In Search of an Easy Witness: Exponential Time vs. Probabilistic Polynomial Time
(Extended Abstract)

Russell Impagliazzo*
Department of Computer Science
University of California
San Diego, CA 91097-0114
russell@cs.ucsd.edu

Valentine Kabanets†
Institute for Advanced Study
Princeton, NJ 08540
kabanets@ias.edu

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

April 5, 2001

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 to say 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 open question in complexity theory is whether polynomial-time probabilistic algorithms are more powerful than their deterministic counterparts. A concrete example

*Research Supported by NSF Award CCR-9734911, Sloan Research Fellowship BR-3311, grant #93025 of the joint US-Czechoslovak Science and Technology Program, and USA-Israel BSF Grant 97-00188

†Most of this research was done while the author was at the Department of Computer Science, University of Toronto. 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.

is the question whether $\mathsf{BPP} \stackrel{?}{=} \mathsf{P}$. Despite a common belief 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 yet if $\mathsf{NEXP} = \mathsf{BPP}$.

A number of conditional derandomization results are known which are based on the assumption that EXP contains Boolean functions of "high" circuit complexity [NW94, BFNW93, ACR96, IW97, STV99]. For instance, it is shown in [IW97] that $\mathsf{BPP} = \mathsf{P}$ if $\mathsf{DTIME}(2^{O(n)})$ contains a language that requires Boolean circuits of size $2^{\Omega(n)}$. The 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 certain *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 $\mathsf{BPP} \subseteq \mathsf{SUBEXP}$ when $\mathsf{EXP} \subset \mathsf{P/poly}$.

In order to motivate the importance of this question, let us imagine for a moment that $\mathsf{BPP} = \mathsf{P}$. How might one prove this? There are, essentially, only three conceivable proof scenarios. *(i)* A *non-constructive* proof showing that, for each BPP algorithm, there *exists* a deterministic polynomial-time algorithm deciding the same language, without giving any hint on how such a deterministic algorithm can be designed. *(ii)* A *constructive* proof describing a deterministic polynomial-time algorithm for estimating the acceptance probability of a given Boolean function; here, the algorithm is given as input an encoding of a Boolean circuit computing this function, and must output the fraction of inputs accepted by the circuit, to within a small additive error. *(iii)* A *constructive* proof describing a deterministic polynomial-time algorithm for estimating the acceptance probability of a given Boolean function, using *oracle access* to this function; that is, the deterministic algorithm does not see the encoding of the circuit computing the function (although it is assumed that the function is computable by a "small" circuit), but only gets to see the value of the function on some polynomial number of inputs.

Currently, all of the research on derandomizing BPP, under various assumptions, has actually proved the strongest type of derandomization (scenario (iii) above), under the same assumptions. In particular, these results construct a pseudorandom generator, which is now known to be equivalent to finding a hard function in EXP. Thus, to prove further derandomization results, we need either to give lower bounds for function in EXP, or to weaken our goals to less powerful notions of derandomization, such as scenarios (i) or (ii) above. Proving circuit lower bounds is hard. This raises the question: can we prove derandomization results along the lines of (i) or (ii) without proving circuit lower bounds?

In this paper, we give a negative answer for the intermediate form of derandomization in scenario (ii), which can be reformulated as the conjecture that promise-BPP is contained in SUBEXP. We show that putting promise-BPP even in NSUBEXP, nondeterministic subexponential time, would simultaneously prove a circuit lower bound for a problem in NEXP, nondeterministic exponential time. 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 derandomization along the lines of scenario (ii) is possible without assuming or proving circuit lower bounds for NEXP.

Another piece of evidence that it will be difficult to show $\mathsf{EXP} \neq \mathsf{BPP}$ (or $\mathsf{NEXP} \neq \mathsf{MA}$) 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 $\mathsf{NP} = \mathsf{P}$, then $\mathsf{NEXP} = \mathsf{EXP}$ 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. 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 randomness 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 under the assumption that NEXP $\subset$ P/poly.

A similar idea was applied in [Kab00] 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 give the necessary definitions and 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 6), where the class AM is a probabilistic version of NP (see the next section for the definitions).

In Section 3, we show that NEXP $\subset$ P/poly iff NEXP = MA. Section 4 contains several interesting corollaries of this equivalence, related to the circuit approximation problem and natural proofs.

Using similar techniques, we obtain a few other results in Section 5. We prove in Section 5.1 a "gap" theorem for MA which says that either MA = NEXP, or MA can be simulated infinitely often in nondeterministic subexponential time with sublinear advice. We also show in Section 5.2 that every NEXP search problem can be solved in deterministic time $2^{\mathsf{poly}(n)}$, under the assumption that NEXP = AM.

In Section 6, we establish our downward closure results for ZPP, RP, BPP, and MA.

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

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 class $\mathsf{SUBEE} = \cap_{\epsilon>0}\mathsf{DTIME}(2^{2^{\epsilon n}})$.

We shall also need the definitions of the classes MA and AM [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 MA-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 Pseudorandom Generators

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 n [2] 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 $2^{O(l(n))}$.

2.3 Conditional Derandomization

Theorem 1 ([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 $\{0,1\}^n$.

It is not difficult to see that 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 $k \in \mathbb{N}$.

It is also easy to see 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.

More precisely, 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 for every input $x = 1^n$

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

Theorem 1 now readily implies the following.

Theorem 2. *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 $\mathsf{MA} \subseteq \mathsf{NSUBEXP}/a(n^{o(1)})$.*

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 $\epsilon > 0$ we have $\mathsf{MA} \subseteq \mathsf{io}\text{-}[\mathsf{NTIME}(2^{n^\epsilon})/a(n^\epsilon)]$.

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

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 $\mathsf{AM} \subseteq \mathsf{NSUBEXP}$. More precisely, we have the following.

Theorem 3 ([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 $\mathsf{AM} \subseteq \mathsf{NSUBEXP}/a(n^{o(1)})$.*

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 $\epsilon > 0$ we have $\mathsf{AM} \subseteq \mathsf{io}\text{-}[\mathsf{NTIME}(2^{n^\epsilon})/a(n^\epsilon)]$.

2.4 Our Main Tools

2.4.1 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 $|y| \leqslant l(|x|)$ for some function $l : \mathbb{N} \to \mathbb{N}$. That is, we need to compute the Boolean function $f_R(x)$ defined by

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

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 4. *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. □

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 5. *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 input 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 2.3, 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 5 yields the following derandomization result for AM, based on a uniform hardness assumption.

Theorem 6. *If $\mathsf{NEXP} \neq \mathsf{EXP}$, then, for every $\epsilon > 0$, we have $\mathsf{AM} \subseteq \mathsf{io}\text{-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$.* [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 5, 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 3 (statement 2). □

We also prove the following

Theorem 7. *If $\mathsf{NE} \cap \mathsf{coNE} \not\subseteq \mathsf{io}\text{-}\mathsf{DTIME}(2^{2^{\epsilon n}})$ for some $\epsilon > 0$, then $\mathsf{AM} = \mathsf{NP}$.*

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

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

Proof. Let $\epsilon > 0$ be arbitrary. We define a padded version of any given language $L \in \mathsf{BPE}$ by $L_{\mathrm{pad}} = \{x0^{2^{|x|}-|x|+i} \mid x \in L,\ 0 \leqslant i < 2^{|x|}\}$. 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 $y = x0^{2^n-n+i}$, 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 $r0^{m-n}$.

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 $y = x0^{2^n-n+i}$, 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 $A(x0^{2^n-n+i}) = \chi_L(x)$. 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. □

3 Uniform versus Nonuniform Complexity of NEXP

Babai, Fortnow, Nisan, and Wigderson [BFNW93], improving on a result of Karp and Lipton [KL82], show that if $\mathsf{EXP} \subset \mathsf{P/poly}$, then $\mathsf{EXP} = \mathsf{MA}$. Here we will prove

Theorem 9. $\mathsf{NEXP} \subset \mathsf{P/poly} \Leftrightarrow \mathsf{NEXP} = \mathsf{MA}$.

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 9.

Theorem 10. *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 [BFNW93], we get that assumption (1) implies that $\mathsf{EXP} = \mathsf{AM} = \mathsf{MA}$. By Theorem 6, we get from assumption (2) that, for every $\epsilon > 0$, $\mathsf{AM} \subseteq \mathsf{io}\text{-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$. Combining the two implications, we get that $\mathsf{EXP} \subseteq \mathsf{io}\text{-}[\mathsf{NTIME}(2^n)/n]$.

We shall need the following easy claim.

Claim 11. *Assumption* (1) *implies that there is a universal constant* $d_0 \in \mathbb{N}$ *such that*

$$\mathsf{NTIME}(2^n)/n \subset \mathsf{SIZE}(n^{d_0})$$

almost everywhere.

Proof of Claim 11. 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 (1), 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 $O((2n)^k) \in O(n^k)$. The claim follows if we take $d_0 = k + 1$. □

Thus, our assumptions allow us to conclude, using Claim 11, that there is a universal constant $d_0 \in \mathbb{N}$ such that $\mathsf{EXP} \subset \mathsf{io\text{-}SIZE}(n^{d_0})$. But, this is impossible by a straightforward diagonalization. □

Corollary 12. *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 10, and $\mathsf{EXP} = \mathsf{MA}$ by the result of [BFNW93] mentioned above. □

Remark 13. The proof of Theorem 10 actually shows that if $\mathsf{NEXP} \subset \mathsf{P/poly}$, then for every language $L \in \mathsf{EXP}$ there is a constant $c \in \mathbb{N}$ such that all sufficiently large strings $x \in L$ have at least one witness that can be described by a Boolean circuit of size $|x|^c$.

The other direction of Theorem 9 was proved by Dieter van Melkebeek [van Melkebeek, personal communication, September 2000]. For the proof, we shall need the following claim whose proof is similar to that of Claim 11.

Claim 14. *If* $\mathsf{NEXP} = \mathsf{EXP}$*, then there is a universal constant* $c_0 \in \mathbb{N}$ *such that* $\mathsf{NTIME}(2^n) \subseteq \mathsf{DTIME}(2^{n^{c_0}})$.

Theorem 15 (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 1, the latter yields that $\mathsf{MA} \subseteq \mathsf{io\text{-}NTIME}(2^n)$.

Thus, our assumptions (3) and (4) together with Claim 14 imply that there is a universal constant c_0 such that $\mathsf{EXP} \subseteq \mathsf{io\text{-}DTIME}(2^{n^{c_0}})$, contradicting the Time Hierarchy Theorem. □

Proof of Theorem 9. The proof follows immediately from Corollary 12 and Theorem 15. □

4 Some Corollaries of Theorem 9

4.1 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.

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 $f(x)$ 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 $q \in [f(x)-1/6, f(x)+1/6]$.

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 polynomial-size circuit). 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 the size of a smallest circuit deciding f, and all of these queries are computed *before* obtaining the value of f on any one of them.

Theorem 16. *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

$$\mathsf{MA} \subseteq \mathsf{io\text{-}NSUBEXP}/n^{o(1)}. \tag{5}$$

This can be shown to imply that $\mathsf{NEXP} \neq \mathsf{MA}$ (otherwise, if $\mathsf{NEXP} = \mathsf{MA}$, then (5) yields a contradiction to the deterministic Time Hierarchy Theorem). Hence, by Theorem 9, we conclude that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$.

(1) $\Rightarrow$ (3). This follows from Theorem 1, by noticing that the truth table of an n-variable Boolean function f in NEXP can be computed in nondeterministic time $\mathsf{poly}(2^n)$ when given as advice the n-bit number $a = |\{f^{-1}(1)\}|$ (we just guess a truth table of size 2^n which contains exactly a ones, and then guess and check the corresponding a witnesses). □

Remark 17. The big open question is whether an analogue of Theorem 16 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 promise-BPP $\subseteq$ P, where promise-BPP $\subseteq$ P 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

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

As an immediate consequence of Theorem 16, we obtain the following.

Corollary 18. promise-BPP $\subseteq$ P $\Rightarrow$ NEXP $\not\subset$ P/poly.

Obviously, if promise-BPP $\subseteq$ P, then BPP = P. However, the converse is *not* known to hold. If the converse were to hold, then Theorem 16 would yield that BPP = P $\Rightarrow$ NEXP $\not\subset$ P/poly, and hence, derandomizing BPP would be as hard as proving circuit lower bounds for NEXP.

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

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

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 2, these hard Boolean functions can then be used to derandomize MA, yielding NEXP $\neq$ MA. Now the claim follows by Theorem 9. □

Remark 20. Note the following subtlety in our proof of Theorem 19. 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 21. Here the interesting open problem is to try to prove a "deterministic" version of Theorem 19. That is, does the existence of a P-natural property useful against P/poly imply that EXP $\not\subset$ P/poly?

5 Other Results

5.1 A Gap Theorem for MA

The following statement can be interpreted as a "gap" theorem for MA: either MA is as powerful as NEXP, or MA is significantly "smaller" than NEXP.

Theorem 22. *Exactly one of the following holds:*

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

The proof of this theorem will follow from the two statements below.

Theorem 23. *If* MA $\neq$ NEXP, *then for every* $\epsilon > 0$, MA $\subseteq$ io-[NTIME$(2^{n^\epsilon})/n^\epsilon$].

Proof. If MA $\neq$ NEXP, then, by Theorem 9, NEXP $\not\subset$ P/poly. This, in turn, implies that there is a poly(2^n) algorithm B and a family $\{x_n\}_{n\geqslant 0}$ of n-bit strings x_n such that B on input x_n nondeterministically generates the truth table of an n-variable Boolean function f_n so that the following holds: for every $d \in \mathbb{N}$, there are infinitely many n where f_n is of circuit complexity greater than n^d.

Indeed, let $L \in$ NEXP $\setminus$ 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 input, B nondeterministically guesses c_n strings $y_i \in L \cap \{0,1\}^n$ together with their certificates $z_i \in \{0,1\}^{2^{\mathsf{poly}(n)}}$. 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.

Applying Theorem 2 (statement 2) concludes the proof. □

Theorem 24. *If* MA = NEXP, *then* MA $\not\subseteq$ io-[NTIME$(2^n)/n$].

The proof of Theorem 24 relies on the following version of the Time Hierarchy Theorem.

Claim 25. EXP $\not\subseteq$ io-[DTIME$(2^{n^{c_0}})/n$], *for any fixed* $c_0 \in \mathbb{N}$.

Proof. The proof is by diagonalization. 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 $2^{n^{c_0}}$-time Turing machine of description of size n that uses an advice string of size n; clearly, we have $|S_n| \leqslant 2^{2n}$. 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 $2n + 1$ 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 $2^{O(n^{c_0})}$. 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, $L \notin \mathsf{io}\text{-}[\mathsf{DTIME}(2^{n^{c_0}})/n]$. □

Proof of Theorem 24. Suppose that $\mathsf{MA} = \mathsf{NEXP}$ and $\mathsf{MA} \subseteq \mathsf{io}\text{-}[\mathsf{NTIME}(2^n)/n]$. If $\mathsf{MA} = \mathsf{NEXP}$, then $\mathsf{NEXP} = \mathsf{EXP}$, and hence, by Claim 14, there is a universal constant $c_0 \in \mathbb{N}$ such that $\mathsf{NTIME}(2^n)/n \subseteq \mathsf{DTIME}(2^{n^{c_0}})/n$. But, then $\mathsf{EXP} = \mathsf{MA} \subseteq \mathsf{io}\text{-}[\mathsf{DTIME}(2^{n^{c_0}})/n]$, contradicting Claim 25. □

5.2 Search versus Decision for Exponential Time

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

Proof. For the sake of contradiction, suppose that $\mathsf{NEXP} = \mathsf{AM}$, but some NEXP search problem cannot be solved in deterministic time $2^{\mathsf{poly}(n)}$.

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 every NEXP search problem is solvable in deterministic time $2^{\mathsf{poly}(n)}$. Hence, by our assumption, 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 5 and Theorem 3, we obtain that, for every $\epsilon > 0$, $\mathsf{AM} \subseteq \mathsf{io}\text{-}[\mathsf{NTIME}(2^{n^\epsilon})/n^\epsilon]$. Similarly to the proof of Theorem 24, our assumption that $\mathsf{NEXP} = \mathsf{EXP} = \mathsf{AM}$ yields, by Claim 14, that there is a universal constant $c_0 \in \mathbb{N}$ such that $\mathsf{EXP} = \mathsf{AM} \subseteq \mathsf{io}\text{-}[\mathsf{DTIME}(2^{n^{c_0}})/n]$, contradicting Claim 25. □

6 Downward Closure for Probabilistic Classes

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 $\mathsf{P} = \mathsf{NP} \Rightarrow \mathsf{DTIME}(\mathsf{polylog}(n)) = \mathsf{NTIME}(\mathsf{polylog}(n)) \cap \mathsf{coNTIME}(\mathsf{polylog}(n)) = \mathsf{RTIME}(\mathsf{polylog}(n))$; see also [BFNW93] and [HIS85]. We prove several downward closure results for probabilistic complexity classes. In particular, we establish the following

Theorem 28. $\mathsf{EXP} = \mathsf{BPP} \Leftrightarrow \mathsf{EE} = \mathsf{BPE}$.

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

Theorem 29 ([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 30. *If* $\mathsf{EXP} \neq \mathsf{BPP}$, *then, for every* $\epsilon > 0$, *we have* $\mathsf{BPE} \subseteq \mathsf{io}\text{-}[\mathsf{DTIME}(2^{2^{\epsilon n}})/n]$.

Proof. If $\mathsf{EXP} \neq \mathsf{BPP}$, then, by Theorem 29, the assumption of Lemma 8 is satisfied, and hence, our claim follows. □

We also need the following stronger version of the Time Hierarchy Theorem.

Claim 31. $\mathsf{EE} \not\subseteq \mathsf{io}\text{-}[\mathsf{DTIME}(2^{2^{c_0 n}})/(2^n - n)]$, *for any fixed* $c_0 \in \mathbb{N}$.

Proof. The proof is by a simple diagonalization. Let $s(n) = 2^n - n$. Define a language as follows. On inputs of length n, we construct all truth tables of the first n Turing machines run for time $2^{2^{c_0 n}}$ with all advice strings of length $s(n)$ or smaller; there are at most $n2^{s(n)+1} < 2^{2^n}$ 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. □

Proof of Theorem 28. $\Rightarrow$. 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 30, $\mathsf{BPE} \subseteq \mathsf{io}\text{-}[\mathsf{DTIME}(2^{2^n})/n]$. But then so is EE, contradicting Claim 31. □

Using similar ideas, we also prove the following results.

Theorem 32. $\mathsf{EXP} = \mathsf{ZPP} \Leftrightarrow \mathsf{EE} = \mathsf{ZPE}$.

Theorem 33. $\mathsf{EXP} = \mathsf{RP} \Leftrightarrow \mathsf{EE} = \mathsf{RE}$.

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

7 Concluding Remarks and Open Problems

As we mentioned in the Introduction, our result that hard Boolean functions are required for derandomizing MA (Corollary 12) 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 under the assumption that $\mathsf{NEXP} \subset \mathsf{P/poly}$ (Corollary 12) does *not* relativize; however, the converse implication (Theorem 15) does. The proof of the collapse of NEXP to EXP under the same assumption (Theorem 10) uses a nonrelativizing result from [BFNW93], but we do not know whether the statement of Theorem 10 itself does not relativize. The proof of Theorem 27 uses only relativizing techniques, and hence, the statement also relativizes. Also, Lance Fortnow [Fortnow, personal communication, December, 2000] pointed out that all of our downward closure results from Section 6 have alternative, *relativizing* proofs. On the other hand, the gap theorem for MA (Theorem 23) is proved using non-relativizing techniques, but we do not know if the statement itself relativizes.

As we mentioned in Section 4, one open problem is to decide if the assumption $\mathsf{promise\text{-}BPP} \subseteq \mathsf{P}$ 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 $\mathsf{EXP} \not\subset \mathsf{P/poly}$.

We also would like to mention a few other open questions. One question is to show that Theorem 10 does (or does not) relativize. Another question is whether Theorem 34 can be improved to have the conclusion $\mathsf{NEXP} = \mathsf{MA}$, rather than $\mathsf{NEXP} \cap \mathsf{coNEXP} = \mathsf{MA}$. 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 $\mathsf{EEE} = \mathsf{BPEE} \Rightarrow \mathsf{EE} = \mathsf{BPE}$, 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 15) in our paper. We want to thank the anonymous referee for bringing [BH92] to our attention. The second author also wishes to thank Steve Cook and Charlie Rackoff for many helpful discussions.

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

References

[ACR96] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. Hitting properties of hard Boolean operators and its consequences on BPP. manuscript, 1996.

[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.

[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.

[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.

[Kab00] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. In *Proceedings of the Fifteenth Annual IEEE Conference on Computational Complexity*, pages 150–157, 2000.

[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.

[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.