Electronic Colloquium on Computational Complexity, Report No. 51 (2009)

# The Pervasive Reach of Resource-Bounded Kolmogorov Complexity in Computational Complexity Theory*

Eric Allender†
Rutgers University
Piscataway, NJ, USA
allender@cs.rutgers.edu

Michal Koucký‡
Institute of Mathematics of the Academy of Sciences of the Czech Republic
Prague, Czech Republic
koucky@math.cas.cz

Detlef Ronneburger§
Bloomberg LP
dronneburgr@bloomberg.net

Sambuddha Roy
India Research Laboratory, IBM, New Delhi, India
sambuddha@in.ibm.com

*Much of this material appeared in preliminary form as "Derandomization and Distinguishing Complexity" in the proceedings of the 2003 IEEE Conference on Computational Complexity [AKRR03].

†Partially supported by NSF grants DMS-0652582, CCF-0830133, and CCF-0832787.

‡Supported in part by grant NSF grant CCF-0514703 and by grant GA ČR 201/07/P276, project No. 1M0021620808 of MŠMT ČR, Institutional Research Plan No. AV0Z10190503 and grant IAA100190902 of GA AV ČR.

§This work was done while at York College, City University of New York.

ISSN 1433-8092

**Abstract**

We continue an investigation into resource-bounded Kolmogorov complexity [ABK$^+$06], which highlights the close connections between circuit complexity and Levin's time-bounded Kolmogorov complexity measure Kt (and other measures with a similar flavor), and also exploits derandomization techniques to provide new insights regarding Kolmogorov complexity. The Kolmogorov measures that have been introduced have many advantages over other approaches to defining resource-bounded Kolmogorov complexity (such as much greater independence from the underlying choice of universal machine that is used to define the measure) [ABK$^+$06]. Here, we study the properties of other measures that arise naturally in this framework.

The motivation for introducing yet more notions of resource-bounded Kolmogorov complexity are two-fold:

- to demonstrate that other complexity measures such as branching-program size and formula size can also be discussed in terms of Kolmogorov complexity, and
- to demonstrate that notions such as nondeterministic Kolmogorov complexity and distinguishing complexity [BFL02] also fit well into this framework.

The main theorems that we provide using this new approach to resource-bounded Kolmogorov complexity are:

- A complete set ($R_{\mathrm{KNt}}$) for NEXP/poly defined in terms of strings of high Kolmogorov complexity.
- A lower bound, showing that $R_{\mathrm{KNt}}$ is not in NP $\cap$ coNP.
- New conditions equivalent to the conditions "NEXP $\subseteq$ nonuniform NC$^1$" and "NEXP $\subseteq$ L/poly".
- Theorems showing that "distinguishing complexity" is closely connected to both FewEXP and to EXP.
- Hardness results for the problems of approximating formula size and branching program size.

# 1 Introduction

The goal of this paper is to develop more fully the relationship between Kolmogorov complexity and computational complexity – with particular emphasis on circuit complexity. In so doing, we expand on a theme that dates back to the earliest investigations of the P vs. NP question, and beyond. Let us begin by giving a brief overview of this history.

Karp's landmark paper [Kar72], which demonstrated the widespread applicability of the notion of NP-completeness as a tool for understanding the apparent intractability of computational problems, took Cook's earlier work [Coo71] as its starting point. It is known now, but was not known then, that Levin had made similar discoveries [Lev73] independently at roughly the same time as Cook. Trakhtenbrot [Tra84] has written an informative account, outlining the fundamental questions that engaged that segment of the research community in Russia that was working on theoretical computer science at the time. We now review part of that history.

## 1.1 The Russian Program

As related by Trakhtenbrot, the attention of the Russian research community focused on problems that seemed to require "*perebor*" or brute-force search. One such problem that was of particular interest was the problem (called "Task 4" by Trakhtenbrot [Tra84, p. 390]) of taking as input the truth-table of a Boolean function, and determining if it has Boolean circuits of a given size. More recently, essentially the same computational problem has been studied under the name MCSP, for the "Minimum Circuit Size Problem" [KC00]. Levin has said that he delayed publication of his work on the complexity of SAT ([Lev73]) because he had been hoping to capture MCSP in this framework [Lev03]. Nearly four decades later, it is still not known if MCSP is NP-complete, and few seem to expect that it really is complete under Karp reductions [KC00].

Trakhtenbrot further relates that it was recognized that MCSP was similar in spirit to the problem of taking a binary string as input and determining its time-bounded Kolmogorov complexity. More precisely, Trakhtenbrot describes a problem (called "Task 5" in [Tra84, p. 392]) defined in terms of some fixed "universal" Turing machine $U$ and a fixed time bound $t(n)$ (such as $t(n) = n^2$) where one takes as input a string $x$ of length $n$ and determines if there is a description $d$ of a given length, such that $U(d) = x$ in time $t(n)$. Thus these two related threads of inquiry were already being discussed in Russia in the 1960s – although there was no theorem explicitly linking the two threads. In the same way that MCSP is not known to be NP-complete, Ko showed that the question of whether computing this sort of time-bounded Kolmogorov complexity is NP-hard cannot be settled by relativizing methods [Ko91].

A third thread dating to this period is also discussed by Trakhtenbrot, when he mentions a different notion of time-bounded Kolmogorov complexity that was introduced by Levin. Levin developed this notion as a tool for proving the second theorem of his 1971 paper [Lev73], in which he presents an optimal search algorithm for NP problems. (This definition, $\mathrm{Kt}(x)$, does not actually appear in Levin's 1971 paper and the earliest published definition seems to be more than a decade later [Lev84].) The difference between this definition and the time-bounded Kolmogorov complexity mentioned in the preceding paragraph lies in the way that time is incorporated into the complexity measure. Rather than fixing the time bound $t(n)$ that $U$ can use to produce $x$ from a short description, instead $\mathrm{Kt}(x)$ is defined by minimizing $m + \log t$, where $m$ is the length of the description $d$ such that $U(d) = x$ in $t$ time steps. (Formal definitions are found in Section 2.) It seems that no connection was suggested between Levin's Kt measure and the Minimum Circuit Size Problem MCSP until roughly three decades later.

## 1.2 Weaving the threads together

The connections between these three threads were made more explicit just a few years ago. In 2002, it was shown [ABK$^+$06] that if $x$ is a string of length $2^m$ (and thus can be viewed as the truth table of a function $f_x$), then $\mathrm{Kt}(x)$ is roughly the same as the size of the smallest *oracle* circuit computing $f_x$, where the oracle is a complete set for $\mathrm{E} = \mathrm{DTime}(2^{O(n)})$. Furthermore, the set $R_{\mathrm{Kt}}$, defined as the set of all $x$ such that $\mathrm{Kt}(x) \geq |x|$ (the so-called Kt-*random* strings) is complete for EXP under P/poly reductions.

This turned out to be a manifestation of a more general phenomenon. A new variant of time-bounded Kolmogorov complexity in the spirit of Levin's Kt was presented, denoted KT [ABK$^+$06], where $\mathrm{KT}(x)$ is polynomially-related to the size of the smallest circuit computing $f_x$ (and in the relativized setting $\mathrm{KT}^A(x)$ is polynomially-related to the size of the smallest circuit with oracle gates for $A$ computing $f_x$). Thus the difference between solving MCSP and computing $\mathrm{KT}(x)$ amounts to not much more than determining the "size" of the smallest circuit for $f_x$ using different notions of "size" (such as counting the number of wires in a circuit as opposed to counting the number of symbols in an encoding of the circuit). Furthermore, the sets $R_{\mathrm{KT}^A}$ and $\mathrm{MCSP}^A$ turned out to be complete for $\mathrm{PSPACE}$, EXP, EXPSPACE, and doubly-exponential time, etc, for the appropriate choice of $A$ [ABK$^+$06]. For the important case when $A = \emptyset$ (i.e., for the problem MCSP for circuits without oracle gates), we still have no completeness theorems, although it is known that factoring and other problems that are conjectured to give rise to cryptographically secure one-way functions are reducible to MCSP and $R_{\mathrm{KT}}$ via probabilistic reductions [ABK$^+$06, KC00]. More generally, it is known that *no* function computable in polynomial time is cryptographically secure relative to MCSP [ABK$^+$06, KC00].

The reductions and completeness results that classify the complexity of $R_{\mathrm{Kt}}$ and related problems [ABK$^+$06] all rely on *derandomization* techniques [BFNW93, IW97, IW98]. The current paper is motivated largely by the desire to understand how other previously-studied notions of Kolmogorov complexity and other derandomization techniques relate to each other.

## 1.3 Variants of Resource-Bounded Kolmogorov Complexity

Before we state the contributions of this paper, let us briefly recall the main variants of resource-bounded Kolmogorov complexity. Li and Vitanyi discuss three different approaches to defining time-bounded Kolmogorov complexity in their book, which is the standard reference for the field [LV93]:

1. Levin's Kt measure, which we have already discussed.
2. The measures $\mathrm{C}^t$ and $\mathrm{K}^t$ that result by adapting the standard plain and prefix Kolmogorov complexity (C and K, respectively) by allowing the universal machine $U$ only time $t(n)$ to produce a string of length $n$ as output. $\mathrm{C}^t$ and $\mathrm{K}^t$ are polynomially related, and thus for the purposes of this paper we group them together.
3. *Distinguishing Complexity*, denoted $\mathrm{CD}^t$ and $\mathrm{KD}^t$, depending on whether one is using the plain or prefix version of this notion.

Distinguishing complexity was introduced by Sipser [Sip83] as a tool in his original proof showing that BPP lies in the polynomial hierarchy. Briefly, $\mathrm{CD}^t(x)$ is the length of the shortest description $d$ such that $U^d(y)$ runs in time $t(|y|)$ and accepts if and only if $y = x$.

Buhrman, Fortnow, and Laplante conducted a thorough study of $\mathrm{CD}^t$ complexity [BFL02], and also introduced a nondeterministic variant of $\mathrm{CD}^t$, which they denote $\mathrm{CND}^t$.

## 1.4 Our Contributions

The main technical contributions of this paper can be enumerated:

1. We present definitions of deterministic and nondeterministic distinguishing complexity (KDt and KNDt, respectively) that are in the style of Levin's Kt measure, and share some of the advantages that Kt enjoys over $\mathrm{C}^t$ and related measures, such as less dependence on the choice of universal machine $U$, and closer connections with circuit complexity.
2. We observe that KNDt is more-or-less equivalent to a different nondeterministic Kolmogorov complexity measure KNt that is even more directly analogous to Kt and is more obviously connected to nondeterministic circuit complexity.
3. We show that $R_{\mathrm{KNt}}$ (the set of strings having high KNt complexity) is complete for NEXP/poly under P/poly truth-table reductions and hard for FewEXP under NP-Turing reductions, and draw connections between KNt complexity and techniques that have been developed for derandomizing AM [MV99, SU05, SU07, SU06].
4. We show that $R_{\mathrm{KNt}}$ is not in NP $\cap$ coNP. In contrast, note that we still have no good lower bounds for $R_{\mathrm{Kt}}$.
5. We observe that $R_{\mathrm{KDt}}$ shares with $R_{\mathrm{Kt}}$ the property of being complete for EXP. However, if $R_{\mathrm{KDt}}$ and $R_{\mathrm{Kt}}$ are polynomially-related, then EXP $=$ FewEXP.
6. We demonstrate the wide applicability of definitions in the mold of Kt and KT, by introducing measures KF and KB that are polynomially related to formula size and branching program size, respectively. We show that factoring Blum integers is efficiently reducible the problem of approximating KB and KF complexity.

7. We show that NEXP is contained in nonuniform $NC^1$ if and only if KNt and KF are polynomially-related, and obtain several other statements that are equivalent to this collapse. Many important questions in complexity theory can be re-stated equivalently in terms of questions about the relationships among different variants of resource-bounded Kolmogorov complexity.

The rest of the paper is organized as follows. In Section 2 we present our basic definitions and background information. In Section 3 we present our results characterizing the computational complexity of various problems relating to resource-bounded Kolmogorov complexity. In Section 4 we study nondeterministic Kolmogorov complexity in connection with various tools of derandomization. In Section 5 we investigate the relationship between Kolmogorov complexity and various possible collapses of NEXP to smaller classes. In Section 6 we study distinguishing complexity, and in particular study the consequences that would follow if some of these measures were polynomially related. We continue this investigation of possible polynomial relationships among various measures in Section 7. Finally, we offer some concluding remarks in Section 8.

# 2 Definitions of Resource-Bounded Kolmogorov Complexity Measures

## 2.1 Universal Turing Machines

Our definitions are not overly sensitive to the particular choice of model of computation, but to avoid ambiguity we will be precise about the model that we use. We use essentially the same model of Turing machines that was considered in [BIS90, ABK$^+$06]. The machine has one read-only input tape of length $n$, a constant number of read-write working tapes of infinite length, and a read-write input address tape. At every time step the machine can modify the content of its read-write tapes using the appropriate heads and move these heads left or right by one tape cell. It can also query the content of the input bit whose address is written on the input address tape. If there is no such input bit the reply to the query is the symbol "*".

Beside considering deterministic Turing machines we use also *nondeterministic* and more general *alternating* Turing machines. These machines have in addition to deterministic states also existential and universal states. We refer the reader to [Pap94] for more background on nondeterministic and alternating Turing machines. An alternating machine runs in space $s$ and time $t$ on a given input if each of its possible computations on that input uses space at most $s$ and runs for at most $t$ steps.

In the case where the machine is an oracle Turing machine (possibly having more than one oracle), for each oracle the machine has one read-write oracle tape. At every step the machine can query any of its oracles about whether the string written on the corresponding oracle tape belongs to the oracle set. We also allow finite oracles. For a finite oracle $y \in \{0,1\}^*$, the machine obtains as an answer to its query $i$ bit $y_i$ if $i \leq |y|$ and "*" otherwise. Note that the input tape behaves like an oracle tape accessing a finite oracle.

A place of central importance is occupied by *universal* machines. Since we are concerned with time and space bounded computation we will require the universal machines to be space and time efficient. We formalize this requirement further. Using the technique of Hennie and Stearns [HS66] and Fürer [Für82, Für84] we can establish the following proposition.

**Proposition 1 (minimal simulation overhead)** *1. There is a deterministic (nondeterministic/alternating) Turing machine $U$ with two work tapes, such that for any deterministic (nondeterministic/alternating) oracle Turing machine $M$ there is a constant $c_M$ so that for any finite oracle $d$ there is a finite oracle $d'$ of length at most $|d| + c_M$ such that for any oracle $A$ and input $x$, $U^{A,d'}(x)$ accepts iff $M^{A,d}(x)$ accepts. The com-*

*putation time of $U$ is at most $c_M t \log t$ and the space used is at most $c_M s$, where $M^{A,d}(x)$ runs for time $t$ and uses space $s$. Furthermore, if $M$ is a two-tape machine, then the running time of $U$ is bounded by $c_M t$.*

2. *There is a nondeterministic (alternating) Turing machine $U$ with two work tapes, such that for any nondeterministic (alternating) oracle Turing machine $M$ there is a constant $c_M$ so that for any finite oracle $d$ there is a finite oracle $d'$ of length at most $|d| + c_M$ such that for any oracle $A$ and input $x$, $U^{A,d'}(x)$ accepts iff $M^{A,d}(x)$ accepts. The computation time of $U$ is at most $c_M t$, where $M^{A,d}(x)$ runs in time $t$.*

We call any machine $U$ that satisfies the first part of the previous proposition a *universal* Turing machine; note that we require our universal Turing machines to be space and time efficient in simulating other machines. We call a *fast universal* Turing machine any machine $U$ that satisfies the second part of the previous proposition. Note that the term "fast universal Turing machine" is reserved for nondeterministic and alternating machines.

**Definition 2** *A Turing machine $U$ is* universal *if it satisfies all properties stated in Part 1 of Proposition* 1. *A Turing machine $U$ is* fast universal *if it satisfies all properties stated in Part 2 of Proposition* 1.

## 2.2 The measures Ks, Kt, KS and KT

The history of Levin's Kt measure was discussed briefly in the introduction. The formal definition of Kt that we present below is equivalent to the original definition ([Lev84]) up to an additive logarithmic term. We will use the definition that was used in our earlier paper ([ABK+06], which also introduced the measures KT and KS), because it provides us with a uniform framework in which to present the new definitions that are the primary focus of this paper.

**Definition 3** *Let $U$ be a deterministic Turing machine.*

$$\begin{aligned}
\mathrm{Kt}_U(x) &= \min\{|d| + \log t \,:\, \forall b \in \{0,1,*\} \\
&\qquad \forall i \le n+1\ U^d(i,b) \text{ runs in} \\
&\qquad \text{time } t \text{ and accepts iff } x_i = b\} \\
\mathrm{KT}_U(x) &= \min\{|d| + t \,:\, \forall b \in \{0,1,*\} \\
&\qquad \forall i \le n+1\ U^d(i,b) \text{ runs in} \\
&\qquad \text{time } t \text{ and accepts iff } x_i = b\} \\
\mathrm{Ks}_U(x) &= \min\{|d| + \log s \,:\, \forall b \in \{0,1,*\} \\
&\qquad \forall i \le n+1\ U^d(i,b) \text{ runs in} \\
&\qquad \text{space } s \text{ and accepts iff } x_i = b\} \\
\mathrm{KS}_U(x) &= \min\{|d| + s \,:\, \forall b \in \{0,1,*\} \\
&\qquad \forall i \le n+1\ U^d(i,b) \text{ runs in} \\
&\qquad \text{space } s \text{ and accepts iff } x_i = b\}
\end{aligned}$$

*Here, we say that $x_i = *$ if $i > |x|$.*

Universal Turing machines provide time efficient simulations of other machines, so if $U$ is a universal Turing machine and $U'$ is any other Turing machine, then $\mathrm{Kt}_U(x) \le \mathrm{Kt}_{U'}(x) + c \cdot \log|x|$, $\mathrm{KS}_U(x) \le c \cdot \mathrm{KS}_{U'}(x)$, $\mathrm{Ks}_U(x) \le \mathrm{Ks}_{U'}(x) + c \cdot \log|x|$, and $\mathrm{KT}_U(x) \le c \cdot \mathrm{KT}_{U'}(x) \log|x|$. Hence, none of these complexity measures

changes much when one changes from one universal machine $U$ to another. As usual in studies of Kolmogorov complexity, we will choose a fixed universal Turing machine $U$ and use the notation Kt, KS, Ks and KT to refer to $\text{Kt}_U, \text{KS}_U, \text{Ks}_U$ and $\text{KT}_U$.

In the traditional study of Kolmogorov complexity without resource bounds, the choice of universal machine $U$ affects the measures $\text{K}(x)$ and $\text{C}(x)$ only by additive constant terms [LV93]. In contrast, the choice of $U$ affects the value of $\text{Kt}(x)$ by an additive logarithmic term, and it affects $\text{KT}(x)$ by a multiplicative logarithmic factor. This comes from the slight slow-down that is incurred in the simulation of $U'$ by $U$. Some of the other measures that we will study are affected to an even greater degree by the choice of the universal machine $U$. However, the situation is much better for KT and Kt and other measures in this vein, than it is for measures such as $\text{C}^t$ and $\text{K}^t$ [ABK$^+$06].

It is worthwhile mentioning how the definition of Kt given here differs from the one given by Levin [Lev84]. In Levin's original definition, the machine $U$ is required to produce the entire string $x$ as output, whereas here the machine $U$ gets index $i$ and has to determine the $i$-th bit of string $x$. This change allows the running time to be sublinear in the length of $x$. As this would allow the machine $U$ not to be aware of the actual length of $x$, we stipulate that for $i = |x| + 1$ the output should be "*". Since our definition does not require $U$ to produce any output, but merely to accept or reject, it is well-suited for generalization to nondeterministic and alternating machines.

## 2.3 The measures KB and KF

The definition of KT complexity is motivated in large part by the fact that $\text{KT}(x)$ is a good estimate of the circuit size required to compute the function $f_x$ that has $x$ as its truth table [ABK$^+$06]. More precisely, for any string $x$ of length $2^m$, let $\text{Size}^A(x)$ denote the number of wires in the smallest oracle circuit with oracle $A$ that computes $f_x$. The following theorem holds:

**Theorem 4** *[ABK$^+$06] There is a constant $c$ such that for any oracle $A$ and any string $x$ of length $2^m$,*

1. $\text{Size}^A(x) \leq c(\text{KT}^A(x))^2\,(\text{KT}^A(x)^2 + \log|x|)$ *and*
2. $\text{KT}^A(x) \leq c\,(\text{Size}^A(x))^2\,(\log \text{Size}^A(x) + \log\log|x|)$.

But circuit size is only one of many possible interesting measures of the "complexity" of $f$. There is also great interest in knowing the size of the smallest branching program computing $f$, as well as the size of the smallest Boolean formula representing $f$. Do these notions of complexity also give rise to a natural notion of Kolmogorov complexity? In this subsection, we answer this question by presenting definitions having the same general flavor as KT.

First, we need to present some background information about branching programs and Boolean formulae. For our purposes, a *Boolean formula* is a circuit with AND and OR gates of fan-in two and fan-out one (except for the output gate, which has fan-out zero), where the inputs to the circuit are literals from $\{x_i, \overline{x_i} : 1 \leq i \leq n\}$. The *size* of a formula is the number of gates; a formula represents a Boolean function on $x_1, \ldots x_n$ in the obvious way.

A *branching program* is a directed acyclic graph with a single source and two sinks labeled 1 and 0, respectively. Each non-sink node in the graph is labeled with a variable in $\{x_1, \ldots, x_n\}$ and has two edges leading out of it: one labeled 1 and one labeled 0. A branching program computes a function $f$ on input $x = x_1 \ldots x_n$ by first placing a pebble on the source node. At any time when the pebble is on a node $v$ labeled $x_i$, the pebble is moved to the (unique) vertex $u$ that is reached by the edge labeled 1 if $x_i = 1$ (or by the edge labeled 0 if $x_i = 0$). If the

pebble eventually reaches the sink labeled $b$, then $f(x) = b$. Details and background on branching programs can be found in a standard text, such as the one by Vollmer [Vol99]. The *size* of a branching program is the number of nodes in the graph.

**Definition 5** *Let $U_1$ be a deterministic Turing machine, and let $U_2$ be an alternating Turing machine.*

$$\begin{aligned}
\mathrm{KB}_{U_1}(x) &= \min\{|d| + 2^s : \forall b \in \{0,1,*\} \\
&\qquad \forall i \leq n+1\ U_1^d(i,b) \textit{ runs in} \\
&\qquad \textit{space } s \textit{ and accepts iff } x_i = b\} \\
\mathrm{KF}_{U_2}(x) &= \min\{|d| + 2^t : \forall b \in \{0,1,*\} \\
&\qquad \forall i \leq n+1\ U_2^d(i,b) \textit{ runs in} \\
&\qquad \textit{time } t \textit{ and accepts iff } x_i = b\}
\end{aligned}$$

*Here, we say that $x_i = *$ if $i > |x|$.*

Similarly to Kt and KT, $\mathrm{KB}_{U_1}$ and $\mathrm{KF}_{U_2}$ are almost invariant under the choice of $U_1$ and $U_2$. More precisely, if $U_1$ is a deterministic universal machine and $U_1'$ is any other deterministic machine then for some constant $c > 0$, for all $x$, $\mathrm{KB}_{U_1}(x) \leq (\mathrm{KB}_{U_1'}(x))^c$. Furthermore, if $U_2$ is a fast universal alternating Turing machine and $U_2'$ is any other alternating machine then there exists a constant $c > 0$ such that for all $x$, $\mathrm{KF}_{U_2}(x) \leq (\mathrm{KF}_{U_2'}(x))^c$. Thus for the rest of the paper we will fix one universal machine $U_1$ and one fast universal alternating machine $U_2$ and measure KB and KF with respect to them.

The following simple proposition shows the relationship between KB and branching program size, and between KF and Boolean formula size.

**Proposition 6** *There exists a constant $c \geq 1$ such that for any string $x$ of length $2^n$ representing the truth table of a function $f$, if* BPSIZE*$(x)$ denotes the size of the smallest branching program computing $f$, and* FSIZE*$(x)$ denotes the size of the smallest Boolean formula representing $f$ then*

1. $(\mathrm{KB}(x))^{1/c} \leq \mathrm{BPSIZE}(x) + \log|x| \leq (\mathrm{KB}(x) + \log|x|)^c$*; and*
2. $(\mathrm{KF}(x))^{1/c} \leq \mathrm{FSIZE}(x) + \log|x| \leq (\mathrm{KF}(x) + \log|x|)^c$.

*Proof.* To prove the first inequality consider a deterministic machine $U_1$ that takes as its finite oracle $d$ a string $1^m 0 w$, where $m$ is a positive integer and $w$ is an encoding of a branching program in variables $x_1, \ldots, x_m$. $U_1$ assumes some natural encoding of branching programs where a branching program of size $S$ is represented by a string $w$ of length $S(2\lceil \log S \rceil + \lceil \log m \rceil)$. There are such encodings. On input $(i,b)$, $U_1^d(i,b)$ first checks whether $i$ is a string of length at most $m+1$ representing an integer between 1 and $2^m$. If not, it accepts iff $b = *$ otherwise it evaluates the branching program represented by $w$ on the string $z$ that is the $m$-bit string representing $i-1$, and $U_1$ accepts iff the output of the branching program equals to $b$. Clearly $U_1$ can be implemented to run in space $O(\log|w| + \log m)$. If $x$ is a $2^m$-bit binary string that represents a Boolean function computable by a branching program of size $S$ then $\mathrm{KB}_{U_1}(x) \leq m + S(2\lceil \log S \rceil + \lceil \log m \rceil) + 2^{O(\log|w| + \log m)} \leq (mS)^{c'}$ for some constant $c'$ depending only on $U_1$. The first inequality follows because KB is measured with respect to some fixed *universal* Turing machine.

We prove now the second inequality. Let $U_1$ be the fixed universal reference Turing machine for KB. We can construct another machine $U_1'$ that on input $i \in \{0,1\}^*$ with oracle $d$ will simulate the computation of $U_1^d(i', 1)$

where $i'$ is the integer binary represented by $i$ incremented by 1. If $|i| = m$ then $U_1'$ will require only $O(\log m)$ extra bits of memory for the simulation. For a string $x$ of length $2^m$, let $d$ and $s$ be such that $|d|+2^s = KB(x)$ and for all $1 \leq i \leq n$, $U_1^d(i,1)$ runs in space $s$ and accepts iff $x_i = 1$. Clearly, if $f_x$ is the Boolean function represented by $x$ then $U_1'^d(i)$ accepts iff $f_x(i) = 1$. $U_1'$ with oracle $d$ on any $m$-bit input $i$ runs in space $s + O(\log m)$ so there are at most $2^{O(s+\log m)}$ distinct reachable configurations on inputs of size $m$ with oracle $d$ where the constant in big-$O$ depends only on $U_1$. By the usual technique the graph of its configurations can be turned into a branching program of size $2^{O(s+\log m)}$ where $d$ will be hardwired into the program. Thus, $\text{BPSIZE}(x) \leq (\text{KB} + \log|x|)^c$ for some constant $c$ depending only on $U_1$.

Next we provide the main idea of how to prove the third inequality. Let $x$ be a string of length $2^m$ representing a Boolean function $f_x$ computable by a Boolean formula of size $k$. Using usual techniques the formula can be turned into a balanced formula of size at most $k^4$. Furthermore, by replication of subformulas it can be turned into a completely balanced formula with alternating AND and OR gates of size at most $k^8$. Thus, the formula is completely uniform now and it is fully determined by the sequence of literals (variables and negated variables) accessed at the leaves. This sequence suitably encoded into a string of length $k^8 \cdot 2m$ will be part of the advice $d$ to an alternating machine $U_2$ that we construct. The machine $U_2$ with advice $d = 0^\ell 1^m 0 w$ on input $(i,1)$ (assuming $i$ is of length at most $m+1$) will spend $O(\ell)$ time reading the first $\ell$ bits of $d$, alternating between universal and existential states for $\ell$ alternations (so that its alternation tree mimics the depth $\ell$ balanced formula), then by reading $w$ it will determine which literal the formula should read in the leaf corresponding to the path from the root of the formula given by the path in the alternating tree. It reads the bit of $i-1$ corresponding to this literal, and accepts if the literal evaluates to 1. On input $(i,0)$ it computes similarly but existential and universal states are interchanged and each computation accepts if the corresponding literal evaluates to 0. On input $(i,*)$ it accepts iff $i = 2^m + 1$ which can be decided in alternating time $O(\log|d|)$. Few more details need to be explained. $U_2$ checks in parallel that $i$ has the right size, which takes alternating time at most $O(\log|d|)$. Furthermore, the input to the formula should be the $m$-bit binary representation of $i-1$. Any particular bit of that representation can be determined from $i$ in alternating time $O(\log m)$. Hence, $U_2$ works in time that is linear in the depth of the formula and $\log m$, i.e., in time $O(\log \text{FSIZE}(x) + \log\log|x|)$. By the choice of the fixed fast universal reference Turing machine for KF we conclude that $\text{KF}(x) \leq (\text{FSIZE}(x) + \log|x|)^c$ for some constant $c$.

The last inequality follows easily, by constructing a Boolean formula that simulates the computation of an alternating Turing machine [Ruz81]. □

## 2.4 The Nondeterministic Measures KNT and KNt

In the preceding section, we saw that a variant of Kolmogorov complexity defined using *alternating* universal machines captures certain aspects of Boolean formula size. In this subsection, we investigate similar measures defined using *nondeterministic* machines. By doing so, we will find a natural complete set for NEXP/poly, and we will see how to use the tools of Kolmogorov complexity to provide a new perspective on the techniques that have been developed to derandomize nondeterministic classes such as AM [MV99, SU05].

Here are the nondeterministic variants of Kt and KT:

**Definition 7** *Let $U$ be a nondeterministic Turing machine.*

$$\begin{aligned}
\text{KNt}_U(x) &= \min\{|d| + \log t \,:\, \forall b \in \{0,1,*\} \\
&\qquad \forall i \leq n+1\ U^d(i,b) \textit{ runs in} \\
&\qquad \textit{time } t \textit{ and accepts iff } x_i = b\} \\
\text{KNT}_U(x) &= \min\{|d| + t \,:\, \forall b \in \{0,1,*\}
\end{aligned}$$

$\forall i \leq n+1\ U^d(i,b)$ *runs in*
*time* $t$ *and accepts iff* $x_i = b\}$

As in the definition for Kt and KT, any fast universal machine $U$ has the property that for all $U'$ there is some constant $c > 0$ such that for any $x$, we have $\text{KNt}_U(x) \leq \text{KNt}_{U'}(x) + c$ and $\text{KNT}_U(x) \leq c \cdot \text{KNT}_{U'}(x)$.

In precisely the same way that $\text{KT}(x)$ is polynomially related to the size of (deterministic) circuits computing the function whose truth table is given by $x$, KNT is polynomially related to *strong nondeterministic* circuit size. We recall for the reader the definitions of nondeterministic and strong nondeterministic circuits:

**Definition 8** *A* nondeterministic Boolean circuit $C$ *contains, in addition to* AND, OR, *and* NOT *gates, choice gates of fan-in* $0$*. The circuit evaluates to* $1$ *on an input* $x$*, and we say that* $C(x) = 1$*, if there is some assignment of truth values to the choice-gates that makes the circuit evaluate to* $1$*. A co-nondeterministic circuit* $C$ *is defined similarly: the circuit evaluates to* $1$ *on an input* $x$*, and we say that* $C(x) = 1$*, if every assignment of truth values to the choice-gates makes the circuit evaluate to* $1$*. Otherwise* $C(x) = 0$*.*

*Similarly, a* strong nondeterministic circuit $C$ *computing a function* $f$ *has, in addition to its usual output, an extra output bit, called the flag. For any input* $x$*, and any setting of the choice-gates, if the flag is on, the circuit should output the correct value of* $f(x)$*. Furthermore, for any* $x$*, there should be some setting of the choice-gates that turns the flag on. It is easy to see that a Boolean function* $f$ *has a strong nondeterministic circuit of size* $O(s(n))$ *if and only if* $f$ *has a nondeterministic circuit of size* $O(s(n))$ *and a co-nondeterministic circuit of size* $O(s(n))$*.*

**Proposition 9** *Let* StrongSIZE*(*$x$*) denote the size of the smallest strong nondeterministic circuit computing the function with truth table* $x$*. There is a constant* $c > 1$ *such that for any string* $x$ *of length* $2^m$

$$(1/c)\text{KNT}(x)^{1/3} \leq \text{StrongSIZE}(x) + \log|x| \leq c(\text{KNT}(x) + \log|x|)^3.$$

*Proof.* To prove the first inequality we first design the following machine $U_1$. Machine $U_1$ takes as its oracle a string $d = 1^m 0 w$, where $w$ is a description of a strong nondeterministic circuit with inputs $x_1, \ldots, x_m$. Machine $U_1$ with oracle $d$ on input $(i, b)$ first checks whether $i$ represents an integer between $1$ and $2^m$. If not then it accepts iff $b = *$, otherwise it simulates $w$ on the string $z$ that is the $m$-bit binary representation of $i - 1$ (with the choice gates set nondeterministically). If the output flag of the circuit is set and the output equals $b$ then $U_1$ accepts $(i, b)$. Clearly, the machine $U_1$ uses its oracle $d = 1^m 0 w$ as a description of the function computed by $w$. It is easy to see that one can choose an encoding of circuits such that a circuit of size $S$ is encoded into $w$ of size $O(S(\log S + \log m))$ and such that $U_1$ works in time $O(|d| + |i|)^2$. Hence, for some $c' > 1$ depending only on $U_1$, $\text{KNT}_{U_1}(x) \leq c'(\text{StrongSIZE}(x) + \log|x|)^3$. The first inequality follows by the properties of the *fast* universal Turing machine in the definition of KNT.

Next we argue the correctness of the second inequality. Let $U_2$ be the fast universal Turing machine from the definition of KNT. By the Cook-Levin Theorem, for any integers $m, t \geq 1$, there is a nondeterministic circuit $C_{m,t}$ of size $O((t+m)^3)$ such that on any input $d \in \{0,1\}^t$, $i \in \{0,1\}^m$ and $b \in \{0,1\}$, $C_{m,t}(d, i, b)$ evaluates to 1 iff $U_2^d(i+1, b)$ accepts in time at most $t$. Here, $i$ is interpreted as an integer between $0$ and $2^m - 1$. For $x$ of length $2^m$, let $d \in \{0,1\}^*$ and integer $t \geq 1$ be such that $|d| + t = \text{KNT}(x)$ and for all integers $i \leq 2^m$, $U_2^d(i, b)$ accepts in time $t$ iff $x_i = b$. The strongly nondeterministic circuit for the function $f$ represented by $x$ consists of two copies of $C_{m,t}$: $C_{m,t}(d, i, 1)$ and $C_{m,t}(d, i, 0)$, the output is given by the output of $C_{m,t}(d, i, 1)$ and the flag by $C_{m,t}(d, i, 1) \vee C_{m,t}(d, i, 0)$. Here, $d$ is hardwired and $i$ is the input to the circuit. □

There is a close connection between KNt complexity and circuit complexity, too. Namely, $\text{KNt}(x)$ is polynomially related to oracle circuit size, on circuits that have oracles for a set that is complete for NEXP. This follows from Theorem 4, when combined with the following theorem.

**Theorem 10** *If $A$ is a set complete for* NE *under many-one linear-time reductions then there is a constant $c > 1$ such that for any $x$,*

$$(1/c)\text{KNt}(x) \leq \text{KT}^A(x) \leq c(\text{KNt}(x) + \log|x|)\log|x|.$$

*Proof.* Let us prove the first inequality. Let $U_1$ be a universal Turing machine used to measure $\text{KT}^A$, the KT complexity where the universal machine has access to the oracle for $A$. Since $A$ is in NE, by Theorem 21, $\overline{A}$ is in NE/lin. We can construct a machine $U_1'$ that simulates machine $U_1$ but instead of asking queries to $A$ it nondeterministically evaluates the queries by itself. Assuming that $U_1'$ is provided with the proper advice for $\overline{A}$, the machine $U_1'$ will simulate $U_1$ in nondeterministic exponential time. More precisely, let $d \in \{0,1\}^*$, $b \in \{0,1\}$ and $t \in \mathbb{N}$. There exists $w \in \{0,1\}^{t+1}$ such that for any $i \in \{0,1\}^*$ if $U_1$ with oracles $A$ and $d$ on input $(i,b)$ runs in time $t$, then $U_1'$ with oracle $1^{|w|}0wd$ on input $(i,b)$ runs in time $t \cdot 2^{O(t)}$ and accepts iff $U_1$ accepts with oracle $A$. The constant in the big-$O$ of the running time depends only on $U_1$ and $A$. Thus if $\text{KT}^A(x) = |d| + t$ then $\text{KNt}_{U_1'}(x) \leq |d| + O(t)$. Since KNt is measured relative to a *fast* universal Turing machine, the first inequality follows.

For the second inequality, let $U_2$ be the fixed universal machine relative to which we measure KNt. Consider the language $L = \{(d, 1^t, i, b) : \ d, i \in \{0,1\}^*;\ b \in \{0,1\};\ t \in \mathbb{N}$; machine $U_2$ with oracle $d$ accepts $(i,b)$ in time $2^{t+|d|+|i|}\}$. We construct a machine $M$ that with oracle $A$ and $1^t0d$ on input $(i,b)$, where $d, i \in \{0,1\}^*$, $b \in \{0,1\}$, $t \in \mathbb{N}$, checks whether $(d, 1^t, i, b)$ is in $L$. Since $L$ is reducible to $A$ in linear time, $M$ with oracle $A$ and $1^t0d$ on input $(i,b)$ works in time linear in $|d| + t + |i|$. Hence, for any $x$ of length $2^m$, $\text{KT}^A_M(x) \leq c(\text{KNt}(x) + \log|x|)$ for some constant $c$ that depends only on $U_2$ and $A$. The inequality follows by the choice of universal machine for KT. □

# 3 How Hard Is It to Compute These Measures?

## 3.1 Review of lower bounds for Kt, KS, and KT

In this section, we briefly review some relevant facts about the complexity of the sets of strings with high resource-bounded Kolmogorov complexity. First, let us present a definition that will make precise what we mean by "sets of strings with high resource-bounded Kolmogorov complexity".

**Definition 11** *For any Kolmogorov complexity measure $\text{K}\mu$, define $R_{\text{K}\mu}$ to be the set $\{x : \text{K}\mu(x) \geq |x|\}$.*

We remark that our theorems are not very sensitive to this threshold of "randomness". Every theorem that we state regarding $R_{\text{K}\mu}$ carries over to the set $\{x : \text{K}\mu(x) \geq |x|^\epsilon\}$, for any fixed $\epsilon > 0$.

The sets $R_{\text{Ks}}, R_{\text{Kt}}$ and $R_{\text{KS}}$ are complete for EXPSPACE, EXP and PSPACE, respectively, under P/poly reductions [ABK+06]. These hardness results both follow from a very general theorem (Theorem 14 below) that shows how to reduce *any* "PSPACE-robust" set $A$ to any set that contains many strings but has no strings of low $\text{KT}^A$-complexity.

**Definition 12** *A set $A$ is* PSPACE-robust *if* $\text{P}^A = \text{PSPACE}^A$.

**Definition 13** *A set $A$ is said to have* polynomial density *if $A$ contains at least $2^n/n^k$ strings of every length $n$, for some integer $k$.*

**Remark:** The complete sets of most of the familiar "large" complexity classes (such as $\mathrm{PSPACE}$, EXP and larger time and space complexity classes) are easily seen to be $\mathrm{PSPACE}$-robust.

**Theorem 14** *[ABK$^+$06, Theorem 31] Let $A$ be any* PSPACE*-robust set. Let $L$ have polynomial density, such that for every $x \in L$, $\mathrm{KT}^A(x) > |x|^\gamma$ for some constant $\gamma > 0$. Then $A$ is reducible to $L$ via $\leq_{\mathrm{tt}}^{\mathrm{P/poly}}$ reductions.*

The general idea behind the proof of Theorem 14 is quite simple, once one has some basic tools of derandomization at one's disposal. In particular, Babai, Fortnow, Nisan, and Wigderson [BFNW93] developed a pseudorandom generator that allows one to build, from any $\mathrm{PSPACE}$-robust set $A$, a pseudorandom generator that takes input of length $n^\epsilon$ and produces output of length $n$, with the property that, if $T$ is any statistical test that can distinguish the output of the pseudorandom generator from truly random inputs, it must be the case that $A \leq_{\mathrm{tt}}^{\mathrm{P/poly}} T$. The theorem follows, since the output of the pseudorandom generator has low $\mathrm{KT}^A$ complexity, and thus any set that contains many strings but has no strings of low $\mathrm{KT}^A$ complexity is a good statistical test. We review some other aspects of this reduction later in this paper, in the proof of Theorem 49.

The set $R_{\mathrm{KT}}$ is in coNP and is not known to be complete for any interesting complexity class. However, no one-way function is cryptographically secure relative to $R_{\mathrm{KT}}$ [ABK$^+$06]. Again, the general idea of the proof is quite simple, once some important tools from cryptography are in hand: Any cryptographically-secure one-way function can be used to construct pseudorandom function generators [HILL99, GGM86, RR97]. The functions produced by pseudorandom function generators have low KT-complexity, and thus an oracle for $R_{\mathrm{KT}}$ allows one to crack any pseudorandom function generator, which in turn provides the power to invert any one-way function on a significant fraction of the inputs.

Somewhat stronger results were shown for specific examples of problems from cryptography (such as factoring and computing discrete logs). These problems were shown to be BPP-reducible to $R_{\mathrm{KT}}$ [ABK$^+$06].

Although these completeness and hardness results provide useful information about the complexity of $R_{\mathrm{KT}}, R_{\mathrm{KS}}$, and $R_{\mathrm{Kt}}$, we have disappointingly few *unconditional* lower bounds on their complexity. It is known that none of these problems are in (nonuniform) $\mathrm{AC}^0$ [ABK$^+$06], but we have absolutely no stronger lower bounds. Even the most complex of these three sets, $R_{\mathrm{Kt}}$, which is hard for EXP under P/poly reductions, might conceivably be recognized by linear-size depth three Dlogtime-uniform circuits of AND, OR, and PARITY gates!

## 3.2 The Complexity of $R_{\mathrm{KF}}$ and $R_{\mathrm{KB}}$

For each of the two measures KF and KB, the sets of random strings $R_{\mathrm{KB}}$ and $R_{\mathrm{KF}}$ lie in coNP. Can we prove better upper bounds on their complexity? Can we prove any intractability results?

As discussed in Section 3.1, related questions have been posed about the set $R_{\mathrm{KT}}$, and Kabanets and Cai posed similar questions earlier for the related Minimum Circuit Size Problem (MCSP) [KC00]. Although we are not able to reduce the factorization problem to $R_{\mathrm{KB}}$ and $R_{\mathrm{KF}}$ (as was accomplished for $R_{\mathrm{KT}}$ in [ABK$^+$06]), we can come close.

In this section we prove that factoring Blum Integers can be done in $\mathrm{ZPP}^{R_{\mathrm{KF}}}$ and $\mathrm{ZPP}^{R_{\mathrm{KB}}}$. (For an oracle $A$, a function $f$ is in $\mathrm{ZPP}^A$ if there exists a procedure computed by a probabilistic oracle machine with oracle $A$ that on input $x$, on every halting path, produces $f(x)$, and the expected running time is polynomial.) We use results of [NR97] and [BBR99] in order to accomplish this. We define the following computational problem.

*Blum Integer Factorization*: Given a Blum Integer $N \in \mathbb{N}$, find the primes $P$ and $Q$ such that $1 < P \leq Q$ and $N = PQ$. (A $2n$-bit integer $N$ is called a *Blum Integer* if $N = PQ$, where $P$ and $Q$ are two primes such that $P \equiv Q \equiv 3 \bmod 4$.)

**Theorem 15** *Blum Integer Factorization is in* $\mathrm{ZPP}^{R_{\mathrm{KF}}} \cap \mathrm{ZPP}^{R_{\mathrm{KB}}}$*, i.e., there are* $\mathrm{ZPP}^{R_{\mathrm{KF}}}$ *and* $\mathrm{ZPP}^{R_{\mathrm{KB}}}$ *procedures that on input* $N$ *that is a Blum Integer produce factors* $P$ *and* $Q$ *of* $N$*.*

*Proof.* Naor and Reingold construct a pseudo-random function ensemble $\{f_{N,r}(x) : \{0,1\}^n \to \{0,1\}\}_{N,r}$ with the following two properties (Construction 5.2 and Corollary 5.6 of [NR97]):

1. There is a $\mathrm{TC}^0$ circuit computing $f_{N,r}(x)$, given $2n$-bit integer $N$, $4n^2+2n$-bit string $r$ and $n$-bit string $x$.
2. For every probabilistic oracle Turing machine $M$, that on its $2n$-bit input asks queries of length only $n$, and any constant $\alpha > 0$, there is a probabilistic Turing machine $\mathcal{A}$, such that for any $2n$-bit Blum Integer $N = PQ$, if
$$|\Pr[M^{f_{N,r}}(N) = 1] - \Pr[M^{R_n}(N) = 1]| > 1/n^{\alpha}$$
where $R_n = \{g : \{0,1\}^n \to \{0,1\}\}_n$ is a uniformly distributed random function ensemble and the probability is taken over the random string $r$ and the random bits of $M$, then $\Pr[\mathcal{A}(N) \in \{P,Q\}] > 1/n$.

Their factoring construction relativizes, i.e., the properties of $\{f_{N,r}(x)\}_{N,r}$ hold even if $M$ and $\mathcal{A}$ have an access to the same auxiliary oracle.

Let $f_{N,r}(x)$ be computable by a $\mathrm{TC}^0$ circuit of size $n^{c'}$, and hence, by an $\mathrm{NC}^1$ circuit of size $n^{c''}$, for some constants $c', c'' > 1$. Let $x_1, x_2, \ldots, x_{2^n}$ denote strings in $\{0,1\}^n$ under lexicographical ordering. Clearly, there is a constant $c > 1$, such that for all large enough $n$, all $2n$-bit integers $N$ and all $4n^2+2n$-bit strings $r$, the string obtained by concatenating $f_{N,r}(x_1), f_{N,r}(x_2), \ldots, f_{N,r}(x_{n^c})$ has KF-complexity less than $n^c/2$. Fix such a $c$ and consider the following oracle Turing machine $M$ with oracles $R_{\mathrm{KF}}$ and a function $g$:

- On $2n$-bit input $N$, $M$ asks oracle $g$ queries $x_1, x_2, \ldots, x_{n^c}$ to get answers $y_1, y_2, \ldots, y_{n^c}$. Then, $M$ accepts if $y_1 y_2 \cdots y_{n^c} \in R_{\mathrm{KF}}$ and rejects otherwise.

It is easy to see that if $g \in \{f_{N,r}(x)\}_{N,r}$ then $M$ always rejects, for $n$ large enough. On the other hand, if $g$ is taken uniformly at random from $R_n$, then $y_1 y_2 \cdots y_{n^c}$ is a random string and the probability that $M$ accepts is at least $1 - 2^{-n/2}$. Hence, $|\Pr[M^{f_{N,r}(x)}(N) = 1] - \Pr[M^{R_n}(N) = 1]| > 1/2$, for $n$ large enough. By the properties of $f_{N,r}(x)$ we can conclude that there is a probabilistic Turing machine $\mathcal{A}$ with oracle $R_{\mathrm{KF}}$ that factors $N$ with non-negligible probability. We can reduce the error to zero by verifying the output of $\mathcal{A}$.

Since any function that is computable by $\mathrm{NC}^1$ circuits is computable by branching programs of polynomial size, by considering branching programs instead of $\mathrm{NC}^1$ circuits we get that Blum Integer Factorization is in $\mathrm{ZPP}^{R_{\mathrm{KB}}}$. □

We close off this section with the only unconditional lower bound that we have on $R_{\mathrm{KF}}$ and $R_{\mathrm{KB}}$.

**Proposition 16** *None of the sets* $R_{\mathrm{Kt}}$, $R_{\mathrm{KT}}$, $R_{\mathrm{KS}}$, $R_{\mathrm{KB}}$ *and* $R_{\mathrm{KF}}$ *are in* $\mathrm{AC}^0$.

This proposition follows from the proof of [ABK+06, Corollary 22], and also carries over to the other Kolmogorov measures $\mathrm{K}\mu$ discussed elsewhere in the paper.

## 3.3 Hardness of Approximation

Many computational problems that complexity theory studies are decision problems for which an answer is always either "yes" or "no". Other problems that are of interest in computational complexity are optimization problems.

Examples of optimization problems are the Maximum Clique — what is the size of the largest clique in $G$ — and the Minimum Circuit Size Problem — what is the size of the smallest circuit computing a Boolean function $f$ given by its truth table?

For some optimization problems efficient (polynomial time) algorithms are known. For others, no efficient algorithm is known. Moreover, it is known that some optimization problems are hard for NP. Given that the exact solution of such an optimization problem may be hard to find one can try to find at least an approximation to the solution. Many optimization problems are known for which even finding an approximation cannot be done efficiently, unless something unlikely is true, such as $\mathrm{P} = \mathrm{NP}$. For example, [Hås99] shows that the Maximum Clique cannot be approximated up to factor $n^{1-\epsilon}$ in polynomial time, unless $\mathrm{P} = \mathrm{NP}$.

In this section we study the following optimization problems — given a truth table of a function $f$, what is the smallest size of a circuit, a branching program or a formula, respectively, that computes $f$. We show that under certain plausible complexity assumptions these optimization problems are hard to approximate.

Related questions about approximating the size of the smallest $\mathrm{AC}^0$ circuits for a given truth table were investigated in [AHM+08]. For the seemingly even more restrictive problem of approximating the size of the smallest DNF formula that represents a given truth table, approximating the size is known to be hard for NP [Fel09, AHM+08].

For a minimization problem $f : \Sigma^* \to \mathbb{N}$ we say that $g : \Sigma^* \to \mathbb{N}$ *approximates $f$ up to factor* $r : \mathbb{N} \to \mathbb{N}$ if for all $x \in \Sigma^*$, $1 \leq g(x)/f(x) \leq r(|x|)$. For a complexity class $\mathcal{C}$ we say that $f$ *cannot be approximated up to factor $r$ in $\mathcal{C}$* if no $g \in \mathcal{C}$ approximates $f$ up to factor $r$.

We recall definitions of two more problems that are believed to be computationally difficult.

*Integer Factorization*: Given a composite integer $N \in \mathbb{N}$, find two integers $P$ and $Q$ such that $1 < P \leq Q$ and $N = PQ$.

*Discrete Logarithm*: Given three integers $x, z, N$, $1 \leq x, z < N$, find an $i$ such that $x = z^i \bmod N$ if such $i$ exists.

The following result is implicit in [ABK+06]:

**Theorem 17** *Let $0 < \gamma < 1$ be a constant and $B$ be a set of at least polynomial density such that for any $x \in B$,* $\mathrm{SIZE}(x) > |x|^\gamma$. *Then Integer Factorization and Discrete Logarithm are in* $\mathrm{BPP}^B$.

This theorem implies the non-approximability of circuit size.

**Theorem 18** *For any $0 < \epsilon < 1$,* $\mathrm{SIZE}(x)$ *cannot be approximated up to factor $n^{1-\epsilon}$ in* BPP, *unless Integer Factorization and Discrete Logarithm is in* BPP.

*Proof.* Assume that for some $0 < \epsilon < 1$, there is a function $g \in \mathrm{BPP}$ that approximates $\mathrm{SIZE}(x)$ up to factor $n^{1-\epsilon}$. We will show that this implies that Integer Factorization and Discrete Logarithm are in BPP.

Consider the set $B = \{x \in \{0,1\}^*;\ g(x) > |x|^{1-\epsilon/2}\}$. Clearly, $B \in \mathrm{BPP}$. Since for all $x \in \{0,1\}^*$, $1 \leq g(x)/\mathrm{SIZE}(x) \leq n^{1-\epsilon}$, we have that for all $x \in B$, $\mathrm{SIZE}(x) > |x|^{\epsilon/2}$ and also for all $x \in \{0,1\}^*$, if $\mathrm{SIZE}(x) \geq |x|^{1-\epsilon/2}$ then $x \in B$. By [Lup59], almost all truth tables $x \in \{0,1\}^*$ require circuits of size at least $O(n/\log n)$. Hence, $B$ is of at least polynomial density. By Theorem 17, Integer Factorization and Discrete Logarithm are in $\mathrm{BPP}^{\mathrm{BPP}} \subseteq \mathrm{BPP}$. (In the case of Integer Factorization we can actually verify correctness of the result to get ZPP computation instead of BPP.) □

Similar non-approximability results can be obtained for formula and branching program sizes. A proof similar to the proof of Theorem 15 yields the following claim.

**Theorem 19** *Let $0 < \gamma < 1$ be a constant and $B$ be a set of at least polynomial density such that for any $x \in B$, $\mathrm{BPSIZE}(x) > |x|^\gamma$ or for any $x \in B$, $\mathrm{FSIZE}(x) > |x|^\gamma$. Then there is a $\mathrm{ZPP}^B$ procedure that on input $N$ that is a Blum Integer produces factors $P$ and $Q$ of $N$.*

As a corollary to this theorem we obtain:

**Theorem 20** *For any $0 < \epsilon < 1$, $\mathrm{BPSIZE}(x)$ and $\mathrm{FSIZE}(x)$ cannot be approximated up to factor $n^{1-\epsilon}$ in* BPP*, unless Blum Integer Factorization is in* ZPP.

In Theorems 18 and 20, a function $f$ is computable in BPP if there is a polynomial time probabilistic machine $M$ such that for any $x$, $\Pr[M(x) = f(x)] \geq 2/3$. However, the results hold for an even stronger notion of non-approximability: For any $0 < \epsilon < 1$, if there is a polynomial time probabilistic machine $M$ such that for all $x$, $\Pr[1 \leq M(x)/\mathrm{BPSIZE}(x) \leq n^{1-\epsilon}] \geq 2/3$ or $\Pr[1 \leq M(x)/\mathrm{FSIZE}(x) \leq n^{1-\epsilon}] \geq 2/3$ then Blum Integer Factorization is in ZPP. Similarly, if there is a polynomial time probabilistic machine $M$ such that for all $x$, $\Pr[1 \leq M(x)/\mathrm{SIZE}(x) \leq n^{1-\epsilon}] \geq 2/3$ then Integer Factorization and Discrete Logarithm are in BPP. These results follow by essentially the same proofs as Theorems 18 and 19; one has only to observe that the derandomization results that we use hold not only relative to *oracles* that distinguish between random and pseudorandom strings but also relative to *probabilistic procedures* that distinguish between random and pseudorandom strings with non-negligible probability.

## 3.4 The Complexity of $R_{\mathrm{KNt}}$

In this subsection, we prove our main results regarding KNt complexity. We prove an upper bound, showing that $R_{\mathrm{KNt}}$ is in $\mathrm{P}^{\mathrm{NEXP}}$ (and observe that this class is contained in NEXP/poly) and we show that $R_{\mathrm{KNt}}$ is complete for NEXP/poly under P/poly reductions. Furthermore, we present an unconditional lower bound, showing that $R_{\mathrm{KNt}}$ is not in NP $\cap$ co-NP. (This result presents a stark contrast to what we are able to prove about $R_{\mathrm{Kt}}$, which is still not known to lie outside of P, or even outside of Dlogtime-uniform $\mathrm{AC}^0[2]$.)

### 3.4.1 Properties of NE, NEXP, and $\mathrm{P}^{\mathrm{NEXP}}$

Before we can present our results about $R_{\mathrm{KNt}}$, it is necessary to present some fundamental facts about the complexity classes that are most closely related to $R_{\mathrm{KNt}}$.

In this paper, we will need to refer both to NE $=$ NTime$(2^{O(n)})$ and NEXP $=$ NTime$(2^{n^{O(1)}})$, as well as their deterministic counterparts E $=$ DTime$(2^{O(n)})$ and EXP $=$ DTime$(2^{n^{O(1)}})$. We will also have occasion to refer to the class FewEXP, which is defined as the class of languages accepted by NEXP machines that have no more than $2^{n^{O(1)}}$ accepting computations on inputs of length $n$. FewE is defined similarly in terms of NE machines that have no more than $2^{O(n)}$ accepting computations on inputs of length $n$.

The following theorem is a well-known "folklore" theorem, although the only citation we know to give is to Fortnow's Computational Complexity weblog [For04, vM03].

**Theorem 21** *(Folklore)* NE/lin $=$ coNE/lin.

*Proof.* It suffices to show that coNE $\subseteq$ NE/lin. Let $A \in$ coNE, where $M$ is an NE machine accepting the complement of $A$. Define $a_n$ to be the advice string for length $n$, where $a_n$ is the binary encoding of the number

of strings of length $n$ in $A$; note that $a_n$ has a linear number of bits. Here is an NE/lin algorithm for $A$: On input $x$ of length $n$, nondeterministically pick $2^n - a_n$ strings of length $n$, and attempt to find an accepting path of $M$ for each of these strings. (If this attempt is unsuccessful, then abort.) At this point, we have a list of *all* of the strings of length $n$ that are *not* in $A$. Accept if and only if $x$ is not on the list. □

**Corollary 22** NEXP/poly = coNEXP/poly.

Similar techniques allow us to show that any NEXP-complete set is PSPACE-robust; recall from Section 3.1 that a set $A$ is PSPACE-robust if $\mathrm{P}^A = \mathrm{PSPACE}^A$. This extends a result of Hemachandra [Hem89], where it is shown that $\mathrm{P}^{\mathrm{NE}} = \mathrm{NP}^{\mathrm{NE}} = \ldots = \Sigma_k^{p,\mathrm{NE}}$. (Note also that $\mathrm{P}^{\mathrm{NE}} = \mathrm{P}^{\mathrm{NEXP}}$, since there are sets that are polynomial-time many-one complete for NEXP in NE.)

**Theorem 23** $\mathrm{P}^{\mathrm{NE}} = \mathrm{PSPACE}^{\mathrm{NE}}$. *In fact,* $\mathrm{P}^{\mathrm{NE}}$ *is also equal to the class* $\mathrm{NEXP}^{\mathrm{NE}}$ *if we restrict the* NEXP *oracle machine to pose queries of length polynomial in the length of the input.*

*Proof.* Let $A$ be accepted by a nondeterministic oracle machine $M$ that runs for time $2^{n^k}$ and asks queries of length at most $n^c$, and has as oracle a set $B \in$ NE. Let $C$ be the set $\{(1^n, m) :$ there are at least $m$ strings of length $\leq n$ in $B\}$. Clearly, $C \in$ NE. Using binary search, a polynomial time machine with access to $C$ (or to an NE-complete oracle) can determine exactly how many strings of length at most $n^c$ are in $B$.

Now consider the set $D$, defined as the set of pairs $(x, m)$ for which there is a set $S \subseteq B$ with exactly $m$ strings of length at most $|x|^c$, such that $M^S(x)$ accepts. $D$ is easily seen to lie in NEXP (simply guess the $m$ strings, guess an accepting computation for each of the strings to verify that it is in $B$, and then simulate the computation of $M(x)$ using the $m$ strings as an oracle). Thus $D$ is reducible in polynomial time to a set in NE.

Thus in $\mathrm{P}^{\mathrm{NE}}$ one can compute the precise value $m$ such that $B$ has $m$ strings of length at most $n^c$, and then find out if $(x, m) \in D$, which is equivalent to $x \in A$. □

**Corollary 24** $\mathrm{P}^{\mathrm{NE}} = \mathrm{PSPACE}^{\mathrm{NE}} = \mathrm{EXP}_{\mathrm{tt}}^{\mathrm{NP}}$ *(where* $\mathrm{EXP}_{\mathrm{tt}}^{\mathrm{NP}}$ *denotes the class of problems accepted by deterministic oracle machines running in time* $2^{n^{O(1)}}$ *with an oracle in* NP*, with the property that the complete list of oracle queries to be posed is composed before the first query is made). This mode of oracle access is known as "nonadaptive" or "truth-table" reducibility; the queries may be of exponential length.*

*Proof.* The first equality is from Theorem 23. The second equality is from [Hem89, Theorem 4.10.2]. □

Combining the techniques of Theorems 23 and 21, we obtain the following equality.

**Theorem 25** $\mathrm{PSPACE}^{\mathrm{NEXP}}$/poly = $\mathrm{P}^{\mathrm{NEXP}}$/poly = NEXP/poly.

*Proof.* Fortnow credits Buhrman with the observation (stated without proof) that $\mathrm{EXP}_{\mathrm{tt}}^{\mathrm{NP}}$ is contained in NEXP/poly [For04]. By Corollary 24, this implies the theorem. For completeness, we give a simple direct proof.

By Theorem 23, it suffices to show that $\mathrm{P}^{\mathrm{NEXP}}$/poly is contained in NEXP/poly. Let $A \in \mathrm{P}^{\mathrm{NEXP}}$ be recognized by an oracle machine $M$ running in time $n^c$ with oracle $B \in$ NEXP. Our NEXP/poly algorithm for $A$ will use an advice sequence, listing for each $m \leq n^c$ the number $a_m$ of strings of length $m$ in $B$. Our NEXP/poly algorithm

will guess $a_m$ strings of length $m$ and guess accepting computations verifying that each of the strings is in $B$. Then it will simulate $M(x)$ using this list of strings as the oracle, accepting if and only if $x \in A$. □

In order to prove our lower bound for $R_{\mathrm{KNt}}$ in Section 3.4.4, we need to establish some conditional collapse results. In particular, we need to show that if NEXP is in (NP ∩ co-NP)/poly, then NEXP = PSPACE. (We prove a stronger result below in Theorem 28.) Vinodchandran proves a related result, showing that if EXP ⊆ (NP ∩ co-NP)/poly, then EXP = AM [Vin04, Theorem 8]. We make use of a stronger hypothesis than this result of Vinodchandran (assuming an upper bound on NEXP instead of EXP), but in order to conclude that NEXP = AM we would first need to argue that, under this assumption, EXP = NEXP. Instead of presenting the argument in that form, we first present the following theorem, which improves Vinodchandran's result (obtaining the conclusion EXP = AM from a weaker hypothesis).

**Theorem 26** EXP ⊆ (AM ∩ coAM)/poly *if and only if* EXP = AM.

*Proof.* The backward implication is trivial (since EXP is closed under complement, and thus if EXP = AM we have EXP = AM ∩ co-AM ⊆ (AM ∩ coAM)/poly).

For the forward implication, we follow the example of the proof that if EXP ⊆ P/poly, then EXP = MA [BFNW93]. That argument proceeds by observing that every problem in EXP has a two-prover interactive proof [BFL91] where, moreover, the strategy of the provers is computable in EXP. Thus if EXP ⊆ P/poly, each problem in EXP can be solved by an MA protocol where Merlin first sends Arthur the circuits computing the provers' strategies, and then Arthur uses the circuits to simulate the rest of the multi-prover interactive proof.

We use the weaker assumption that EXP ⊆ (AM ∩ coAM)/poly. Thus each problem in EXP has an $\mathrm{MA}^{\mathrm{AM} \cap \text{co-AM}}$ protocol, where Merlin sends Arthur the advice sequence used for the (AM ∩ coAM)/poly algorithms for the provers, and then Arthur uses the AM ∩ co-AM oracle to simulate the multi-prover protocol. The result follows since $\mathrm{MA}^{\mathrm{AM} \cap \text{co-AM}} \subseteq \mathrm{AM}^{\mathrm{AM} \cap \text{co-AM}} = \mathrm{AM}$ [Sch89]. □

**Corollary 27** $\mathrm{P}^{\#\mathrm{P}} \subseteq$ (AM ∩ coAM)/poly *if and only if* $\mathrm{P}^{\#\mathrm{P}} = \mathrm{AM}$.
PSPACE ⊆ (AM ∩ coAM)/poly *if and only if* PSPACE = AM.

*Proof.* #P and PSPACE have interactive proofs where the strategy of the prover is computable in #P and PSPACE respectively [LFKN92, Sha92]. The rest of the argument is the same as in Theorem 26. □

The same strategy fails in proving a similar result for NEXP, since the strategies of the provers for the two-prover interactive proofs for NEXP are not known to be computable in NEXP. However, a different strategy succeeds.

**Theorem 28** NEXP ⊆ (AM ∩ coAM)/poly *if and only if* NEXP = AM.

*Proof.* As in the proof of Theorem 26, the backward implication is trivial (since AM ⊆ EXP ⊆ NEXP).

For the forward implication, assume that NEXP ⊆ (AM ∩ coAM)/poly. This clearly implies that NEXP ⊆ $\Sigma_2^p$/poly.

Assume, for the sake of contradiction, that NEXP ≠ EXP. It is known that this implies that AM ⊆ io-NTime$(2^n)/n$ [IKW01] (where this means that, for every problem $A \in$ EXP, there is a problem in NTime$(2^n)/n$ that agrees with $A$ for infinitely many input lengths $n$). There is a problem $B \in$ NE that is hard for NTime$(2^n)$ under *linear-time*

reductions. It follows from the preceding paragraph that there is a constant $c > 0$ and a $\Sigma_2^p$ algorithm running in time $n^c$ with $n^c$ bits of advice, accepting $B$, and thus EXP $\subseteq$ io-$\Sigma_2$time$(n^d)/n^d$ for some $d > 0$. However, a straightforward diagonalization argument in the style of Kannan [Kan82] shows that this inclusion does not hold. (That is, an exponential-time algorithm can simulate the first $n$ $\Sigma_2$time$(n^d)$ algorithms on each of the $2^{n^d}$ advice sequences of length $n^d$ and on each of the lexicographically first $n^{2d}$ strings of length $n$. There must be some function on these $n^{2d}$ strings that differs from each of these $n2^{n^d}$ functions. Select one such function. This defines a function in EXP that is not in io-$\Sigma_2$time$(n^d)/n^d$.)

Thus we can conclude that EXP $=$ NEXP under this assumption. The theorem now follows from Theorem 26. □

For completeness, we mention two more equivalences in the same vein.

**Corollary 29** EXP$^{\mathrm{NP}} \subseteq$ (AM $\cap$ coAM)/poly *if and only if* EXP$^{\mathrm{NP}} =$ AM.
FewEXP $\subseteq$ (AM $\cap$ coAM)/poly *if and only if* FewEXP $=$ AM.

*Proof.* Again, the backward implications are trivial.

If EXP$^{\mathrm{NP}} \subseteq$ (AM $\cap$ coAM)/poly, then clearly EXP$^{\mathrm{NP}} \subseteq$ EXP/poly, which implies that EXP$^{\mathrm{NP}} =$ EXP [BH92]. Thus we have EXP$^{\mathrm{NP}} =$ EXP $\subseteq$ (AM $\cap$ coAM)/poly, which implies EXP$^{\mathrm{NP}} =$ AM by Theorem 26.

If FewEXP $\subseteq$ (AM $\cap$ coAM)/poly, then clearly FewEXP $\subseteq$ EXP/poly, which implies that FewEXP $=$ EXP [AKS95]. Thus we have FewEXP $=$ EXP $\subseteq$ (AM $\cap$ coAM)/poly, which implies FewEXP $=$ AM by Theorem 26. □

### 3.4.2 An Upper Bound for $R_{\mathrm{KNt}}$

**Theorem 30** $R_{\mathrm{KNt}} \in \mathrm{P}^{\mathrm{NE}}$.

*Proof.* By Theorem 23, it suffices to show that $R_{\mathrm{KNt}}$ is in $\mathrm{PSPACE}^{\mathrm{NE}}$. From the definition, it is clear that a string $x$ is *not* in $R_{\mathrm{KNt}}$ if and only if there is a string $d$ of length less than $|x|$ and a time $t < 2^{|x|}$ such that $\forall i \in \{1, \ldots, |x|+1\} \forall b \in \{0, 1, *\}$ $U^d(i, b)$ runs in time $t$ and accepts if and only if $x_i = b$, where $|d|+\log t < |x|$. A PSPACE machine can cycle through each choice of $d$ and $t$ and use an oracle in NE to answer questions about whether the nondeterministic universal machine $U$ accepts the given input in the allotted time. □

It is natural to wonder if there is a better upper bound on the complexity of $R_{\mathrm{KNt}}$. In the next subsection, we show that $R_{\mathrm{KNt}}$ is complete for NEXP/poly under P/poly reductions, which is some evidence that $R_{\mathrm{KNt}}$ cannot be too much easier than $\mathrm{P}^{\mathrm{NE}}$ – but it is actually rather weak evidence, since it is still not proven that NEXP/poly $\neq$ P/poly. Although it might seem that any algorithm determining membership in $R_{\mathrm{KNt}}$ would *have* to solve problems that are hard for both NE and coNE, we know of no unlikely consequences that follow if $R_{\mathrm{KNt}}$ were to lie in NE. In particular, the following proposition shows that it is unlikely that there is a polynomial-time many-one reduction (or even a truth-table reduction) from NE (or coNE) to $R_{\mathrm{KNt}}$.

**Proposition 31** *If there is a unary language in* NEXP $-$ $\mathrm{P}_{tt}^{\mathrm{NP}}$*, then* $R_{\mathrm{KNt}}$ *is not hard for* NEXP *under polynomial-time truth-table reductions.*

*Proof.* Let $A$ be a unary language in NEXP $-$ $\mathrm{P}_{tt}^{\mathrm{NP}}$, and assume that there is a polynomial-time truth-table reduction computed by a machine $M$, reducing $A$ to $R_{\mathrm{KNt}}$. Each query $q$ that is posed by $M$ on input $0^n$ has $\mathrm{Kt}(q) = O(\log n)$. Thus, all queries that do not have length $O(\log n)$ lie outside of $R_{\mathrm{KNt}}$, and hence in order to compute the value of the reduction, it suffices to determine if $q \in R_{\mathrm{KNt}}$ for those queries $q$ of length $O(\log n)$. Since $R_{\mathrm{KNt}} \in \mathrm{P}^{\mathrm{NE}}$ via an algorithm that asks queries of length linear in the input size, each such query of $M$ can be answered by making unary queries to an oracle in NP. This shows that $A \in \mathrm{P}_{tt}^{\mathrm{NP}}$, contrary to assumption. □

The hypothesis to this proposition seems quite plausible. It is known that there are problems that lie in NEXP $-$ $\mathrm{P}_{tt}^{\mathrm{NP}}$ [Moc96, FLZ94], but it does not appear to be known if there are any *unary* languages in this difference.

### 3.4.3 A Completeness Result for $R_{\mathrm{KNt}}$

**Theorem 32** *$R_{\mathrm{KNt}}$ is complete for* NEXP/poly *under* P/poly *reductions.*

*Proof.* It was established in the preceding section that $R_{\mathrm{KNt}} \in$ NEXP/poly. Hardness follows immediately from Theorem 23 (which tells us that any set $A$ that is complete for NE is PSPACE-robust), Theorem 10 (which tells us that the strings in $R_{\mathrm{KNt}}$ have high $\mathrm{KT}^A$ complexity), and Theorem 14 (which tells us that these conditions imply that $A \leq_{\mathrm{tt}}^{\mathrm{P/poly}} R_{\mathrm{KNt}}$). □

We do not know if $R_{\mathrm{KNt}}$ is hard for NEXP under NP reductions. However, we are able to show hardness for some important subclasses of NEXP.

**Theorem 33** $\mathrm{UEXP} \subseteq \mathrm{NP}^{R_{\mathrm{KNt}}}$.

*Proof.* When Babai *et al.* showed that EXP $\in$ P/poly implies EXP $=$ MA [BFNW93], a crucial step involved observing that the strategies of the provers in the MIP protocols for EXP [BFL91] are computable in exponential time. Analysis of the MIP protocols for NEXP [BFL91] reveals that the strategies of the provers for some language $A \in$ NEXP *can* be computed in NEXP if there is a language in NEXP that encodes the bits of an accepting computation path for every string $x \in A$.

Although this condition is not known to hold for every $A \in$ NEXP, it does hold for every $A \in$ UEXP. Let $A$ be accepted by a UEXP machine $M$, and consider the set $B : \{(x, i, b) :$ the unique accepting path of $M$ on input $x$ has $b$ as its $i$th bit$\}$. Clearly, $B \in$ UEXP $\subseteq$ NEXP. Thus the strategy of the provers is in NEXP and hence by Theorem 32 can be computed by an oracle circuit of polynomial size with an oracle for $R_{\mathrm{KNt}}$.

Thus we obtain a $\mathrm{MA}^{R_{\mathrm{KNt}}}$ protocol for $A$: Merlin sends Arthur the oracle circuits $C_1, C_2$ that compute the provers' strategies, and then Arthur uses his probabilistic bits to simulate the MIP protocol, using the circuits $C_1$ and $C_2$ along with the oracle $R_{\mathrm{KNt}}$, to compute the answers provided by the provers in the MIP protocol.

We now appeal to the following lemma:

**Lemma 34** *[ABK+06] Let $C$ be any oracle and $L$ be a set such that $L \in \mathrm{P/poly}^C$ and for every $x \in L, \mathrm{KT}^C(x) > |x|^\gamma$ for some constant $\gamma > 0$. then* $\mathrm{MA}^L = \mathrm{NP}^L$.

Letting $C$ be any set complete for NE, and letting $L$ be $R_{\mathrm{KNt}}$, and appealing to Theorem 10, we see that the hypothesis of the lemma is satisfied. Thus $A \in \mathrm{NP}^{R_{\mathrm{KNt}}}$. □

Building on this proof, we can prove a stronger result.

**Theorem 35** FewEXP $\subseteq$ NP$^{R_{\mathrm{KNt}}}$.

*Proof.* Let $A \in$ FewEXP be accepted by a NEXP machine $M$ that has no more than $2^{n^k}$ accepting computation paths on any input $x$. We appeal to the following well-known hashing theorem:

**Theorem 36** *([FKS82][Lemma 2], [Meh82][Theorem B]) Let $S$ be a set of $2^{O(n^k))}$ numbers, each of whose binary representation has at most $2^{n^k}$ bits. Then there is some prime number $p$ with $O(n^k)$ bits such that for any $x \neq y$ in $S$, $x \not\equiv y (\mathrm{mod}\, p)$.*

Now let $B = \{(x, i, b, p, q) : p$ is a prime number with $O(|x|^k)$ bits, $0 \leq q < p$, and there is an accepting path $y$ of $M$ on input $x$, such that $y$ has $b$ as its $i$th bit and $y \equiv q (\mathrm{mod}\, p)\}$. Clearly, $B \in$ NEXP.

Assume for the moment that $x, p$ and $q$ are such that there is exactly one accepting computation of $M$ on input $x$ (and recall from Theorem 36 that there must always be such a pair $(p, q)$ for any $x \in A$). Then an exponential-time machine $M'$ with an oracle for $B$ can query the strings $(x, i, b, p, q)$ for all $b \in \{0, 1\}$ and all $1 \leq i \leq 2^{n^k}$ and construct an accepting computation path of $M$ on input $x$, and given this information $M'$ can then compute the strategies of the MIP provers to show that $x \in A$, given a good pair $(p, q)$. (We are not concerned with the behavior of $M'$ when given a bad pair $(p, q)$.) Since the queries made by $M'$ are all of length polynomial in $|x|$, it follows from Theorem 23 that the language computed by $M'$ lies in P$^{\mathrm{NE}}$, and hence by Theorems 32 and 25 it lies in P/poly$^{R_{\mathrm{KNt}}}$.

Thus we obtain an MA protocol for $A$, where in the first step Merlin sends Arthur a good pair $(p, q)$ along with the oracle circuits that simulate $M'$ when provided $R_{\mathrm{KNt}}$ as an oracle. The rest of the proof proceeds exactly as in the proof of Theorem 33. □

Our observations about the complexity of the MIP protocols for UEXP and FewEXP also provide us with the following corollary, which is analogous to the theorem that NEXP $\subseteq$ P/poly if and only if NEXP $=$ MA [IKW01].

**Corollary 37** UEXP $\subseteq$ P/poly *if and only if* UEXP $=$ MA
FewEXP $\subseteq$ P/poly *if and only if* FewEXP $=$ MA

*Proof.* In each case, the forward implication follows from our observation that the provers' strategies for MIP protocols for these classes also lie in the same classes. (Namely, Merlin guesses the circuits for the provers' strategies and sends them to Arthur.)

The other implications follow from exactly the same argument given by Impagliazzo, Kabanets, and Wigderson, in proving the analogous implication [IKW01]. □

### 3.4.4 An Unconditional Lower Bound for $R_{\mathrm{KNt}}$

**Theorem 38** $R_{\mathrm{KNt}} \notin$ NP $\cap$ co-NP.

*Proof.* If $R_{\mathrm{KNt}} \in$ NP $\cap$ co-NP, then NEXP $\subseteq$ P$^{R_{\mathrm{KNt}}}$/poly $\subseteq$ (NP $\cap$ co-NP)/poly by Theorem 32. By Theorem 28, this implies NEXP $=$ AM and hence NEXP $=$ PSPACE.

However, it is known that any polynomially-dense set that has no strings of KS-complexity $n^\epsilon$ is hard for PSPACE under ZPP reductions [ABK$^+$06]. Since $R_{\mathrm{KNt}}$ is dense and has no strings of low KS complexity, it follows that

NEXP $=$ PSPACE $\subseteq$ ZPP$^{R_{\mathrm{KNt}}} \subseteq$ ZPP$^{\text{NP} \cap \text{co-NP}} =$ NP $\cap$ co-NP, in contradiction to the nondeterministic time hierarchy theorem. □

# 4 Nondeterministic Kolmogorov Complexity

Earlier work has shown that many of the techniques that have been developed to derandomize BPP can be re-cast in terms of arguments in resource-bounded Kolmogorov complexity [All01, ABK$^+$06]. In this section, we investigate the extent to which a similar program can be carried out to study the techniques that have been developed to derandomize AM. In order to state our results, we first recall some standard definitions.

**Definition 39** *[Lon82] A* StrongNP-procedure *computing a function $f$ is a polynomial time nondeterministic procedure, so that every computation path on input $x$ either produces $f(x)$ or rejects. Furthermore, at least one computation path must produce $f(x)$.*

We will also refer to functions computable in SNP/log. For this, we assume that, for each input length $n$, there is an advice string $a_n$ of length $O(\log n)$, and a nondeterministic machine as above that produces $f(x)$ on every non-rejecting computation path on input $(x, a_{|x|})$. We place no restrictions on the behavior of the nondeterministic machine on inputs $(x, z)$ where $z \neq a_{|x|}$.

**Definition 40** *[ACR98] A* hitting set generator *for a class of circuits $\mathcal{C}$ and threshold $\alpha$ is a procedure $G$ that maps $0^n$ to a set $H_n$ of polynomial size with the property that, for every circuit in $\mathcal{C}$ on $n$ inputs that accepts at least $\alpha 2^n$ strings in $\Sigma^n$, the circuit accepts an element of $H_n$.*

**Definition 41** *[All89] Let $A$ be a language and let $\mathrm{K}\mu$ be a Kolmogorov complexity measure. We define the Kolmogorov complexity of $A$ for length $n$ as*

$$\mathrm{K}\mu_A(n) = \min\{\mathrm{K}\mu(x) \ : \ |x| = n \text{ and } x \in A\}$$

*If $A \cap \Sigma^n = \emptyset$ then $\mathrm{K}\mu_A(n)$ is undefined.*

A typical question that will concern us is the question of how rapidly $\mathrm{K}\mu_A(n)$ can grow, for $A$ residing in various complexity classes. For example, consider the following theorem:

**Theorem 42** *[All01, For01, ISW99, KRC00, ACR98] The following are equivalent:*

1. Kt *and* KT *are exponentially far apart. (That is, there is some $\epsilon > 0$ such that for all large $n$ there is a string $x \in \Sigma^n$ such that* $\mathrm{KT}(x) > 2^{\epsilon(\mathrm{Kt}(x)+\log|x|)}$.*)*
2. *For all polynomially-dense $A \in$* P/poly, $\mathrm{Kt}_A(n) = O(\log n)$.
3. *There is a language $A \in$* E *and a constant $\epsilon > 0$ such that, for all large $n$, there is no circuit of size $2^{\epsilon n}$ accepting $A^{=n}$.*
4. *There are pseudorandom generators $G$ computable in time $n^{O(1)}$, such that $G : \Sigma^{k \log n} \to \Sigma^n$.*
5. *There are hitting set generators for* P/poly *and threshold $\frac{1}{2}$ computable in polynomial time.*

One of the most important theorems in the literature on derandomization is that each of these conditions implies P = BPP [IW97]. Not all work in derandomization has been aimed at BPP; there has also been a significant amount of work aimed at discovering conditions that imply AM = NP. In particular, Klivans and van Melkebeek proved that if if there is a set in NE ∩ coNE that does not have oracle circuits circuits of subexponential size that make nonadaptive queries to SAT, then AM = NP [KvM02]. This was improved by Miltersen and Vinodchandran [MV99], who proved that the same conclusion follows from the formally weaker assumption that there is a set in NE ∩ coNE that does not have strong nondeterministic circuits of subexponential size, by showing that this assumption implies that there is a hitting-set generator computable in NP for co-nondeterministic circuits. Shaltiel and Umans [SU05] subsequently presented a better construction of a hitting-set generator that hits co-nondeterministic as well as nondeterministic circuits. In an earlier version of this paper, we considered several conditions that had been studied in relation to derandomizing AM, and showed that they are all equivalent [AKRR03]. Subsequently, Shaltiel and Umans improved this, to show that even the condition studied by Klivans and van Melkebeek is equivalent to the others [SU06].

In contrast to Theorem 42, we are not able to show that the hypotheses that have been used to derandomize AM have equivalent restatements in terms of resource-bounded Kolmogorov complexity. However, we are able to use these derandomization techniques to prove a partial analog of Theorem 42:

**Theorem 43** *The following are equivalent:*

1. $\exists \epsilon > 0,\ \forall n,\ \exists x \in \Sigma^n$ $\mathrm{KNT}(x) > 2^{\epsilon(\mathrm{KNt}(x)+\log|x|)}$. *(That is,* KNT *and* KNt *are nearly as far apart as possible.)*
2. *For all polynomially-dense $A$ in* coNP/poly, $\mathrm{KNt}_A(n) = O(\log n)$.
3. *For all polynomially-dense $A$ in* NP/poly, $\mathrm{KNt}_A(n) = O(\log n)$.
4. *For all polynomially-dense $A$ in* NP/poly ∩ coNP/poly, $\mathrm{KNt}_A(n) = O(\log n)$.
5. $\exists A \in$NE/lin, $\exists a$ *such that $A$ requires strong nondeterministic circuits of size* $2^{an}$.
6. $\exists A \in$NE/lin, $\exists a$ *such that $A$ requires nondeterministic circuits of size* $2^{an}$.
7. *There exist* SNP/log *computable hitting set generators for nondeterministic polynomial-size circuits and threshold* $\frac{1}{n^c}$ *(and similar conditions for co-nondeterministic and strong circuits).*

**Remark:** We wish to call attention to the equivalence of conditions 2 and 3. For some notions of complexity such as KT, there are polynomially-dense sets in coNP with essentially maximal KT complexity (such as $R_{\mathrm{KT}}$), whereas there are good reasons to believe that every polynomially-dense language in NP/poly has low KT-complexity. (Rudich gives evidence for this conjecture in [Rud97].) In contrast, we see here that the KNT complexity of dense sets in NP and coNP are similar.

*Proof.* (1 ⇔ 5) This equivalence is proved similarly to related statements in [All01]. Given any sequence of strings $x_1, x_2, \ldots$ with $|x_m| = n = 2^m$, where $\mathrm{KNT}(x_m)$ is large and $\mathrm{KNt}(x_m)$ is logarithmic in $\mathrm{KNT}(x_m)$ (and hence is logarithmic in $|x_m|$), define the language $A$ to be the set such that the truth table of $A^{=n}$ is given by $x_m$. Since $\mathrm{KNt}(x_m) = O(\log 2^m) = O(m)$, it is immediate that $A \in$ NE/lin. (The description of $x_m$ showing that $\mathrm{KNt}(x_m)$ is small gives the advice sequence for the NE/lin upper bound.) We need to show that $A$ requires large strong nondeterministic circuits. But this is immediate from Proposition 9, since by hypothesis $\mathrm{KNT}(x_m) \geq |x_m|^\epsilon$ for some $\epsilon > 0$.

Conversely, given any language $A \in$ NE/lin that requires exponential-size strong nondeterministic circuits, the length-$n$ prefices of the characteristic sequence $\chi_A$ have logarithmic KNt complexity and KNT complexity $n^\epsilon$ for some $\epsilon > 0$.

($5 \Rightarrow 6$) We prove the contrapositive, $\neg 6 \Rightarrow \neg 5$. Thus assume that every $A \in$ NE has "small" nondeterministic circuits (that is, of size less than $2^{an}$ for any $a > 0$). By Theorem 21, $\overline{A} \in$ NE/lin, and hence also has "small" nondeterministic circuits. This yields co-nondeterministic circuits for $A$; we can combine the two circuits to get strong nondeterministic circuits for $A$. This proves $\neg 5$. (Similar observations are made by Shaltiel and Umans [SU05].)

($6 \Rightarrow 5$) This is trivial; a strong nondeterministic circuit yields a nondeterministic circuit of roughly the same size.

($5 \Rightarrow$ 2, 3, 4, and 7) Shaltiel and Umans show that for any constant $c$ there is a function $G_x(0^n)$ computable in deterministic polynomial time with the property that if $x$ is a string of length $n^{O(1)}$ that is the truth table of a function requiring exponential size strong nondeterministic circuits, then $G_x(0^n)$ produces a set $H_{x,n}$ that is a hitting set for both nondeterministic and co-nondeterministic circuits of size $n^c$ with threshold $\frac{1}{n^c}$ [SU05, Corollaries 6.6 and 6.9].

It is now straightforward to obtain a hitting set generator in SNP/log. By assumption there is a set $A \in$ NE/lin that requires large strong nondeterministic circuits. Let $h(m)$ be the advice sequence for length $m$ (having length $O(m)$). Let $n = 2^m$. On input $0^n$ with advice $h(m)$ having length $O(\log n)$ we can, in nondeterministic polynomial time, guess and verify the string $x$ that is the truth table for $A^{=m}$, and then run the generator $G_x(0^n)$.

It is easy to see that any string in the hitting set output by a SNP/log computable hitting set generator has logarithmic KNt complexity; this completes the proof of this implication.

The implications ($7 \Rightarrow 4$) ($2 \Rightarrow 4$), and ($3 \Rightarrow 4$) are either trivial or follow via the argument above. Thus it suffices to prove ($4 \Rightarrow 5$).

($4 \Rightarrow 5$). Define $A = \{x : |x| = 5m \text{ and } \mathrm{KNt}(x) > m\}$. We claim that $A$ is in NE/lin. To see this, recall that for a string $x$ of length $5m$, $\mathrm{KNt}(x) \leq m$ implies $\exists d, |d| \leq m, \forall i\ U^d(i,b)$ has an accepting path iff $x_i = b$, where $U$ is a universal nondeterministic Turing machine running for $2^m$ steps. In order to enumerate all $x$'s of length $5m$ that have $\mathrm{KNt}(x) \leq m$, we will define a nondeterministic procedure that will allow us to exclude from consideration those $d$'s that are not valid descriptions of strings. Define $\alpha$ to be the number of strings $d$ of length $\leq m$ that are indeed valid descriptions of strings of length $5m$, (i.e., there exists an $x$ for which $\forall i\ U^d(i,b)$ has an accepting path iff $x_i = b$), and we define $\beta$ to be the number of "recognizably bad" descriptions, that is, those strings $d$ of length $\leq m$ for which $\forall i \leq 5m+1,\ \exists b \in \{0,1,*\}, U^d$ accepts $(i,b)$ and for some $i$ and some $b' \neq b \in \{0,1,*\}, U^d$ accepts both $(i,b)$ and $(i,b')$. Our NE/lin machine, on input $x$ of length $5m$, takes $\alpha$ and $\beta$ as advice (each of length $O(m)$). First it guesses $\beta$ "recognizably bad" descriptions and verifies that they are indeed bad by guessing accepting paths for both $(i,b)$ and $(i,b')$. Then it guesses $\alpha$ other strings (corresponding to candidate "good" $d$'s), and guesses accepting paths for all of them and prints out the corresponding strings. All of this takes time exponential in $m$. Now we can accept $x$ if and only if it is not in the list that has been generated.

Now we need to show that $A$ requires large strong nondeterministic circuits. Assume otherwise, so that for every $c$ there is some $n$ such that there is a strong nondeterministic circuit of size $2^{n/c}$ deciding $A$ for inputs of length $n$. Then we can construct a polynomially-dense language $B \in$ NP/poly $\cap$ coNP/poly of the form $B = \{y : |y| = n$ and the prefix of $y$ of length $c_n \cdot \log n$ is in $A\}$ where $c_n$ is chosen (nonuniformly, as part of the advice sequence) to be as large as possible, so that the membership test for $A$ can be implemented in size $n$ via a strong nondeterministic circuit. By assumption, the sequence of numbers $(c_n)$ is unbounded. It follows that $\mathrm{KNt}_B(n) \neq O(\log n)$, contrary to our assumption. □

Although none of the conditions of the preceding theorem are known to imply AM = NP, it is trivial to observe that they imply AM $\subseteq$ NP/log. It is worth mentioning that these conditions do imply a nontrivial inclusion for AM:

**Theorem 44** *If there exists $A \in$*NE/lin*, such that $A$ requires strong nondeterministic circuits of size $2^{an}$, for some*

$a > 0$, *then* AM $\in \mathrm{P}^{\mathrm{NP}[\log n]}$.

*Proof.* As in [MV99], to determine if $x$ is in a set $B \in$ AM, we model the Arthur-Merlin game using a nondeterministic circuit with input $x$ and some probabilistic inputs $y$. Let $C_x$ be the result of hardwiring the bits of $x$ into this circuit; then $x \in B \Rightarrow C_x$ accepts every $y$, and $x \notin B \Rightarrow C_x$ rejects at least half of the strings $y$. Thus it suffices to use our NP oracle to determine if there is a string $y$ that is rejected by $C_x$. By parts 7 and 2 of the preceding theorem, if such a string $y$ exists, then there is such a string with $\mathrm{KNt}(y) = O(\log n)$.

Thus it suffices to design a $\mathrm{P}^{\mathrm{NP}[\log n]}$ procedure to determine if there is a string $y$ with $\mathrm{KNt}(y) \leq c \log n$ such that the nondeterministic circuit $C_x$ rejects $y$.

As in the proof of (4 $\Rightarrow$ 5) of the previous theorem, let $\alpha$ be the number of good descriptions of length at most $c \log n$ and let $\beta$ be the number of "recognizably bad" descriptions $d$ of length at most $c \log n$. The numbers $\alpha$ and $\beta$ can be computed in $O(\log n)$ queries to an NP oracle of the form "do there exist $\geq j$ strings $(d_1, d_2, \cdots, d_j)$ of length at most $c \log n$ such that for all $m$ and all $i \leq |y| + 1$ there is a $b \in \{0, 1, *\}$ such that $U^{d_m}(i, b)$ has an accepting path?" and "do there exist $\geq j$ strings $(d_1, d_2, \cdots, d_j)$ of length at most $c \log n$ such that for all $m, i$ there is a $b$ such that $U^{d_m}(i, b)$ accepts and there is some $i \leq |y| + 1$ for which there are $b \neq b' \in \{0, 1, *\}$ such that $U^{d_m}(i, b)$ and $U^{d_m}(i, b')$ each have an accepting path?" Having computed $\alpha$ and $\beta$ we can ask one more query to an NP oracle to determine if there are $\beta$ bad descriptions and $\alpha$ good descriptions such that $C_x$ accepts all of the strings $y$ described by the $\alpha$ good descriptions. □

One might wonder how reasonable it is to expect that a condition such as $\exists \epsilon > 0,\ \forall n,\ \exists x \in \Sigma^n$, $\mathrm{KNT}(x) > 2^{\epsilon(\mathrm{KNt}(x) + \log |x|)}$ should hold (saying that KNT and KNt are nearly as far apart as possible). The following proposition shows that, for at least some of the measures that we consider, there are unconditional results of this form that one can prove.

**Proposition 45** $\forall n,\ \exists x,\ \in \Sigma^n$, $\mathrm{KS}(x) > n^{1/2}$ *and* $\mathrm{Ks}(n) \leq 2 \log n + O(1)$.

*Proof.* A straightforward diagonalization shows that there is a set $A$ in $\mathrm{DSpace}(2^n)$ that is not in io-$\mathrm{DSpace}(2^{2n/3}/2^{2n/3})$. Let $x_n$ be the length $n$ prefix of the characteristic sequence $\chi_A$. It is immediate that $\mathrm{Ks}(x_n) \leq 2 \log n + O(1)$ (since it is described by the number $n$ and a program for $A$, which can be simulated in space $O(2^n)$. Assume for the sake of contradiction that $\mathrm{KS}(x_n) \leq n^{1/2}$, and let $m$ be the largest number such that $x_n$ contains the membership information for all strings of length $m$. Clearly $n \geq 2^m \geq n/2$. Thus there is a description $d$ of length $n^{1/2} \leq 2^{(m+1)/2}$ such that for all $x$ of length at most $m$, $U^d(x, b)$ runs in space $\leq 2^{(m+1)/2}$ and accepts if and only if $x \in A$. This is counter to our choice of $A$. □

# 5 KF Complexity and the NEXP $\subseteq$ NC$^1$ Question

Derandomization techniques were used in [IKW01] to show that NEXP $\subseteq$ P/poly if and only if NEXP = MA; it was observed in [All01] that this is also equivalent to conditions concerning the Kt-complexity of sets in P. In this section we conduct a similar investigation of the question of whether or not NEXP is contained in nonuniform NC$^1$.

In order to formulate the appropriate generalization of the equivalence NEXP $\subseteq$ P/poly if and only if NEXP = MA, it will be helpful to present a technical definition. We begin by recalling the definition of IP[P/poly].

**Definition 46** *[AKS95]* $\mathrm{IP}[\mathrm{P/poly}]$ *is the class of languages having an interactive proof system where the strategy of the honest prover can be computed by a* P/poly *circuit family (also see [AK99], where the multiple prover class* $\mathrm{MIP}[\mathrm{P/poly}]$ *is observed to be the same as* $\mathrm{IP}[\mathrm{P/poly}]$*).*

Clearly $\mathrm{IP}[\mathrm{P/poly}] \subseteq \mathrm{MA} \cap \mathrm{P/poly}$ (because Merlin can guess the circuit that implements the Prover's strategy and send it to Arthur); it appears to be a proper subclass of MA (since otherwise NP $\subseteq$ P/poly). If NEXP $\subseteq$ P/poly, the proof of [IKW01] actually shows that NEXP $= \mathrm{IP}[\mathrm{P/poly}]$. We now define an analogous subclass of MA$\cap$ nonuniform $\mathrm{NC}^1$.

**Definition 47** $\mathrm{MIPNC}^1$ *refers to the class of languages for which there is a 2-prover one-round interactive proof protocol where the strategy of each honest prover can be implemented by a (nonuniform)* $\mathrm{NC}^1$ *circuit family and the computation of the verifier is computable by a uniform (probabilistic)* $\mathrm{NC}^1$ *circuit family. (Although it is important that the verifier's circuits be uniform, our results do not depend crucially on the exact notion of uniformity. They hold for* P*-uniformity and for* DLOGTIME*-uniformity.)*

We could likewise define $\mathrm{IPNC}^1$ as the class of languages similar to the above for a single-prover constant-round interactive proof protocol, but we can easily see that $\mathrm{MIPNC}^1$ and $\mathrm{IPNC}^1$ coincide.

**Definition 48** *Every* NEXP *search problem is solvable in* $\mathcal{C}$ *if for every* $k$ *and* every NEXP *machine* $M$ *running in time* $2^{n^k}$ *there is a function* $f$ *in* $\mathcal{C}$ *with the following property. If* $x$ *is accepted by* $M$*, then there is a witness* $w \in \{0,1\}^{2^{|x|^{O(k)}}}$ *encoding an accepting computation path of* $M$ *on input* $x$*, such that for all* $i \leq 2^{|x|^{O(k)}}$ $f(x,i,b) = 1$ *if and only if* $w_i = b$*. (That is, the unique string* $w = b_1 b_2 \ldots b_{2^{|x|^{O(k)}}}$ *such that* $f(x,i,b_i) = 1$ *encodes an accepting computation path of* $M$ *on input* $x$*.)*

**Theorem 49** *The following are equivalent:*

1. *For all* $A \in$ NP, $\mathrm{KF}_A(n) = \log^{O(1)} n$*.*
2. *For all* $A \in$ DLOGTIME*-uniform* $\mathrm{AC}^0$*,* $\mathrm{KF}_A(n) = \log^{O(1)} n$*.*
3. *All* NEXP *search problems are solvable in nonuniform* $\mathrm{NC}^1$*.*
4. NEXP $\subseteq$ *nonuniform* $\mathrm{NC}^1$*.*
5. NEXP $= \mathrm{MIPNC}^1$*.*

*Proof.* Items (1) and (2) are easily seen to be equivalent, as in the remark before Proposition 61.

The proof that (2 $\Rightarrow$ 3) is immediate, once the following two assertions are established:

- (2) $\Rightarrow$ EXP $\subseteq \mathrm{NC}^1$.
- (2) $\Rightarrow$ NEXP search problems are solvable in EXP.

Assume both of these assertions hold. Then for a given NEXP search problem solved in exponential time by machine $M$, the language $\{(x,i,b) :$ the $i$th bit output by $M$ on input $x$ is $b\}$ is in $\mathrm{NC}^1$. The existence of such circuit families for NEXP search problems is precisely what is meant by condition (3). Let us examine each assertion in turn.

Let $A \in$ EXP. Let $B = \{w : w \text{ is a prefix of } \chi_A\}$. $B$ is clearly in P and (since we have already observed that (2 $\Rightarrow$ 1)) our assumption tells us that $\mathrm{KF}_B n = \log^{O(1)}(n)$. Now Proposition 6 allows us to conclude that $A \in \mathrm{NC}^1$.

For the second assertion, let $M$ be any NEXP machine, and consider the language $C = \{y10^x : \text{where } y \in \{0,1\}^{2^{|x|^k}} \text{ is a witness that } M \text{ accepts } x\}$. $C$ is in DLOGTIME-uniform $\mathrm{AC}^0$ (assuming an appropriate encoding of witnesses) and by (2) if there is any string in $C^{=n}$ then there is a string in $C^{=n}$ with small KF complexity. The exponential-time algorithm solving this search problem involves taking input $x$ and searching through all short descriptions and seeing if any of the strings thus described encodes an accepting computation path of $M$ on input $x$.

($3 \Rightarrow 4$) This implication requires some explanation. In most cases, assuming that a search problem is easy trivially implies that the decision problem is easy. However, when we assume that every NEXP search problem is solvable in $\mathrm{NC}^1$, it means only that there is an $\mathrm{NC}^1$ circuit $C$ such that $C(x)$ describes an exponentially long witness for membership when such a witness for $x$ exists. It is not *obvious* that there is any easy way to detect when $C(x)$ is describing such a witness. Thus this implication does require proof.

Certainly (3) implies that NP search problems are solvable in $\mathrm{NC}^1$. Let $A \in$ NP be accepted by NP-machine $M$, and let $C$ be a circuit solving the search problem defined by $M$. Thus $x \in A$ if and only if $C(x,1)C(x,2)\cdots C(x,n^k)$ encodes an accepting computation of $M$. This latter condition can also be checked in $\mathrm{NC}^1$, which implies NP $\subseteq$ (nonuniform) $\mathrm{NC}^1$. NP being contained in $\mathrm{NC}^1$ easily implies that $\Sigma_2^p$ is contained in $\mathrm{NC}^1$. On the other hand, by [IKW01], if NEXP search problems are solvable in P/poly, then NEXP is in $\Sigma_2^p$.

($4 \Rightarrow 5$) To prove this implication, observe that by [IKW01] if NEXP $\subseteq$ P/poly then NEXP $=$ MA $=$ PSPACE. By [CCL94], we know that PSPACE has 2-prover, 1-round interactive proof systems, where the honest provers are in PSPACE. Also we note that the verifier's protocol is very easy to compute; it sends random sequences to each prover and receives from the provers sequences of polynomials on which it performs (in parallel) some consistency checks. The consistency checks involve field operations, which are computable by DLOGTIME-uniform $\mathrm{TC}^0$ circuits [HAB02]. All the queries to the provers are made in one round (and hence are non-adaptive). Since by assumption, PSPACE $\subseteq \mathrm{NC}^1$, we have that every language in NEXP is also in $\mathrm{MIPNC}^1$.

($5 \Rightarrow 2$) Now we prove this implication. We largely follow [IKW01], where it is shown that if NEXP $\subset$ P/poly, then NEXP search can be performed by P/poly circuits. More precisely, we will show that if there is a set in P with large KF-complexity, then for every $\epsilon > 0$, $\mathrm{MIPNC}^1 \subset io - [\mathrm{NTime}(2^{n^\epsilon})/n^\epsilon]$. As in [IKW01] this latter condition implies either that $\mathrm{MIPNC}^1$ is a proper subset of NEXP (which is to say that condition (5) is false) or else EXP $\neq$ NEXP (which also easily implies that condition (5) is false).

Let $A \in \mathrm{MIPNC}^1$, where the verifier's strategy is computable by a P-uniform family of probabilistic $\mathrm{NC}^1$ circuits $\{C_n\}$. Let $p$ be a polynomial, such that $C_n$ uses at most $p(n)$ probabilistic bits. Our strategy to determine if $x \in A$ is

1. Construct the circuit $C = C_{|x|}$.
2. Nondeterministically guess $\mathrm{NC}^1$ circuits $D, D'$ that might implement the strategies of the provers in the $\mathrm{MIPNC}^1$ protocol for $A$.
3. Construct a circuit $B$ that, given an input $y$ of length $p(n)$
   (a) Uses $C$ to compute the query that gets posed to each prover in the $\mathrm{MIPNC}^1$ protocol for $A$ on input $x$ and probabilistic sequence $y$.
   (b) Uses $D$ and $D'$ to answer the queries.
   (c) Uses $C$ to compute the actions of the verifier.
4. Estimate the probability that $B$ accepts a randomly-chosen string $y$.

By the definition of $\mathrm{MIPNC}^1$, if $x \in A$ then there are fan-in two circuits $D$ and $D'$ implementing the strategy of the provers (where the depth of $D$ and $D'$ is bounded by $d \log n$ for some constant $d$ depending only on $A$) such that the circuit $B$ accepts *all* of the inputs $y$, whereas if $x \notin A$, then *no* provers (and hence also no provers computed by small circuits $D$ and $D'$) can cause $B$ to accept more than one-third of the inputs $y$.

All of the steps in this algorithm are easily computable in NP except for the final step 4. In order to complete the argument that $\mathrm{MIPNC}^1 \subset io - [\mathrm{NTime}(2^{n^\epsilon})/n^\epsilon]$, it suffices to show that for infinitely many input lengths $n$, there is an advice string of length $n^\epsilon$ such that a nondeterministic machine running in time $2^{n^\epsilon}$ can estimate the probability that a circuit with fan-in two and depth $b \log p(n)$ accepts a randomly-chosen input of length $p(n)$ (where the constant $b$ and the polynomial $p$ depend only on our language $A$, and do not depend on $\epsilon$).

As in [ABK$^+$06], we will make use of the hardness-versus-randomness techniques of [NW94, BFNW93]. In particular, some of the results of [NW94, BFNW93, KvM02] are summarized in [ABK$^+$06] in the following form.

**Definition 50** *For all large $n$, any $\epsilon > 0$ and any Boolean function $f : \{0,1\}^{n^{\epsilon/3}} \to \{0,1\}$ there is a pseudo-random generator $G_{f,\epsilon}^{\mathrm{BFNW}} : \{0,1\}^{n^\epsilon} \mapsto \{0,1\}^{p(n)}$ with the property that the function $G_{f,\epsilon}^{\mathrm{BFNW}}$ is computable in space $O(n^\epsilon)$ given access to the Boolean function $f$, and such that the following theorem holds.*

**Theorem 51 ([BFNW93, KvM02])** *There is a constant $k'$ depending on $\epsilon$ such that if $T$ is a set such that $|\Pr_{r \in U_{p(n)}}[r \in T] - \Pr_{x \in U_{n^\epsilon}}[G_{f,\epsilon}^{\mathrm{BFNW}}(x) \in T]| \geq 1/3$, then there exists an oracle circuit $C$ of size $n^{k'}$ with oracle $T$ that computes $f$ and queries $T$ non-adaptively.*

Closer examination of the proof techniques that are used in [BFNW93, KvM02] shows that the circuit $C$ computing the reduction can actually be implemented as a *constant depth* circuit of MAJORITY gates and oracle gates. Thus it can be implemented as a circuit of depth $k \log n$ for some constant $k$, consisting of oracle gates (where there is no path in the circuit from one oracle gate to another) and AND and OR gates of fan-in two.

Now we can state our $io - [\mathrm{NTime}(2^{n^\epsilon})/n^\epsilon]$ algorithm to estimate the probability that an $\mathrm{NC}^1$ circuit accepts. Let $L$ be a language in $\mathrm{DTime}(n^k)$ such that for every $\ell$ there exist infinitely many $m$ such that $\mathrm{KF}_L(m) > \log^\ell m$. By our assumption that condition (2) fails, such a set $L$ exists.

On input $x$ of length $n$, our advice string will be a number $m$ with approximately $n^\delta$ bits with $\delta = \epsilon/3$, such that $L$ contains strings of length $m$, and all strings of length $m$ in $L$ have high KF complexity. Our nondeterministic algorithm will guess a string $z$ of length $m$ and verify that $z \in L$. This takes time $2^{O(n^\epsilon)}$. Let $f$ be the Boolean function on inputs of length $\lceil \log m \rceil$ (roughly $n^\epsilon$) whose truth table has $z$ as a prefix (and is zero elsewhere). By our assumption on $L$ (combined with Proposition 6), there exist infinitely many $m$ such that function $f$ requires Boolean formulae of size greater than $p(n)^{k+b}$. For any input length $n$ for which a corresponding $m = 2^{O(n^\epsilon)}$ exists, the probability that circuit $B$ accepts can be estimated by counting the fraction of strings $y$ of length $n^\epsilon$ such that $B$ accepts $G_{f,\epsilon}^{\mathrm{BFNW}}(y)$. This fraction must be within one-third of the true probability (since otherwise $f$ is computed by a formula of size $p(n)^{k+b}$, by Theorem 51).

Since $G_{f,\epsilon}^{\mathrm{BFNW}}(y)$ is computable in space $n^\epsilon$, the entire computation to estimate the acceptance probability of the $\mathrm{NC}^1$ circuit $B$ (and to recognize language $A$) takes time $2^{O(n^\epsilon)}$.

This completes the proof. □

The following definition of MIPL combined with an analogous proof yields Theorem 53

**Definition 52** MIPL *corresponds to the class of languages for which there is a 2-prover one-round interactive proof protocol where the strategy of each prover can be implemented in* L/poly *and the verifier is in* L.

**Theorem 53** *The following are equivalent :*

1. NEXP $\subseteq$ L/poly
2. *All* NEXP *search problems are solvable in* L/poly
3. *For all* $A \in$ P, $\mathrm{KB}_A(n) = \log^{O(1)} n$.
4. NEXP $=$ MIPL

For completeness, we summarize what is known about $\mathrm{K}\mu_A$ for $A \in$ P for the other measures $\mathrm{K}\mu$ that we have considered.

**Theorem 54** *The following equivalences hold:*

- *([All01]) All* NEXP *search problems are solvable in* P/poly *if and only if* $\forall A \in$ P, $\mathrm{KT}_A(n) = \log^{O(1)} n$.
- *All* NEXP *search problems are solvable in* NP/poly *if and only if* $\forall A \in$ P, $\mathrm{KNT}_A(n) = \log^{O(1)} n$.
- *All* NEXP *search problems are solvable in* PSPACE *if and only if* $\forall A \in$ P, $\mathrm{KS}_A(n) = \log^{O(1)} n$.
- *([All89]) All* NEXP *search problems are solvable in* EXP *if and only if* $\forall A \in$ P, $\mathrm{Kt}_A(n) = \log^{O(1)} n$.
- *For all* $A \in$ P, $\mathrm{Ks}_A(n) = O(\log n)$.

Note that in all cases the upper bounds on solvability of search problems for NEXP are given by *nonuniform* classes, except for the cases concerning PSPACE and EXP. However, it is easy to see that a NEXP search problem is solvable in PSPACE (or EXP) if and only if it is solvable in PSPACE/poly (or EXP/poly, respectively). This is because PSPACE provides enough resources to cycle through all advice sequences of polynomial length; similar observations were made by [BH92]. Note also that we do not have a crisp statement that is equivalent to every set $A \in$ P having $\mathrm{KNt}_A(n)$ bounded by $\log^{O(1)} n$. See also the remark after Proposition 61, regarding the KDt complexity of sets in P.

*Proof.* In order to see that $\mathrm{Ks}_A(n) = O(\log n)$ for all $A \in$ P, it suffices to observe that there is a trivial algorithm that runs in space $2^{O(|n|)}$ that takes the string $n$ as input and searches for the lexicographically least $x \in A^{=n}$, and produces this string $x$ as output.

All of the rest of the implications are proved similarly to each other. For the backward direction, in each case, the first step is to show that the given assumption implies that every NEXP search problem is solvable in EXP. It will suffice to consider the weakest of these assumptions; namely:

$$\forall A \in \mathrm{P},\ \mathrm{Kt}_A(n) = \log^{O(1)} n.$$

Let $M$ be any NEXP machine, and consider the language $C = \{y10^x$ : where $y \in \{0,1\}^{2^{|x|^k}}$ is a witness that $M$ accepts $x\}$. $C$ is in P and by assumption, if there is any string in $C^{=n}$ then there is a string in $C^{=n}$ with small Kt complexity. The exponential-time algorithm solving this search problem involves taking input $x$ and searching through all short descriptions and seeing if any of the strings thus described encodes an accepting computation path of $M$ on input $x$.

Thus for any NEXP search problem there is a deterministic exponential-time machine $M'$ solving it. Hence the language $\{(x, i, b)$ : the $i$th bit output by $M'$ on input $x$ is $b\} \in$ EXP, and hence it is $\mathrm{AC}^0$-reducible to a set $A \in$ E.

In order to complete the proof of the backward direction, it suffices to show that this set $A$ is contained in the appropriate class. The set $\{w : w \text{ is a prefix of the characteristic sequence of } A\}$ is in P (and contains exactly one string of each length $n$). Results such as Proposition 9 and Theorem 4 now suffice to give the appropriate upper bound for $A$.

For the forward direction, let $A$ be any set in P, and consider the search problem defined by the NE machine $M$ that takes input $n$ and guesses a string $x$ of length $n$, accepting if and only if $x \in A$. By hypothesis, there is a function in a given class that takes as input $(n, i)$ and returns the $i$th bit of some string of length $n$ in $A$. In each case, this is precisely what is needed in order to provide the desired upper bound on $\mathrm{K}\mu_A(n)$. □

# 6 Distinguishing Complexity

Recall from Section 1.3 that there are three main kinds of resource-bounded Kolmogorov complexity that have been studied:

- Definitions in the spirit of Levin's Kt measure.
- Definitions similar to $\mathrm{C}^t$ and $\mathrm{K}^t$ for various time bounds $t$.
- Distinguishing complexity.

Thus far in this paper, we have introduced a number of other measures in the spirit of Levin's Kt measure, where these new measures bear close relationships to different complexity classes (much in the same way as varying the time bound $t$ causes the measures $\mathrm{C}^t$ and $\mathrm{K}^t$ to be related to different complexity classes). But the measures that we have introduced have more in common with $\mathrm{C}^t$ and $\mathrm{K}^t$ (in which a description allows a machine to *produce* a string) than with distinguishing complexity (in which a description allows a machine to *recognize* a string $x$ when it sees it). In this section, we investigate the topic of distinguishing complexity, in the spirit of Levin's Kt measure.

Distinguishing complexity dates back to the work of Sipser [Sip83], and it has been studied in more depth by Fortnow and Kummer [FK96] and by Buhrman, Fortnow, and Laplante [BFL02]. In all cases, the focus has been on the polynomial-time bounded versions of distinguishing complexity. Here are the formal definitions of the Distinguishing Complexity measures:

**Definition 55** *Let $p$ be a polynomial, and let $U_1$ ($U_2$) be a universal (nondeterministic) Turing machine.*

- $\mathrm{CD}^p(x)$ *is defined to be the minimum $|d|$ such that $U_1^d(y)$ accepts in time $p(|x|)$ if and only if $y = x$.*
- $\mathrm{CND}^p(x)$ *is defined to be the minimum $|d|$ such that $U_2^d(y)$ accepts in time $p(|x|)$ if and only if $y = x$.*

The following definitions are the most natural way to formulate notions of Distinguishing Complexity more in line with Kt and KNt complexity:

**Definition 56** *Let $U_1$ be a fixed deterministic Turing machine, and let $U_2$ be a fixed nondeterministic Turing machine.*

$$\begin{aligned} \mathrm{KDt}_{U_1}(x) &= \min\{|d| + \log t \,:\, \forall y \in \Sigma^{|x|}\ U_1^d(y) \\ &\qquad \textit{runs in time } t \textit{ and accepts iff } x = y\} \\ \mathrm{KNDt}_{U_2}(x) &= \min\{|d| + \log t \,:\, \forall y \in \Sigma^{|x|}\ U_2^d(y) \\ &\qquad \textit{runs in time } t \textit{ and accepts iff } x = y\} \end{aligned}$$

As usual, we select fixed universal deterministic Turing machines $U_1$ and nondeterministic $U_2$, and define KDt to be $\text{KDt}_{U_1}$, and KNDt to be $\text{KNDt}_{U_2}$. Via standard arguments it follows that for all $U'$, we have $\text{KDt}(x) \leq \text{KDt}_{U'}(x) + c\log|x|$ for some constant $c$, and for all $U''$, we have $\text{KNDt}(x) \leq \text{KNDt}_{U''}(x) + c$. It is clear that $\text{KNDt}(x) - O(1) \leq \text{KDt}(x) \leq \text{Kt}(x) + O(\log|x|)$.

We see no useful way to define a measure bearing the same relationship to KDt as KT bears to Kt, because with Distinguishing Complexity, the machine $U$ has access to the *entire* string $x$, and this would seem to entail run-times that are at least linear.

When discussing strings having logarithmic distinguishing complexity, it makes little difference if one uses $\text{CD}^p$ or KDt complexity (and similarly $\text{CND}^p$ and KNDt agree closely with each other for strings of logarithmic complexity). As we shall see, for a number of the situations where $\text{CD}^p$ and $\text{CND}^p$ complexity have been studied previously, KDt and KNt are just as useful.

We observe next that KNDt is essentially the same thing as KNt, up to logarithmic terms.

**Theorem 57** $\text{KNDt}(x) = \text{KNt}(x) + \Theta(\log|x|)$

*Proof.* Showing that $\text{KNDt}(x) \leq \text{KNt}(x) + O(\log|x|)$ is an easy exercise. Conversely, if $\text{KNDt}(x)$ is small (using description $d$), then a nondeterministic machine with oracle $d' = (d, n)$, given input $(i, b)$ can guess $x \in \Sigma^n$ and if $U^d(x)$ accepts, then accept iff the $i$th bit of $x$ is $b$. Analysis of the run times easily yields that $\text{KNDt}(x) \leq \text{KNt}(x) + O(\log|x|)$. □

Since KNDt is indistinguishable from KNt from our standpoint, we will not refer to KNDt any further.

Since KNDt is so closely related to KNt, one is quickly led to ask if KDt is similarly related to Kt. At first glance, the following proposition would seem to indicate that they *are* closely related:

**Proposition 58** *$R_{\text{Kt}}$ and $R_{\text{KDt}}$ are both complete for* EXP *under* P/poly *truth-table reductions and* NP*-Turing reductions.*

*Proof.* For $R_{\text{Kt}}$ this is proved in [ABK+06], and in fact hardness holds for any polynomially dense set containing no strings of low Kt-complexity. Since $\text{Kt}(x) > \text{KDt}(x) - O(\log|x|)$ it follows that $R_{\text{KDt}}$ is also hard for EXP. Membership in EXP is easy to show. □

Nonetheless, it seems unlikely that Kt and KDt are polynomially related; we show below that this would imply the collapse of some exponential-time complexity classes. Fortnow and Kummer made related observations about $\text{CD}^p$ and $\text{C}^p$ [FK96]; for a given polynomial $p$, they showed that if there was a polynomial $p'$ such that $\text{C}^{p'}(x) \leq \text{CD}^p(x)$, then every sparse set in FewP is in P, which in turn is equivalent to FewE = E [RRW94]. Here, we show that if KDt and Kt are polynomially related, it not only *implies* a collapse of related classes, but is in fact *equivalent* to a certain collapse. In order to state this precisely, we need some additional definitions.

**Definition 59** *We say that* FewEXP search instances are EXP-solvable *if, for* every NEXP *machine $N$ and every $k$ there is an* EXP *machine $M$ with the property that if $N$ has fewer than $2^{|x|^k}$ accepting paths on input $x$, then $M(x)$ produces one of these accepting paths as output if there is one. We say that* FewEXP decision instances are EXP-solvable *if, for* every NEXP *machine $N$ and every $k$ there is an* EXP *machine $M$ with the property that if $N$ has fewer than $2^{|x|^k}$ accepting paths on input $x$, then $M(x)$ accepts if and only if $N(x)$ accepts.*

**Remark:** Note that we do not require that $N$ is a FewEXP machine, i.e., we do not require that $N$ have a small number of accepting paths on *every* input.

**Theorem 60** *The following statements are equivalent.*

1. $\forall x, \mathrm{Kt}(x) \leq \mathrm{KDt}(x)^{O(1)}$
2. FewEXP *search instances are* EXP-*solvable (or* EXP/poly-*solvable).*
3. FewEXP *decision instances are* EXP-*solvable (or* EXP/poly-*solvable).*
4. $\forall\, L \in \mathrm{P}$, $\mathrm{Kt}_L(n) \leq (\log |L^{=n}| + \log n)^{O(1)}$
5. $\forall\, L \in$ *Dlogtime-uniform* $\mathrm{AC}^0$, $\mathrm{Kt}_L(n) \leq (\log |L^{=n}| + \log n)^{O(1)}$
6. $\forall x,\ \forall y$, $\mathrm{KDt}(x) \leq (\mathrm{KDt}(xy) + \log |xy|)^{O(1)}$

*Proof.* It is immediate that 2 $\Rightarrow$ 3, 4 $\Rightarrow$ 5, and 1 $\Rightarrow$ 6. We will now prove 3 $\Rightarrow$ 1, 1 $\Rightarrow$ 4, 5 $\Rightarrow$ 2, and 6 $\Rightarrow$ 1.

(3 $\Rightarrow$ 1) Consider a NEXP machine $M$ that with oracle $d$ on input $(1^t, i, b, n)$ guesses a string $y \in \{0,1\}^n$, runs $U^d(y)$ for $2^t$ steps and then accepts iff $y_i = b$ and $U^d(y)$ accepts. If $\mathrm{KDt}(x) \leq |d| + t$, where $d$ is a distinguishing description for a string $x \in \{0,1\}^n$ and $t$ is sufficiently large, then there is exactly one accepting path of $M^d$ on input $(1^t, i, x_i, |x|)$; there is no accepting path of $M^d$ on $(1^t, i, \overline{x_i}, |x|)$, for all $1 \leq i \leq |x|$. Note that the run-time $2^t$ must be at least $|x|$, since $U$ must read every bit of $x$ in order to distinguish $x$ from all other strings of the same length. By our assumption, there is a deterministic machine $N$ running in exponential time, that with oracle $d$ on input $(1^t, i, b, |x|)$, given some polynomial advice $h$, can decide whether $M^d$ accepts $(1^t, i, b, |x|)$ or not.[1] Thus, given $d$, $t$, $|x|$ and the advice $h$, we can generate $x$ bit by bit in time exponential in $(|d| + t + \log |x| + |h|)^{O(1)}$. Thus $\mathrm{Kt}(x) \leq (|d + t + \log |x| + |h|)^{O(1)} = (|d| + t)^{O(1)} \leq \mathrm{KDt}(x)^{O(1)}$.

(1 $\Rightarrow$ 4) Buhrman, Fortnow, and Laplante use hashing to show that for any set $L$ there is a polynomial time algorithm with oracle access to $L$, such that for every $x \in L$ there is a description $d_x$ of length $2 \log \left|L^{=|x|}\right| + O(\log |x|)$, such that the algorithm accepts $(z, d_x)$ if and only if $z = x$ [BFL02]. If $L \in \mathrm{P}$, then oracle access to $L$ is not necessary, and for every $x \in L$ we conclude that $\mathrm{KDt}(x) \leq 2 \log \left|L^{=|x|}\right| + O(\log |x|)$. Assuming that KDt and Kt are polynomially related we obtain $\mathrm{Kt}(x) \leq (\log \left|L^{=|x|}\right| + \log |x|)^{O(1)}$.

(5 $\Rightarrow$ 2) Let $L$ be decidable by a nondeterministic machine $N$ running in time $2^{n^k}$, for $k \geq 1$. Define the set $C = \{w10^x$ : where $w \in \{0,1\}^{2^{2|x|^k}}$ is a witness that $N(x)$ accepts $\}$. (Here, we identify $x$ with the integer having binary representation $1x$.) Clearly, we can choose an encoding of NEXP computations so that $C \in$ Dlogtime-uniform $\mathrm{AC}^0$. Let $x$ be a string, such that $N(x)$ has few accepting paths, i.e., $|C^{=n_x}| \leq 2^{|x|^{O(1)}}$, where $n_x = 2^{|x|^k} + x + 1$. By assumption, there is a witness $w$ with $\mathrm{Kt}(w10^x) \leq |x|^{O(1)}$. So in order to find a witness for $x \in L$ we just need to search through all strings $y$ with $\mathrm{Kt}(y) \leq |x|^{O(1)}$, which can be done in exponential time.

(6 $\Rightarrow$ 1) Assume that there is a constant $c \geq 1$, such that for every string $z$ and every prefix $x$ of $z$, $\mathrm{KDt}(x) \leq (\mathrm{KDt}(z) + \log |z|)^c$. Let $z$ be a string of length $n$. If $\mathrm{KDt}(z) \geq n^{1/c}$, then clearly $\mathrm{Kt}(x)$ will be bounded by $\mathrm{KDt}(z)^{c'}$ for some constant $c'$, which establishes the claim in this case. Thus assume that $\mathrm{KDt}(z) < n^{1/c}$. Let $a = \max\{\mathrm{KDt}(x);\ x \text{ is a prefix of } z\}$. By assumption, $a \leq (\mathrm{KDt}(z) + \log |z|)^c < n$. We construct a sequence $S_a, \ldots, S_n$ of sets with $|S_i| \leq 2^a$, where $S_i$ contains (at least) all of the strings $x$ of length $i$ such that $x$ and every prefix of $x$ has KDt complexity $\leq a$. We initially start with $S_a = \{0,1\}^a$ and then proceed iteratively as follows.

$$
\begin{aligned}
S_{i+1} \quad := \quad & \big\{ s \in S_i \circ \{0,1\} \,:\, \exists d_s \in \{0,1\}^a \\
& \quad U^{d_s}(s) \text{ accepts , and if } s' \in S_i \circ \{0,1\} \\
& \quad \text{and } s \neq s' \text{ then } U^{d_s}(s') \text{ rejects } \big\}
\end{aligned}
$$

[1] Note that it would have been sufficient to use a formally weaker assumption, dealing only with the case where there is a single accepting path. Similarly, in condition 4, it would be sufficient to consider only those $n$ for which $|L^{=n}| = 1$.

It is fairly straightforward to verify that these sets have the property mentioned above, namely that they are not too big and that they contain all the simple strings having simple prefices. Thus $z \in S_n$. Let $x$ be any prefix of $z = xy$, having length $i$. Observe that there is an algorithm running in time $n2^{O(a)}$ that takes input $(n, a, i, j)$, and computes each set $S_a, \ldots, S_i$ and then produces as output the $j$-th string in $S_i$. Hence, $\mathrm{Kt}(x) \leq |(n, a, i, j)| + \log(n2^{O(a)}) + O(\log n) = O(a + \log n) = (\mathrm{KDt}(xy) + \log |xy|)^{O(1)}$. □

**Remark:** The final condition of Theorem 60 deserves some comment. For all of the other resource-bounded Kolmogorov complexity measures $\mathrm{K}\mu$ studied in this paper (other than KDt) it is easy to see that the following three conditions are equivalent:

- For all $A \in$ NP, $\mathrm{K}\mu_A(n) \leq \log^{O(1)} n$.
- For all $A \in$ P, $\mathrm{K}\mu_A(n) \leq \log^{O(1)} n$.
- For all $A \in$ DLOGTIME-uniform $\mathrm{AC}^0$, $\mathrm{K}\mu_A(n) \leq \log^{O(1)} n$.

(For a proof, see Theorem 3 in [All01].) The simple observation that forms the main part of the proof of this equivalence is the fact that for all $x$ and $y$, $\mathrm{K}\mu(x)$ can be bounded by $\mathrm{K}\mu(xy) + \log |xy|$. (That is, the complexity of a string does not *decrease* by much if more information is appended to it, according to $\mathrm{K}\mu$.) Distinguishing complexity does not seem to work this way; appending some information to $x$ may make the KDt complexity plummet. The next proposition shows that the last two of these three conditions are equivalent, but it remains unknown if they are equivalent to the first condition.

**Proposition 61** *The following are equivalent:*

- *For all* $A \in$ P, $\mathrm{KDt}_A(n) \leq \log^{O(1)} n$.
- *For all* $A \in$ DLOGTIME-*uniform* $\mathrm{AC}^0$, $\mathrm{KDt}_A(n) \leq \log^{O(1)} n$.

*Proof.* It suffices to prove that the second condition implies the first. Thus assume that the second condition holds, and let $A \in \mathrm{DTime}(n^k)$ be recognized by some deterministic machine $M$ running in time $kn^k$. Let $B$ be the set $\{(x, C_0, C_1, \ldots, C_m) : |(x, C_0, C_1, \ldots, C_m)| = \ell |x|^\ell$, $C_0$ encodes an initial configuration of $M$ on input $x$, $C_m$ encodes an accepting configuration of $M$, $C_i \vdash C_{i+1}$ for all $i < m\}$, where $\ell$ is chosen so that there is a string in $A$ of length $n$ iff there is a string in $B$ of length $\ell n^\ell$. For a string $x$ in $A$ we denote the corresponding string in $B$ by $y_x$ (if there is any). It is easy to see that $B \in$ Dlogtime-uniform $\mathrm{AC}^0$. Let $U$ be the universal machine used to define KDt. In order to show that $\mathrm{KDt}_A(n) \leq \log^{O(1)} n$, assume that there is some string $x$ of length $n$ in $A$ and if there are several pick one such that there is a corresponding string $y_x$ in $B$, and a description $d$ showing that $\mathrm{KDt}(y_x) \leq \log^{O(1)} n$, i.e., $U^d(y_x)$ accepts in time $n^{\log^{O(1)} n}$, and there is no other string $z$ of length $|y_x|$ for which $U^d(z)$ accepts. A new deterministic machine $U'$ can use oracle $d$ to perform the following computation on input $w$: Compute the candidate string $y_w$ by simulating $M$ on $w$, and simulate $U^d(y_w)$. If $U^d(y_w)$ accepts, then $U'$ accepts $w$. By choice of $d$, the only string of length $|x|$ that $U'^d$ accepts is $x$ itself. The proposition now follows, by the properties of the universal machine defining KDt, and by analyzing the run time of the machine $U'$. □

**Remark:** The proof of the preceding proposition shows one could define a somewhat artificial (and messy) notion of what it means for all NEXP search problems to be "solvable in UEXP/poly," which would be equivalent to $\mathrm{KDt}_A(n) \leq \log^{O(1)} n$ for all $A \in$ P.

The preceding discussion indicates that it is unlikely that KDt is polynomially-related to Kt. Since KDt is intermediate between KDt and KNt, one might ask if KDt instead is polynomially-related to KNt. The following theorem shows both that this is unlikely, and that KDt and Kt again share several similarities.

**Theorem 62** *The following are equivalent:*

1. $\mathrm{KDt}(x) \leq \mathrm{KNt}(x)^{O(1)}$
2. $\mathrm{Kt}(x) \leq \mathrm{KNt}(x)^{O(1)}$
3. NEXP $\subseteq$ EXP/poly.

*Proof.* (2 $\Rightarrow$ 1) This is trivial.

(1 $\Rightarrow$ 2) If $\mathrm{KDt}(x)$ is always polynomially bounded by $\mathrm{KNt}(x)$, then it follows that for every $x$ and $y$ we have $\mathrm{KDt}(x) \leq \mathrm{KDt}(xy) + \log(|xy|)^{O(1)}$. Hence Theorem 60 yields $\mathrm{Kt}(x) \leq \mathrm{KDt}(x)^{O(1)}$, and we obtain the desired conclusion.

(1 $\Leftrightarrow$ 3) This is established in Corollary 63. □

**Remark:** This theorem is similar in spirit to [BFL02, Theorem 7.6], in which it is shown that P = NP if and only if either $\mathrm{C}^{p'}(x|y)$ or $\mathrm{CD}^{p''}(x|y)$ is polynomially-related to $\mathrm{CND}^{p}(x|y)$ for appropriate polynomials $p, p'$, and $p''$.

# 7 Which Kolmogorov Measures are Polynomially Related?

In the preceding section, we saw that KNDt and KNt are polynomially related, but that it is unlikely that KDt and Kt are. In this section, we examine what would happen if some of the other resource-bounded Kolmogorov complexity measures that we have introduced should happen to be polynomially related. We state the following theorem, and in the appendix we provide a table summarizing the relationships.

**Theorem 63** *The following equivalences hold:*

1. KF *vs. the rest:*
   - $\mathrm{KF}(x) \leq (\mathrm{KB}(x) + \log|x|)^{O(1)}$ *if and only if* L $\subseteq$ *nonuniform* $\mathrm{NC}^1$.
   - $\mathrm{KF}(x) \leq (\mathrm{KT}(x) + \log|x|)^{O(1)}$ *if and only if* P $\subseteq$ *nonuniform* $\mathrm{NC}^1$.
   - $\mathrm{KF}(x) \leq (\mathrm{KNT}(x) + \log|x|)^{O(1)}$ *if and only if* NP/poly $\cap$ coNP/poly = *nonuniform* $\mathrm{NC}^1$.
   - $\mathrm{KF}(x) \leq (\mathrm{KS}(x) + \log|x|)^{O(1)}$ *if and only if* $\mathrm{PSPACE} \subseteq$ *nonuniform* $\mathrm{NC}^1$.
   - $\mathrm{KF}(x) \leq (\mathrm{Kt}(x) + \log|x|)^{O(1)}$ *if and only if* EXP $\subseteq$ *nonuniform* $\mathrm{NC}^1$.
   - $\mathrm{KF}(x) \leq (\mathrm{KNt}(x) + \log|x|)^{O(1)}$ *if and only if* NEXP $\subseteq$ *nonuniform* $\mathrm{NC}^1$.
2. KB *vs. the rest:*
   - $\mathrm{KB}(x) \leq (\mathrm{KT}(x) + \log|x|)^{O(1)}$ *if and only if* P $\subseteq$ L/poly.
   - $\mathrm{KB}(x) \leq (\mathrm{KNT}(x) + \log|x|)^{O(1)}$ *if and only if* NP/poly $\cap$ coNP/poly $\subseteq$ L/poly.
   - $\mathrm{KB}(x) \leq (\mathrm{KS}(x) + \log|x|)^{O(1)}$ *if and only if* $\mathrm{PSPACE} \subseteq$ L/poly.

- $\mathrm{KB}(x) \leq (\mathrm{Kt}(x) + \log|x|)^{O(1)}$ *if and only if* EXP $\subseteq$ L/poly.
- $\mathrm{KB}(x) \leq (\mathrm{KNt}(x) + \log|x|)^{O(1)}$ *if and only if* NEXP $\subseteq$ L/poly.

3. KT *vs. the rest:*

- $\mathrm{KB}(x) \leq (\mathrm{KNT}(x) + \log|x|)^{O(1)}$ *if and only if* NP/poly $\cap$ coNP/poly $=$ P/poly.
- $\mathrm{KT}(x) \leq (\mathrm{KS}(x) + \log|x|)^{O(1)}$ *if and only if* $\mathrm{PSPACE} \subseteq$ P/poly.
- $\mathrm{KT}(x) \leq (\mathrm{Kt}(x) + \log|x|)^{O(1)}$ *if and only if* EXP $\subseteq$ P/poly.
- $\mathrm{KT}(x) \leq (\mathrm{KNt}(x) + \log|x|)^{O(1)}$ *if and only if* NEXP $\subseteq$ P/poly.

4. KNT *vs. the rest:*

- $\mathrm{KS}(x) = (\mathrm{KNT}(x) + \log|x|)^{O(1)}$ *if and only if* $\mathrm{PSPACE} \subseteq$ NP/poly.
- $\mathrm{Kt}(x) = (\mathrm{KNT}(x) + \log|x|)^{O(1)}$ *if and only if* EXP $\subseteq$ NP/poly.
- $\mathrm{KNt}(x) = (\mathrm{KNT}(x) + \log|x|)^{O(1)}$ *if and only if* NEXP $\subseteq$ NP/poly.

5. KS *vs. the rest:*

- $\mathrm{KS}(x) \leq (\mathrm{Kt}(x) + \log|x|)^{O(1)}$ *if and only if* EXP $\subseteq$ PSPACE.
- $\mathrm{KS}(x) \leq (\mathrm{KNt}(x) + \log|x|)^{O(1)}$ *if and only if* NEXP $\subseteq$ PSPACE/poly.

6. Kt *vs. the rest:*

- $\mathrm{Kt}(x) \leq (\mathrm{KNt}(x) + \log|x|)^{O(1)}$ *if and only if* NEXP $\subseteq$ EXP/poly.
- $\mathrm{Kt}(x) \leq (\mathrm{Ks}(x) + \log|x|)^{O(1)}$ *if and only if* EXPSPACE $\subseteq$ EXP/poly.

*Proof.* All the proofs of the equivalences follow essentially the same pattern and spelling all of them out would serve little purpose. Thus we provide a sample proof of one of the equivalences and point out the remaining subtleties. We prove: $\mathrm{KB}(x) \leq (\mathrm{KNt}(x) + \log|x|)^{O(1)}$ if and only if NEXP $\subseteq$ L/poly.

($\Rightarrow$) Assume that there is a constant $c > 0$ such that for all $x$, $\mathrm{KB}(x) \leq (\mathrm{KNt}(x) + \log|x|)^c$. Let $A$ be a language in NEXP. Denote by $t_n$ the truth table of $A$ restricted to strings of size $n$. By Corollary 22, $B = \{(1^n, i, (t_n)_i) : n \in \mathbb{N},\ 1 \leq i \leq n\}$ is in NEXP/poly. Hence, there is $k > 1$ such that for all $n$, $\mathrm{KNt}(t_n) \leq kn^k$. Thus, by our assumption, $\mathrm{KB}(t_n) \leq (2kn)^{ck}$. Furthermore, $\mathrm{BPSIZE}(t_n) \leq k'n^{k'}$ for some constant $k' > 1$. The implication follows by noting that L/poly consists precisely of functions computable by branching programs of polynomial size.

($\Leftarrow$) Assume that NEXP $\subseteq$ L/poly. Let $A$ be the set $\{(d, i, b, 1^j) : j \in \mathbb{N},\ d \in \{0,1\}^{\leq j},\ 1 \leq i \leq 2^j,\ b \in \{0,1\},\ U^d(i,b)$ accepts in time $2^j\}$ where $U$ is the universal machine that is used to define KNt. $A \in$ NEXP and thus lies in L/poly. Thus for any $m$ there is an advice string $a_m$ of length $m^\ell$ such that queries of length $O(m)$ to $A$ can be answered in space $O(\log m)$ using access to the advice string $a_m$.

Pick an arbitrary string $x$. Let $\mathrm{KNt}(x) = m$. Thus there is a description $d$ of length at most $m$ such that for all $i \leq |x|$, $U^d(i,b)$ runs in time at most $2^m$ and accepts if and only if $x_i = b$. Consider the oracle machine $M$ that uses oracle $(d, a_m, m)$, and on input $(i,b)$ determines if $(d, i, b, 1^m)$ is in $A$. This machine $M$ with oracle $(d, a_m, m)$ runs in space $O(\log m)$, and accepts if and only if $U^d(i,b)$ accepts (which happens if and only if $x_i = b$). The length of the description $(d, a, m)$ is $O(m^\ell)$, and thus $\mathrm{KB}_M(x) \leq (m + \log|x|)^c$ for some constant $c$. The theorem now follows, by the properties of the universal machine $U$ used to measure KB.

There are also some minor subtleties that arise in the implications involving KNT. We illustrate with a sample equivalence. We prove: $\mathrm{KNt}(x) = (\mathrm{KNT}(x) + \log|x|)^{O(1)}$ if and only if NEXP $\subseteq$ NP/poly.

($\Leftarrow$) Assume that $\mathrm{KNt}(x) = (\mathrm{KNT}(x) + \log|x|)^{O(1)}$, and let $A \in$ NEXP. Since coNEXP $\subseteq$ NEXP/poly it is easy to see that the $2^{n+1}$-bit prefix of the characteristic sequence of $A$ has $\mathrm{KNt}^A$ complexity $n^{O(1)}$ and by assumption also has KNT complexity $n^{O(1)}$. Thus there is a description $d_n$ of length $O(n^k)$ for some $k$ such that for all $x \leq 2^{n+1}$, the nondeterministic universal Turing machine $U^{d_n}(x, b)$ runs in at most $n^k$ time and accepts if and only if $x \in A$. The description $d_n$ can be used as an advice sequence to show that $A \in$ NP/poly.

($\Rightarrow$) Conversely, if NEXP $\in$ NP/poly, it follows that NEXP/poly = coNEXP/poly = NP/poly = coNP/poly. If $\mathrm{KNt}(x) = m$, then there is a description $d$ such that the nondeterministic machine $U^d(i, b)$ runs in time $2^m$ and accepts if and only if $x_i = b$. The language $\{(d, i, b, 1^m) : U^d$ accepts $(i, b)$ accepts in time $2^m\}$ is in NEXP, and by assumption lies in NP/poly $\cap$ coNP/poly. It follows that $(d, h_1, h_2)$ is a suitable description of the string described by $d$, where $h_1$ and $h_2$ are the advice strings for the NP/poly and coNP/poly algorithms, respectively, to show that the $\mathrm{KNT}(x)$ is polynomially related to $\mathrm{KNt}(x)$. This completes the proof of this implication.

The following subtlety is involved in establishing equivalences with EXP $\subseteq$ PSPACE. Instead of proving equivalence with EXP $\subseteq$ PSPACE one proves equivalence with EXP $\subseteq$ PSPACE/poly using a proof similar to above proof and then invokes the following lemma. □

**Lemma 64** EXP $\subseteq$ PSPACE/poly *if and only if* EXP $\subseteq$ PSPACE.

*Proof.* We only show that if EXP $\subseteq$ PSPACE/poly then EXP = PSPACE.

This follows since EXP has a complete set $A$ that is self-reducible [Bal90]. That is, there is a polynomial time oracle machine $M$ that decides membership in $A$ using $A$ as an oracle, with the property that on input $x$, M asks queries only to words that lexicographically precede $x$. If $A$ is in PSPACE/poly, then let $M'$ be a PSPACE machine that accepts $A$ using advice sequence $a_n$. Now a PSPACE machine can determine if a string $x$ is in $A$ by searching through all possible advice sequences $b$ of length polynomial in $|x|$ until it finds a sequence $b$ with the property that, for all strings $y$ that lexicographically precede $x$, running $M'(y)$ with advice $b$ agrees with the result of running $M(y)$, where any oracle query $z$ asked by $M$ is answered by running $M'(z)$ with advice $b$. By assumption, at least one such sequence exists, and thus one will be found. Running $M'(x)$ with this advice sequence correctly determines if $x$ is in $A$.

The other implication is trivial. □

We close this section with a brief discussion of how to define relativized measures of the form $\mathrm{KB}^A$ and $\mathrm{KF}^A$, since we have found measures of the form $\mathrm{KT}^A$ to be quite useful. Unfortunately, there are substantial difficulties that arise when attempting to provide oracle access to an alternating machine [Bus88], and thus we do not know of a useful way to define $\mathrm{KF}^A$. The situation is somewhat better for $\mathrm{KB}^A$, but it does require us to alter the conventions that we have adopted thus far.

Up until this point in the paper, we have followed the convention that space-bounded oracle Turing machines must respect the space bound on their oracle tapes. Indeed, this is the customary convention when considering classes of the form $\mathrm{PSPACE}^A$. However, there is an equally venerable tradition of allowing a logspace-Turing reduction to ask queries of polynomial length [LL76]. In order to prove the following theorem (which allows us to relate $\mathrm{KT}^A$ and $\mathrm{KB}^B$) we found that it was more appropriate to define $\mathrm{KB}^B$ in terms of oracle Turing machines that have a write-only oracle tape that is *not* subject to the space bound.

**Theorem 65** *Let $A$ and $B$ be oracles. Then* $\mathrm{KB}^B(x) \leq (\mathrm{KT}^A(x) + \log|x|)^{O(1)}$ *if and only if* $\mathrm{P}^A \subseteq \mathrm{L}^B$*/poly.*

*Proof.* Assume that, for all $x$, $\mathrm{KB}^B(x) \leq (\mathrm{KT}^A(x) + \log|x|)^k$. Let $C$ be any language in $\mathrm{P}^A$. By Theorem 4, the $2^{n+1}$-bit prefix of the characteristic sequence $\chi_C$ has $\mathrm{KT}^A$ complexity $n^{O(1)}$, and by assumption has $\mathrm{KB}^B$

complexity $O(n^c)$ for some $c$. Thus there is a description $d_n$ of length $O(n^c)$ such that for all $x \leq 2^{n+1}$, $U^{B,d_n}(x,b)$ uses space at most $O(\log n)$ (not counting the space that is used on the oracle tape, which must be at most $n^{O(1)}$) and accepts if and only if $x \in C$. The conclusion that $C \in \mathrm{L}^B$/poly now follows.

Conversely, assume that $\mathrm{P}^A \subseteq \mathrm{L}^B$/poly. Let $\mathrm{KT}^A(x) = m$. Thus there is a description $d$ of length at most $m$ such that for all $i \leq |x|$, $U^{A,d}(i,b)$ runs in time at most $m$ and accepts if and only if $x_i = b$. Let $C$ be the set $\{(d,i,b,1^j) : U^{A,d}(i,b)$accepts in time $j\}$. $C \in \mathrm{P}^A$ and thus lies in $\mathrm{L}^B$/poly. Thus there is an advice string $a$ of length $(m + \log|x|)^l$ such that queries of length $O(m + \log|x|)$ to $C$ can be answered in space $O(\log(m + \log|x|))$ using queries to $B$ and access to the advice string $a$. Consider the oracle machine $M$ that uses oracles $B$ and $(d,a,m)$, and on input $(i,b)$ determines if $(d,i,b,1^m)$ is in $C$. This machine $M$ runs in space $O(\log(m + \log|x|))$ using queries to $B$, and accepts if and only if $U^{A,d}(i,b)$ accepts (which happens if and only if $x_i = b$). The length of the description $(d,a,m)$ is $O(m^l)$, and thus $\mathrm{KB}_M(x) \leq m^l + (m + \log|x|)^c$ for some constant $c$. The theorem now follows, by the properties of the universal machine $U$. □

# 8 Concluding Comments

We began this paper with a brief historical review, pointing out that parallel investigations of computational complexity theory and resource-bounded Kolmogorov complexity date back to the dawn of the theory of NP-completeness. We cited the historical survey by Trakhtenbrot [Tra84], and recalled how the various computational tasks discussed by Trakhtenbrot (in particular, Task 4 and Task 5) relate to the themes that are the focus of the present paper.

This may be the appropriate time to mention that Trakhtenbrot also mentioned one more problem (called "Task 1" [Tra84, p. 388]) that can be captured using the tools presented in this paper. Trakhtenbrot's Task 1 involves taking a truth table $f$ as input, and determining the size of the smallest "switching circuit" for $f$ – which is roughly the same as the size of the smallest *nondeterministic* branching program for $f$. We are reluctant to introduce yet another resource-bounded Kolmogorov complexity measure into this paper, but we remark that this notion could be captured by changing the definition of KB by using a nondeterministic universal Turing machine (and indeed, some nondeterministic space-bounded Kolmogorov complexity measures that are polynomially related to KS have been introduced [All06]). Alternatively, such a measure is polynomially-related to $\mathrm{KB}^A$ where $A$ is any problem complete for NL.

It is natural to wonder if it is possible (and useful) to define even more restrictive notions of Kolmogorov complexity, in order to discuss subclasses of $\mathrm{NC}^1$. One could consider placing more restrictions on the universal alternating machine in the definition for KF complexity, for instance by restricting the number of alternations, or by making it deterministic. At first glance, it seems that one might obtain a measure that is related to depth $k$ $\mathrm{AC}^0$ circuit size for fixed $k$ – but it seems that such machines cannot do much interesting computation on input $(i,b)$ with oracle $d$ without looking at all of $i$, which means that their running time is so high that the framework developed here does not yield a very interesting measure. Is there a useful definition that can be developed to capture this notion?

For the more "limited" notions of Kolmogorov complexity KB and KF, we are not able to prove as strong intractability results as were proved for KT in [ABK+06]. However, it is not clear that this needs to be the case. For instance, although it is not known if the minimum circuit size problem is NP-complete, it *is* complete when restricted to DNF circuits [Czo99, Mas79, Fel09, AHM+08]. Is there a natural, restricted notion of Kolmogorov complexity, for which the "random" strings do indeed provide a complete set for coNP? Vazirani and Vazirani present a related problem that is complete under randomized reductions [VV83], but the computational problem that they present does not capture a very satisfactory notion of Kolmogorov complexity.

# References


[ABK+06] E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35:1467–1493, 2006.

[ACR98] A. E. Andreev, A. E. F. Clementi, and J. D. P. Rolim. A new general derandomization method. *J. ACM*, 45(1):179–213, 1998.

[AHM+08] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M. E. Saks. Minimizing disjunctive normal form formulas and $AC^0$ circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008.

[AK99] V. Arvind and J. Köbler. Graph isomorphism is low for $ZPP^{NP}$ and other lowness results. Technical Report TR99-033, Electronic Colloquium on Computational Complexity, 1999.

[AKRR03] E. Allender, M. Koucký, D. Ronneburger, and S. Roy. Derandomization and distinguishing complexity. In *Proceedings of the 18th IEEE Conference on Computational Complexity*, pages 209–220, 2003.

[AKS95] V. Arvind, J. Köbler, and R. Schuler. On helping and interactive proof systems. *International Journal of Foundations of Computer Science (IJFCS)*, 6(2):137–153, 1995.

[All89] E. Allender. Some consequences of the existence of pseudorandom generators. *Journal of Computer and System Sciences*, 39:101–124, 1989.

[All01] E. Allender. When worlds collide: Derandomization, lower bounds, and Kolmogorov complexity. In *Proc. Conf. on Found. of Software Technology and Theo. Comp. Sci. (FST&TCS)*, volume 2245 of *Lecture Notes in Computer Science*, pages 1–15, 2001.

[All06] E. Allender. NL-printable sets and nondeterministic kolmogorov complexity. *Theor. Comput. Sci.*, 355(2):127–138, 2006.

[Bal90] J. L. Balcázar. Self-reducibility. *J. Comput. Syst. Sci.*, 41(3):367–388, 1990.

[BBR99] E. Biham, D. Boneh, and O. Reingold. Breaking generalized Diffie-Hellmann modulo a composite is no easier than factoring. In *Information Processing Letters 70(2)*, pages 83–87, 1999.

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

[BFL02] H. Buhrman, L. Fortnow, and S. Laplante. Resource-bounded Kolmogorov complexity revisited. *SIAM Journal on Computing*, 31(3):887–905, 2002.

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

[BH92] H. Buhrman and S. Homer. Superpolynomial circuits, almost sparse oracles and the exponential hierarchy. In *Proc. Conf. on Found. of Software Technology and Theo. Comp. Sci. (FST&TCS)*, volume 652 of *Lecture Notes in Computer Science*, pages 116–127, 1992.

[BIS90] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within $NC^1$. *Journal of Computer and System Sciences*, 41(3):274–306, December 1990.

[Bus88] J. F. Buss. Relativized alternation and space-bounded computation. *J. Comput. Syst. Sci.*, 36(3):351–378, 1988.

[CCL94] J.-Y. Cai, A. Condon, and R. J. Lipton. PSPACE is provable by two provers in one round. *Journal of Computer and System Sciences*, 48:183–193, 1994.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 151–158, 1971.

[Czo99] S. L. A. Czort. The complexity of minimizing disjunctive normal form fomulas. Master's thesis, University of Aarhus, 1999.

[Fel09] V. Feldman. Hardness of approximate two-level logic minimization and pac learning with membership queries. *J. Comput. Syst. Sci.*, 75(1):13–26, 2009.

[FK96] L. Fortnow and M. Kummer. On resource-bounded instance complexity. *Theoretical Computer Science*, 161(1–2):123–140, 1996.

[FKS82] M. Fredman, J. Kómlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 165–169, 1982.

[FLZ94] Bin Fu, Hong-Zhou Li, and Yong Zhong. An application of the translational method. *Mathematical Systems Theory*, 27(2):183–186, 1994.

[For01] L. Fortnow. Comparing notions of full derandomization. In *Proc. IEEE Conf. on Computational Complexity '01*, pages 28–34, 2001.

[For04] L. Fortnow. A little theorem. Entry for Friday, January 30, 2004, in the Computational Complexity Weblog. Available at http://blog.computationalcomplexity.org/2004/01/little-theorem.html, 2004.

[Für82] M. Fürer. The tight deterministic time hierarchy. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 8–16, 1982.

[Für84] M. Fürer. Data structures for distributed counting. *Journal of Computer and System Sciences*, 28(2):231–243, 1984.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[HAB02] W. Hesse, E. Allender, and D.A.M. Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

[Hås99] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[Hem89] L. A. Hemachandra. The strong exponential hierarchy collapses. *J. Comput. Syst. Sci.*, 39(3):299–322, 1989.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:1364–1396, 1999.

[HS66] F. Hennie and R. Stearns. Two-tape simulation of multitape turing machines. *Journal of the ACM*, 13:533–546, 1966.

[IKW01] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *Proc. IEEE Conf. on Computational Complexity*, pages 2–12, 2001.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 181–190, 1999.

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ if $E$ requires exponential circuits: Derandomizing the XOR lemma. In *Proc. ACM Symp. on Theory of Computing (STOC) '97*, pages 220–229, 1997.

[IW98] R. Impagliazzo and A. Wigderson. Randomness vs. time: de-randomization under a uniform assumption. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 734–743, 1998.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1-3):40–56, 1982.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. New York, 1972.

[KC00] V. Kabanets and J.-Y. Cai. Circuit minimization problem. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 73–79, 2000.

[Ko91] K.-I Ko. On the complexity of learning minimum time-bounded Turing machines. *SIAM Journal on Computing*, 20:962–986, 1991.

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

[KvM02] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 2002. To appear; a preliminary version appeared in STOC '99.

[Lev73] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9:265–266, 1973.

[Lev84] L. A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.

[Lev03] L. Levin. Personal communication. 2003.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39:859–868, 1992.

[LL76] R. Ladner and N. Lynch. Relativization of questions about log space reducibility. *Mathematical Systems Theory*, 10:19–32, 1976.

[Lon82] T. J. Long. Strong nondeterministic polynomial-time reducibilities. *Theor. Comput. Sci.*, 21:1–25, 1982.

[Lup59] O. B. Lupanov. A method of circuit synthesis. *Izvestiya VUZ, Radiofizika*, 1(1):120–140, 1959.

[LV93] M. Li and P. Vitanyi. *Introduction to Kolmogorov Complexity and its Applications.* Springer, 1993.

[Mas79] W.J. Masek. Some NP-complete set covering problems. Unpublished manuscript, 1979.

[Meh82] K. Mehlhorn. On the program size of perfect and universal hash functions. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 170–175, 1982.

[Moc96] S. Mocas. Separating classes in the exponential-time hierarchy from classes in PH. *Theor. Comput. Sci.*, 158:221–231, 1996.

[MV99] P. Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 71–80, 1999.

[NR97] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudorandom functions. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 458–467, 1997.

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

[Pap94] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994.

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

[RRW94] R. P. N. Rao, J. Rothe, and O. Watanabe. Upward separation for FewP and related classes. *Inf. Process. Lett.*, 52(4):175–180, 1994.

[Rud97] S. Rudich. Super-bits, demi-bits, and $\tilde{\text{NP}}$/qpoly-natural proofs. In *Proceedings of RANDOM*, volume 1269 of *Lecture Notes in Computer Science*, 1997.

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981.

[Sch89] U. Schöning. Probabilistic complexity classes and lowness. *J. Comput. Syst. Sci.*, 39(1):84–100, 1989.

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

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 330–335, 1983.

[SU05] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, 2005.

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

[SU07] R. Shaltiel and C. Umans. Low-end uniform hardness vs. randomness tradeoffs for AM. In *STOC*, pages 430–439, 2007. To appear in SIAM J. Comput.

[Tra84] B. A. Trakhtenbrot. A survey of Russian approaches to perebor (brute-force searches) algorithms. *IEEE Annals of the History of Computing*, 6(4):384–400, 1984.

[Vin04] N. V. Vinodchandran. $\text{AM}_{\text{exp}} \not\subseteq (\text{NP} \cap \text{coNP})/\text{poly}$. *Inf. Process. Lett.*, 89(1):43–47, 2004.

[vM03] D. van Melkebeek. Personal communication. 2003.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.

[VV83] U. V. Vazirani and V. V. Vazirani. A natural encoding scheme proved probabilistic polynomial complete. *Theoretical Computer Science*, 24:291–300, 1983.



ECCC ISSN 1433-8092

http://eccc.hpi-web.de/