In Search of an Easy Witness: Exponential Time vs. Probabilistic Polynomial Time

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

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

Avi Wigderson‡
Department of Computer Science
Hebrew University
Jerusalem, Israel 91904
and
Institute for Advanced Study
Princeton, NJ 08540
avi@ias.edu

May 22, 2002

Abstract

Restricting the search space $\{0,1\}^n$ to the set of truth tables of "easy" Boolean functions on $\log n$ variables, as well as using some known hardness-randomness tradeoffs, we establish a number of results relating the complexity of exponential-time and probabilistic polynomial-time complexity classes. In particular, we show that $\mathsf{NEXP} \subset \mathsf{P/poly} \Leftrightarrow \mathsf{NEXP} = \mathsf{MA}$; this can be interpreted as saying that *no* derandomization of MA (and, hence, of $\mathsf{promise\text{-}BPP}$) is possible *unless* NEXP contains a hard Boolean function. We also prove several downward closure results for ZPP, RP, BPP, and MA; e.g., we show $\mathsf{EXP} = \mathsf{BPP} \Leftrightarrow \mathsf{EE} = \mathsf{BPE}$, where EE is the double-exponential time class and BPE is the exponential-time analogue of BPP.

1 Introduction

One of the most important question in complexity theory is whether probabilistic algorithms are more powerful than their deterministic counterparts. A concrete formulation is the open question of whether $\mathsf{BPP} \stackrel{?}{=} \mathsf{P}$. Despite growing evidence that BPP can be derandomized (i.e., simulated deterministically) without a significant increase in the running time, so far it has not been ruled out that $\mathsf{NEXP} = \mathsf{BPP}$.

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

†This research was done while the author was at the Department of Computer Science, University of Toronto, and at the School of Mathematics, Institute for Advanced Study, Princeton. At IAS, the research was supported by NSF grant DMS 97-29992.

‡This research was supported by grant number 69/96 of the Israel Science Foundation, founded by the Israel Academy for Sciences and Humanities, and partially supported by NSF grants CCR-9987845 and CCR-9987077.

A number of conditional derandomization results are known which are based on the assumption that EXP contains hard Boolean functions, i.e., those of "high" circuit complexity [NW94, BFNW93, ACR98, IW97, STV99]. For instance, it is shown in [IW97] that BPP = P if $\mathsf{DTIME}(2^{O(n)})$ contains a language that requires Boolean circuits of size $2^{\Omega(n)}$. Results of this form, usually called *hardness-randomness tradeoffs*, are proved by showing that the truth table of a "hard" Boolean function can be used to construct a *pseudorandom generator*, which is then used to derandomize BPP or some other probabilistic complexity class. It is well known that such pseudorandom generators exist if and only if there exist hard Boolean functions in EXP. However, it is not known whether the existence of hard Boolean functions in EXP is actually *necessary* for derandomizing BPP. That is, it is not known if $\mathsf{BPP} \subseteq \mathsf{SUBEXP} \Rightarrow \mathsf{EXP} \not\subset \mathsf{P/poly}$.

Obtaining such an implication would yield a "normal form" for derandomization, because hardness-vs.-randomness results actually conclude that BPP can be derandomized in a very specific way. Think of a probabilistic algorithm, after fixing its input x, as defining a Boolean function $f_x(r)$ on the "random bits" r. Since the algorithm is fast, we know f_x is "easy", i.e., has low circuit complexity. For an algorithm accepting a language L in BPP, f_x is either almost always 1 (if $x \in L$) or almost always 0 (otherwise). To decide which, it suffices to approximate the fraction of r's with $f_x(r) = 1$ to within a constant additive error. To do this, the derandomization first computes all possible sequences that are outputs of a generator G, say $r_1, ...r_t$, and tries $f_x(r_i)$ for each i. (If G is a pseudo-random generator, the final output is the majority of the bits $f_x(r_i)$. Other constructions such as the hitting-set derandomization from [ACR98] are more complicated, but have the same general form.)

In particular,

1. We never use the acceptance probability guarantees for the algorithm on other inputs. Thus, we can derandomize algorithms even when acceptance seperations aren't guaranteed for all inputs, i.e., we can derandomize $Promise - \mathsf{BPP}$ ([For01, KRC00]). Intuitively, this means that randomized heuristics, that only perform well on some inputs, can also be simulated by a deterministic algorithm that performs well on the same inputs as the randomized algorithm.

2. The derandomization procedure only uses f_x as an oracle. Although its correctness relies on the *existence* of a small circuit computing f_x, the circuit itself is only used in a "black box" fashion.

Derandomization along the lines above is equivalent to proving circuit lower bounds, which seems difficult. One might hope to achieve derandomization unconditionally by relaxing the above restrictions. In particular, one could hope that it is easier to approximate the acceptance probability of a circuit using the circuit itself than it is treating it as a black box. In fact, recent results indicate that, in general, having access to the circuit computing a function is stronger than having the function as an oracle ([BGI$^+$01, Bar01]).

However, we show that this hope is ill-founded: for non-deterministic algorithms solving the approximation problem for circuit acceptance, oracle access is just as powerful as access to the circuit. In particular, any (even non-deterministic) derandomization of $Promise - \mathsf{BPP}$ yields a circuit lower bound for NEXP, and hence a "black-box" circuit approximation algorithm running in non-deterministic subexponential time. Thus, unconditional results in derandomization require either making a distinction between BPP and $Promise - \mathsf{BPP}$, or proving a circuit lower bound for NEXP.

More precisely, we show that $\mathsf{NEXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{MA} = \mathsf{NEXP}$, and hence no derandomization of MA is possible unless there are hard functions in NEXP. Since derandomizing promise-BPP also allows one to derandomize MA, the conclusion is that no full derandomization result is possible without assuming or proving circuit lower bounds for NEXP.

Another piece of evidence that it will be difficult to show EXP $\neq$ BPP (or NEXP $\neq$ MA) comes from the *downward closure* results for these classes. It is a basic fact in computational complexity that the equalities of complexity classes "translate upwards". For example, if NP = P, then NEXP = EXP by a simple padding argument. Thus, a separation at a "higher level" implies a separation at a "lower level", which suggests that "higher-level" separations are probably harder to prove. We show that separating EXP from BPP is *as hard as* separating their higher time-complexity analogues. More precisely, we show that EXP = BPP iff EE = BPE, where EE is the class of languages accepted in deterministic time $2^{2^{O(n)}}$ and BPE is the $2^{O(n)}$-time analogue of BPP. We prove similar downward closures for ZPP, RP, and MA. [1]

Main Techniques One of the main ideas that we use to derive our results can be informally described as the "easy witness" method, invented by Kabanets [Kab01]. It consists in searching for a desired object (e.g., a witness in a NEXP search problem) among those objects that have concise descriptions (e.g., truth tables of Boolean functions of low circuit complexity). Since there are few binary strings with small descriptions, such a search is more efficient than the exhaustive search. On the other hand, if our search fails, then we obtain a certain "hardness test", an efficient algorithm that accepts only those binary strings which do not have small descriptions. With such a hardness test, we can guess a truth table of a hard Boolean function, and then use it as a source of pseudorandomness via known hardness-randomness tradeoffs.

Recall that the problem Succinct-SAT is to decide whether a propositional formula is satisfiable when given a Boolean circuit which encodes the formula (e.g., the truth table of the Boolean function computed by the circuit is an encoding of the propositional formula); it is easy to see that Succinct-SAT is NEXP-complete. Thus, the idea of reducing the search space for NEXP problems to "easy" witnesses is suggested by the following natural question: Is it true that every satisfiable propositional formula that is described by a "small" Boolean circuit must have at least one satisfying assignment that can also be described by a "small" Boolean circuit? We will show that this is indeed the case if NEXP $\subset$ P/poly.

This idea was applied in [Kab01] to RP search problems in order to obtain certain "uniform-setting" derandomization of RP. In this paper, we consider NEXP search problems, which allows us to prove our results in the standard setting.

Remainder of the paper In Section 2, we present the necessary background. In Section 3, we describe our main technical tools. In particular, as an application of the "easy witness" method, we show that nontrivial derandomization of AM can be achieved under the uniform complexity assumption that NEXP $\neq$ EXP (cf. Theorem 18), where the class AM is a probabilistic version of NP (see the next section for the definitions).

In Section 4, we prove several results on complexity of NEXP. In particular, Section 4.1 contains the proof of the equivalence NEXP $\subset$ P/poly $\Leftrightarrow$ NEXP = MA. In Section 4.2, we show that every NEXP search problem can be solved in deterministic time $2^{\mathsf{poly}(n)}$, if NEXP = AM; we also prove that, if NEXP $\subset$ P/poly, then every language in NEXP has membership witnesses of polynomial circuit complexity.

Section 5 contains several interesting implications of our main result from Section 4.1 for the circuit approximation problem and natural proofs.

[1] Such closure results were also obtained by Fortnow and Miltersen [Fortnow, personal communication, July 2000], independently of our work.

In Section 6, we establish our downward closure results for ZPP, RP, BPP, and MA. We also prove "gap" theorems for ZPE, BPE, and MA; in particular, our gap theorem for ZPE states that either $\mathsf{ZPE} = \mathsf{EE}$, or ZPE can be simulated infinitely often in deterministic sub-double-exponential time.

Concluding remarks and open problems are given in Section 7.

2 Preliminaries

2.1 Complexity Classes

We assume that the reader is familiar with the standard complexity classes such as P, NP, ZPP, RP, and BPP (see, e.g., [Pap94]). We will need the two exponential-time deterministic complexity classes $\mathsf{E} = \mathsf{DTIME}(2^{O(n)})$ and $\mathsf{EXP} = \mathsf{DTIME}(2^{\mathsf{poly}(n)})$, and their nondeterministic analogues NE and NEXP. We define $\mathsf{SUBEXP} = \cap_{\epsilon>0}\mathsf{DTIME}(2^{n^\epsilon})$ and $\mathsf{NSUBEXP} = \cap_{\epsilon>0}\mathsf{NTIME}(2^{n^\epsilon})$. We will use the "exponential-time analogues" of the probabilistic complexity classes BPP, RP, and ZPP: $\mathsf{BPE} = \mathsf{BPTIME}(2^{O(n)})$, $\mathsf{RE} = \mathsf{RTIME}(2^{O(n)})$, and $\mathsf{ZPE} = \mathsf{ZPTIME}(2^{O(n)})$. We also define the double-exponential time complexity classes $\mathsf{EE} = \mathsf{DTIME}(2^{2^{O(n)}})$, $\mathsf{NEE} = \mathsf{NTIME}(2^{2^{O(n)}})$, and the classes $\mathsf{SUBEE} = \cap_{\epsilon>0}\mathsf{DTIME}(2^{2^{\epsilon n}})$ and $\mathsf{NSUBEE} = \cap_{\epsilon>0}\mathsf{NTIME}(2^{2^{\epsilon n}})$.

We shall also need the definitions of the classes MA and AM [Bab85, BM88]. The class MA can be viewed as a "nondeterministic version" of BPP, and is defined as follows. A language $L \subseteq \{0,1\}^*$ is in MA iff there exists a polynomial-time decidable predicate $R(x,y,z)$ and a constant $c \in \mathbb{N}$ such that, for every $x \in \{0,1\}^n$, we have

$$x \in L \Rightarrow \exists y \in \{0,1\}^{n^c} : \mathbf{Pr}_{z\in\{0,1\}^{n^c}}[R(x,y,z) = 1] \geqslant 2/3, \text{ and}$$
$$x \notin L \Rightarrow \forall y \in \{0,1\}^{n^c} : \mathbf{Pr}_{z\in\{0,1\}^{n^c}}[R(x,y,z) = 1] \leqslant 1/3.$$

The class AM, a "probabilistic version" of NP, consists of all binary languages L for which there is a polynomial-time decidable predicate $R(x,y,z)$ and a constant $c \in \mathbb{N}$ such that, for every $x \in \{0,1\}^n$, we have

$$x \in L \Rightarrow \mathbf{Pr}_{z\in\{0,1\}^{n^c}}[\exists y \in \{0,1\}^{n^c} : R(x,y,z) = 1] \geqslant 2/3, \text{ and}$$
$$x \notin L \Rightarrow \mathbf{Pr}_{z\in\{0,1\}^{n^c}}[\exists y \in \{0,1\}^{n^c} : R(x,y,z) = 1] \leqslant 1/3.$$

We shall also use the exponential-time version of MA, denoted as $\mathsf{MA\text{-}E}$, where the strings y and z from the definition of MA are of length 2^{cn}, rather than n^c.

For an arbitrary function $s : \mathbb{N} \to \mathbb{N}$, we define the nonuniform complexity class $\mathsf{SIZE}(s)$ to consist of all the families $f = \{f_n\}_{n\geqslant 0}$ of n-variable Boolean functions f_n such that, for all sufficiently large $n \in \mathbb{N}$, f_n can be computed by a Boolean circuit of size at most $s(n)$. Similarly, for any oracle A, we define the class $\mathsf{SIZE}^A(s)$ to contain the families of n-variable Boolean functions computable by *oracle* circuits of size at most $s(n)$ with A-oracle gates.

Let $\mathcal{C}$ be any complexity class over an alphabet Σ. We define the class $\mathcal{C}/\mathsf{poly}$ to consist of all languages L for which there is a language $M \in \mathcal{C}$ and a family of strings $\{y_n\}_{n\geqslant 0}$, where $y_n \in \Sigma^{\mathsf{poly}(n)}$, such that the following holds for all $x \in \Sigma^n$:

$$x \in L \Leftrightarrow (x, y_n) \in M.$$

More generally, for any function $t : \mathbb{N} \to \mathbb{N}$, we define the class $\mathcal{C}/t$ by requiring that $y_n \in \Sigma^{O(t(n))}$.

Finally, for an arbitrary complexity class $\mathcal{C}$ over an alphabet Σ, we define

$$\mathsf{io}\text{-}\mathcal{C} = \{L \subseteq \Sigma^* \mid \exists M \in \mathcal{C} \text{ such that } L \cap \Sigma^n = M \cap \Sigma^n \text{ infinitely often}\}.$$

2.2 Nondeterministic Generation of Hard Strings

As we shall see below, the truth table of a hard Boolean function can be used in order to approximate the acceptance probability of a Boolean circuit of appropriate size. Thus, an "efficient" algorithm for generating hard strings (the truth tables of hard Boolean functions) would yield an "efficient" derandomization procedure for probabilistic algorithms.

Usually, one talks about *deterministic* algorithms for generating hard strings. For example, the existence of such algorithms follows from the assumptions such as $\mathsf{EXP} \not\subset \mathsf{P/poly}$ or $\mathsf{E} \not\subset \mathsf{SIZE}(2^{o(n)})$. In some cases, however, we can afford to use *nondeterministic* algorithms for generating hard strings. We formalize this with the following definition.

We say that a Turing machine M *nondeterministically generates* the truth table of an n-variable Boolean function of circuit complexity at least $s(n)$, for some function $s : \mathbb{N} \to \mathbb{N}$, if on input 1^n

1. there is at least one accepting computation of M, and
2. whenever M enters an accepting state, the output tape of M contains the truth table of some n-variable Boolean function of circuit complexity at least $s(n)$.

The following lemma will be useful.

Lemma 1. *Suppose* $\mathsf{NEXP} \not\subset \mathsf{P/poly}$*. Then there is a* $\mathsf{poly}(2^n)$*-time Turing machine which, given an advice string of size* n*, nondeterministically generates* 2^n*-bit truth tables of* n*-variable Boolean functions* f_n *satisfying the following: for every* $d \in \mathbb{N}$ *and infinitely many* $n \in \mathbb{N}$*,* f_n *has circuit complexity greater than* n^d*.*

Proof. By a simple padding argument, we have that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ implies $\mathsf{NE} \not\subset \mathsf{P/poly}$. Let $L \in \mathsf{NE} \backslash \mathsf{P/poly}$ be any language. Suppose also that x_n is the binary encoding of the cardinality $c_n = |L \cap \{0,1\}^n|$; obviously, the length of x_n is at most $\log_2 2^n = n$. Then we can nondeterministically construct the truth table of the Boolean function deciding $L \cap \{0,1\}^n$ with the following algorithm B. Given x_n as advice, B nondeterministically guesses c_n strings $y_i \in L \cap \{0,1\}^n$ together with their certificates $z_i \in \{0,1\}^{2^{O(n)}}$. After B verifies the correctness of its guess, it outputs the 2^n-bit binary string t which has 1 in exactly those positions that correspond to the guessed y_i's, and 0 elsewhere. $\square$

As follows from the proof Lemma 1, the nondeterministic algorithm B, given appropriate advice, generates a *unique* truth table for every n. In general, however, we will allow our nondeterministic generating algorithm to output different hard strings on different accepting computation paths.

2.3 Hierarchy Theorems

We shall need several separation results that are provable by diagonalization.

Theorem 2. *For any fixed* $c \in \mathbb{N}$*,* $\mathsf{EXP} \not\subset \mathsf{io}\text{-}\mathsf{SIZE}(n^c)$*.*

Proof. By counting, we have that, for all sufficiently large $n \in \mathbb{N}$, there is an n-variable Boolean function of circuit complexity $2n^c > n^c$. The lexicographically first such function can be constructed in deterministic time $2^{n^{c+1}}$. $\square$

Theorem 3. *For any fixed* $c \in \mathbb{N}$*,* $\mathsf{EXP} \not\subseteq \mathsf{io}\text{-}[\mathsf{DTIME}(2^{n^c})/n^c]$*.*

Proof. For a given $n \in \mathbb{N}$, let S_n be the set of the truth tables of all n-variable Boolean functions computable by some deterministic 2^{n^c}-time Turing machine of description of size n that uses an advice string of size at most n^c; clearly, we have $|S_n| \leqslant 2^{2n^c}$. Define the truth table $t = t_1 \dots t_{2^n}$ of an n-variable Boolean function not in S_n as follows. The first bit t_1 has the value opposite to that of the first bit of the majority of strings in S_n. Let S_n^1 be the subset of S_n that contains the strings with the first bit equal to t_1; the size of S_n^1 is at most a half of the size of S_n. We define t_2 to have the value opposite to that of the second bit of the majority of strings in S_n^1; this leaves us with the subset S_n^2 of S_n^1 of half the size. After we have eliminated all the strings in S_n (which will happen after at most $2n^c + 1$ steps), we define the remaining bits of t to be 0.

It is easy to see that the string t can be constructed in deterministic time $2^{n^{c^3}}$. We define our language $L \in \mathsf{EXP}$ so that, for every $x \in \{0,1\}^n$, $x \in L$ iff the corresponding position in t is 1. By construction, $L \notin$ io-$[\mathsf{DTIME}(2^{n^c})/n^c]$. $\square$

Theorem 4. *For any fixed $c \in \mathbb{N}$, $\mathsf{EE} \not\subseteq$ io-$[\mathsf{DTIME}(2^{2^{cn}})/cn]$.*

Proof. Define a language as follows. On inputs of length n, we construct all truth tables of the first n Turing machines run for time $2^{2^{cn}}$ with all advice strings of length cn or smaller; there are at most $n2^{cn+1} \ll 2^{2^n}$ such truth tables. Then we enumerate all 2^{2^n} possible truth tables of n-variable Boolean functions, and use the first one that is not on our list. We output the value of our input in this table. $\square$

We shall need the following auxiliary lemmas whose proof relies on the existence of universal Turing machines.

Lemma 5. *If $\mathsf{NEXP} \subset \mathsf{P/poly}$, then there is a fixed constant $d_0 \in \mathbb{N}$ such that $\mathsf{NTIME}(2^n)/n \subset \mathsf{SIZE}(n^{d_0})$.*

Proof. Let $L \in \mathsf{NTIME}(2^n)/n$ be any binary language. Then there is a language $M \in \mathsf{NTIME}(2^n)$ and a sequence $\{y_n\}_{n\geqslant 0}$ of binary strings $y_n \in \{0,1\}^n$ such that, for every $x \in \{0,1\}^n$,

$$x \in L \Leftrightarrow (x, y_n) \in M.$$

Consider the following nondeterministic Turing machine U. On input (i, x) of size n, where $i \in \mathbb{N}$ and $x \in \{0,1\}^*$, the machine U runs in time 2^{2n}, simulating the ith nondeterministic Turing machine M_i on input x; the machine U accepts iff M_i accepts.

By assumption, there is some constant $k \in \mathbb{N}$ such that the language of U can be decided by Boolean circuits of size n^k almost everywhere. It follows that every language $M \in \mathsf{NTIME}(2^n)$ can be decided by Boolean circuits of size $(|i| + n)^k \in O(n^k)$, where i is the constant-size description of a nondeterministic 2^n-time Turing machine deciding M. Consequently, every language $L \in \mathsf{NTIME}(2^n)/n$ can be decided by Boolean circuits of size $O((2n)^k)$, which is in $O(n^k)$. The claim follows if we take $d_0 = k + 1$. $\square$

Lemma 6. *If $\mathsf{NEXP} = \mathsf{EXP}$, then there is a fixed constant $d_0 \in \mathbb{N}$ such that $\mathsf{NTIME}(2^n)/n \subseteq \mathsf{DTIME}(2^{n^{d_0}})/n$.*

Proof. For an arbitrary $L \in \mathsf{NTIME}(2^n)/n$, there is a nondeterministic 2^n-time Turing machine M and a sequence of n-bit advice strings a_n such that an n-bit string $x \in L$ iff $M(x, a_n)$ accepts.

Let U be the universal Turing machine for the class $\mathsf{NTIME}(2^n)$. By the assumption $\mathsf{NEXP} = \mathsf{EXP}$, we get that there is a constant $k \in \mathbb{N}$ such that the language of U is in $\mathsf{DTIME}(2^{n^k})$. The universality of U implies that the language of M is in $\mathsf{DTIME}(2^{n^{d_0}})$, for $d_0 = k + 1$. $\square$

Lemma 7. *If* $\mathsf{NEE} = \mathsf{EE}$*, then there is a fixed constant* $d_0 \in \mathbb{N}$ *such that* $\mathsf{NTIME}(2^{2^n})/n \subseteq \mathsf{DTIME}(2^{2^{d_0 n}})/n$.

Proof. The proof is virtually identical to that of Lemma 6. □

Combining the hierarchy theorems and the auxiliary lemmas above, we obtain the following.

Corollary 8. *If* $\mathsf{NEXP} \subset \mathsf{P/poly}$*, then* $\mathsf{EXP} \not\subseteq$ io-$[\mathsf{NTIME}(2^n)/n]$.

Proof. If $\mathsf{NEXP} \subset \mathsf{P/poly}$, then, by Lemma 5, there is a fixed $d_0 \in \mathbb{N}$ such that $\mathsf{NTIME}(2^n)/n \subset \mathsf{SIZE}(n^{d_0})$. The claim now follows by Theorem 2. □

Corollary 9. *If* $\mathsf{NEXP} = \mathsf{EXP}$*, then* $\mathsf{NEXP} \not\subseteq$ io-$[\mathsf{NTIME}(2^n)/n]$.

Proof. If $\mathsf{NEXP} = \mathsf{EXP}$, then, by Lemma 6, there is a fixed constant $d_0 \in \mathbb{N}$ such that $\mathsf{NTIME}(2^n)/n \subseteq \mathsf{DTIME}(2^{n^{d_0}})/n$. Applying Theorem 3 concludes the proof. □

Corollary 10. *If* $\mathsf{NEE} = \mathsf{EE}$*, then* $\mathsf{NEE} \not\subseteq$ io-$[\mathsf{NTIME}(2^{2^n})/n]$.

Proof. If $\mathsf{NEE} = \mathsf{EE}$, then, by Lemma 7, there is a fixed constant $d_0 \in \mathbb{N}$ such that $\mathsf{NTIME}(2^{2^n})/n \subseteq \mathsf{DTIME}(2^{2^{d_0 n}})/n$. By Theorem 4, the conclusion is immediate. □

2.4 Pseudorandom Generators and Conditional Derandomization

For more background on pseudorandom generators and derandomization, the reader is referred to the book by Goldreich [Gol99], as well as the surveys by Miltersen [Mil01] and Kabanets [Kab02].

A *generator* is a function $G : \{0,1\}^* \to \{0,1\}^*$ which maps $\{0,1\}^{l(n)}$ to $\{0,1\}^n$, for some function $l : \mathbb{N} \to \mathbb{N}$; we are interested only in the generators with $l(n) < n$.

For any oracle A, we say that a generator $G : \{0,1\}^{l(n)} \to \{0,1\}^n$ is $\mathsf{SIZE}^A(n)$*-pseudorandom* if, for any n-input Boolean circuit C of size [2] n with A-oracle gates, the following holds:

$$|\mathbf{Pr}_{x\in\{0,1\}^{l(n)}}[C(G(x)) = 1] - \mathbf{Pr}_{y\in\{0,1\}^n}[C(y) = 1]| \leqslant 1/n.$$

For the case of the empty oracle A, we will omit the mention of A and simply call the generator $\mathsf{SIZE}(n)$*-pseudorandom*.

Finally, we call a generator $G : \{0,1\}^{l(n)} \to \{0,1\}^n$ *quick* if its output can be computed in deterministic time $2^{O(l(n))}$.

Theorem 11 ([BFNW93, KM99]). *There is a polynomial-time computable function* $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *with the following properties. Let* A *be any oracle. For every* $\epsilon > 0$*, there exist* $\delta < \epsilon$ *and* $d \in \mathbb{N}$ *such that*

$$F : \{0,1\}^{2^{n^\delta}} \times \{0,1\}^{n^\epsilon} \to \{0,1\}^n,$$

and if r *is the truth table of an* n^δ*-variable Boolean function of* A*-oracle circuit complexity at least* $n^{\delta d}$*, then the function* $G_r(s) = F(r,s)$ *is a* $\mathsf{SIZE}^A(n)$*-pseudorandom generator mapping* $\{0,1\}^{n^\epsilon}$ *into* $\{0,1\}^n$.

[2] Such a circuit C may not use some of its n inputs.

As observed in [Yao82, NW94], a quick $\mathsf{SIZE}(n)$-pseudorandom generator $G : \{0,1\}^{n^\epsilon} \to \{0,1\}^n$ allows one to simulate every BPP algorithm in deterministic time $2^{n^{k\epsilon}}$, for some $k \in \mathbb{N}$. Goldreich and Zuckerman [GZ97] show that a quick $\mathsf{SIZE}(n)$-pseudorandom generator $G : \{0,1\}^{n^\epsilon} \to \{0,1\}^n$ allows one to decide every MA language in nondeterministic time $2^{n^{k\epsilon}}$, for some $k \in \mathbb{N}$. Thus, if we can "efficiently" generate the truth tables of Boolean functions of superpolynomial circuit complexity, then we can derandomize MA, by placing it in nondeterministic subexponential time. Note that, for the case of BPP, we need a deterministic algorithm for generating hard Boolean functions, but, for the case of MA, a nondeterministic algorithm suffices.

Theorem 11 readily implies the following.

Theorem 12. *1. Suppose that there is a $\mathsf{poly}(2^n)$-time Turing machine which, given an advice string of size $a(n)$ for some $a : \mathbb{N} \to \mathbb{N}$, nondeterministically generates 2^n-bit truth tables of n-variable Boolean functions f_n satisfying the following: for every $d \in \mathbb{N}$ and all sufficiently large $n \in \mathbb{N}$, f_n has circuit complexity greater than n^d. Then, for every $\epsilon > 0$, $\mathsf{MA} \subseteq \mathsf{NTIME}(2^{n^\epsilon})/a(n^\epsilon)$.*

2. If the Boolean functions f_n from Statement (1) above are such that, for every $d \in \mathbb{N}$ and infinitely many $n \in \mathbb{N}$, f_n has circuit complexity greater than n^d, then, for every $\epsilon > 0$, $\mathsf{MA} \subseteq \text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/a(n^\epsilon)]$.

Klivans and van Melkebeek [KM99] show that a quick $\mathsf{SIZE}^{\mathrm{SAT}}(n)$-pseudorandom generator $G : \{0,1\}^{n^\epsilon} \to \{0,1\}^n$ allows one to simulate every language in AM in nondeterministic time $2^{n^{k\epsilon}}$, for some $k \in \mathbb{N}$. Thus, if the truth tables of Boolean functions of superpolynomial SAT-oracle circuit complexity can be generated nondeterministically in time polynomial in their length, then $\mathsf{AM} \subseteq \mathsf{NSUBEXP}$ (see also [MV99] for derandomization of AM under weaker assumptions). More precisely, we have the following.

Theorem 13 (following [KM99]). *1. Suppose there is a $\mathsf{poly}(2^n)$-time algorithm which, given an advice string of length at most $a(n)$ for some $a : \mathbb{N} \to \mathbb{N}$, nondeterministically generates 2^n-bit truth tables of n-variable Boolean functions f_n satisfying the following: for every $d \in \mathbb{N}$ and all sufficiently large $n \in \mathbb{N}$, f_n has SAT-oracle circuit complexity greater than n^d. Then, for every $\epsilon > 0$, $\mathsf{AM} \subseteq \mathsf{NTIME}(2^{n^\epsilon})/a(n^\epsilon)$.*

2. If the functions f_n from Statement (1) above are such that, for every $d \in \mathbb{N}$ and infinitely many $n \in \mathbb{N}$, f_n has SAT-oracle circuit complexity greater than n^d, then, for every $\epsilon > 0$, $\mathsf{AM} \subseteq \text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/a(n^\epsilon)]$.

Stronger derandomization results hold for BPP, MA, and AM, under stronger complexity assumptions. In particular, Impagliazzo and Wigderson [IW97] show that a quick $\mathsf{SIZE}(n)$-pseudorandom generator $G : \{0,1\}^{O(\log n)} \to \{0,1\}^n$ can be constructed from a given truth table of a $O(\log n)$-variable Boolean function of circuit complexity at least $n^{\Omega(1)}$. Since this result relativizes (see [KM99]), we get the following.

Theorem 14 ([IW97, KM99]). *There is a polynomial-time computable function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ with the following properties. Let A be any oracle. For every $\epsilon > 0$, there exist $c, d \in \mathbb{N}$ such that*

$$F : \{0,1\}^{n^c} \times \{0,1\}^{d \log n} \to \{0,1\}^n,$$

and if r is the truth table of an $c \log n$-variable Boolean function of A-oracle circuit complexity at least $n^{\epsilon c}$, then the function $G_r(s) = F(r, s)$ is a $\mathsf{SIZE}^A(n)$-pseudorandom generator mapping $\{0,1\}^{d \log n}$ into $\{0,1\}^n$.

Note that if there is a deterministic $\mathsf{poly}(2^n)$-time algorithm that generates the truth tables of n-variable Boolean functions of circuit complexity at least $2^{\Omega(n)}$, then $\mathsf{BPP} = \mathsf{P}$; and if this algorithm is zero-error probabilistic, then $\mathsf{BPP} = \mathsf{ZPP}$.

We also have the following version of Theorem 13.

Theorem 15 ([KM99]). *1. Suppose there is a constant $\epsilon > 0$ and a $\mathsf{poly}(2^n)$-time algorithm which, given an advice string of length at most $a(n)$ for some $a : \mathbb{N} \to \mathbb{N}$, nondeterministically generates 2^n-bit truth tables of n-variable Boolean functions f_n satisfying the following: for all sufficiently large $n \in \mathbb{N}$, f_n has SAT-oracle circuit complexity at least $2^{\epsilon n}$. Then $\mathsf{AM} \subseteq \mathsf{NP}/a(O(\log n))$.*

2. If the functions f_n from Statement (1) above are such that, for infinitely many $n \in \mathbb{N}$, f_n has SAT-oracle circuit complexity at least $2^{\epsilon n}$, then we have $\mathsf{AM} \subseteq \text{io-}[\mathsf{NP}/a(O(\log n))]$.

3 Our Main Tools

3.1 Easy Witnesses and Hard Functions

In several applications below, we will need to decide whether a polynomial-time checkable relation $R(x, y)$ has a satisfying assignment (witness) $y \in \{0,1\}^*$ for a given input $x \in \{0,1\}^*$, where $|y| = l(|x|)$ for some function $l : \mathbb{N} \to \mathbb{N}$. That is, we need to compute the Boolean function $f_R(x)$ defined by

$$f_R(x) = 1 \text{ iff } \exists y \in \{0,1\}^{l(|x|)} : R(x, y) \text{ holds.}$$

To simplify the notation, we shall assume that $l(n) = 2^n$, i.e., that $f_R(x)$ is the characteristic function of a language in NE. Our approach will be to enumerate all possible truth tables $\hat{y}$ of Boolean functions on $n = |x|$ variables that are computable by A-oracle circuits of size $s(n)$, for some oracle $A \in \mathsf{EXP}$ and a function $s : \mathbb{N} \to \mathbb{N}$ (where $s(n) \geqslant n$) and check whether $R(x, \hat{y})$ holds for at least one of them.

Let $T_{A,s}(n)$ denote the set of truth tables of n-variable Boolean functions computable by A-oracle circuits of size $s(n)$. Then, instead of computing $f_R(x)$, we will be computing the following Boolean function $\hat{f}_{R,A,s}(x)$:

$$\hat{f}_{R,A,s}(x) = 1 \text{ iff } \exists y \in T_{A,s}(|x|) : R(x, y) \text{ holds.}$$

The following easy lemma shows that the set $T_{A,s}(n)$ can be efficiently enumerated.

Lemma 16. *For any fixed oracle $A \in \mathsf{EXP}$, there is a constant $c \in \mathbb{N}$ such that the set $T_{A,s}(n)$ can be enumerated in deterministic time $2^{s(n)^c}$, for any function $s : \mathbb{N} \to \mathbb{N}$.*

Proof. Let $A \in \mathsf{DTIME}(2^{n^d})$ for some $d \in \mathbb{N}$. Then the value of an A-oracle circuit on an n-bit input can be computed in deterministic time $\mathsf{poly}(s(n))2^{(s(n))^d}$, since the circuit of size $s(n)$ can query the oracle A on strings of size at most $s(n)$, and these oracle queries can be answered by running the deterministic 2^{n^d}-time Turing machine deciding A. Thus, the truth table of an n-variable Boolean function computed by such a circuit can be found in deterministic time $2^n \mathsf{poly}(s(n))2^{(s(n))^d}$, by evaluating the circuit on each n-bit input. Since the total number of A-oracle circuits of size s is at most $2^{O(s \log s)}$, the lemma follows. $\square$

It follows that the Boolean function $\hat{f}_{R,A,s}$ defined above is computable in deterministic time $2^{s(n)^d}$, for some $d \in \mathbb{N}$, which is less than the trivial upper bound $2^{O(n)}2^{2^n}$ (of a "brute-force" deterministic algorithm for $f_R(x)$) whenever $s(n) \in 2^{o(n)}$. For example, if $s(n) \in \mathsf{poly}(n)$, then the function $\hat{f}_{R,A,s}$ is computable in deterministic time $2^{\mathsf{poly}(n)}$, i.e., $\hat{f}_{R,A,s}$ is the characteristic function of a language in EXP. If $f_R = \hat{f}_{R,A,s}$, then we get a nontrivial deterministic algorithm for computing f_R. If $f_R \neq \hat{f}_{R,A,s}$, then we get a nondeterministic $\mathsf{poly}(2^n)$-time algorithm which, given a "short" advice string, generates the truth table of an n-variable Boolean function of "high" A-oracle circuit complexity. More precisely, the following is true.

Lemma 17. *Let $R(x,y)$ be any polynomial-time decidable relation defined on $\{0,1\}^n \times \{0,1\}^{2^n}$, let $A \in \mathsf{EXP}$ be any language, and let $s : \mathbb{N} \to \mathbb{N}$ be any function. Let $f_R(x)$ and $\hat{f}_{R,A,s}(x)$ be the Boolean functions defined above. If $f_R \neq \hat{f}_{R,A,s}$, then there is a nondeterministic $\mathsf{poly}(2^n)$-time algorithm B and a family $\{x_n\}_{n\geqslant 0}$ of n-bit strings with the following property: for infinitely many $n \in \mathbb{N}$, the algorithm B on advice x_{n+1} nondeterministically generates the truth table of an n-variable Boolean function of A-oracle circuit complexity greater than $s(n)$.*

Proof. If $f_R \neq \hat{f}_{R,A,s}$, then for infinitely many $n \in \mathbb{N}$ there exists a string $z_n \in \{0,1\}^n$ such that $f_R(z_n) = 1$ but $\hat{f}_{R,A,s}(z_n) = 0$. For those $n \in \mathbb{N}$ where such a string z_n exists, we define $x_{n+1} = 1z_n$ (i.e., the string z_n preceded with a 1); for the remaining $n \in \mathbb{N}$, we define $x_{n+1} = 0^{n+1}$.

It is easy to see that the following nondeterministic algorithm B is the required one: on input $1z \in \{0,1\}^{n+1}$, nondeterministically guess a $y \in \{0,1\}^{2^n}$, verify that $R(z,y)$ holds, output y, and halt in the accepting state; on input 0^{n+1}, output 0^{2^n}, and halt in the accepting state. □

Using the relationship between Boolean functions of high circuit complexity and pseudorandom generators that was described in Section 2.4, we obtain that if $f_R \neq \hat{f}_{R,A,s}$ for some $A \in \mathsf{EXP}$ and $s(n) \in n^{\Omega(1)}$, then certain derandomization of probabilistic algorithms is possible. For example, Lemma 17 yields the following derandomization result for AM, based on the assumption that $\mathsf{NEXP} \neq \mathsf{EXP}$.

Theorem 18. *If $\mathsf{NEXP} \neq \mathsf{EXP}$, then, for every $\epsilon > 0$, we have $\mathsf{AM} \subseteq$ io-$[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$.* [3]

Proof. It follows by a simple padding argument that if for every polynomial-time decidable relation $R(x,y)$ defined on $\{0,1\}^n \times \{0,1\}^{2^n}$ there is a $d \in \mathbb{N}$ such that $f_R = \hat{f}_{R,\mathrm{SAT},n^d}$, then $\mathsf{NEXP} \subseteq \mathsf{EXP}$. Hence, our assumption that $\mathsf{NEXP} \neq \mathsf{EXP}$ implies, by Lemma 17, that there is a $\mathsf{poly}(2^n)$-time algorithm which, given an advice string of length $a(n) = n+1$, nondeterministically generates the truth table of an n-variable Boolean function f_n such that, for every $d \in \mathbb{N}$, there are infinitely many n where f_n has SAT-oracle circuit complexity greater than n^d. The claim now follows by Theorem 13 (statement 2). □

Under a stronger assumption, we show that $\mathsf{AM} = \mathsf{NP}$. The same conclusion is known to hold under certain *nonuniform* hardness assumptions [KM99, MV99], and the assumption that NP is hard in a certain "uniform" setting [Lu00].

Theorem 19. *If $\mathsf{NE} \cap \mathsf{coNE} \not\subseteq$ io-$\mathsf{DTIME}(2^{2^{\epsilon n}})$ for some $\epsilon > 0$, then $\mathsf{AM} = \mathsf{NP}$.*

Proof. Consider all pairs (R_+, R_-) of polynomial-time decidable relations defined on $\{0,1\}^n \times \{0,1\}^{2^n}$ such that $f_{R_+}(x) = \neg f_{R_-}(x)$ for all $x \in \{0,1\}^n$. If, for every such pair (R_+, R_-) and every

[3] We should note that this is a very weak conditional derandomization result for AM, since it is known unconditionally that $\mathsf{AM} \subset \mathsf{NP/poly}$ and, obviously, $\mathsf{AM} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$.

$\epsilon > 0$, there are infinitely many n where $f_{R_+}(x) = \hat{f}_{R_+,\mathrm{SAT},2^{\epsilon n}}(x)$ for all $x \in \{0,1\}^n$, then we get by a simple padding argument that, for every $\epsilon > 0$, $\mathsf{NE} \cap \mathsf{coNE} \subseteq \text{io-}\mathsf{DTIME}(2^{2^{\epsilon n}})$. Thus, under the assumption of the theorem, there is a pair (R_+, R_-) of polynomial-time decidable relations defined on $\{0,1\}^n \times \{0,1\}^{2^n}$ such that, for some $\epsilon > 0$ and all sufficiently large n, we have $f_{R_+}(x) \neq \hat{f}_{R_+,\mathrm{SAT},2^{\epsilon n}}(x)$ for at least one $x \in \{0,1\}^n$. This implies that there is a $\mathsf{poly}(2^n)$-time algorithm B that nondeterministically generates 2^n-bit truth tables of n-variable Boolean functions f_n such that, for all sufficiently large n, f_n has SAT-oracle circuit complexity $2^{\Omega(n)}$.

Indeed, let $\{0,1\}^n = \{x_1, \dots, x_{2^n}\}$, let $y_1, \dots, y_{2^n} \in \{0,1\}^{2^n}$ be any strings such that $R_+(x_i, y_i) = 1$ or $R_-(x_i, y_i) = 1$ for all $1 \leqslant i \leqslant 2^n$, and let $Y = y_1 \dots y_{2^n}$ be the concatenation of all the y_i's. Note that such a Y can be found nondeterministically in time $2^{O(n)}$. It is clear that, for all sufficiently large n, such a string Y is the truth table of an $2n$-variable Boolean function of SAT-oracle circuit complexity greater than $2^{\epsilon n}$. Hence, the existence of the required algorithm B follows.

Applying Theorem 15 (statement 1) with $a(n) = 0$, we conclude that $\mathsf{AM} = \mathsf{NP}$. ☐

Essentially the same argument as in Theorem 19 (but using Theorem 13 (statement 2) instead of Theorem 15 (statement 1)), we also get the following.

Theorem 20. *If* $\mathsf{NEXP} \cap \mathsf{coNEXP} \neq \mathsf{EXP}$, *then* $\mathsf{AM} \subseteq \text{io-}\mathsf{NTIME}(2^{n^\epsilon})$, *for every* $\epsilon > 0$.

3.2 P-Sampleable Distributions and Padding

A family of probability distributions $\mu = \{\mu_n\}_{n \geqslant 0}$ is P-*sampleable* if there is a polynomial $p(n)$ and a polynomial-time Turing machine M such that the following holds: if $r \in \{0,1\}^{p(n)}$ is chosen uniformly at random, then the output of $M(n, r)$ is an n-bit string distributed according to μ_n.

For any language $L \subseteq \{0,1\}^*$, we define its *characteristic function* $\chi_L : \{0,1\}^* \to \{0,1\}$ so that $\chi_L(x) = 1$ iff $x \in L$.

Lemma 21. *Suppose that, for every language* $L \in \mathsf{BPP}$, *every* $\epsilon > 0$, *and every* P*-sampleable distribution family* $\mu = \{\mu_n\}_{n \geqslant 0}$, *there is a deterministic* 2^{n^ϵ}*-time algorithm* A *such that* $\mathbf{Pr}_{x \leftarrow \mu_n}[A(x) \neq \chi_L(x)] < 1/n$ *for infinitely many* $n \in \mathbb{N}$. *Then, for every* $\epsilon > 0$, $\mathsf{BPE} \subseteq \text{io-}[\mathsf{DTIME}(2^{2^{\epsilon n}})/n]$.

Proof. Let $\epsilon > 0$ be arbitrary. We define a padded version of any given language $L \in \mathsf{BPE}$ by $L_{\mathrm{pad}} = \{x 0^{2^{|x|} - |x| + i} \mid x \in L,\ 0 \leqslant i < 2^{|x|}\}$. Clearly, $L_{\mathrm{pad}} \in \mathsf{BPP}$.

Note that, for every $n \in \mathbb{N}$ and $0 \leqslant i < 2^n$, the number of "interesting" strings $y = x 0^{2^n - n + i}$, for some $x \in \{0,1\}^n$, is 2^n, which is at most their length $m = 2^n + i$. Hence, the uniform distribution μ_m on the set of such y's will assign each y the probability at least $1/m$. It is easy to see that this probability distribution is P-sampleable: for $m = 2^n + i$, where $0 \leqslant i < 2^n$, and $r \in \{0,1\}^n$, we define $M(m, r)$ to output $r 0^{m-n}$.

By the assumption, there is a 2^{m^ϵ}-time algorithm A such that, for infinitely many $m \in \mathbb{N}$, $\mathbf{Pr}_{y \leftarrow \mu_m}[A(y) \neq \chi_{L_{\mathrm{pad}}}(y)] < 1/m$. For infinitely many $m = 2^n + i$, where $0 \leqslant i < 2^n$, this algorithm A must be correct on *every* string $y = x 0^{2^n - n + i}$, since each such y has probability at least $1/m$ according to μ_m. Thus, there are infinitely many lengths $n \in \mathbb{N}$ such that, for some $0 \leqslant i < 2^n$, we have for every $x \in \{0,1\}^n$ that $A(x 0^{2^n - n + i}) = \chi_L(x)$. Using the n-bit encodings of such i's as advice, we obtain a deterministic algorithm with linear-length advice that runs in sub-double-exponential time and correctly decides L infinitely often. ☐

4 Complexity of NEXP

In this section, we prove several theorems relating uniform and nonuniform complexity of NEXP.

4.1 NEXP versus MA

Babai, Fortnow, and Lund [BFL91, Corollary 6.10], based on an observation by Nisan, improved a result of Albert Meyer [KL82] by showing the following.

Theorem 22 ([BFL91]). $\mathsf{EXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{EXP} = \mathsf{MA}$.

Here we will prove

Theorem 23. $\mathsf{NEXP} \subset \mathsf{P/poly} \Leftrightarrow \mathsf{NEXP} = \mathsf{MA}$.

Buhrman and Homer [BH92] proved that $\mathsf{EXP}^{\mathsf{NP}} \subset \mathsf{P/poly} \Rightarrow \mathsf{EXP}^{\mathsf{NP}} = \mathsf{EXP}$, but left open the question whether $\mathsf{NEXP} \subset \mathsf{P/poly} \Rightarrow \mathsf{NEXP} = \mathsf{EXP}$. Resolving this question is the main step in our proof of Theorem 23.

Theorem 24. *If* $\mathsf{NEXP} \subset \mathsf{P/poly}$*, then* $\mathsf{NEXP} = \mathsf{EXP}$.

Proof. Our proof is by contradiction. Suppose that

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

but

$$\mathsf{NEXP} \not\subseteq \mathsf{EXP}. \tag{2}$$

By Theorem 22, we get that assumption (1) implies that $\mathsf{EXP} = \mathsf{AM} = \mathsf{MA}$. By Theorem 18, we get from assumption (2) that, for every $\epsilon > 0$, $\mathsf{AM} \subseteq \text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$. Combining the two implications, we get that $\mathsf{EXP} \subseteq \text{io-}[\mathsf{NTIME}(2^n)/n]$. This and assumption (1) contradict Corollary 8. □

Corollary 25. *If* $\mathsf{NEXP} \subset \mathsf{P/poly}$*, then* $\mathsf{NEXP} = \mathsf{MA}$.

Proof. If $\mathsf{NEXP} \subset \mathsf{P/poly}$, then $\mathsf{NEXP} = \mathsf{EXP}$ by Theorem 24, and $\mathsf{EXP} = \mathsf{MA}$ by Theorem 22. □

Remark 26. Buhrman, Fortnow, and Thierauf [BFT98] show that $\mathsf{MA\text{-}E} \not\subset \mathsf{P/poly}$. Combined with a simple padding argument, their result yields the following implication: $\mathsf{MA} = \mathsf{NP} \Rightarrow \mathsf{NEXP} \not\subset \mathsf{P/poly}$. Our Corollary 25 is a significant strengthening of this implication.

The other direction of Theorem 23 was proved by Dieter van Melkebeek [van Melkebeek, personal communication, September 2000].

Theorem 27 (van Melkebeek). *If* $\mathsf{NEXP} = \mathsf{MA}$*, then* $\mathsf{NEXP} \subset \mathsf{P/poly}$.

Proof. Suppose that

$$\mathsf{NEXP} = \mathsf{MA}, \tag{3}$$

but

$$\mathsf{NEXP} \not\subset \mathsf{P/poly}. \tag{4}$$

The assumption (3) implies that $\mathsf{NEXP} = \mathsf{EXP}$, and so by (4) we get that $\mathsf{EXP} \not\subset \mathsf{P/poly}$. By Theorem 11, the latter yields that $\mathsf{MA} \subseteq \text{io-}\mathsf{NTIME}(2^n)$. Applying Corollary 9 concludes the proof. □

Proof of Theorem 23. The proof follows immediately from Corollary 25 and Theorem 27. □

4.2 Search versus Decision for NEXP

It is well known that if $\mathsf{NP} = \mathsf{P}$, then every NP search problem can be solved in deterministic polynomial time. Here, by an NP search problem, we mean the problem of finding, for a given input string x, a witness string y of length at most polynomial in the length of x such that $R(x, y)$ holds, where $R(x, y)$ is a polynomial-time decidable binary relation. Assuming that $\mathsf{NP} = \mathsf{P}$, we can find such a string y in polynomial time, fixing it "bit by bit". That is, we find y by asking a series of NP questions of the form: "Is there a y with a prefix y_0 such that $R(x, y)$?"

The same approach fails in the case of NEXP search problems. Suppose that $\mathsf{NEXP} = \mathsf{EXP}$. Let $R(x, y)$ be a predicate decidable in time $2^{\mathsf{poly}(|x|)}$, and the NEXP search problem is to find, given a string x, a witness string y of length at most $2^{\mathsf{poly}(|x|)}$ such that $R(x, y)$ holds. When we attempt to find a y satisfying $R(x, y)$ by encoding prefixes y_0 of y as part of the instance, we eventually get an instance whose size is *exponential* in $|x|$, the size of the original instance. Being able to solve such an instance in deterministic exponential time would only give us a double-exponential time algorithm for solving the original search problem, which is not better than solving it by "brute force".

Thus, apparently, the assumption $\mathsf{NEXP} = \mathsf{EXP}$ does not suffice to conclude that every NEXP search problem is solvable in deterministic time $2^{\mathsf{poly}(n)}$. The following theorem of Impagliazzo and Tardos [IT89] gives some evidence to this effect.

Theorem 28 ([IT89]). *There is an oracle relative to which* $\mathsf{NEXP} = \mathsf{EXP}$*, and yet there is a* NEXP *search problem that cannot be solved deterministically in less than double exponential time.*

Under the stronger assumption that $\mathsf{NEXP} = \mathsf{AM}$, we obtain the desired conclusion for NEXP search problems.

Theorem 29. *If* $\mathsf{NEXP} = \mathsf{AM}$*, then every* NEXP *search problem can be solved in deterministic time* $2^{\mathsf{poly}(n)}$.

The proof will follow from the next theorem.

Theorem 30. *If* $\mathsf{NEXP} = \mathsf{AM}$*, then for every language* $L \in \mathsf{NEXP}$ *there is a constant* d *such that every sufficiently large* n*-bit string* $x \in L$ *has at least one witness* $y \in \{0,1\}^{2^{\mathsf{poly}(n)}}$ *that can be described by a SAT-oracle circuit of size at most* n^d.

Proof. The proof is by contradiction. It is easy to see by a simple padding argument that if, for every polynomial-time decidable relation $R(x, y)$ defined on $\{0,1\}^n \times \{0,1\}^{2^n}$, there is a $d \in \mathbb{N}$ such that $f_R = \hat{f}_{R,\mathrm{SAT},n^d}$, then the conclusion of the Theorem is true. So, let us suppose that there is a polynomial-time decidable relation $R(x, y)$ on $\{0,1\}^n \times \{0,1\}^{2^n}$ such that, for every $d \in \mathbb{N}$, we have $f_R \neq \hat{f}_{R,\mathrm{SAT},n^d}$.

Applying Lemma 17 and Theorem 13, we obtain that, for every $\epsilon > 0$, $\mathsf{AM} \subseteq \mathsf{io}\text{-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$. Together with our assumption that $\mathsf{NEXP} = \mathsf{EXP} = \mathsf{AM}$, this contradicts Corollary 9. $\square$

Proof of Theorem 29. By Theorem 30, witnesses for any language in NEXP can be found in deterministic exponential time by enumerating all SAT-oracle circuits of some fixed polynomial size and checking whether any of these circuits encodes a witness. $\square$

We conclude this section by showing that, if $\mathsf{NEXP} \subset \mathsf{P/poly}$, then every language in NEXP has membership witnesses of polynomial circuit complexity.

Theorem 31. *If* $\mathsf{NEXP} \subset \mathsf{P/poly}$*, then for every language* $L \in \mathsf{EXP}$ *there is a constant* $d \in \mathbb{N}$ *such that every sufficiently large* n*-bit string* $x \in L$ *has at least one witness that can be described by a Boolean circuit of size at most* n^d*.*

Proof. The assumption $\mathsf{NEXP} \subset \mathsf{P/poly}$ implies by Theorem 23 that $\mathsf{NEXP} = \mathsf{MA}$. For the sake of contradiction, suppose that the conclusion of our theorem does not hold. Then, similarly to the proof of Theorem 30 above, we conclude that there is a polynomial-time decidable relation $R(x, y)$ on $\{0,1\}^n \times \{0,1\}^{2^n}$ such that, for every $d \in \mathbb{N}$, we have $f_R \neq \hat{f}_{R,\varnothing,n^d}$.

Applying Lemma 17 and Theorem 12, we obtain that, for every $\epsilon > 0$, $\mathsf{MA} \subseteq$ io-$[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$. Combined with our assumption that $\mathsf{NEXP} = \mathsf{EXP} = \mathsf{MA}$, this contradicts Corollary 9. ☐

5 Implications for Circuit Approximation and Natural Properties

In this section, we present two implications of our Theorem 23 for the problem of circuit approximation and natural properties of Razborov and Rudich [RR97]. In Section 5.1, we show that (for nondeterministic Turing machines with sublinear amount of advice) if the problem of circuit approximation can be solved efficiently at all, then it can also be solved efficiently with only *oracle access* to the Boolean circuit to be approximated. In Section 5.2, we show that the mere existence of an NP-natural property useful against P/poly already implies the existence of a hard Boolean function in NEXP.

5.1 Circuit Approximation

Recall that the *Circuit Acceptance Probability Problem (CAPP)* is the problem of computing the fraction of inputs accepted by a given Boolean circuit. This problem is easily solvable in probabilistic polynomial time, and, in a certain sense, is "complete" for promise-BPP (see, e.g., [KRC00, For01]).

We say that CAPP can be *nontrivially approximated* if, for every $\epsilon > 0$, there is a nondeterministic 2^{n^ϵ}-time algorithm which, using advice of size n^ϵ, approximates the acceptance probability of any given Boolean circuit of size n, to within an additive error $1/6$, for infinitely many input sizes n. Here, when we say that a *nondeterministic* algorithm M approximates a real-valued function $g(x)$ to within $1/6$, for all $x \in \{0,1\}^n$, we mean that

1. for every $x \in \{0,1\}^n$, there is an accepting computation of M on x, and
2. every accepting computation of M on x outputs a rational number $q \in [g(x) - 1/6, g(x) + 1/6]$.

We say that an algorithm M for approximating CAPP is *"black-box"* if M is given only oracle access to an input Boolean function f (computable by a circuit of size n). That is, M is allowed to query the value of f on any binary string α, but M is *not* allowed to view the actual syntactic representation of any circuit computing f.

Finally, we say that a "black-box" algorithm M for approximating CAPP is *non-adaptive* if the queries asked by M on a given input Boolean function f depend *only* on n, and all of these queries are computed *before* obtaining the value of f on any one of them.

Theorem 32. *The following assumptions are equivalent.*

1. $\mathsf{NEXP} \not\subset \mathsf{P/poly}$.
2. *CAPP can be nontrivially approximated.*

3. CAPP can be nontrivially approximated by a "black-box" non-adaptive algorithm.

Proof Sketch. (3) $\Rightarrow$ (2). Trivial.

(2) $\Rightarrow$ (1). It is not difficult to see that if CAPP can be nontrivially approximated, then, for every $\epsilon > 0$,

$$\mathsf{MA} \subseteq \text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]. \tag{5}$$

This implies that $\mathsf{NEXP} \neq \mathsf{MA}$, since otherwise we would contradict Corollary 9. Hence, by Theorem 23, we conclude that $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

(1) $\Rightarrow$ (3). This follows immediately from Lemma 1 and Theorem 11. $\square$

Remark 33. The big open question is whether an analogue of Theorem 32 can be proved where all "nondeterministic" assumptions are replaced by the corresponding "deterministic" assumptions. In particular, we want to know if the existence of a *deterministic* efficient algorithm for approximating CAPP is equivalent to the existence of a *deterministic* efficient algorithm for the same problem with the *additional* property of being "black-box" and non-adaptive.

Note that the existence of a deterministic polynomial-time algorithm that approximates the acceptance probability of a given Boolean circuit to within an additive error 1/6 is equivalent to the statement that $\mathsf{promise\text{-}BPP} \subseteq \mathsf{promise\text{-}P}$, which means the following: for every probabilistic polynomial-time algorithm M, there is a deterministic polynomial-time algorithm A such that A accepts every element in the set

$$\{x \in \{0,1\}^* : \mathbf{Pr}[M(x) \text{ accepts}] > 2/3\}$$

and A rejects every element in the set

$$\{x \in \{0,1\}^* : \mathbf{Pr}[M(x) \text{ accepts}] < 1/3\}.$$

The statement $\mathsf{promise\text{-}BPP} \subseteq \mathsf{promise\text{-}SUBEXP}$ is interpreted similarly, with the deterministic algorithm A running in subexponential time.

As an immediate consequence of Theorem 32, we obtain the following.

Corollary 34. $\mathsf{promise\text{-}BPP} \subseteq \mathsf{promise\text{-}SUBEXP} \Rightarrow \mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

Obviously, if $\mathsf{promise\text{-}BPP} \subseteq \mathsf{promise\text{-}P}$, then $\mathsf{BPP} = \mathsf{P}$. However, the converse is *not* known to hold. If the converse were to hold, then Theorem 32 would yield that $\mathsf{BPP} = \mathsf{P} \Rightarrow \mathsf{NEXP} \not\subseteq \mathsf{P/poly}$, and hence, derandomizing BPP would be as hard as proving circuit lower bounds for NEXP.

5.2 Natural Properties

Razborov and Rudich [RR97] argue that all known proofs of circuit lower bounds for nonmonotone Boolean functions consist of two parts. First, one defines a certain "natural" property of Boolean functions (or such a property is implicit in the proof) so that any family of Boolean functions that satisfies this property must require "large" circuits. Then one shows that a particular explicit family of Boolean functions satisfies this "natural" property.

We consider the scenario where one has made the first step (defined an appropriate property of Boolean functions), but cannot (does not know how to) prove that some explicit Boolean function satisfies this property. Does the existence of such a property alone yield any circuit lower bounds for explicit Boolean functions? We will argue that the answer is *yes*, if one considers a NEXP-complete function explicit.[4]

Recall that a family $\mathcal{F} = \{\mathcal{F}_n\}_{n>0}$ of nonempty subsets $\mathcal{F}_n$ of n-variable Boolean functions is called P-*natural* if it satisfies the following conditions:

1. **constructiveness** the language T consisting of the truth tables of Boolean functions in $\mathcal{F}$ is in P, and

2. **largeness** there is a $c \in \mathbb{N}$ such that, for every $N = 2^n$, we have $|T_N| \geqslant 2^N/N^c$, where $T_N = T \cap \{0,1\}^N$.

By replacing P with NP in the constructiveness condition above, we obtain an NP-*natural* property.

Finally, a property $\mathcal{F}$ is called *useful against* P/poly if, for every family of Boolean functions $f = \{f_n\}_{n>0}$, the following holds: if $f_n \in \mathcal{F}_n$ for infinitely many n, then $f \notin$ P/poly.

Theorem 35. *If there exists an* NP*-natural property (even without the largeness condition) that is useful against* P/poly*, then* NEXP $\not\subset$ P/poly.

Proof Sketch. The existence of an NP-natural property allows us to guess and certify Boolean functions of superpolynomial circuit complexity, nondeterministically in time polynomial in the size of their truth tables; note that this does not require the largeness condition. By Theorem 12, these hard Boolean functions can then be used to derandomize MA, yielding NEXP $\neq$ MA. Now the claim follows by Theorem 23. ☐

Remark 36. Note the following subtlety in our proof of Theorem 35. Although we conclude that NEXP $\not\subset$ P/poly, we do *not* prove that any Boolean function in NEXP actually satisfies the given natural property.

Remark 37. Here the interesting open problem is to try to prove a "deterministic" version of Theorem 35. That is, does the existence of a P-natural property useful against P/poly imply that EXP $\not\subset$ P/poly?

6 Downward Closures and Gap Theorems

The results showing that a collapse of higher complexity classes implies a collapse of lower complexity classes are known as *downward closure* results. Very few such results are known. For example, Impagliazzo and Naor [IN88] prove that P = NP $\Rightarrow$ DTIME(polylog(n)) = NTIME(polylog(n)) $\cap$ coNTIME(polylog(n)) = RTIME(polylog(n)); see also [BFNW93] and [HIS85]. We prove several downward closure results for probabilistic complexity classes. Along the way, we also obtain "gap" theorems for the complexity of BPE, ZPE, and MA.

Note: Fortnow [For01] gives much simpler proofs of the downward closures presented in this section. However, our techniques also allow us to establish the gap theorems that do not seem to follow from [For01].

6.1 Case of BPP

Here we establish the following

Theorem 38. EXP = BPP $\Leftrightarrow$ EE = BPE.

[4] Usually, by an *explicit* Boolean function, one means a function in NP.

Our proof will rely on the following result by Impagliazzo and Wigderson [IW98] on the derandomization of BPP under a uniform hardness assumption.

Theorem 39 ([IW98]). *Suppose that* $\mathsf{EXP} \neq \mathsf{BPP}$*. Then, for every binary language* $L \in \mathsf{BPP}$ *and every* $\epsilon > 0$*, there is a deterministic* 2^{n^ϵ}*-time algorithm* A *satisfying the following condition: for every* P*-sampleable distribution family* $\mu = \{\mu_n\}_{n \geqslant 0}$*, there are infinitely many* $n \in \mathbb{N}$ *such that* $\mathbf{Pr}_{x \leftarrow \mu_n}[A(x) \neq \chi_L(x)] < 1/n$*.*

This allows to prove the following.

Theorem 40. *If* $\mathsf{EXP} \neq \mathsf{BPP}$*, then, for every* $\epsilon > 0$*, we have* $\mathsf{BPE} \subseteq \text{io-}[\mathsf{DTIME}(2^{2^{\epsilon n}})/n]$*.*

Proof. If $\mathsf{EXP} \neq \mathsf{BPP}$, then, by Theorem 39, the assumption of Lemma 21 is satisfied, and hence, our claim follows. ☐

Proof of Theorem 38. $\Rightarrow$. If $\mathsf{EXP} = \mathsf{BPP}$, then by padding, we conclude $\mathsf{EE} = \mathsf{BPE}$.

$\Leftarrow$. Assume $\mathsf{BPE} = \mathsf{EE}$, but $\mathsf{BPP} \neq \mathsf{EXP}$. By Theorem 40, $\mathsf{BPE} \subseteq \text{io-}[\mathsf{DTIME}(2^{2^n})/n]$. But then so is EE, contradicting Theorem 4. ☐

As a corollary to Theorem 40, we obtain the following.

Theorem 41 (Gap Theorem for BPE). *Exactly one of the following holds:*

1. $\mathsf{BPE} = \mathsf{EE}$*, or*

2. *for every* $\epsilon > 0$*,* $\mathsf{BPE} \subseteq \text{io-}[\mathsf{DTIME}(2^{2^{\epsilon n}})/n]$*.*

Proof. First, by Theorem 4, statements (1) and (2) cannot both hold at the same time. Now, if statement (1) does not hold, then, by padding, we get $\mathsf{EXP} \neq \mathsf{BPP}$, which implies statement (2) via Theorem 40. ☐

6.2 Cases of ZPP and RP

In this section, we prove the following results.

Theorem 42. $\mathsf{EXP} = \mathsf{ZPP} \Leftrightarrow \mathsf{EE} = \mathsf{ZPE}$*.*

Theorem 43. $\mathsf{EXP} = \mathsf{RP} \Leftrightarrow \mathsf{EE} = \mathsf{RE}$*.*

The proof of Theorem 42 will rely on the following result implicit in [IW98].

Theorem 44 ([IW98]). *Suppose that* $\mathsf{EXP} \neq \mathsf{BPP}$*. Then, for every binary language* $L \in \mathsf{ZPP}$ *and every* $\epsilon > 0$*, there is a deterministic* 2^{n^ϵ}*-time algorithm* A *satisfying the following conditions:*

1. *for every* $x \in \{0,1\}^*$*, we have* $A(x) \in \{\chi_L(x), ?\}$*, where* $\chi_L(x)$ *is* 1 *if* $x \in L$*, and is* 0 *if* $x \notin L$*, (i.e.,* $A(x)$ *either outputs the correct answer, or says "don't know"), and*

2. *for every* P*-sampleable distribution family* $\mu = \{\mu_n\}_{n \geqslant 0}$*, there are infinitely many* $n \in \mathbb{N}$ *such that* $\mathbf{Pr}_{x \leftarrow \mu_n}[A(x) = ?] < 1/n$*.*

As a corollary, we can prove

Theorem 45. *If* $\mathsf{EXP} \neq \mathsf{BPP}$*, then, for every* $\epsilon > 0$*, we have* $\mathsf{ZPE} \subseteq \text{io-}\mathsf{DTIME}(2^{2^{\epsilon n}})$*.*

Proof. If $\mathsf{EXP} \neq \mathsf{BPP}$, then the conclusion of Theorem 44 holds. Proceeding exactly as in the proof of Lemma 21, we obtain that, for every language $L \in \mathsf{BPP}$ and every $\epsilon > 0$, there is a deterministic $2^{2^{\epsilon n}}$-time algorithm A satisfying the following: there are infinitely many $n \in \mathbb{N}$ such that, for some $0 \leqslant i < 2^n$, we have $A(x0^{2^n-n+i}) = \chi_L(x)$ for every $x \in \{0,1\}^n$.

At that point in the proof of Lemma 21, we took the binary encodings of such "good" i's as advice. However, in the present case we know that, by condition 1 of Theorem 44, our algorithm A *never* gives a wrong answer, though it may output ?. Hence, we can simply try all possible i's and check if A outputs 0 or 1 on any of them. That is, our new algorithm B is the following: On input $x \in \{0,1\}^n$, accept x if there is a $0 \leqslant i < 2^n$ such that $A(x0^{2^n-n+i}) = 1$, and reject otherwise. It is easy to see that B correctly decides L infinitely often, and that the running time of B is sub-double-exponential. $\square$

Before we can prove our downward closure result, we need to show that the assumption of Theorem 45 can be weakened to say $\mathsf{EXP} \neq \mathsf{ZPP}$. To this end, we prove the following.

Lemma 46. *If, for some* $\epsilon > 0$, $\mathsf{ZPE} \not\subseteq \text{io-}\mathsf{DTIME}(2^{2^{\epsilon n}})$, *then* $\mathsf{BPP} = \mathsf{ZPP}$.

Proof. The proof is very similar to that of Theorem 19. For a given language $L \in \mathsf{ZPE}$, there are two polynomial-time decidable predicates $R_+(x,y)$ and $R_-(x,y)$ such that, for some $c \in \mathbb{N}$, we have for every $x \in \{0,1\}^n$ that

$$x \in L \Rightarrow \mathbf{Pr}_{y\in\{0,1\}^{2^{cn}}}[R_+(x,y)=1] \geqslant 1/2 \text{ and } \mathbf{Pr}_{y\in\{0,1\}^{2^{cn}}}[R_-(x,y)=1] = 0,$$
$$x \notin L \Rightarrow \mathbf{Pr}_{y\in\{0,1\}^{2^{cn}}}[R_+(x,y)=1] = 0 \text{ and } \mathbf{Pr}_{y\in\{0,1\}^{2^{cn}}}[R_-(x,y)=1] \geqslant 1/2.$$

Without loss of generality, we may assume that $c = 1$.

If, for all such pairs (R_+, R_-) and every $\epsilon > 0$, there are infinitely many n where $f_{R_+}(x) = \hat{f}_{R,\varnothing,2^{\epsilon n}}(x)$ for every $x \in \{0,1\}^n$, then it follows by a simple padding argument that $\mathsf{ZPE} \subseteq \text{io-}\mathsf{DTIME}(2^{2^{\epsilon n}})$ for every $\epsilon > 0$. Hence, by our assumption, we have some pair (R_+, R_-) and some $\epsilon > 0$ such that, for all sufficiently large n, $f_{R_+}(x) \neq \hat{f}_{R,\varnothing,2^{\epsilon n}}(x)$ for at least one $x \in \{0,1\}^n$.

Proceeding as in the proof of Theorem 19, we obtain the existence of a $\mathsf{poly}(2^n)$-time algorithm that nondeterministically generates the truth tables of $2n$-variable Boolean functions of circuit complexity $2^{\Omega(n)}$. This algorithm outputs the string $Y = y_1 \dots y_{2^n}$, where $y_i \in \{0,1\}^{2^n}$, such that, for each $x_1, \dots, x_{2^n} \in \{0,1\}^n$, either $R_+(x_i, y_i) = 1$ or $R_-(x_i, y_i) = 1$. However, in our case, this algorithm can be viewed as zero-error probabilistic because of the abundance of witnesses for $x \in L$ and for $x \notin L$. Once we have such an algorithm, we conclude that $\mathsf{BPP} = \mathsf{ZPP}$, by applying Theorem 14. $\square$

Now we can strengthen Theorem 45.

Theorem 47. *If* $\mathsf{EXP} \neq \mathsf{ZPP}$*, then, for every* $\epsilon > 0$*, we have* $\mathsf{ZPE} \subseteq \text{io-}\mathsf{DTIME}(2^{2^{\epsilon n}})$.

Proof. We prove the contrapositive. Suppose that, for some $\epsilon > 0$, $\mathsf{ZPE} \not\subseteq \text{io-}\mathsf{DTIME}(2^{2^{\epsilon n}})$. Then, by Theorem 45, we get $\mathsf{EXP} = \mathsf{BPP}$, and, by Lemma 46, we get $\mathsf{BPP} = \mathsf{ZPP}$. $\square$

Proof of Theorem 42. $\Rightarrow$. This follows by a simple padding argument.

$\Leftarrow$. Suppose that $\mathsf{EE} = \mathsf{ZPE}$ but $\mathsf{EXP} \neq \mathsf{ZPP}$. Then, by Theorem 47, we have $\mathsf{EE} = \mathsf{ZPE} \subseteq \text{io-}\mathsf{DTIME}(2^{2^n})$, contrary to Theorem 4. $\square$

The proof of Theorem 43 is now immediate.

Proof of Theorem 43. $\Rightarrow$. This follows by a simple padding argument.

$\Leftarrow$. If $\mathsf{EE} = \mathsf{RE}$, then $\mathsf{EE} = \mathsf{ZPE}$, and hence, by Theorem 42, we get $\mathsf{EXP} = \mathsf{ZPP} = \mathsf{RP}$. ☐

Theorem 47 yields the following.

Theorem 48 (Gap Theorem for ZPE). *Exactly one of the following holds:*

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

Proof. The proof is very similar to that of Theorem 41. ☐

6.3 Case of MA

For MA, we only know how to prove the following downward closure statement, which is weaker than what we expect to be true. [5]

Theorem 49. $\mathsf{NEE} = \mathsf{MA\text{-}E} \Rightarrow \mathsf{NEXP} \cap \mathsf{coNEXP} = \mathsf{MA}$.

Proof. The proof is by contradiction. Suppose that $\mathsf{NEE} = \mathsf{MA\text{-}E}$, but that $\mathsf{NEXP} \cap \mathsf{coNEXP} \neq \mathsf{MA}$. The latter assumption implies that

1. either $\mathsf{NEXP} \cap \mathsf{coNEXP} \neq \mathsf{EXP}$,
2. or $\mathsf{EXP} \neq \mathsf{MA}$.

We will show that in each of these two cases one gets that $\mathsf{MA} \subseteq \text{io-}\mathsf{NSUBEXP}$.

Indeed, if $\mathsf{NEXP} \cap \mathsf{coNEXP} \neq \mathsf{EXP}$, then it follows by Theorem 20 that $\mathsf{MA} \subseteq \mathsf{AM} \subseteq \text{io-}\mathsf{NSUBEXP}$. On the other hand, if $\mathsf{EXP} \neq \mathsf{MA}$, then it follows by Theorem 22 that $\mathsf{EXP} \not\subset \mathsf{P/poly}$. That is, one can generate deterministically in polynomial time (without any advice!) the truth tables of Boolean functions of superpolynomial circuit complexity (infinitely often), and hence, by Theorem 12 (statement 2), we again obtain that $\mathsf{MA} \subseteq \text{io-}\mathsf{NSUBEXP}$.

Now it follows by a simple padding argument that if $\mathsf{MA} \subseteq \text{io-}\mathsf{NSUBEXP}$, then $\mathsf{MA\text{-}E} \subseteq \text{io-}\mathsf{NSUBEE}/n$ (where the advice of length n is used to point to the correct length, as in the proof of Lemma 21).

Finally, we observe that our assumptions $\mathsf{NEE} = \mathsf{MA\text{-}E}$ and $\mathsf{MA\text{-}E} \subseteq \text{io-}\mathsf{NSUBEE}/n$ contradict Corollary 10. ☐

We conclude this section with the following gap theorem for MA.

Theorem 50 (Gap Theorem for MA). *Exactly one of the following holds:*

1. $\mathsf{MA} = \mathsf{NEXP}$, *or*
2. *for every* $\epsilon > 0$, $\mathsf{MA} \subseteq \text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$.

Proof. If $\mathsf{MA} \neq \mathsf{NEXP}$, then, by Theorem 23, $\mathsf{NEXP} \not\subset \mathsf{P/poly}$. Applying Lemma 1 and Theorem 12 (statement 2) implies that, for every $\epsilon > 0$, $\mathsf{MA} \subseteq \text{io-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$.

On the other hand, if both $\mathsf{MA} = \mathsf{NEXP}$ and $\mathsf{MA} \subseteq \text{io-}[\mathsf{NTIME}(2^n)/n]$, then we get a contradiction by Corollary 9. ☐

[5]The statement that we actually wish to prove is the following: $\mathsf{NEE} = \mathsf{MA\text{-}E} \Rightarrow \mathsf{NEXP} = \mathsf{MA}$.

7 Concluding Remarks and Open Problems

As we mentioned in the Introduction, our result that hard Boolean functions are required for derandomizing MA (Corollary 25) has the following consequence: If there is an efficient deterministic algorithm for estimating the acceptance probability of a given Boolean circuit (and, hence, MA can be derandomized), then NEXP requires superpolynomial circuit size. Thus, hard Boolean functions are also required for derandomizing promise-RP, promise-BPP, and the class APP introduced in [KRC00].

We would like to point out which of our theorems relativize, and which do not. It follows from the results in [BFT98] that the collapse of NEXP to MA when NEXP $\subset$ P/poly (Corollary 25) does *not* relativize; although, the only nonrelativizing ingredient in our proof is the the old result from [BFL91] that EXP $\subset$ P/poly $\Rightarrow$ EXP $=$ MA. The converse implication (Theorem 27) relativizes. The proof of NEXP $\subset$ P/poly $\Rightarrow$ NEXP $=$ EXP (Theorem 24) uses the same nonrelativizing result from [BFL91], but we do not know whether the statement of Theorem 24 itself does not relativize. The proof of Theorem 29 uses only relativizing techniques, and hence, the statement relativizes. Also, Fortnow [For01] shows that all of our downward closure results from Section 6 have proofs that *relativize*. On the other hand, the gap theorems for BPE, ZPE, and MA (Theorems 41, 48, and 50) are proved using non-relativizing techniques, but we do not know if the statements relativize.

As we mentioned in Section 5, one open problem is to decide if the assumption promise-BPP $\subseteq$ promise-P is equivalent to the existence of a deterministic polynomial-time algorithm for CAPP which is "black-box" and non-adaptive. Another open problem is to decide if the existence of a P-natural property useful against P/poly yields EXP $\not\subset$ P/poly.

We also would like to mention a few other open questions. One question is to show that Theorem 24 does (or does not) relativize. Another question is whether Theorem 49 can be improved to have the conclusion NEXP $=$ MA, rather than NEXP $\cap$ coNEXP $=$ MA. Finally, it is interesting to try to generalize our downward closures to higher time complexity classes; the techniques in this paper (as well as those used by Lance Fortnow for the relativizing proofs) fail to show that EEE $=$ BPEE $\Rightarrow$ EE $=$ BPE, where EEE is the class of languages decidable in triple-exponential time and BPEE is the double-exponential version of BPP.

Acknowledgements The authors would like to thank Lance Fortnow, Dieter van Melkebeek, and Salil Vadhan for their comments; special thanks are due to Dieter van Melkebeek for pointing out an error in an early version of this paper, as well as for allowing us to include his theorem (Theorem 27) in our paper. We want to thank an anonymous referee of the conference version of this paper for bringing [BH92] to our attention. The second author also wishes to thank Steve Cook and Charlie Rackoff for many helpful discussions. Finally, we thank anonymous referees of the journal version of this paper for many helpful comments and suggestions.

References

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

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

[Bar01] B. Barak. How to go beyond the black-box simulation barrier. In *Proceedings of the Forty-Second Annual IEEE Symposium on Foundations of Computer Science*, pages 106–115, 2001.

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

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

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

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In J. Kilian, editor, *Advances in Cryptology – CRYPTO'2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, Berlin, Germany, 2001.

[BH92] H. Buhrman and S. Homer. Superpolynomial circuits, almost sparse oracles and the exponential hierarchy. In R. Shyamasundar, editor, *Proceedings of the Twelfth Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*, pages 116–127, Berlin, Germany, 1992. Springer Verlag.

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

[For01] L. Fortnow. Comparing notions of full derandomization. In *Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity*, pages 28–34, 2001.

[Gol99] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics series.* Springer Verlag, 1999.

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

[HIS85] J. Hartmanis, N. Immerman, and V. Sewelson. Sparse sets in NP–P: EXPTIME versus NEXPTIME. *Information and Control*, 65:158–181, 1985.

[IN88] R. Impagliazzo and M. Naor. Decision trees and downward closures. In *Proceedings of the Third Annual IEEE Conference on Structure in Complexity Theory*, pages 29–38, 1988.

[IT89] R. Impagliazzo and G. Tardos. Decision versus search problems in super-polynomial time. In *Proceedings of the Thirtieth Annual IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1989.

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

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

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

[Kab02] V. Kabanets. Derandomization: A brief overview. *Bulletin of the European Association for Theoretical Computer Science*, 76, 2002. (also available as ECCC TR02-008).

[KL82] R.M. Karp and R.J. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(3-4):191–209, 1982. (preliminary version in STOC'80).

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

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

[Lu00] C.-J. Lu. Derandomizing Arthur-Merlin games under uniform assumptions. In *Proceedings of the Eleventh Annual International Symposium on Algorithms and Computation (ISAAC'00)*, 2000.

[Mil01] P.B. Miltersen. Derandomizing complexity classes. In S. Rajasekaran P. Pardalos, J. Reif, and J. Rolim, editors, *Handbook of Randomized Computing*, volume II. Kluwer Academic Publishers, 2001. (a draft is available at `www.brics.dk/~bromille`).

[MV99] P.B. Miltersen and N.V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science*, pages 71–80, 1999.

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

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

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

[STV99] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 537–546, 1999.

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