly $A$ has to receive some information (in the form of an oracle) about $f$ to achieve this task. Several such oracles were considered in the literature, the most natural one being an oracle to $f$ itself.

A classic result in computational learning theory by Bshouty, Cleve, Gavaldà, Kannan, and Tamon (1996) is that there is an algorithm that exactly learns SAT with respect to the concept class SIZE($n^k$) and hypothesis class SIZE($n^{k+3}$) (for any $k > 1$). The algorithm runs in probabilistic expected polynomial-time with access to a SAT oracle. Of course, we do not know if SAT has polynomial-size circuits. Indeed in the case that we consider, SAT cannot be computed by $n^{10}$-size circuits. So how does the algorithm of Bshouty, Cleve, Gavaldà, Kannan, and Tamon (1996) behave when SAT is not even in the hypothesis class SIZE($n^{k+3}$)? Fortnow, Pavan, and Sengupta (2008) observed that in this case the algorithm outputs a poly($n$)-long list of SAT instances such that *every* circuit of size $n^k$ fails to compute correctly the SAT-value of at least one of them.[7] We call this *a list of counterexamples*.

We proceed in two steps. First, in section Section 4.2 we show that if there is a *deterministic* learning algorithm that outputs a polynomially-long list of counterexamples, then there is an explicit function that requires exponentially large circuits (see Lemma 4.3). The complexity of computing this function is directly related to the complexity of the learning algorithm. In particular, if the algorithm runs in deterministic polynomial-time with access to an NP oracle, then there is a function in $\mathrm{E}^{\mathrm{NP}}$ that requires exponentially large circuits. Next, in section Section 4.3 we show (in Theorem 4.5), based on ideas from Bshouty, Cleve, Gavaldà, Kannan, and Tamon (1996); Chakaravarthy and Roy (2008); Fortnow, Pavan, and Sengupta (2008), that there is a *deterministic* algorithm that uses an oracle to prAM for learning counterexamples (recall that the result of Fortnow, Pavan, and Sengupta (2008) only gives a randomized algorithm, so it is not good for us). By the hypothesis of Theorem 4.1, we can replace the prAM oracle with an NP oracle and then the lower bound follows in Section 4.4.

**4.2. Learning Counterexamples Implies Exponential-Size Lower Bounds.** In this section we show the connection between the problem of learning counterexamples for SAT and exponential-size lower bounds. First, we formally define the problem of learning counterexamples.

[7] We mention that a slightly stronger statement was later given by Atserias (2006).

DEFINITION 4.2. *Let $f : \{0,1\}^* \to \{0,1\}$ be a function, and $\mathcal{C}$ a family of Boolean circuits such that $f \notin \mathcal{C}$. We say that an algorithm $A$ learns $\ell = \ell(n)$ counterexamples for $f$ with respect to the concept class $\mathcal{C}$, if for every $n$ for which $f^n \notin \mathcal{C}^n$, on input $1^n$, the algorithm outputs a list of at most $\ell$ strings $x_1, \ldots, x_\ell$ of $n$-bit length such that for every circuit $C \in \mathcal{C}^n$, there exists $1 \leq i \leq \ell$ such that $C(x_i) \neq f(x_i)$.*

The following lemma states that deterministic learning of counterexamples for SAT implies exponential-size lower bounds.

LEMMA 4.3. *Suppose that for some $c > k > 4$ there is a deterministic algorithm $A$ with access to an* NP *oracle that runs in time* $\mathrm{poly}(n)$*, such that $A$ learns a list of $n^c$ counterexamples for $SAT^n$ with respect to the concept class* $\mathrm{SIZE}(n^k)$*, for infinitely many $n \in \mathbb{N}$. Then there is a constant $\delta > 0$ (that depends only on $k$ and $c$), and a language in the class $\mathrm{E}^{\mathrm{NP}}$ that cannot be computed by circuits of size $2^{\delta n}$, for infinitely many input lengths $n$.*

Before we give the formal proof we briefly present the intuition. By the hypothesis, the algorithm $A$ generates, in polynomial-time with access to an NP oracle, a list of counterexamples $(\phi_1, ..., \phi_\ell)$ for some polynomial $\ell(n) := n^c > n^k$. It holds that every $n^k$-size circuit fails on at least one instance in the list. It is tempting to take the function $f(i) := \mathrm{SAT}(\phi_i)$ as our hard function. However, this does not quite work. Since we cannot assume any particular property regarding the order of the $\phi_i$'s, it is hypothetically possible that the location of a formula in the list determines its satisfiability (e.g., every even formula in the list is satisfiable and every odd is unsatisfiable). Furthermore, since $\ell > n^k$, a circuit of size $n^k$ cannot necessarily determine the index of a formula from the formula itself, thus it is possible that the list is indeed hard for circuits of size $n^k$ but $f$ itself is easy. Instead we show that if $f$ is easy for circuits of size $n^k$ then the hardness of the counterexamples stems from the fact that it is hard to generate their description (under some canonical representation of Boolean formulas). That is, we show that the function $h(i,j) = [\text{the } j\text{-th bit in the description of } \phi_i]$ is sufficiently hard for Boolean circuits.

More precisely, we do a case analysis. If an indexing function $g$: $g(\phi_i) = i$ is easy, then we can prove that $f(i) = \mathrm{SAT}(\phi_i)$ is hard since otherwise a small circuit can compute all the SAT-values of the counterexamples $\phi_1, ..., \phi_\ell$ using small circuits for $f$ and $g$. This contradicts the hardness of the counterexamples. Otherwise, we can prove that $h$ defined above is hard since, if $h$ is easy, a small circuit for $h$ can be used to compute $g$, which contradicts the hardness

of $g$.

Let us proceed with the proof.

PROOF. Fix a sufficiently large $n$ so that no $n^k$-size circuit solves $\mathrm{SAT}^n$. In this case, $A$ outputs a list of $\ell(n)$ counterexamples. Let $(\phi_1, ..., \phi_\ell)$ be the list sorted in lexicographical order so that $\phi_1 < \cdots < \phi_\ell$. Define $m := \lceil \log \ell(n) \rceil \leq \lceil c \log n \rceil$.

In the sequel we define several functions on different input lengths. A superscript denotes the input length of each function. Consider the following function $g^n : \{0,1\}^n \to \{0,1\}^m$ defined as $g^n(\phi) = i$ if $\phi = \phi_i$ for some $1 \leq i \leq \ell$, and $g^n(\phi) = 0$ otherwise. We consider two cases, whether (I) an $n^{k-1}$-size circuit can compute $g^n$ or (II) not.

*Case* (I): In this case, we prove that no circuit of size $n^{k-1} \geq 2^{\frac{k-1}{c}(m-1)}$ can compute the function $f^m(i) = \mathrm{SAT}(\phi_i)$. For contradiction, we assume that some $n^{k-1}$-size circuit $C_f$ can compute $f^m$. By the hypothesis of Case (I), we have an $n^{k-1}$-size circuit $C_g$ that computes $g^n$. Using $C_f$ and $C_g$, we can obtain an $n^k$-size circuit $C$ that computes the SAT-values of all the counterexamples $\{\phi_1, ..., \phi_\ell\}$, which contradicts the hardness of the counterexamples. The circuit $C$ is constructed as follows. Let $\phi$ be a given instance of $n$-bit length.

1. Run $C_g(\phi)$. If the output is 0, then output 0 and quit. Otherwise let $i \in [\ell]$ be the output of $C_g(\phi)$.
2. Output $C_f(i)$.

Obviously, the size of this circuit $C$ is at most $3n^{k-1} \leq n^k$ for a sufficiently large $n$ and it correctly computes $\mathrm{SAT}(\phi_i)$ for any $i$.

Moreover, the function $f$ can be computed in $\mathrm{poly}(n) = 2^{O(m)}$ time using an NP oracle as follows. Let $i \in \{0,1\}^m$ be an input.

1. Run $A$ and lexicographically sort the output formulas. The resulting list is $(\phi_1, ..., \phi_\ell)$.

2. Invoke the NP oracle to determine if $\phi_i \in \mathrm{SAT}$, and output the result.

Therefore, $f^m$ is hard against $2^{\frac{k-1}{c}(m-1)}$-size circuits and computable in $2^{O(m)}$ time using an NP oracle.

*Case* (II): In this case, we prove that no $n^{k-3}$-size circuit can compute yet another function $h$ defined as $h^{m'}(i,j) = [\text{the } j\text{-th bit in the description of } \phi_i \in \{0,1\}^n]$, where $m' := m + \lceil \log n \rceil = \lceil \log \ell(n) \rceil + \lceil \log n \rceil = \Theta(\log n)$. For contradiction, we assume that $h^{m'}$ can be computed by an $n^{k-3}$-size circuit $C_h$. Then, we can

compute $g^n$ by a small circuit $C'$ that uses $C_h$, contradicting the hardness of $g^n$, i.e., the hypothesis for Case (II).

The circuit $C'$ computes $g^n$ as follows. Let $\phi$ be a given SAT instance of $n$-bit length.

1. Perform a binary search on the list $(\phi_1, \ldots, \phi_\ell)$ to find the index $i$ such that $\phi = \phi_i$, if $\phi$ is in the list. Each comparison in the binary search, against the formula with index $i'$, is done by computing $\phi_{i'} = (C_h(i', 1), \ldots, C_h(i', n))$ and checking whether $\phi$ is lexicographically equal, larger or smaller than $\phi_{i'}$.
2. Output the obtained index $i$ if the binary search succeeds, otherwise output 0.

The binary search can be implemented by a circuit of size $O(|C_h| n \log \ell) = O(n^{k-2} \log n)$. Therefore, the size of $C'$ is at most $n^{k-1}$, which contradicts the hardness of $g^n$. Also, $h^{m'}$ is computable in $\mathrm{poly}(n) = 2^{O(m')}$ time using an NP oracle as follows. Let $(i, j) \in \{0,1\}^{m'}$ be a given instance.

1. Run $A$ and sort the output formulas. The resulting list is $(\phi_1, ..., \phi_\ell)$.
2. Output the $j$-th bit of $\phi_i$.

Therefore, $h^{m'}$ is hard against $n^{k-3} \geq 2^{\frac{k-3}{c+1}(m'-2)}$-size circuits and computable in $\mathrm{poly}(n) = 2^{O(m')}$ time using an NP oracle.

We showed that either $f^m$ or $h^{m'}$ has the required hardness for a fixed input length. By setting $\delta := \frac{k-3}{c+1}$, we get that either $f$ or $h$ is a hard function for circuits of size $2^{\delta r}$ for infinitely many $r \in \mathbb{N}$, while both functions are computable in deterministic time $2^{O(r)}$ with access to an NP oracle. □

**4.3. Learning Counterexamples with an Oracle to Promise** AM**.** In this section we show how to deterministically learn counterexamples for SAT with a prAM oracle. We first give an overview and then present the proof.

**4.3.1. Overview.** Our starting point is a result by Fortnow, Pavan, and Sengupta (2008), which essentially states that if SAT does not have size-$n^{k+3}$ circuits, then there is a short list of counterexamples for every size-$n^k$ circuit. More precisely, their result is that if SAT is hard for size-$n^{k+3}$ circuits, then there is a list of satisfiable formulae $L = (\phi_1, \ldots \phi_\ell)$, $\ell = \mathrm{poly}(n)$, such that every size-$n^k$ circuit $C$ fails on some $\phi$ in $L$, where by "fail" we mean that $C$ cannot be used to find a satisfying assignment for $\phi$. (Note that if $C$ were a circuit for SAT then it would not fail on $\phi$; in particular we could use $C$

to find the lexicographically least satisfying assignment for $\phi$ via downwards self-reducibility.)

In proving their result, Fortnow, Pavan, and Sengupta (2008) gave a probabilistic argument showing that given *any* collection of size-$n^k$ circuits, there is a satisfiable formula $\phi$ such that at least a third of the collection fail on it. This is our first ingredient in obtaining a deterministic oracle algorithm for finding counterexamples.

Our second ingredient is based on ideas of Chakaravarthy and Roy (2008), who show a deterministic polynomial time algorithm with oracle access to prAM for the task of finding a small circuit for SAT, assuming such a circuit exists. They do this in iterations, where the outcome of iteration $i$ is a circuit $C_i$ that fails on only a fraction of the formulas that previously found circuits $C_1, \ldots, C_{i-1}$ all fail on. Also obtained in iteration $i$ is an integer $\gamma_i$, which is an estimate for the number of formulas that the circuits $C_1, \ldots, C_i$ all fail on.

We adapt this algorithm to our setting, roughly speaking, by swapping circuits with formulas. That is, instead of trying to find a circuit $C_i$ that fails on as few of the remaining formulas as possible, in each iteration we try to find a formula $\phi_i$ that causes to fail as many of the remaining circuits as possible. Our first ingredient ensures that within poly$(n)$ iterations we end up with a list $L$ on which every circuit fails.

Once we obtain $L$, we turn it into a list of counterexamples by modestly growing the list. To be specific, for each $\phi \in L$ we add to $L$ all formulas $\phi'$ that would be queried if one was using a circuit for SAT to find the lexicographically least satisfying assignment for $\phi$.

The full proof in the next subsection is laid out as follows. We begin with the Goldwasser and Sipser (1989) result that shows that the approximate lower bound problem (mentioned in Section 1.2) is in prAM; we state this in Lemma 4.4 and later use it several times. We state the main result of this section, an algorithm for learning counterexamples, in Theorem 4.5, and prove it throughout the rest of the section. We present the above-mentioned first ingredient in Lemma 4.8; the second ingredient makes up the remaining material. Claim 4.11 together with Claim 4.12 describes the first task of each iteration, namely finding $\phi_i$ as described above. The second task of each iteration, namely finding $\gamma_i$, is explained in Claim 4.13.

**4.3.2. Proof.** We will need an Arthur-Merlin protocol for the approximate lower bound problem mentioned in Section 1.2. Such a protocol was given by Goldwasser and Sipser (1989). The formulation that we use is taken from

Bogdanov and Trevisan (2006).

LEMMA 4.4. *(Bogdanov and Trevisan (2006); Goldwasser and Sipser (1989)) Let $C$ denote the description of a Boolean circuit, $a$ an integer in binary, $\epsilon$ a number in the range $[0,1]$ given as $\epsilon^{-1}$ in unary. Define the following promise problem $\Pi$:*

- ***Yes instances:*** $(C,a,\epsilon) \in \Pi^Y$ *if* $|C^{-1}(1)| \geq a$.
- ***No instances:*** $(C,a,\epsilon) \in \Pi^N$ *if* $|C^{-1}(1)| \leq (1-\epsilon)a$.

$\Pi \in$ prAM.

PROOF (Sketch). The protocol works as follows. Arthur sends to Merlin a random hash function $h : \{0,1\}^m \to \{0,1\}^k$, where $m$ is the input length of $C$ and $k \approx \log a$. Merlin sends Arthur a string $x$ that satisfies $h(x) = 0 \cdots 0$ and $C(x) = 1$, and Arthur verifies that this is indeed the case. If $|C^{-1}(1)| \geq a$ then Merlin can find such an $x$ with high probability. On the other hand, if $|C^{-1}(1)|$ is much smaller than $a$, then there is no such $x$ with high probability. □

We now present the deterministic algorithm that learns counterexamples for SAT with a prAM oracle.

THEOREM 4.5. *Suppose that for some $k > 4$, SAT $\notin$ SIZE$(n^{k+3})$. There is a promise problem $\Gamma \in$ prAM and a polynomial-time deterministic oracle algorithm $A$, such that for every function $f : \{0,1\}^* \to \{0,1\}$ that agrees with $\Gamma$, for every input length $n$ for which SAT$^n$ does not have circuits of size $n^{k+3}$, $A^f$ learns $O(n^{k+1})$ counterexamples for SAT$^n$ with respect to the concept class* SIZE$(n^k)$.

PROOF. Let us first set up the following notation. For a Boolean circuit $C$ on $n$ inputs, define the circuit $S(C)$ that on an input formula $\phi$ of length $n$, attempts to find the lexicographic first satisfying assignment for $\phi$, via the downward self-reducibility property of SAT using $C$ to solve the SAT instances along the search path. If $S(C)$ finds a satisfying assignment it outputs the assignment and otherwise it outputs 0. From now on we fix $k$ to be an arbitrary integer greater than 4. For a list $L$ of satisfiable formulas, we consider the set of circuits that are consistent with $L$, in the following sense.

DEFINITION 4.6. *For a list of* satisfiable *formulas $L = (\phi_1, \ldots \phi_\ell)$ each of description length $n$, we denote by $T_L$ the set of $n^k$-size circuits that are consistent with $L$ in the following sense: $C \in T_L$ if and only if $S(C)$ finds a satisfying assignment for every $\phi_j \in L$.*

With this notation we can now describe the learning algorithm. Below we will define several promise problems in prAM and allow the algorithm oracle access to all of them (or to functions that agree with them to be more accurate). We can then combine them into a single promise problem in prAM as discussed in Section 2.

The algorithm has two stages. The first stage runs in iterations. Every iteration step $i$, passes to step $i+1$ a list $L_i$ of satisfiable formulas $\phi_1, \ldots, \phi_i$ each of description length $n$, as well as a number $1 \leq \gamma_i \leq 2^{3n^k \log n^k}$, where $\gamma_i$ is an estimate for $|T_{L_i}|$ such that

$$|T_{L_i}| \leq \gamma_i \leq \left(1 - \frac{1}{n^2}\right)^{-1} |T_{L_i}|. \tag{4.7}$$

Initially we set $L_0 := \emptyset$. This means that $T_{L_0}$ contains all the circuits of size $n^k$, and we therefore set $\gamma_0 := 2^{3n^k \log n^k}$.

The algorithm works in such a way that for every $i > 0$, $|T_{L_i}| \leq \frac{4}{5}|T_{L_{i-1}}|$. Thus for some $I \leq \lceil (\log_{5/4} 2) \cdot 3n^k \log n^k \rceil$, $T_{L_I} = \emptyset$ at which stage we will terminate the loop and with it the first stage of the algorithm.

Note that at the end of the first stage we already have a list of counterexamples, but those are only counterexamples for the search circuits $S(C)$, $C \in \mathrm{SIZE}(n^k)$. In fact, the list after the end of the first stage is very easy for decision circuits because it only contains satisfiable formulas. This is the reason that we need the second stage.

In the second stage, the algorithm uses its oracle as a SAT solver (clearly SAT $\in$ prAM) to generate for every $\phi_j \in L_I$ the list of formulas that are queried along the search path (via the downward self-reducibility property of SAT) for the lexicographic first satisfying assignment to $\phi_j$ (recall that every $\phi_j$ is satisfiable). We may assume w.l.o.g. that all the formulas thus generated are of description length $n$. The algorithm outputs all these formulas as the list of counterexamples. Note that this list contains $O(n^{k+1})$ formulas. By the fact that $T_{L_I} = \emptyset$ it follows that for every $C \in \mathrm{SIZE}(n^k)$, $S(C)$ fails to find a satisfying assignment for at least one $\phi_j \in L_I$. This means that $C$ errs on at least one query along the search path for a satisfying assignment for $\phi_j$, and this query appears in the list that the learning algorithm outputs. It therefore follows that the algorithm indeed outputs a list of counterexamples for the concept class $\mathrm{SIZE}(n^k)$.

It remains to describe iteration step $i > 0$, given a list $L_{i-1} = (\phi_1, \ldots, \phi_{i-1})$ and $\gamma_{i-1}$ as above. We need some more notation. For a list $L$ of satisfiable formulas all of the same length $n$ and one additional satisfiable formula $\rho$ of length $n$, it is clear that $T_{L\cup\{\rho\}} \subseteq T_L$. Define the set $G_{L,\rho}$ to be $T_L \setminus T_{L\cup\{\rho\}}$.

Fortnow, Pavan and Sengupta used a probabilistic argument (similar to the one in the result of Bshouty, Cleve, Gavaldà, Kannan, and Tamon (1996)) to prove the following lemma.

LEMMA 4.8. *(Fortnow, Pavan, and Sengupta (2008)) If $SAT^n$ cannot be computed by circuits of size $n^{k+3}$, then for every list $L$ of satisfiable formulas each of length $n$, there exists a satisfiable formula $\phi$ of length $n$ such $|T_{L\cup\{\phi\}}| \leq \frac{2}{3}|T_L|$ if $T_L \neq \emptyset$.*

For completeness, we provide the proof of this lemma.

PROOF. We choose $m := 36n$ circuits $C_1, ..., C_m$ from $T_L$ independently and uniformly at random. Let $C$ be a circuit taking majority vote of $C_1, ..., C_m$. The size of $S(C)$ is at most $n^{k+3}$. By the assumption, there is a formula $\phi_C$ on which $S(C)$ fails. Since $S(C)$ never fails on unsatisfiable formulas, $\phi_C$ is satisfiable.

We call a formula $\rho$ bad if $|T_{L\cup\{\rho\}}| > (2/3)|T_L|$. Fix a bad $\rho$. We then have $\Pr[|T_{L\cup\{\rho\}} \cap \{C_1, ..., C_m\}| \leq (1/2)m] < 2^{-2n}$ by the Chernoff bounds. By the union bound, $\Pr[\exists$bad $\rho$ such that $|T_{L\cup\{\rho\}} \cap \{C_1, ..., C_m\}| \leq (1/2)m] < 2^{-n}$ since the number of bad formulas is at most $2^n$.

Notice that more than half of $C_1, ..., C_m$ fails on $\phi_C$ for any $C_1, ..., C_m$ since $C$ fails on $\phi_C$. Thus, the probability that $\phi_C$ is not bad is nonzero (at least $1 - 2^{-n}$) from the above inequality. It follows that there exists $\phi$ such that $|T_{L\cup\{\phi\}}| \leq (2/3)|T_L|$. □

By this lemma there exists a $\phi$, such that

$$\begin{aligned} |G_{L_{i-1},\phi}| &\geq \frac{1}{3}|T_{L_{i-1}}| \\ &\geq \frac{1}{4}\left(1 - \frac{1}{n^2}\right)^{-1}|T_{L_{i-1}}| \\ &\geq \frac{1}{4}\gamma_{i-1}. \end{aligned} \tag{4.9}$$

(Recall that $|T_{L_{i-1}}| \leq \gamma_{i-1} \leq (1 - \frac{1}{n^2})^{-1}|T_{L_{i-1}}|$.)

We would like to find such a formula $\phi$ and then set $L_i := L_{i-1} \cup \{\phi\}$. We will not achieve quite that, but we will show how to find a $\phi$ such that

$$|G_{L_{i-1},\phi}| \geq \frac{1}{5}\gamma_{i-1} \geq \frac{1}{5}|T_{L_{i-1}}|. \tag{4.10}$$

Claim 4.11. *There is a promise problem $\Pi_1 \in$ prAM and a deterministic polynomial-time procedure, that when given as input the set $L_{i-1}$ and an estimate $\gamma_{i-1}$ for $|T_{L_{i-1}}|$ that satisfies Inequality (4.7), as well as oracle access to any function that agrees with $\Pi_1$, outputs a Boolean formula $\phi$ of length $n$ that satisfies Inequality (4.10).*

Proof. The instances of $\Pi_1$ are of the form $(1^m, (\rho_1, \ldots, \rho_\ell), p, a)$, where $m, \ell > 0$ are arbitrary integers, $\rho_j \in \{0,1\}^m$ for every $1 \leq j \leq \ell$, $p \in \{0,1\}^b$ for some integer $0 \leq b \leq m$, and $a$ is an integer between 0 and $2^{3m^k \log m^k}$ (in binary representation). We define $\Pi_1$ as follows:

- **Yes instances:** $(1^m, (\rho_1, \ldots, \rho_\ell), p, a) \in \Pi_1^Y$ if $\rho_1, \ldots, \rho_\ell$ are all satisfiable Boolean formulas and there exists an $s \in \{0,1\}^{m-b}$ such that $\rho = p \circ s$ is a satisfiable formula and $|G_{(\rho_1,\ldots,\rho_\ell),\rho}| \geq a$.
- **No instances:** $(1^m, (\rho_1, \ldots, \rho_\ell), p, a) \in \Pi_1^N$ if either at least one of $\rho_1, \ldots, \rho_\ell$ is not satisfiable, or for every $s \in \{0,1\}^{m-b}$, $\rho = p \circ s$ is not a satisfiable formula, or $\rho_1, \ldots, \rho_\ell$ are all satisfiable and for every $s \in \{0,1\}^{m-b}$ for which $\rho = p \circ s$ is a satisfiable formula, $|G_{(\rho_1,\ldots,\rho_\ell),\rho}| \leq (1 - \frac{1}{m^2})a$.

Claim 4.12. $\Pi_1 \in$ prAM.

Proof. The protocol is as follows. Merlin sends a string $s \in \{0,1\}^{m-b}$. Let $\rho = p \circ s$. Merlin also sends satisfying assignments for all of $\rho_1, \ldots, \rho_\ell, \rho$. If he fails to do so, Arthur rejects.

Define the circuit $C$ (which both Merlin and Arthur construct on their own) that on input a description of a circuit $B$ of size $m^k$, checks whether $S(B)$ finds a satisfying assignment to all of $\rho_1, \ldots, \rho_\ell$ but fails to find a satisfying assignment to $\rho$. If so it outputs 1 and otherwise 0. Note that $C$ computes the characteristic function of $G_{(\rho_1,\ldots,\rho_\ell),\rho}$. Arthur and Merlin run the lower bound protocol from Lemma 4.4 on input $(C, a, \frac{1}{m^2})$. Arthur accepts/rejects according to whether he accepts/rejects the lower bound protocol.

It is easy to verify that the protocol runs in time that is polynomial in its input length. We next argue about the completeness and soundness.

**Completeness:** If $\rho_1, \ldots, \rho_\ell$ are all satisfiable Boolean formulas and there exists an $s \in \{0,1\}^{m-b}$ such that $\rho = p \circ s$ is a satisfiable formula and $|G_{(\rho_1,\ldots,\rho_\ell),\rho}| \geq a$, then Merlin can find and send such an $s$ as well as satisfying assignments to $\rho_1, \ldots, \rho_\ell, \rho$, and then the completeness follows from the completeness of the lower bound protocol.

**Soundness:** If one of $\rho_1, \ldots, \rho_\ell$ is not satisfiable or there is no $s$ such that $p \circ s$ is satisfiable, then Arthur will reject after the first message of Merlin with probability 1. Otherwise for every $s$, for which $\rho = p \circ s$ is satisfiable, $|G_{(\rho_1,\ldots,\rho_\ell),\rho}| \leq (1 - \frac{1}{m^2})a$, and the soundness follows from the soundness of the lower bound protocol. □

We show how to find, with the help of $\Pi_1$, a formula $\phi$ that satisfies Inequality ((4.10)). We will do that iteratively where in each iteration we will set another bit of $\phi$. Let $\mu_0 := \lfloor \frac{1}{4}\gamma_{i-1} \rfloor$. Recall, by Lemma 4.8, that there exists a formula that satisfies Inequality (4.9). The most significant bit (MSB) of such a formula is either 0 or 1. In other words at least one of the following is true: $(1^n, L_{i-1}, 0, \mu_0) \in \Pi_1^Y$ and/or $(1^n, L_{i-1}, 1, \mu_0) \in \Pi_1^Y$. We query the $\Pi_1$ oracle on the input $(1^n, L_{i-1}, 0, \mu_0)$. If the answer is 1 we set the MSB of $\phi$ to 0, otherwise we set it to 1. Note that if we set the MSB to 1 then necessarily it is the MSB of a formula that satisfies Inequality (4.9). However, if we set it to 0, this is not necessarily the case. The reason is that the query $(1^n, L_{i-1}, 0, \mu_0)$ may fall outside the promise of $\Pi_1$. What we are assured of, though, is that if the $\Pi_1$ oracle answered 1 on $(1^n, L_{i-1}, 0, \mu_0)$ then it is not in $\Pi_1^N$. That is, there is a satisfiable formula $\phi$ whose MSB is 0 such that $|G_{L_{i-1},\phi}| \geq (1 - \frac{1}{n^2})\mu_0$. We set $\mu_1 = (1 - \frac{1}{n^2})\mu_0$ and continue. In the $j$'th iteration, suppose that we already fixed a prefix $p$ of length $j-1$ such that we know that there is a suffix that creates a satisfiable formula $\phi = p \circ s$ for which $|G_{L_{i-1},\phi}| \geq \mu_{j-1}$, then we query the $\Pi_1$ oracle on $(1^n, L_{i-1}, p \circ 0, \mu_{j-1})$ and set the next bit to 0 if the answer is 1 and otherwise we set it to 1. By the same argument as above we are guaranteed that the new prefix has a suffix such that the resulting formula $\phi$ satisfies $|G_{L_{i-1},\phi}| \geq (1 - \frac{1}{n^2})\mu_{j-1}$. We then set $\mu_j := (1 - \frac{1}{n^2})\mu_{j-1}$ and continue. After $n$ iterations we hold a formula $\phi$ of length $n$ such that

$$\begin{aligned} |G_{L_{i-1},\phi}| &\geq \left(1 - \frac{1}{n^2}\right)\mu_n \geq \left(1 - \frac{1}{n^2}\right)^n \mu_0 \\ &\geq \frac{1}{4}\left(1 - \frac{1}{n^2}\right)^n \gamma_{i-1} \geq \frac{1}{5}\gamma_{i-1} \\ &\geq \frac{1}{5}|T_{L_{i-1}}|. \end{aligned}$$

□

By Claim 4.11 we can find $\phi$ that satisfies Inequality (4.10). We then set $L_i = L_{i-1} \cup \{\phi\}$. By the definition of $|G_{L_{i-1},\phi}|$, we get that $|G_{L_{i-1},\phi}| = |T_{L_{i-1}} \setminus T_{L_i}| \geq \frac{1}{5}|T_{L_{i-1}}|$ which implies that $|T_{L_i}| \leq \frac{4}{5}|T_{L_{i-1}}|$ as required.

Next we show how to compute an estimate $\gamma_i$ for $|T_{L_i}|$. First we check whether $T_{L_i} = \emptyset$, in which case we terminate the main loop and move to the second stage of the algorithm. Note that whether $T_{L_i} = \emptyset$ is a coNP statement: for every $C$ of size $n^k$, $S(C)$ fails to find a satisfying assignment for at least one $\phi_j \in L_i$. Thus we can query the prAM oracle to check that. If $T_{L_i} \neq \emptyset$, the next claim shows that we can compute $\gamma_i$ as required (with oracle to prAM). This completes the description of the $i$'th iteration and hence the description of the algorithm.

Claim 4.13. *There is a promise problem $\Pi_2 \in$ prAM and a deterministic polynomial-time procedure that when given as input the set $L_i$, as well as oracle access to any function that agrees with $\Pi_2$ outputs a number $\gamma_i \in [2^{3n^k \log n^k}]$, such that $|T_{L_i}| \leq \gamma_i \leq (1 - \frac{1}{n^2})^{-1}|T_{L_i}|$.*

Proof. Let $\Pi_2$ be the following promise problem on instances $(1^m, (\rho_1, \ldots, \rho_\ell), a)$, where $m, \ell > 0$ are arbitrary integers, $\rho_1, \ldots, \rho_\ell$ are all of length $m$, and $a$ is an integer between 0 and $2^{3m^k \log m^k}$ (in binary representation):

- **Yes instances:** $(1^m, (\rho_1, \ldots, \rho_\ell), a) \in \Pi_2^Y$ if $\rho_1, \ldots, \rho_\ell$ are all satisfiable formulas and $|T_{(\rho_1,\ldots,\rho_\ell)}| \geq a$.
- **No instances:** $(1^m, (\rho_1, \ldots, \rho_\ell), a) \in \Pi_2^N$ if either at least one of $\rho_1, \ldots, \rho_\ell$ is not a satisfiable formula, or they are all satisfiable and $|T_{(\rho_1,\ldots,\rho_\ell)}| \leq (1 - \frac{1}{m^2})a$.

Notice that $\Pi_2$ is designed similarly to $\Pi_1$ in Claim 4.11, and hence it is easy to see that $\Pi_2 \in$ prAM by inspecting the proof of Claim 4.12 (the differences from $\Pi_1$ are that we do not check the existence of the satisfiable formula $\rho = p \circ s$ and we check the lower bound of $|T_{(\rho_1,\ldots,\rho_\ell)}|$ rather than $|G_{(\rho_1,\ldots,\rho_\ell),\rho}|$.)

By the definition of $\Pi_2$, we know that for every $a \leq |T_{L_i}|$ a $\Pi_2$ oracle answers 1 on the query $(1^n, L_i, a)$, and for every $a \geq (1 - \frac{1}{n^2})^{-1}|T_{L_i}|$ a $\Pi_2$ oracle answers 0 on the query $(1^n, L_i, a)$. For values of $a$ in between these two bounds we have no guarantee on the oracle's answers. The algorithm conducts a binary search on the set $[2^{3n^k \log n^k}]$ to find an $a$ such that the $\Pi_2$ oracle answers 0 on $(1^n, L_i, a)$ but 1 on $(1^n, L_i, a-1)$ (forcing the answer on $(1^n, L_i, 0)$ to be 1). Such a search takes $O(n^{k+1})$ time and we are guaranteed that for the $a$ that we find,

$$|T_{L_i}| \leq a \leq \left(1 - \frac{1}{n^2}\right)^{-1} |T_{L_i}|.$$

We then set $\gamma_i := a$. □
□

**4.4. Derandomization Implies Exponential-Size Lower Bounds.** We now combine the ingredients from the previous paragraphs to prove Theorem 4.1.

PROOF. (of Theorem 4.1) We condition on whether SAT $\in$ SIZE($n^{10}$) or not.
**Case 1: SAT $\in$ SIZE($n^{10}$).** By hypothesis, $\mathrm{P}^{\mathrm{prAM}} = \mathrm{P}^{\mathrm{P}^{\mathrm{NP}}} = \mathrm{P}^{\mathrm{NP}}$. For every $0 < \delta < 1$, Kannan (1982) showed that on an input $1^n$, we can compute in $\Sigma_3^{\mathrm{P}}$ the lexicographic first truth-table of length $n$ ($n = 2^m$) of a function (on $m$ inputs) whose circuit complexity is at least $n^\delta$. By Chakaravarthy and Roy (2008), if SAT $\in$ SIZE($n^{10}$), the polynomial-time hierarchy collapses to $\mathrm{P}^{\mathrm{prAM}}$, and hence to $\mathrm{P}^{\mathrm{NP}}$ by our hypothesis. In particular $\Sigma_3^{\mathrm{P}} \subseteq \mathrm{P}^{\mathrm{NP}}$ and we can compute the truth-table of the hard function in this class. By translation to the exponential level, this implies that there is a function in $\mathrm{E}^{\mathrm{NP}}$ that cannot be computed by circuits of size $2^{\delta n}$ (for all sufficiently large $n$).
**Case 2: SAT $\notin$ SIZE($n^{10}$).** Let $\Gamma$ be the promise problem in prAM from Theorem 4.5. By hypothesis, prAM $\subseteq \mathrm{P}^{\mathrm{NP}}$ so there is a function $f : \{0,1\}^* \to \{0,1\}$ in $\mathrm{P}^{\mathrm{NP}}$ that agrees with $\Gamma$. By Theorem 4.5 there is a polynomial-time deterministic oracle algorithm $A$, such that for every input length $n$ for which $\mathrm{SAT}^n$ does not have circuits of size $O(n^{10})$, $A^f$ learns a poly($n$)-long list of counterexamples for $\mathrm{SAT}^n$ with respect to the concept class SIZE($n^7$). The function that $A^f$ computes is in the class $\mathrm{P}^{\mathrm{P}^{\mathrm{NP}}} = \mathrm{P}^{\mathrm{NP}}$. This implies by Lemma 4.3, that there is a constant $\delta > 0$ and a Boolean function in the class $\mathrm{E}^{\mathrm{NP}}$ that cannot be computed by circuits of size $2^{\delta n}$ (for infinitely many input lengths $n$). □

## 5. Concluding Remarks

Our proofs show that there is a deterministic linear-exponential-time algorithm $A$ and a promise problem $\Pi$ in prAM, such that for every function $f$ that agrees with $\Pi$, $A^f$ computes a language that requires Boolean circuits of linear-exponential size. We point out that in both proofs the hard function that is constructed depends on the specific $f$ that is used as an oracle, i.e., on the values of $f$ outside of the promise. In the elementary proof this dependence arises in determining the minority vote of the circuits diagonalized against, whereas in the learning-based proof it arises in finding the counterexamples. This is the reason that we do not get a lower bound for a language in the class

$\mathrm{E}^{\mathrm{prAM}}$. Recall that a language is in this class if for *every* oracle that agrees with the promise, the algorithm computes the same function (i.e., the values of the function do not depend on values of the oracle outside the promise). Nevertheless, our results do imply the best *known* exponential-size lower bound, namely the one for the class $\mathrm{E}^{\Sigma_2^{\mathrm{P}}}$. This is because the $\Pi_2^{\mathrm{P}}$ simulation of our prAM oracle (Fürer, Goldreich, Mansour, Sipser, and Zachos (1989)) gives an explicit function in $\mathrm{E}^{\Sigma_2^{\mathrm{P}}}$ that requires circuits of size $\Omega(2^n/n)$. Proving an exponential-size lower bound for the class $\mathrm{E}^{\mathrm{prAM}}$ (and thus improving the best known lower bound) remains an open problem.

Another interesting open problem is to prove a true converse to Miltersen and Vinodchandran (2005); Shaltiel and Umans (2005); Umans (2003). Namely, show that a full derandomization of prAM implies lower bounds against exponential-size *nondeterministic* circuits. This would give a tight connection between hardness and derandomization in the high-end setting. Namely, exponential-size lower bounds (in this case for nondeterministic circuits) would be sufficient and necessary for derandomization (of prAM).

Finally, we would like to point out some similarities between our elementary proof and the proof of Goldreich (2010) that the assertion $\mathrm{prBPP} \subseteq \mathrm{P}$ implies certain canonical derandomizers. The latter proof has two main steps. First a certain search-to-decision reduction for prBPP is proven. Then it is argued that constructing a canonical derandomizer amounts to a certain diagonalization procedure against all fixed polynomial-time *algorithms*. This diagonalization turns out to be a prBPP-search problem. Now assuming $\mathrm{prBPP} \subseteq \mathrm{P}$ and using the search-to-decision reduction, this gives an efficient deterministic construction of a canonical derandomizer. Interestingly, the search-to-decision reduction fixes the solution to the search problem bit by bit where at each stage it uses the fact that approximate counting up to *additive* accuracy is in prBPP. In our proof we also search for an object (truth-table) by diagonalization, this time against all *circuits* of a certain size. We do that by fixing it bit by bit where at each stage we use the fact that approximate counting up to *multiplicative* accuracy is in prAM. We point out that Goldreich (2010) leaves as an open problem whether a result similar to his in the nondeterministic setting (i.e. regarding derandomizers for prAM) is true.

## Acknowledgements

We thank Scott Aaronson, Harry Buhrman and Dieter van Melkebeek for their collaboration on Aaronson, Aydınlıoğlu, Buhrman, Hitchcock, and van Melkebeek (2010), as well as for their contribution to this paper. We thank Amnon

Ta-Shma, Ronen Shaltiel and Osamu Watanabe for helpful comments and discussions, and the anonymous referees for their corrections and suggestions.

DG did most of the research while at Tel-Aviv University and supported by Oded Regev's European Research Council (ERC) Starting Grant. JH was partially supported by an NWO travel grant and by NSF grants 0652601 and 0917417. AK was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (B), 18300002 and the JST-CNRS Strategic International Cooperative Program.

A preliminary version of this paper appeared in Proceedings of the 25th Annual IEEE Conference on Computational Complexity, p.38–49, 2010.

## References

S. Aaronson, B. Aydınlıoğlu, H. Buhrman, J. Hitchcock and D. van Melkebeek, "A note on exponential circuit lower bounds from derandomizing Arthur-Merlin games," ECCC technical report. TR10-174, 2010.

S. Arora and B. Barak, "Complexity Theory: A Modern Approach," Cambridge University Press, 2009.

V. Arvind and P. Mukhopadhyay, "Derandomizing the isolation lemma and lower bounds for circuit size," in *Proceedings of the 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and the 12th International Workshop on Randomization and Computation*, ser. LNCS 5171, 2008, pp. 276–289.

A. Atserias, "Distinguishing SAT from polynomial-size circuits, through black-box queries," in *Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, pp. 88–95, 2006.

L. Babai, L. Fortnow, N. Nisan, and A. Wigderson, "BPP has subexponential simulation unless EXPTIME has publishable proofs," *Computational Complexity*, vol. 3, pp. 307–318, 1993.

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

M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM Journal on Computing*, vol. 13, no. 4, pp. 850-864, 1984.

A. Bogdanov and L. Trevisan, "On worst-case to average-case reductions for NP problems," *SIAM Journal on Computing*, vol. 36, no. 4, pp. 1119–1159, 2006.

N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon, "Oracles and queries that are sufficient for exact learning," *Journal of Computer and System Sciences*, vol. 52, no. 3, pp. 421–433, 1996.

H. Buhrman, L. Fortnow, and T. Thierauf, "Nonrelativizing separations," in *Proceedings of the 13rd Annual IEEE Conference on Computational Complexity*, 1998, pp. 8–12.

J.-Y. Cai, "$\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$," *Journal of Computer and System Sciences*, vol. 73, no. 1, pp. 25–35, 2007.

V. T. Chakaravarthy and S. Roy, "Finding irrefutable certificates for $S_2^P$ via Arthur and Merlin," in *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science*, 2008, pp. 157–168.

Z. Dvir, A. Shpilka, and A. Yehudanoff, "Hardness-randomness tradeoffs for bounded depth arithmetic circuits," *SIAM Journal on Computing*, vol. 39, no. 4, pp. 1279–1293, 2009.

S. Even, A. L. Selman, and Y. Yacobi, "The complexity of promise problems with applications to public-key cryptography," *Information and Control*, vol. 61, no. 2, pp. 159–173, 1984.

L. Fortnow, A. Pavan, and S. Sengupta, "Proving SAT does not have small circuits with an application to the two queries problem," *Journal of Computer and System Sciences*, vol. 74, no. 3, pp. 358–363, 2008.

M. Fürer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos, "On completeness and soundness in interactive proof systems," *Advances in Computing Research 5: Randomness and Computation*, pp. 429–442, 1989.

O. Goldreich, "On promise problems (a survey in memory of Shimon Even [1935-2004])," ECCC technical report. TR05-018, 2005.

O. Goldreich, "Computational Complexity: A Conceptual Perspective," Cambridge University Press, 2008.

O. Goldreich, "In a world of P=BPP," ECCC technical report. TR10-135, 2010.

S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

S. Goldwasser and M. Sipser, "Private coins versus public coins in interactive proof system," in *Advances in Computing Research, Vol. 5: Randomness and Computation*, S. Micali, Ed. JAI Press, 1989, pp. 73–90.

D. Gutfreund and A. Kawachi, "Derandomizing Arthur-Merlin Games and Approximate Counting Implies Exponential-Size Lower Bounds," in *Proceedings of the 25th Annual IEEE Conference on Computational Complexity*, 2010, pp. 38–49.

R. Impagliazzo, V. Kabanets, and A. Wigderson, "In search of an easy witness: exponential time vs. probabilistic polynomial time," *Journal of Computer and System Sciences*, vol. 65, no. 4, pp. 672–694, 2002.

R. Impagliazzo and A. Wigderson, "P=BPP if E requires exponential circuits: Derandomizing the XOR lemma," in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 220–229.

V. Kabanets and R. Impagliazzo, "Derandomizing polynomial identity tests means proving circuit lower bounds," *Computational Complexity*, vol. 13, no. 1-2, pp. 1–46, 2004.

R. Kannan, "Circuit-size lower bound and non-reducibility to sparse sets," *Information and Control*, vol. 55, no. 1–3, pp. 40–56, 1982.

R. M. Karp and R. J. Lipton, "Some connections between nonuniform and uniform complexity classes," in *Proceedings of the 12th Annual Symposium on Theoretical Computer Science*, 1980, pp. 302–309.

J. Kinne, D. van Melkebeek, and R. Shaltiel, "Pseudorandom generators and typically-correct derandomization," in *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and the 13th International Workshop on Randomization and Computation*, 2009, pp. 574–587.

A. R. Klivans and D. van Melkebeek, "Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses," *SIAM Journal on Computing,* vol. 31, no. 5, pp. 1501–1526, 2002.

P. B. Miltersen and N. V. Vinodchandran, "Derandomizing Arthur-Merlin games using hitting sets," *Computational Complexity*, vol. 14, no. 3, pp. 256–279, 2005.

P. B. Miltersen, N. V. Vinodchandran, and O. Watanabe, "Super-polynomial versus half-exponential circuit size in the exponential hierarchy," in *Proceedings of the 5th Annual International Conference on Computing and Combinatorics*, 1999, pp. 210–220.

N. Nisan, "Pseudorandom bits for constant depth circuits," *Combinatorica*, vol. 11, pp. 63–70, 1991.

N. Nisan and A. Wigderson, "Hardness vs. randomness," *Journal of Computer and System Sciences*, vol. 49, pp. 149–167, 1994.

R. Santhanam, "Circuit lower bounds for Merlin-Arthur classes," in *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, 2007, pp. 275–283.

R. Shaltiel and C. Umans, "Simple extractors for all min-entropies and a new pseudo-random generator," *Journal of ACM*, vol. 52, no. 2, pp. 172–216, 2005.

R. Shaltiel and C. Umans, "Pseudorandomness for approximate counting and sampling," *Computational Complexity*, vol. 15, no. 4, pp. 298–341, 2006.

C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell System Technical Journal*, vol. 28, no. 1, pp. 59–98, 1949.

C. Umans, "Pseudo-random generators for all hardnesses," *Journal of Computer and System Sciences*, vol. 67, no. 2, pp. 419–440, 2003.

E. Viola, "Pseudorandom bits for constant-depth circuits with few arbitrary symmetric gates," *SIAM Journal on Computing*, vol. 36, no. 5, pp. 1387–1403, 2007.

A. C. Yao, "Theory and application of trapdoor functions," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, 1982, pp. 80–91.

Manuscript received

Barış Aydınlıoğlu
University of Wisconsin-Madison
baris@cs.wisc.edu

Dan Gutfreund
IBM Research
Haifa, Israel
dannygu@il.ibm.com

John M. Hitchcock
University of Wyoming
jhitchco@cs.uwyo.edu

Akinori Kawachi
Dept. of Math. and Comput. Sci.
Tokyo Institute of Technology
Tokyo, Japan
kawachi@is.titech.ac.jp