Electronic Colloquium on Computational Complexity, Report No. 139 (2010)

Limits on the Computational Power of Random Strings

Eric Allender[*]
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08855, USA
allender@cs.rutgers.edu

Luke Friedman[*]
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08855, USA
lbfried@cs.rutgers.edu

William Gasarch
Dept. of Computer Science
University of Maryland
College Park, MD, 20742
gasarch@cs.umd.edu

September 17, 2010

Abstract

Let $C(x)$ and $K(x)$ denote plain and prefix Kolmogorov complexity, respectively, and let R_C and R_K denote the sets of strings that are "random" according to these measures; both R_K and R_C are undecidable. Earlier work has shown that every set in NEXP is in NP relative to both R_K and R_C, and that every set in BPP is polynomial-time truth-table reducible to both R_K and R_C [ABK06a, BFKL10]. (All of these inclusions hold, no matter which "universal" Turing machine one uses in the definitions of $C(x)$ and $K(x)$.) Since each machine U gives rise to a slightly different measure C_U or K_U, these inclusions can be stated as:

- $\mathrm{BPP} \subseteq \mathrm{DEC} \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{C_U}\}$.
- $\mathrm{NEXP} \subseteq \mathrm{DEC} \cap \bigcap_U \mathrm{NP}^{R_{C_U}}$.
- $\mathrm{BPP} \subseteq \mathrm{DEC} \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{K_U}\}$.
- $\mathrm{NEXP} \subseteq \mathrm{DEC} \cap \bigcap_U \mathrm{NP}^{R_{K_U}}$.

(Here, "DEC" denotes the class of decidable sets.)

It remains unknown whether DEC is *equal* to $\bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{C_U}\}$.

In this paper, we present the *first* upper bounds on the complexity of sets that are efficiently reducible to R_{K_U}. We show:

- $\mathrm{BPP} \subseteq \mathrm{DEC} \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{K_U}\} \subseteq \mathrm{PSPACE}$.
- $\mathrm{NEXP} \subseteq \mathrm{DEC} \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq \mathrm{EXPSPACE}$.

This also provides the first quantitative limits on the applicability of uniform derandomization techniques.

[*]Supported in part by NSF Grants CCF-0830133 and CCF-0832787.

ISSN 1433-8092

1 Introduction

The motivation for the research reported here comes from a body of work that seems to indicate that the set of "random strings" is quite powerful, where "randomness" is defined in terms of Kolmogorov complexity:

Notation 1.1 Let $C(x)$ be the Kolmogorov complexity of the string x. Then

$$R_C = \{x : C(x) \geq |x|\}.$$

(More complete definitions of Kolmogorov complexity can be found in Section 2. Each universal Turing machine U gives rise to a slightly different measure C_U, and hence to various closely-related sets R_{C_U}.)

In particular, consider the following curious inclusions:

Theorem 1.2 *The following inclusions hold:*

- $\text{BPP} \subseteq \{A : A \leq^{\text{p}}_{tt} R_C\}$ *[BFKL10]*
- $\text{PSPACE} \subseteq \text{P}^{R_C}$ *[ABK$^+$06b].*
- $\text{NEXP} \subseteq \text{NP}^{R_C}$ *[ABK06a].*

We call these inclusions "curious" because the upper bounds that they provide for the complexity of problems in BPP, PSPACE and NEXP is not even computable; thus at first glance these inclusions may seem either trivial or nonsensical. However, in spite of the fact that R_C is not computable, it is not straightforward how to make use of R_C via an *efficient* reduction, and hence Theorem 1.2 does not seem trivial. After the failure of attempts to squeeze larger complexity classes into NP^{R_C}, it was even suggested that it might be possible to *characterize* complexity classes in terms of efficient reductions to R_C [ABK06a].

What might such a characterization look like? Since P^{R_C} contains undecidable problems (such as R_C itself), it would surely be necessary to restrict attention to only *decidable* sets. Yet this is not enough, since there are arbitrarily complex decidable sets that are reducible to R_C via very restrictive polynomial-time reductions (polynomial-time disjunctive truth-table reductions, denoted $\leq^{\text{p}}_{dtt}$) [ABK06a, Theorem 16].

A way out of this trap is provided by the insight that these "arbitrarily complex sets" depend in a crucial way on the particular choice of the universal Turing machine U that is picked to provide a formal definition of the Kolmogorov complexity function $C(x)$ (in contrast to the usual situation in Kolmogorov complexity, where one is able to argue that the choice of universal Turing machine is irrelevant). Indeed, the choice of universal Turing machine *is* irrelevant for the inclusions in Theorem 1.2. If we let DEC denote the class of decidable sets, it was shown that one can characterize P as $\text{DEC} \cap \bigcap_U \{A : A \leq^{\text{p}}_{dtt} R_{C_U}\}$. That is, if the choice of U is "factored out" in this way, characterizations of complexity classes *can* be obtained in some cases.

This leads us to the question of whether NEXP is not merely contained in $\text{DEC} \cap \bigcap_U \text{NP}^{R_{C_U}}$ (which is known [ABK06a]), but is actually *equal* to this class. We are not able to answer this question. Worse, we are not able to say whether or not *every* decidable problem lies in $\bigcap_U \text{NP}^{R_{C_U}}$. Still worse, we cannot disprove that the halting problem is in $\text{NP}^{R_{C_U}}$, or in $\text{P}^{R_{C_U}}$, or that it is even $\leq^{\text{p}}_{tt}$-reducible to R_{C_U} for every universal Turing machine U.

1.1 Statement of the Main Theorem

Instead, we make progress on the analogous question, posed in terms of *prefix* complexity $K(x)$, which is the other most-widely-studied variant of Kolmogorov complexity. (See Section 2 for definitions.) Let $R_K = \{x : K(x) \geq |x|\}$ (or, when we need to be explicit about the choice of universal prefix machine U, let $R_{K_U} = \{x : K_U(x) \geq |x|\}$). It is known that $\mathrm{NEXP} \subseteq \bigcap_U \mathrm{NP}^{R_{K_U}}$ [ABK06a], and we show that this inclusion is not far from being optimal, in the following sense. EXPSPACE is the smallest space complexity class that is known to contain NEXP. We show that every decidable set in $\bigcap_U \mathrm{NP}^{R_{K_U}}$ lies in EXPSPACE.

The proof of Buhrman *et al.* [BFKL10], showing that $\mathrm{BPP} \subseteq \{A : A \leq^{\mathrm{p}}_{tt} R_C\}$ also shows that $\mathrm{BPP} \subseteq \bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{K_U}\}$. We show that every decidable set that is in $\bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{K_U}\}$ is in PSPACE (which is the smallest space complexity class that is known to contain BPP). Thus our result shows that the inclusion of [BFKL10] is not too far from optimal. (Previously, absolutely no upper bound had been known for the complexity of this class.)

A stronger inclusion is possible for "monotone" truth-table reductions ($\leq^{\mathrm{p}}_{mtt}$). We show that $\mathrm{DEC} \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{mtt} R_{K_U}\} \subseteq \mathrm{coNP}$.

Our work builds on a construction by Muchnik and Positselsky [MP02]. (See also [AFG10] for an alternative exposition of their main result that we utilize here.) They showed that there is an "optimal prefix free entropy function" H such that the halting problem is not truth-table reducible to the overgraph of H. This is more-or-less equivalent to showing that there is a universal prefix Turing machine U such that the halting problem is not truth-table reducible to the set $\{(x, r) : K_U(x) < r\}$, which in turn immediately implies that the halting problem is not truth-table reducible to R_{K_U}. This contrasts with the fact that, for *every* universal Turing machine U, the halting problem *is* truth-table reducible to R_{C_U} [Kum96]. It also contrasts with the fact that there is a different universal prefix Turing machine U' such that the halting problem is truth-table reducible to $R_{K_{U'}}$ [ABK06a], leading to the rather uncomfortable situation that the question of whether or not R_K is complete under truth-table reductions depends on the choice of universal prefix Turing machine that one uses in order to define the prefix Kolmogorov complexity function $K(x)$. A detailed study of analogous questions that arise when one considers other variants of Kolmogorov complexity (such as various types of "monotone" Kolmogorov complexity) has been carried out by Day [Day09].

Until now, it was not at all clear that it was reasonable to consider the class of problems efficiently reducible to R_K to be a complexity class, since it was conceivable that *every* decidable problem was efficiently reducible to R_K. In contrast, combining our results with prior work we now have the following:

- $\mathrm{BPP} \subseteq \mathrm{DEC} \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{K_U}\} \subseteq \mathrm{PSPACE} \subseteq \mathrm{DEC} \cap \bigcap_U \mathrm{P}^{R_{K_U}}$.
- $\mathrm{NEXP} \subseteq \mathrm{DEC} \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq \mathrm{EXPSPACE}$.

In particular, note that PSPACE is sandwiched in between the classes of decidable problems that are reducible to R_K via truth-table and Turing reductions.

We postpone until Section 4 the discussion of what these results signify. In that section, we also discuss avenues for future research suggested by this work. We do believe that these results take us closer to the goal of characterizing complexity classes in terms of efficient reductions to R_K.

2 Background and Definitions

Definition 2.1 If f is a function mapping some domain to the naturals $\mathbb{N}$, then $ov(f)$, the overgraph of f, is

$$\{(x, y) : f(x) \leq y\}.$$

The following example gives a familiar overgraph, related to (plain) Kolmogorov complexity:

Example 2.2 Fix a Universal Turing machine U. Let $C(x)$ be the size of the shortest s such that $U(s) = x$. Note that

$$ov(C) = \{(x, y) : \text{ there is an } s, |s| \leq y \text{ such that } U(s) = x\}.$$

Note that $ov(C)$ is c.e.

This paper is concerned much more with *prefix complexity.*

Definition 2.3

1. A *prefix Turing machine* is a Turing machine M such that, for all x, if $M(x)$ halts then, for all $y \neq \lambda$, $M(xy)$ does not halt. That is, the domain of M is a prefix code.
2. Let M be a prefix Turing machine. Define $K_M(x)$ to be the size of the shortest s such that $M(s) = x$.
3. A *universal prefix Turing machine* is a prefix Turing machine U such that, for any prefix Turing machine M, there is a constant c such that for all $x, K_U(x) \leq K_M(x) + c$.

Example 2.4 Let M be a prefix Turing machine. Note that

$$ov(K_M) = \{(x, y) : \text{ there is an } s, |s| \leq y \text{ such that } M(s) = x\}.$$

Note that $ov(K_M)$ is c.e.

Definition 2.5 We select some universal prefix Turing machine U and call $K_U(x)$ the *prefix complexity of* x. As usual, we delete the subscript in this case, and denote the prefix complexity of x by $K(x)$. The arbitrary choice of U affects $K(x)$ by at most an additive constant, and in most instances where prefix complexity is studied, the particular choice of U is deemed to be irrelevant. Note however, that in this paper it is important to consider K_U for various machines U.

The following definition was used implicitly by Muchnik and Positselsky [MP02]:

Definition 2.6 A *Prefix Free Entropy Function* f is a function from $\{0, 1\}^*$ to $\mathbb{N}$ such that

- $\sum_{x \in \{0,1\}^*} 2^{-f(x)} \leq 1$ and
- $ov(f)$ is c.e.

The canonical example of a prefix free entropy function is $K(x)$. (K can be seen to be a prefix free entropy function by appeal to the Kraft Inequality; see e.g. [LV08, Theorem 1.11.1].)

Note that if f is a prefix free entropy function, then 2^{-f} is a special case of what Li and Vitányi call a *Lower Semicomputable Discrete Semimeasure* [LV08, Definition 4.2.2]. We recall the *Coding Theorem* (see [LV08, Theorem 4.3.3]), the proof of which yields the following important relationship between prefix free entropy functions and prefix complexity.

Theorem 2.7 *Let f be a prefix free entropy function. Given a machine computing f, one can construct a prefix machine M such that $f(x) = K_M(x) - 1$.*

Proof: The Coding Theorem, as stated and proved in [LV08, Theorem 4.3.3], gives only the *inequality* $f(x) \leq K_M(x) - 3$, where 2^{-f} is a lower semicomputable semimeasure. However, since we are dealing with the special case where f is a prefix free entropy function, we can dispense with some of the technicalities in the proof of [LV08, Theorem 4.3.3]. In particular (using the terminology utilized by Li and Vitányi in their proof) we use intervals I_x, and observe that each I_x contains a *binary* interval having length (exactly) half the length of I_x. Modifying the proof of [LV08, Lemma 4.3.3], to give the label of this interval (instead of the label of the *entire* interval in the case where I_x is itself a binary interval) yields the desired prefix machine. ▮

We will make use of the following easy propositions.

Proposition 2.8 *Let U and U' be prefix Turing machines. Then there is a prefix machine U'' such that $K_{U''}(x) = \min(K_U(x), K_{U'}(x)) + 1$.*

Proof: The domain of U'' is $\{1x : x \text{ is in the domain of } U\} \cup \{0x : x \text{ is in the domain of } U'\}$. ▮

Proposition 2.9 *Given any machine U and constant c, there is a machine U' such that $K_U(x) + c = K_{U'}(x)$*

Proof: The domain of U' is $\{0^c x : x \text{ is in the domain of } U\}$. ▮

In this paper we consider four types of reductions: truth table reductions, monotone truth table reductions, anti-monotone reductions, and Turing reductions.

- *Truth-table reductions.* For a complexity class $\mathcal{R}$ and languages A and B, we say that A *$\mathcal{R}$-truth-table-reduces* to B ($A \leq_{tt}^{\mathcal{R}} B$) if there is a function q computable in $\mathcal{R}$, such that, on an input $x \in \{0,1\}^*$, q produces an encoding of a circuit λ and a list of queries $q_1, q_2, \ldots q_m$ so that for $a_1, a_2, \ldots, a_m \in \{0,1\}$ where $a_i = B(q_i)$, it holds that $x \in A$ if and only if $\lambda(a_1 a_2 \cdots a_m) = 1$. If the function q is polynomial time computable, we say that A *polynomial-time-truth-table-reduces* to B ($A \leq_{tt}^{p} B$).

- *Monotone truth-table reductions.* In the scenario above, if the circuit λ computes a monotone function (i.e. changing any input bit of the function from 0 to 1 cannot change the output of the function from 1 to 0), then we say that A *$\mathcal{R}$-monotone-truth-table-reduces* to B ($A \leq_{mtt}^{\mathcal{R}} B$). If the function q is polynomial time computable, we say that A *polynomial-time-monotone-truth-table-reduces* to B ($A \leq_{mtt}^{p} B$).

- *Anti-monotone truth-table reductions.* In the scenario above, if the circuit λ computes an anti-monotone function (i.e. $\neg\lambda$ is monotone), then we say that A *$\mathcal{R}$-anti-monotone-truth-table-reduces* to B ($A \leq_{amtt}^{\mathcal{R}} B$). If the function q is polynomial time computable, we say that A *polynomial-time-anti-monotone-truth-table-reduces* to B ($A \leq_{amtt}^{p} B$).

- *Turing reductions.* We say that A *$\mathcal{R}$-Turing reduces* to B ($A \leq_{T}^{\mathcal{R}} B$) if there is an oracle Turing machine in class $\mathcal{R}$ that accepts A when given B as an oracle.

3 Main Results

The proof of the following theorem is a modification of Theorem 2.7 of [MP02] combined with ideas from [ABK06a]

Theorem 3.1

$$\mathrm{DEC} \cap \bigcap_U \{A : A \leq_{tt}^p R_{K_U}\} \subseteq \mathrm{PSPACE}$$

Proof:
We will actually prove the statement

$$\mathrm{DEC} \cap \bigcap_U \{A : A \leq_{tt}^p ov(K_U)\} \subseteq \mathrm{PSPACE} \tag{1}$$

The theorem follows, since any query "$x \in R_{K_U}$?" can always be modified to the equivalent query "$(x, |x|-1) \notin ov(K_U)$?", so

$$\mathrm{DEC} \cap \bigcap_U \{A : A \leq_{tt}^p R_{K_U}\} \subseteq \mathrm{DEC} \cap \bigcap_U \{A : A \leq_{tt}^p ov(K_U)\}$$

To prove the statement (1) it suffices to show that

$$L \notin \mathrm{PSPACE} \Rightarrow \exists \text{ a universal prefix machine } U \text{ s.t. } L \not\leq_{tt}^p ov(K_U) \tag{2}$$

Let $L \notin$ PSPACE be given. Our strategy will be to use a diagonalization technique to carefully construct a universal prefix machine U such that $L \not\leq_{tt}^p ov(K_U)$. To do this we will use the standard prefix complexity function K, together with a function $F : \{0,1\}^* \to \mathbb{N}$ that we will construct, to form a function $H : \{0,1\}^* \to \mathbb{N}$ with the following properties.

1. F is a total function and $ov(F)$ is c.e.
2. $H(x) = \min(K(x)+4, F(x)+2)$.
3. $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{4}$.
4. $L \not\leq_{tt}^p ov(H)$.

Claim 1: Given the above properties, $H = K_{U'}$ for some universal prefix machine U' (which by Property 4 ensures that (2) holds).

Proof: By Property 3 we have that $\sum_{x\in\{0,1\}^*} 2^{-F(x)+2} \leq \frac{1}{4}$. Therefore $\sum_{x\in\{0,1\}^*} 2^{-F(x)} \leq 1$, which along with Property 1 means that F is a prefix free entropy function. By Proposition 2.7 we then have that $F+1$ is K_M for some prefix machine M. By Proposition 2.9 we have that $K(x)+3$ is $K_{U''}$ for some universal prefix machine U''. Thus, by Lemma 2.8, $H(x) = \min(K(x)+4, F(x)+2) = \min(K(x)+3, F(x)+1)+1$ is $K_{U'}$ for some universal prefix machine U'. ▮

It remains to show that for a given $L \notin$ PSPACE we can always construct an H with the desired properties. Let us first informally discuss the ideas before providing the formal construction.

Our control over H comes from our freedom in constructing the function F. The construction will occur in stages – at any given time in the construction there will be a "current" version of F which we will denote by F^*. Similarly, there will be a "current" version of K denoted by K^*, which represents our knowledge of K at a given stage. At all times, H^*, our "current" version of H, will be defined as $\min(K^*(x)+4, F^*(x)+2)$.

Originally we set $F^*(x) = 2|x|+2$ and K^* as the empty function. At each stage of the construction we will assume that a new element (x, y) is enumerated into $ov(K)$ according to some fixed enumeration of $ov(K)$. (This is possible since $ov(K)$ is c.e.) When this occurs K^* is updated by setting $K^*(x) = \min(K^*(x), y)$. (Since K^* is a partial function, it is possible that $K^*(x)$ was previously undefined. In this case we set $K^*(x) = y$.) Similarly, during the construction at times we will modify F by enumerating elements into $ov(F)$. Whenever we enumerate an element (x, y) into $ov(F)$, F^* is updated by setting $F^*(x) = \min(F^*(x), y)$.

Let $\gamma_1, \gamma_2, \ldots$ be a list of all possible polynomial time truth table reductions from L to $ov(H)$. This is formed in the usual way: we take a list of all Turing Machines and put a clock of $n^i + i$ on the ith one and we will interpret the output as an encoding of a Boolean circuit on atoms of the form "$(z, r) \in ov(H)$".

We need to ensure that $L \not\leq^p_{tt} ov(H)$. We break this requirement up into an infinite number of requirements:

$$R_e : \gamma_e \text{ is not a polynomial-time tt-reduction of } L \text{ to } ov(H)$$

At stage e of the construction we will begin to attempt to satisfy the requirement R_e. For a particular input x, let $\gamma_e(x)$ be an encoding of a circuit $\lambda_{e,x}$. The output of the circuit $\lambda_{e,x}$ is determined by the truth values of the atoms "$z \in ov(H)$" that label the inputs to the circuit. Define $\lambda_{e,x}[H']$ to be the truth value obtained by taking the circuit $\lambda_{e,x}$ and for each atom "$(z, r) \in ov(H)$" using the truth value of "$(z, r) \in ov(H')$ in its place. In order to satisfy the requirement R_e, we would like to find some x such that $\lambda_{e,x}[H] \neq L(x)$, where $L(x)$ is the characteristic function of L. The problem is that at a given stage s we can "guess" at the value of $\lambda_{e,x}[H]$ by computing $\lambda_{e,x}[H^*]$, but in general we cannot know the value of $\lambda_{e,x}[H]$ for sure, because as H^* evolves the value of $\lambda_{e,x}[H^*]$ may change. The main difficulty is that the function K is out of our control and determining whether $(z, r) \in ov(K)$ is in general an uncomputable task.

We do have *some* influence over the situation though due to our control of F. Indeed, for any atom "$(z, r) \in ov(H)$", we can ensure that the truth value of the atom is 1 by enumerating $(z, r-2)$ into $ov(F)$. (Note that for all x, the value of $H^*(x)$ can only decrease over time). We have to be careful about making these types of changes though; if we are too liberal in modifying F we may violate the condition $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq 1/4$ in the process. Thus the construction becomes a balancing act – we will try to use F to satisfy R_e while at the same time maintaining the invariant that $\sum_{x \in \{0,1\}^*} 2^{-H^*(x)} \leq 1/4$. (In particular, if F_s is the function F^* at the beginning of stage s, for all x we will not want $\lim_{s \to \infty} F_s(x)$ to be very much smaller than $K(x)$).

As part of our solution, for each R_e we will find a suitable witness x and set up a game $\mathcal{G}_{e,x}$ played between us (making moves by enumerating elements into $ov(F)$), and K, who makes moves by enumerating elements into $ov(K)$. (Even though elements are obliviously enumerated into $ov(K)$ according to some fixed enumeration we will treat K as if it is a willful adversary). The witness x will be chosen so that we have a winning strategy: as long as K continues to make legal moves we can respond with changes to F (our own legal moves) that both assure that R_e is satisfied and that $\sum_{x\in\{0,1\}^*} 2^{-H^*(x)} \leq 1/4$.

It is possible that K will cheat by enumerating elements into $ov(K)$ in such a way that it plays an illegal move. In this case we will simply destroy the game $\mathcal{G}_{e,x}$ and start all over again with a new game $\mathcal{G}_{e,x'}$, using a different witness x'. However we will be able to show that if K cheats infinitely often on games associated with a particular requirement R_e, then $\sum_{x\in\{0,1\}^*} 2^{-K(x)}$ diverges; therefore it can only happen finitely often.

The requirements $R_1, R_2, R_3, \ldots$ are listed in priority ordering. If during stage s a move is played on a game $\mathcal{G}_{e,x}$, we say that R_e is "acting". In this case for all $e < e' \leq s$, if $\mathcal{G}_{e',y}$ is the game associated with $R_{e'}$ currently being played, we destroy this game and start a new game $\mathcal{G}_{e',y'}$ with some new witness y'. When this happens we say that each of the $R_{e'}$ has been "injured" by R_e. The reason this works in the end is that at some point $R_1, R_2, \ldots, R_{e-1}$ have stopped acting, so R_e will no longer ever be injured by some higher priority requirement.

3.1 Description of the Game

Before we present the other details of the formal construction, let us describe one of the games $\mathcal{G}_{e,x}$ in more depth and provide some analysis of the game. Let the inputs to the Boolean circuit $\lambda_{e,x}$ (encoded by $\gamma_e(x)$) be labeled by the atoms $\{(z_1, r_1), \ldots, (z_k, r_k)\}$. Let $X_e = \{z_1, \ldots, z_k\}$. Note that the queries in this reduction are of the form: "Is $H(z_i) \leq r_i$?". If $H^*(z_i) \leq r_i$ then we already know $H(z_i) \leq r_i$, so we can replace that input to the circuit with the value $TRUE$ and simplify the circuit accordingly. Renumber the z's, rename k to again be the number of questions, and rename X_e to be the set of all z's being asked about. When we are done we have $\{(z_1, r_1), \ldots, (z_k, r_k)\}$ and we know that $(\forall z_i \in X_e)[H^*(z_i) > r_i]$.

We make one more change to X_e. If there exists an element z_i such that $z_i \in X_e$ and $z_i \in X_{e'}$ for some $e' < e$, then changing H^* on the value z_i during the game $\mathcal{G}_{e,x}$ could affect the game associated with the requirement $R_{e'}$, which would upset our priority ordering. Hence we will take

$$X_e = X_e - \bigcup_{e'<e} X_{e'}.$$

This will ensure that R_e cannot injure any $R_{e'}$ with $e' < e$.

Let $H^*_{e,x}$ be the function H^* when the game $\mathcal{G}_{e,x}$ is first constructed. Let $\epsilon = 2^{-e-i_e-5}$. (How i_e is determined will be explained later). The game $\mathcal{G}_{e,x}$ is played on a labeled DAG. The label of each node of the DAG has the following two parts:

1. A function h that maps X_e to $\mathbb{N}$. The function h provides conjectured values for H restricted to X_e. The function h will be consistent with $H^*_{e,x}$ in that $(\forall i)[h(z_i) \leq H^*_{e,x}(z_i)]$.

2. A truth value VAL, which is the value of $\lambda_{e,x}$ assuming that $(\forall z \in X_e)[H(z) = h(z)]$. Note that this will be either YES or NO indicating that either, under assumption $(\forall z \in X_e)[H(z) = h(z)]$, $\lambda_{e,x}$ thinks $x \in L$ or thinks $x \notin L$.

There is a separate node in the DAG for every possible such function h.

Let us place an upper bound on the size of this DAG. The set X_e contains at most $|x|^e$ queries. For any query z_i, $H(z_i)$ can take at most $2|z_i| + 4$ values (since it is always bounded by $F^*(z_i) + 2$). Note also that $|z_i| \leq |x|^e$. Thus there are at most $(2|x|^e + 4)^{|x|^e}$ possible choices for h. For all large x this is bounded by $2^{|x|^{2e}}$, so note that we can represent a *particular* node in the DAG with $|x|^{2e} + 1$ bits.

We now describe the start node and how to determine the edges of the DAG.

1. There is a node (h, VAL) where $h = H^*_{e,x}$ restricted to X_e. This is the start node and has indegree 0.

2. There is an edge from (h, VAL) to (h', VAL') if for all $z_i \in X_e$, $h(z_i) \geq h'(z_i)$ (so it is possible that H^* could at some point evolve from $H^*_{e,x}$ to h, and then at a later point evolve from h to h'.)

The game $\mathcal{G}_{e,x}$ is played between two players, the YES player and the NO player. Each player has a score, which originally is zero, and represents how much the player has been penalized so far in the game. (In other words a high score is bad). The game starts with a token placed on the start node. The YES player goes first (although this choice is arbitrary – one could also have the first player be the one whose value differs from the value of the start node), after which the players alternate moves.

On a given turn a player can either leave the token where it is or move the token to a new node in the DAG. Suppose a player moves the token from a node t to a node t', where h is the function labeling t and h' is the function labeling t'. In this case we add $\sum_{z_i \in X_e} 2^{-h'(z_i)} - 2^{-h(z_i)}$ to the player's score.

A player can legally move the token from node t to t' if

1. There is an edge from t to t' in the game DAG.

2. The score of the player after making the move does not exceed ϵ.

The YES player wins if the token ends up on a node such that $\mathit{VAL} =$ YES, and the NO player wins if the token ends up on a node such that $\mathit{VAL} =$ NO. Note that because the game is entirely deterministic, for a given game $\mathcal{G}_{e,x}$, either the YES player has a winning strategy or the NO player has a winning strategy. Let $\mathit{val}(\mathcal{G}_{e,x}) = 1$ if the YES player has a winning strategy on the game $\mathcal{G}_{e,x}$ and $\mathit{val}(\mathcal{G}_{e,x}) = 0$ otherwise.

During the actual construction the games will be played between us (the construction) trying to make the computation go one way, and K (which we do not control) trying to make it go (perhaps) another way. We will always ensure that we play the side of the player who has the winning strategy in the game. We will effect our moves by enumerating elements into $ov(F)$, which changes F^* and hence H^*. (To move the token to a node labeled with the function h, we modify H^* so that h equals H^* restricted to the set X_e) The K moves will occur when a new element is enumerated into $ov(K)$ at the beginning of each stage, which changes K^* and hence H^*. (In this case K is moving the token to the node in the game DAG labeled by the new H^*).

The key is that the players' scores measure how much the sum $\sum_{x \in \{0,1\}^*} 2^{-H^*(x)}$ has gone up, which we bound by not allowing a player's score to exceed ϵ. (Of course K is oblivious to the rules of the game and will at times cheat – we take this into account as part of our analysis.) One final note: it is possible that K will simply stop playing a game in the middle and never make another move. This will not matter to us in the construction; what is important is that we have a winning strategy and if K does move we always have a winning response.

3.2 Formal Construction

Stage 0:

- Let F^* initially be defined as $F^*(x) = 2|x| + 2$.

- Let K be the standard prefix complexity function, and K^* initially be the empty function.
- At all times throughout the construction, we have that $H^*(x) = \min(K^*(x) + 4, F^*(x) + 2)$. (In the case where $K^*(x)$ is undefined, let $H^*(x) = F^*(x) + 2$). We will denote by H_s the function H^* as it is at the beginning of stage s.
- For all e, set $i_e = 0$. In the future i_e will be the number of times R_e has been injured by the requirements $R_{e'}$, $1 \leq e' \leq e - 1$.
- Let $HEAP$ be an object that enumerates strings in the normal lexicographical order. So the first time that $HEAP$ is called it returns the string '0', the second time it returns '1', then '00', '01', etc.

Stage s (for $s \geq 1$):

Let (x', y') be the sth element in the fixed enumeration of $ov(K)$. Update K^* by setting $K^*(x') = \min(K^*(x'), y')$. (This automatically updates H^* as well)

(**) For $1 \leq e \leq s$ we consider requirement R_e.

There are two possibilities:

1. There is no game associated with R_e in progress. This can occur because either $e = s$ or because the game associated with R_e was destroyed during the last round.

   In this case we continue to get strings from $HEAP$ until a string x is found that has the following property:

   - If we define a new game $\mathcal{G}_{e,x}$ using the current H^*, then $val(\mathcal{G}_{e,x}) \neq L(x)$, where $L(x)$ is the characteristic function of L.

   We will later show in Claim 4 that in a finite number of steps we will always find such an x.

   Once we have found the string x, construct the game $\mathcal{G}_{e,x}$ in the way described in the previous section and begin the game. For this game, we will play as the YES player if $val(\mathcal{G}_{e,x}) = 1$, and as the NO player if $val(\mathcal{G}_{e,x}) = 0$. (That is, we will always play as the player who has a winning strategy for the game).

2. The game associated with R_e is already in progress (again call this game $\mathcal{G}_{e,x}$).

   There are a few sub-cases to consider.

   (a) K has not changed on X_e at all since the last stage. We do nothing.

   (b) K on X_e has changed so much that the move K plays causes his score to exceed ϵ (i.e. K "cheats"). In this case we destroy the game $\mathcal{G}_{e,x}$.

   (c) It is our turn in $\mathcal{G}_{e,x}$, either because the token is on the start node of the DAG and we are the YES player, or because K on X_e has changed in a way that his score does not exceed ϵ, so he has played a legal move.

   In this case we play the move dictated by our winning strategy (which we can assume we have stored or which we can recompute each time). This may be to do nothing or it may involve moving the token to a new node, in which case we change H^* accordingly by enumerating elements into $ov(F)$.

If either case (b) or (c) occurs, we say that "R_e is acting", in which case for all e' such that $s \geq e' \geq e+1$: Set $i_{e'}$ to $i_{e'}+1$ and destroy the game associated with $R_{e'}$. Note: i_e does *not* change in case (b), even though $\mathcal{G}_{e,x}$ is destroyed.

If R_e acts then proceed to the next stage. Otherwise return to (**) and process the next e.

END OF CONSTRUCTION

Claim 2: For all e, each R_e acts at most finitely often and is satisfied.
Proof of Claim 2:

We prove this by induction on e. Assume that the claim is true for all $e' < e$. We show that the claim holds for e. By the inductive hypothesis there exists a stage s' such that, for all $s \geq s'$, for all $e' < e$, $R_{e'}$ does not act at stage s.

Let $\mathcal{G}_{e,x}$ be the game associated with R_e at stage s. If $\mathcal{G}_{e,x}$ is never destroyed in a later stage, then (by construction) R_e will be satisfied (since for $H = \lim_{s\to\infty} H_s$, our winning strategy ensures that $\gamma_{e,x}[H]$ evaluates to YES if and only if x is not in L).

Suppose that $\mathcal{G}_{e,x}$ is destroyed at some point. Then, since by the inductive hypothesis R_e cannot be injured by higher priority requirements, by the rules of the construction it must be that the "player" K cheats on the game $\mathcal{G}_{e,x}$. In doing this, K is adding at least $\epsilon = 2^{-e-i_e-5}$ to $\sum_{x\in X_e} 2^{-K^*(x)}$ and hence to $\sum_{x\in\{0,1\}^*} 2^{-K^*(x)}$.

Once K cheats and destroys the game $\mathcal{G}_{e,x}$, a new witness x' is found and a new game $\mathcal{G}_{e,x'}$ is started during the next stage. Once again if this game is never destroyed then R_e will be satisfied. If this game is also later destroyed, this means that another $\epsilon = 2^{-e-i_e-5}$ is added to $\sum_{x\in\{0,1\}^*} 2^{-K^*(x)}$. The crucial observation is that since i_e did not change, this is the same ϵ as before.

This process keeps repeating. If the games associated with R_e continue to be destroyed indefinitely, then $\sum_{x\in\{0,1\}^*} 2^{-K(x)} \geq \epsilon + \epsilon + \cdots$ so it diverges. This contradicts K being a prefix free entropy function.

Hence eventually there is some game $\mathcal{G}_{e,x''}$ that is played throughout all the rest of the stages. Since the game DAG for $\mathcal{G}_{e,x''}$ is finite, this means that eventually R_e stops acting and is satisfied.
End of Proof of Claim 2

Claim 3: $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{4}$.
Proof of Claim 3:

We have that $H = \lim_{s\to\infty} H_s$, and thus

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} = \sum_{x\in\{0,1\}^*} 2^{-H_1(x)} + \sum_{s\geq 1} \sum_{x\in\{0,1\}^*} (2^{H_{s+1}(x)} - 2^{H_s(x)}).$$

That is, we can bound the sum by bounding H_1 and by bounding the changes that occur to H over the lifetime of the construction (some of which are made by K, and some by F).

Originally $H^*(x) = F^*(x) + 2 = 2|x| + 4$ for all x, so $\sum_{x\in\{0,1\}^*} 2^{-H_1(x)} = \frac{1}{16}$.

The total contribution that K can make to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ is bounded by $\sum_{x\in\{0,1\}^*} 2^{-K(x)+4}$. Since K is a prefix free entropy function, this contribution is at most 1/16.

Let us now consider the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ due to movements by F on games on which K eventually cheats. On each of these games F contributes less to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ than K, so from the above we can say that the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ while playing these games is at most 1/16.

Finally, let us consider the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ due to movements by F on games that are never destroyed, or are destroyed by higher priority requirements. Consider such games associated with a particular requirement R_e. During the first such game associated with R_e, $i_e = 0$, so F can change at most $\epsilon = 2^{-e-i_e-5} = 2^{-e-5}$. On the second such game associated with R_e, $i_e = 1$, so F can change at most $\epsilon = 2^{-e-6}$. Generalizing, we see that the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ on such games associated with R_e is

$$\sum_{i=5}^{\infty} 2^{-e-i} = 2^{-e} \sum_{i=5}^{\infty} 2^{-i} = 2^{-e-4}$$

Hence, the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ on games that are never destroyed, or are destroyed by higher priority requirements is at most $\sum_{e=1}^{\infty} 2^{-e-4} = \frac{1}{16}$.

Putting all this information together we have that

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} \le \frac{1}{16} + \frac{1}{16} + \frac{1}{16} + \frac{1}{16} \le \frac{1}{4}$$

End of Proof of Claim 3

All that remains is to show that whenever a new game associated with R_e is constructed, a witness x with the appropriate property can be found. Recall that we are searching for an x such that

- If we define a new game $\mathcal{G}_{e,x}$ using the current H^*, then $val(\mathcal{G}_{e,x}) \neq L(x)$, where $L(x)$ is the characteristic function of L.

Claim 4: *In the above situation, a witness with the desired property can always be found in a finite number of steps*

Proof of Claim 4:

Suppose for contradiction that during some stage s for some e we are not able to find such an x. Let y be the last string that was taken from $HEAP$ before this endless search for an x began. This means that for all strings $x > y$ (under the normal lexicographical ordering), when we construct the game $\mathcal{G}_{e,x}$, $val(\mathcal{G}_{e,x}) = L(x)$. But this gives a PSPACE algorithm to decide L, which we now describe.

Hardwire in the value of $L(x)$ for every $x \le y$. Also hardwire in the function H^* at this moment in the construction and $X_{e'}$ for all $e' \le e$. (It is possible to hardwire in H^* because at any given moment in the construction only finitely many elements have been enumerated into $ov(F)$ and $ov(K)$.)

On an input $x \le y$, refer to the lookup table to decide $L(x)$. On an input $x > y$, use the stored values of H^* and the $X_{e'}$'s to construct $\mathcal{G}_{e,x}$ and output $val(\mathcal{G}_{e,x})$. As noted previously, for all large x we can represent a *particular* node in the DAG of $\mathcal{G}_{e,x}$ with $|x|^{2e} + 1$ bits. Despite the fact that there are exponentially many nodes in the graph, an alternating polynomial-time Turing machine can search for winning strategies on the DAG by merely keeping track of a constant number of nodes, where each node is specified by H^* restricted to the set X_e used for $\mathcal{G}_{e,x}$ plus the circuit $\lambda_{e,x}$, which requires at most $|x|^c$ bits for some constant c. Therefore, we can represent any state of the game (i.e., the node where the token currently lies, plus the scores of the players) by a number of bits bounded by a polynomial in $|x|$. Given the functions h and h' for any two nodes in the DAG, along with the scores of each player, it is easy to determine in polynomial time if it is legal to move from h to h', and to compute the scores of each player after the move. (It suffices to verify that for all z, $h(z) \leq h'(z)$, and to add up a polynomial number of rationals of the form $a/2^b$ where $b = n^{O(1)}$. The length of any path in the DAG is bounded by a polynomial in n (since the values of h always decrease). Thus, determining winning strategies is possible in alternating polynomial time (since this amounts to determining if there exists a move of player 1 such that for all moves of player 2 there exists a response by player 1 . . . , ending in a winning position of player 1).

Since alternating polynomial time is equal to PSPACE [CKS81], this contradicts the fact that $L \notin$ PSPACE.

End of Proof of Claim 4

▮

Theorem 3.2

$$\mathrm{DEC} \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{mtt} R_{K_U}\} \subseteq \mathrm{coNP} \cap \mathrm{P/poly}$$

Proof: The containment in P/Poly comes from [ABK06a].

Note that a reduction showing $L \leq^{\mathrm{p}}_{mtt} R_{K_U}$ corresponds to an *anti-monotone* reduction to $ov(K_U)$ (where the only queries are of the form "Is $K_U(z) < |z|$?") Thus this same reduction is an anti-monotone reduction from the complement of L to the complement of R_{K_U}. If we replace each Boolean function in this anti-monotone reduction with its complement, we obtain a *monotone* reduction of L to $ov(K_U)$.

Thus it suffices to show that any set that is $\leq^{\mathrm{p}}_{mtt}$-reducible to the overgraph $ov(K_U)$ for every U is in NP.

The proof of this containment is almost identical to the proof of Theorem 3.1. The only difference is now we consider an arbitrary language $L \notin$ NP, and must show that when a game $\mathcal{G}_{e,x}$ is constructed corresponding to a polynomial time *monotone* truth table reduction γ_e, determining whether $val(\mathcal{G}_{e,x}) = 1$ can be computed in NP. Note that in the monotone case, the NO player of the game has no incentive to ever make a move, as doing so could only change the value of the circuit $\lambda_{e,x}$ from NO to YES. Therefore whether the YES player has a winning strategy in the game depends solely on whether the YES player can legally move the token from the start node to a node u in the game DAG labeled by YES. This is an NP question – the certificate is the node u, which as we have seen can be represented by a polynomial number of bits in $|x|$.

▮

Theorem 3.3

$$\mathrm{DEC} \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq \mathrm{EXPSPACE}$$

Proof: An NP-Turing reduction can be simulated by a truth-table reduction computable in exponential time, where all queries have length bounded by a polynomial in the input length. Carrying out the same analysis as in the proof of Theorem 3.1, but changing the time bound on the truth-table reductions from polynomial to exponential, immediately yields the EXPSPACE upper bound. ▮

4 Perspective and Open Problems

How should one interpret the theorems presented here?

Prior to this work, the inclusion $\mathrm{NEXP} \subseteq \mathrm{NP}^{R_K}$ was a mere curiosity, since it was not clear that it was even meaningful to speak about efficient reductions to an undecidable set. Here, we show that if we view R_K not as merely a single undecidable set, but as a *class* of closely-related undecidable sets (differing only by the "insignificant" choice of the universal Turing machine U), then the decidable sets that are always in NP^{R_K} *is* a complexity class sandwiched between NEXP and EXPSPACE. The obvious question is whether this class is actually *equal* to NEXP (or to EXPSPACE). Any characterization of a complexity class in terms of efficient reductions to a class of undecidable sets would raise the possibility of applying techniques from computability theory to questions in complexity theory, where they had seemed inapplicable previously.

One possible objection to the theorems presented here is that they make use of universal Turing machines U that are far from "natural". However, we see little to be gained in trying to formulate a definition of a "natural" universal Turing machine. Even basic questions such as whether there is a truth-table reduction from the Halting Problem to R_K depend on the choice of the universal Turing machine U, and the only machines for which the answer is known (positive or negative) are all decidedly "unnatural". All of the positive results, showing that problems *are* efficiently reducible to R_K hold using a quite general notion of "universal Turing machine", and we believe that the approach used here and in [ABK06a] to "factor out" the idiosyncrasies of individual universal machines is a more productive route to follow.

It is natural to conjecture that our main theorems hold, even if "DEC$\cap$" is erased from the statement of the theorems. For instance, if A is in $\bigcap_U \mathrm{NP}^{R_{K_U}}$, we conjecture that A is decidable.

We also conjecture that all of our main theorems hold if the prefix complexity function $K(x)$ is replaced everywhere by $C(x)$, although the techniques that we use do not seem sufficient to prove this.

Is R_K just as useful as an oracle as $ov(K)$? All of the upper bounds that we have on the classes of sets reducible to R_K hold also for the classes of sets reducible to $ov(K)$. However, it seems much easier to make use of $ov(K)$ than to use R_K. For instance, for *any* decidable set A, there is a universal prefix machine U such that $A \leq^{\mathrm{p}}_{tt} ov(K_U)$ (by creating a machine U such that $x \in A$ if and only if $K_U(x)$ is odd). (Details will appear in a more complete version of this work.) In contrast, some of us would conjecture that, for any machine U, if $A \leq^{\mathrm{p}}_{tt} R_{K_U}$, then $A \in \mathrm{P/poly}$. (Some partial results in this direction can be found in [ABK06a]; see also [Hit10].)

4.1 Derandomization from Uniform Hardness Assumptions

Work in derandomization that follows the "hardness vs. randomness" paradigm generally falls into two categories: those that proceed from uniform hardness assumptions, and those that rely on nonuniform hardness. The nonuniform approach yields the more general and widely-applicable tools. For instance, the work of Babai, Fortnow, Nisan, and Wigderson [BFNW93] shows that, given *any* function f, one can construct a pseudorandom generator G_f such that, given *any* test T that distinguishes the pseudorandom distribution generated by G_f from the uniform distribution, one can conclude that f has polynomial-size circuits that have access to T as an oracle. (Or, in terms of the contrapositive, if f does *not* have small circuits relative to T, then the generator G_f is secure against T.)

In contrast, there has been much more modest success in providing derandomization tools from *uniform* hardness assumptions. The canonical example here comes from Impagliazzo and Wigderson [IW01] as extended by Trevisan and Vadhan [TV07]. They show that for certain functions in

PSPACE (including some PSPACE-complete problems), one can build a pseudorandom generator G_f such that, given *any* test T that distinguishes the pseudorandom distribution generated by G_f from the uniform distribution, one can conclude that f can be computed in BPP^T. (Or, in terms of the contrapositive, if f is *not* in BPP^T, then the generator G_f is secure against T.) Trevisan and Vadhan showed that a completely black-box reduction, such as is used in the nonuniform setting, can not be used in order to obtain derandomizations from uniform assumptions, but they did not find any *limits* on the applicability of this approach. For instance, it can be used for some PSPACE-complete problems (because some such problems are both downward- and random-self-reducible), but are there some (special) problems, say, complete for doubly-exponential time where the technique can be applied? (Only problems in PSPACE can be downward-self-reducible [Bal90], but it is not *known* that derandomization from uniform assumptions requires downward-self-reducibility.)

If we restrict attention to tests T that contain no strings of low (resource-bounded) complexity, then the BPP^T algorithm can be replaced by a ZPP^T algorithm [ABK$^+$06b], and thus in particular one obtains a restricted NP^T algorithm. If we consider derandomization from a uniform NP^T hardness assumption, then the domain of applicability of these techniques can be pushed from PSPACE to NEXP; that is, it is known that for certain problems f in NEXP (including some NEXP-complete problems), one can build a pseudorandom generator G_f such that, given *any* test T that contains many strings and no strings of low (resource-bounded) complexity (and hence is disjoint from the range of G_f), one can conclude that f can be computed in NP^T [ABK06a]. It is natural to wonder if NEXP is the limit of this approach, or if it can be pushed even further.

The results in this paper provide the first concrete limits to the reach of derandomization from uniform hardness assumptions. Any test of the form R_{K_U} will have to be disjoint from the range of any computable pseudorandom generator, and will thus be an excellent test to distinguish a pseudorandom distribution from the uniform distribution. Any problem that is not in EXPSPACE will lie outside of $\mathrm{NP}^{R_{K_U}}$ for some U. Thus no such problem can be incorporated in this way into a derandomization argument from a uniform hardness assumption.

In particular, consider doubly-exponential time EEXP; let A be complete for EEXP. The set of truth-tables of Boolean functions that require oracle circuits of size $2^{n/2}$ with oracle A is complete for EEXP under P/poly reductions [ABK$^+$06b], but is not known to be complete under NP reductions. Our results show that it *cannot* be shown to be complete under NP reductions (or more restrictive reductions) using only the properties that (1) the set contains many strings and (2) contains no strings of suitably low resource-bounded Kolmogorov complexity, since (a) most likely EEXP is not contained in EXPSPACE, and (b) in that case, some R_{K_U} is an example of a set that satisfies conditions (1) and (2), but is not hard for EEXP under NP reductions.

The theorems presented here all relativize. For instance, for any decidable oracle B, if $A \notin \mathrm{PSPACE}^B$, then there is a universal prefix Turing machine U such that $A \not\leq_{tt}^{\mathrm{P}^B} R_{K_U}$. (Note that, for decidable oracles B, there is no need to "relativize" R_{K_U}. A similar restatement is possible for noncomputable oracles B, too.) However, it seems quite possible to us that, say, if it were possible to *characterize* NEXP in terms of NP^{R_K}, that this might proceed via nonrelativizable techniques. The types of characterizations of complexity classes that we are investigating are quite different than those that have been studied in the past, and we hope that new insights will result as new connections are explored.

Acknowledgments

We thank Adam Day and Bruno Loff for helpful comments.

References

[ABK06a] E. Allender, H. Buhrman, and M. Koucký. What can be efficiently reduced to the Kolmogorov-random strings? *Annals of Pure and Applied Logic*, 138:2–19, 2006.

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

[AFG10] E. Allender, L. Friedman, and W. Gasarch. Exposition of the muchnik-positselsky construction of a prefix free entropy function that is not complete under truth-table reductions. Technical Report TR10-138, Electronic Colloquium on Computational Complexity, 2010.

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

[BFKL10] H. Buhrman, L. Fortnow, M. Koucky, and B. Loff. Derandomizing from random strings. In *25th IEEE Conference on Computational Complexity (CCC)*, pages 58–63. IEEE Computer Society Press, 2010.

[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.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[Day09] A. Day. On the computational power of random strings. *Annals of Pure and Applied Logic*, 160:214–228, 2009.

[Hit10] John M. Hitchcock. Lower bounds for reducibility to the Kolmogorov random strings. In *Proc. Computability in Europe (CiE)*, volume 6158 of *Lecture Notes in Computer Science*, pages 195–200. Springer, 2010.

[IW01] R. Impagliazzo and A. Wigderson. Randomness vs. time: de-randomization under a uniform assumption. *J. Comput. Syst. Sci.*, 63(4):672–688, 2001.

[Kum96] M. Kummer. On the complexity of random strings. In *Proc. of Symp. on Theo. Aspects of Comp. Sci. (STACS)*, volume 1046 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 1996.

[LV08] M. Li and P. Vitanyi. *Introduction to Kolmogorov Complexity and its Applications.* Springer, third edition, 2008.

[MP02] A. A. Muchnik and S. Positselsky. Kolmogorov entropy in the context of computability theory. *Theoretical Computer Science*, 271:15–35, 2002.

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