Limits on the Computational Power of Random Strings

Eric Allender[a], Luke Friedman[a], William Gasarch[b,*]

[a]*Dept. of Computer Science, Rutgers University, New Brunswick, NJ 08855, USA*
[b]*Dept. of Computer Science, University of Maryland, College Park, MD 20742, USA*

Abstract

How powerful is the set of random strings? What can one say about a set A that is efficiently reducible to R, the set of Kolmogorov-random strings? We present the first *upper bound* on the class of computable sets in P^R and NP^R.

The two most widely-studied notions of Kolmogorov complexity are the "plain" complexity $C(x)$ and "prefix" complexity $K(x)$; this gives rise to two common ways to define the set of random strings "R": R_C and R_K. (Of course, each different choice of universal Turing machine U in the definition of C and K yields another variant R_{C_U} or R_{K_U}.) Previous work on the power of "R" (for *any* of these variants) has shown

- BPP $\subseteq \{A : A \leq^{\mathrm{p}}_{tt} R\}$.
- PSPACE $\subseteq \mathrm{P}^R$.
- NEXP $\subseteq \mathrm{NP}^R$.

Since these inclusions hold irrespective of low-level details of how "R" is defined, and since BPP, PSPACE and NEXP are all in Δ^0_1 (the class of decidable languages), we have, e.g.: NEXP $\subseteq \Delta^0_1 \cap \bigcap_U \mathrm{NP}^{R_{K_U}}$.

Our main contribution is to present the first upper bounds on the complexity of sets that are efficiently reducible to R_{K_U}. We show:

- BPP $\subseteq \Delta^0_1 \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{K_U}\} \subseteq$ PSPACE.
- NEXP $\subseteq \Delta^0_1 \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq$ EXPSPACE.

Hence, in particular, PSPACE is sandwiched between the class of sets polynomial-time Turing- and truth-table-reducible to R.

As a side-product, we obtain new insight into the limits of techniques for derandomization from uniform hardness assumptions.

Keywords: Kolmogorov Complexity, Prefix Complexity, Uniform Derandomization, Complexity Classes

*Corresponding author. Tel.: +1 301 405 2698; fax: 301 405 6707.
Email addresses: allender@cs.rutgers.edu (Eric Allender), lbfried@cs.rutgers.edu (Luke Friedman), gasarch@cs.umd.edu (William Gasarch)

1. Introduction

In this paper, we take a significant step toward providing characterizations of some important complexity classes in terms of efficient reductions to *non-computable* sets. Along the way, we obtain new insight into the limits of techniques for derandomization from uniform hardness assumptions.

Our attention will focus on the set of Kolmogorov random strings:

Definition 1. *Let $K(x)$ be the prefix Kolmogorov complexity of the string x. Then*

$$R_K = \{x : K(x) \geq |x|\}.$$

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

The first steps toward characterizing complexity classes in terms of efficient reductions to R_K came in the form of the following curious inclusions:

Theorem 2. *The following inclusions hold:*

- BPP $\subseteq \{A : A \leq^{\mathrm{p}}_{tt} R_K\}$ *[11].*
- PSPACE $\subseteq \mathrm{P}^{R_K}$ *[2].*
- NEXP $\subseteq \mathrm{NP}^{R_K}$ *[1].*

We call these inclusions "curious" because the upper bounds that they provide for the complexity of problems in BPP, PSPACE and NEXP are not even computable; thus at first glance these inclusions may seem either trivial or nonsensical.

A key step toward understanding these inclusions in terms of standard complexity classes is to invoke one of the guiding principles in the study of Kolmogorov complexity: The choice of universal machine should be irrelevant. Theorem 2 actually shows that problems in certain complexity classes are *always* reducible to R_K, no matter *which* universal machine is used to define $K(x)$. That is, combining this insight with the fact that BPP, PSPACE, and NEXP are all contained in Δ^0_1 (the class of decidable languages), we have

- BPP $\subseteq \Delta^0_1 \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{K_U}\}$.
- PSPACE $\subseteq \Delta^0_1 \cap \bigcap_U \mathrm{P}^{R_{K_U}}$.
- NEXP $\subseteq \Delta^0_1 \cap \bigcap_U \mathrm{NP}^{R_{K_U}}$.

The question arises as to how powerful the set $\Delta^0_1 \cap \bigcap_U \{A : A \leq_r R_{K_U}\}$ is, for various notions of reducibility $\leq_r$. Until now, no computable upper bound was known for the complexity of any of these classes. (Earlier work [1] did give an upper bound for a related class defined in terms of a very restrictive notion of reducibility: $\leq^{\mathrm{p}}_{dtt}$ reductions – but this only provided a characterization of P in terms of a class of polynomial-time reductions, which is much less compelling than giving a characterization where the set R_K is actually providing some useful computational power.)

Our main results show that the class of problems reducible to R_K in this way *does* have bounded complexity; hence it is at least plausible to conjecture that some complexity classes can be characterized in this way:
Main Results:

- $\Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{\mathrm{p}} R_{K_U}\} \subseteq$ PSPACE.
- $\Delta_1^0 \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq$ EXPSPACE.

A stronger inclusion is possible for "monotone" truth-table reductions ($\leq_{mtt}^{\mathrm{p}}$). We show that

- $\Delta_1^0 \cap \bigcap_U \{A : A \leq_{mtt}^{\mathrm{p}} R_{K_U}\} \subseteq \mathrm{coNP} \cap \mathrm{P/poly}$.

Combining our results with Theorem 2 we now have:

- $\mathrm{BPP} \subseteq \Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{\mathrm{p}} R_{K_U}\} \subseteq \mathrm{PSPACE} \subseteq \Delta_1^0 \cap \bigcap_U \mathrm{P}^{R_{K_U}}$.
- $\mathrm{NEXP} \subseteq \Delta_1^0 \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq$ EXPSPACE.

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

Our results bear a superficial resemblance to results of Book *et al.* [8, 9, 10], who also studied decidable sets that are reducible in some sense to algorithmically random sets. However, there is really not much similarity at all. Book *et al.* studied the class **ALMOST-R**:

Definition 3. *Let* $\mathbf{R}$ *be a reducibility (e.g,* $\leq_m^p$ *or* $\leq_T^p$*). Then* **ALMOST-R** *is the class of all* B *such that* $\{A : B$ *is* $\mathbf{R}$*-reducible to* $A\}$ *has measure 1.*

Book *et al.* showed that **ALMOST-R** can be characterized as the class of decidable sets that are **R**-reducible to sets whose characteristic sequences are (for example) Martin-Löf random. Thus using such sets as oracles is roughly the same as providing access to unbiased independent coin flips. But a set whose characteristic sequence is Martin-Löf random will contain many strings of low Kolmogorov complexity, and a set such as R_K is very far from being Martin-Löf random. Another contrast is provided by noting that **ALMOST**-NP = AM [21], whereas NP^{R_K} contains NEXP.

1.1. Derandomization from Uniform Hardness Assumptions

In this section, we present some easy consequences of our main result, regarding the question of which problems can be reduced (via a uniform probabilistic reduction) to the problem of distinguishing random and pseudorandom distributions.

Papers in derandomization that follow the "hardness vs. randomness" paradigm generally fall 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 [7] shows that, given *any* function f, one can construct a

pseudorandom generator[1] 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.) As discussed by Gutfreund and Vadhan [14], invoking the terminology introduced by Reingold et al. [22], this is an example of a *fully black-box* construction.

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 [16] as extended by Trevisan and Vadhan [23]. They show that for certain functions f 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.) As discussed by Gutfreund and Vadhan [14], this is still an example of a black-box reduction (although it is not *fully* black-box), since it works for *every* test T that distinguishes random from pseudorandom.

Furthermore, Trevisan and Vadhan showed that a fully black-box construction, such as is used in the nonuniform setting, can *not* yield derandomizations from uniform assumptions [23]. Their argument is nonconstructive, in the sense that they derive a contradiction from the assumption that a fully black-box construction exists, as opposed to presenting a concrete function f for which no fully black-box construction will work. Thus the question remained: For which functions f does such a reduction exist?

Gutfreund and Vadhan considered the question in more detail, and showed that there is no function f outside of BPP^{NP} for which there is a uniform *non-adaptive* black-box reduction, showing that computing f reduces to the problem of distinguishing random pseudorandom distributions [14]. (That is, if the BPP^T algorithm computing f is non-adaptive in the sense that the list of oracle queries is computed before any queries are asked, then $f \in \text{BPP}^{\text{NP}}$.) Gutfreund and Vadhan also considered a more general class of black-box reductions, still maintaining some restrictions on "adaptiveness", and showed that there is no function f outside of PSPACE for which there is a uniform black-box reduction of this restricted class, from computing f to distinguishing random from pseudorandom [14]. It appears that no limits were known at all for general BPP reductions to distinguishing random from pseudorandom.

We now state an easy consequence of our main theorems:

Theorem 4. *If there is a uniform black-box reduction, showing that a computable function f reduces to the problem of distinguishing random and pseudorandom distributions, then $f \in$ EXPSPACE.*

Proof: Let f be a computable Boolean function satisfying the hypothesis. Thus there is a generator G_f mapping strings of length $n/2$ to strings of length n, and such that there

[1]The generator G_f in general is computable in PSPACE^f, but in many important instances it is computable in P^f.

is a polynomial-time oracle Turing machine M such that $M^T(x)$ outputs $f(x)$ with probability at least 2/3 given *any* test T that distinguishes random from pseudorandom (i.e., such that the probabilities of the events $G_f(y) \in T$ and $x \in T$ differ by some δ, for y of length $n/2$ and x of length n, chosen uniformly).

Since G_f is computable, the range of G_f is disjoint from R_K (at least for all large input lengths), and thus $f \in \text{BPP}^{R_{K_U}}$ for every universal prefix machine U (since many strings will be in R_{K_U}, but none of the pseudorandom outputs will be). But $\text{BPP}^{R_K} = \text{ZPP}^{R_K}$ [2], and thus in particular we have $f \in \Delta_1^0 \cap \bigcap_U \text{NP}^{R_{K_U}} \subseteq \text{EXPSPACE}$. ∎

In addition to shedding light on the limitations of black-box reductions to distinguishing random from pseudorandom, Theorem 4 also provides some (slight) insight about the completeness of problems related to the Minimum Circuit Size Problem:

The Minimum Circuit Size Problem (MCSP) (given the truth table of a Boolean function f, and a number s, does f have a circuit of size s?) is a well-known example of a problem in NP that is believed to lie outside of P, but is not widely believed to be NP-complete [17, 2, 6].

For a complexity class $\mathcal{C}$, let $\text{MCSP}^{\mathcal{C}}$ denote an analog of MCSP for $\mathcal{C}$: given the truth table of a Boolean function f, and a number s, does f have an *oracle* circuit of size s, where the oracle is for the standard complete set for $\mathcal{C}$?

It is known [2, 6] that $\text{MCSP}^{\mathcal{C}}$ is complete for $\mathcal{C}$ under P/poly reductions, where $\mathcal{C}$ is any of $\{\text{PSPACE}, \text{EXP}, \text{NEXP}, \text{EXPSPACE}$, doubly- or triply-exponential time and space, etc., $\dots\}$. Completeness under *uniform* reductions is known only for two cases: $\text{MCSP}^{\text{PSPACE}}$ is complete for PSPACE under ZPP reductions, and MCSP^{EXP} is complete for EXP under NP reductions [2]. In the former case, completeness is proved via the Impagliazzo-Wigderson generator [16]; in the latter case completeness is proved via a uniform black-box NP-reduction to distinguishing random from pseudorandom.

Now consider doubly-exponential space EEXPSPACE. Is $\text{MCSP}^{\text{EEXPSPACE}}$ complete for EEXPSPACE under ZPP or even NP reductions? As a consequence of Theorem 4, this question cannot be answered using the techniques that were used to resolve the analogous questions for PSPACE and EXP, which were black-box reductions to distinguishing random from pseudorandom.

2. Background and Definitions

We review some basic facts and definitions about Kolmogorov complexity. For a more detailed treatment of these topics, see [19].

For a fixed universal Turing machine U, the plain Kolmogorov complexity of a string x, $C(x)$, is the size of the shortest s such that $U(s) = x$. This paper is concerned much more with *prefix complexity*. Although its definition (see below) is slightly less natural than the plain complexity, the prefix complexity is an important variant because of certain mathematical properties it possesses. For instance, the plain complexity is not subadditive (i.e. $C(x, y) \leq C(x) + C(y) + c$ does not hold in general for any constant c), and the series $\sum_x 2^{-C(x)}$ diverges, which means it cannot easily be converted into a probability measure. The prefix complexity fixes both of these problems, and is crucial to many of the applications that originally motivated the discovery of

Kolmogorov complexity, such as studying the complexity of infinite sequences and defining a universal prior probability measure that could be used as the basis for inductive reasoning. For an in-depth discussion of the tradeoffs between the plain and prefix complexity, see [19, Chapter 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 $x, K_U(x) \leq K_M(x) + c$.

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 let $K(x)$ denote the prefix complexity of x. It is known that, for some constant c, $C(x) - c \leq K(x) \leq C(x) + c + 2\log|x|$, and hence the two measures are not very far apart from each other. 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.

If f is a function mapping some domain to the naturals $\mathbb{N}$, then $ov(f)$, the overgraph of f, is $\{(x, y) : f(x) \leq y\}$. (For instance, $ov(K_U) = \{(x, y) :$ there exists an s, $|s| \leq y$, such that $U(s) = x\}$).

The following definition was used implicitly by Muchnik and Positselsky [20]: 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 computably enumerable (c.e.)

The canonical example of a prefix free entropy function is $K(x)$. (That K is a prefix free entropy function follows from the Kraft Inequality; see e.g. [19, 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* [19, Definition 4.2.2]. The *Coding Theorem* (see [19, Theorem 4.3.3]) says that, for any lower semicomputable discrete semimeasure 2^{-f} there is a universal prefix machine M such that $f(x) \leq K_M(x) - 3$. For the case of prefix free entropy functions, one can obtain a tighter bound (and replace the inequality with equality):

Proposition 5. *Let f be a prefix free entropy function. Given a machine accepting $ov(f)$, one can construct a prefix machine M such that $f(x) = K_M(x) - 2$.*[2]

[2] In preliminary versions of this work [5, 4], we incorrectly claimed that $f(x) = K_M(x) - 1$. We do not know how to obtain that bound.

Proof: Our proof is patterned on the proof of [19, Theorem 4.3.3].

Since $ov(f)$ is c.e., there is a bijective enumeration function $D : \mathbb{N} \to \{(x, a) : f(x) \leq a\}$. Let $D(0) = (x_0, a_0), D(1) = (x_1, a_1,), \ldots$ be this enumeration. We have that for each x, $\sum_{i \geq f(x)} 2^{-i} \leq 2 \cdot 2^{-f(x)}$ and therefore

$$\sum_{i \geq 0} \frac{1}{2} 2^{-a_i} = \sum_x \sum_{i \geq f(x)} \frac{1}{2} 2^{-i} \leq \sum_x 2^{-f(x)} \leq 1$$

We identify the set of infinite sequences $S = \{0, 1\}^\infty$ with the half-open real interval $[0, 1)$; that is, each real number r between 0 and 1 will be associated with the sequence(s) corresponding to the infinite binary expansion of r. We will associate each pair (x_i, a_i) from the enumeration D with a subinterval $I_i \subseteq S$ as follows:

$I_0 = [0, \frac{1}{2} 2^{-a_0})$, and for $i \geq 1$, $I_i = [\sum_{k<i} \frac{1}{2} 2^{-a_k}, \sum_{k \leq i} \frac{1}{2} 2^{-a_k})$. That is, I_i is the half-open interval of length $\frac{1}{2} 2^{-a_i}$ that occurs immediately after the interval corresponding to the pair (x_{i-1}, a_{i-1}) that appeared just prior to (x_i, a_i) in the enumeration D.

Since $\sum_{i \geq 0} \frac{1}{2} 2^{-a_i} \leq 1$, each $I_i \subseteq S$.

Any *finite* string z also corresponds to a subinterval $\Gamma_z \subseteq S$ consisting of all infinite sequences that begin with z; Γ_z has length $2^{-|z|}$. Given any pair (x_i, a_i), one can determine the interval I_i and find the lexicographically first string z of length $a_i + 2$ such that $\Gamma_z \subseteq I_i$. Since I_i has length $2^{(-a_i+1)}$, it is not too difficult to see that such a string z must exist. (Alternatively, look at the proof of [19, Lemma 4.3.3].) Call this string z_i. Observe that, since the intervals I_i are disjoint, no string z_i is a prefix of any other.

We are now ready to present our prefix machine M. Let M be a machine that, given a string z, uses D to start an enumeration of the intervals I_i until it finds (x_i, a_i) such that $\Gamma_z \subseteq I_i$. If it ever finds such a pair, at this point M determines if $z = z_i$, and if so, outputs x_i. Otherwise the machine enters an infinite loop. Since no string z_i is a prefix of any other, M is a prefix machine.

Now consider the shortest string on which M will output x. Let $T = \{i : x_i = x\}$. For every $j \in T$ there exists a string z_j such that $M(z_j) = x$, and the length of z_j will be $a_j + 2$. We have that $\min_{j \in T} a_j = f(x)$, so $K_M(x) = f(x) + 2$. ∎

We will make use of the following easy propositions.

Proposition 6. *Let M and M' be prefix Turing machines. Then there is a prefix machine M'' such that $K_{M''}(x) = \min(K_M(x), K_{M'}(x)) + 1$.*

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

Proposition 7. *Given any prefix machine M and constant c, there is a prefix machine M' such that $K_M(x) + c = K_{M'}(x)$*

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

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 $a_i = 1$ if and only if $q_i \in B$, 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 ($A \leq_{mtt}^{\mathrm{p}} B$).

- *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 ($A \leq_{amtt}^{\mathrm{p}} B$).

- *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

Theorem 8. $\Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{p} R_{K_U}\} \subseteq$ PSPACE

Proof: The main idea of the proof can be seen as a blending of the approach of [1] with the techniques that Muchnik and Positselsky used to prove Theorem 2.7 of [20]. (See also [3] for an alternative exposition of this theorem of Muchnik and Positselsky.)

We will actually prove the statement

$$\Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{p} ov(K_U)\} \subseteq \text{PSPACE}. \tag{1}$$

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

$$\Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{p} R_{K_U}\} \subseteq \Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{p} ov(K_U)\}.$$

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

$$L \in \Delta_1^0 - \text{PSPACE} \Rightarrow \exists \text{ a universal prefix machine } U \text{ s.t. } L \not\leq_{tt}^{p} ov(K_U). \tag{2}$$

Let $L \in \Delta_1^0 - \text{PSPACE}$ be given. It suffices to show how to incorporate a machine deciding membership in L into the construction of a universal prefix machine U such

that $L \not\leq^p_{tt} ov(K_U)$. (As part of this construction, we use a diagonalization argument designed to foil every $\leq^{\mathrm{p}}_{tt}$ reduction.) 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. $H(x) = \min(K(x) + 5, F(x) + 3)$.
3. $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq \frac{1}{8}$.
4. $L \not\leq^p_{tt} ov(H)$.

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

Proof of Claim 1: By Properties 2 and 3 we have that $\sum_{x \in \{0,1\}^*} 2^{-F(x)+3} \leq \frac{1}{8}$. 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 5 we then have that $F + 2$ is K_M for some prefix machine M. By Proposition 7 we have that $K(x) + 4$ is $K_{U''}$ for some universal prefix machine U''. Therefore, by Proposition 6, $H(x) = \min(K(x) + 5, F(x) + 3) = \min(K(x) + 4, F(x) + 2) + 1$ is $K_{U'}$ for some universal prefix machine U'. ∎

It remains to show that for a given computable set $L \notin$ PSPACE we can always construct functions H and F 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 $\min(K^*(x) + 5, F^*(x) + 3)$.

Originally we set $F^*(x) = 2|x| + 3$ 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 "$(z,r) \in ov(H)$" 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 $(z,r) \in ov(H)$ 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 $(z, r-3)$ 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 this type of change though; if we are too liberal in modifying F we may violate the condition $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq 1/8$ 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 $\sum_{x\in\{0,1\}^*} 2^{-H^*(x)} \leq 1/8$. (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 $\sum_{x\in\{0,1\}^*} 2^{-H^*(x)} \leq 1/8$. Our ability to find such a witness x follows from our assumption that the computable language L is not in PSPACE; if no such witness exists, then membership in L reduces to finding which player has a winning strategy in one of these games, which can be done in PSPACE.

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. This contradicts K being a prefix complexity function. Hence K can only cheat finitely often.[3]

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

[3]This reliance on the convergence of $\sum_{x\in\{0,1\}^*} 2^{-K(x)}$ is the key reason why our proof does not go through in the case where we consider the plain complexity $C(x)$ as opposed to the prefix complexity $K(x)$.

$e < e' \leq s$, 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

Now 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 atoms $\{(z_1, r_1), \ldots, (z_k, r_k)\}$ 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 $e' < e$.

While working on requirement R_e we will need to evaluate the circuit $\lambda_{e,x}$. This will involve answering queries of the form $H(z) \leq r$. There will be two types of queries:

- If $z \in \bigcup_{e'<e} X_{e'}$ then we answer FALSE, which is the correct value unless the appropriate $R_{e'}$ acts. However, if this occurs, then all of the work done on R_e will be wiped out anyway.
- If $x \in X_e$ then we answer with the status of $H^*(z) \leq r$. The key is that we have some control over this if the answer is FALSE and may purposely change it.

Let $H^*_{e,x}$ be the function H^* when the game $\mathcal{G}_{e,x}$ is first constructed. Let $\epsilon = 2^{-e-i_e-6}$. (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 $2|z_i| + 6$ values (since it is always bounded by $F^*(z_i) + 3$). Note also that $|z_i| \leq |x|^e$. Thus there are at most $(2|x|^e + 6)^{|x|^e}$ 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), 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 $val(\mathcal{G}_{e,x}) = 1$ if the YES player has a winning strategy on the game $\mathcal{G}_{e,x}$ and $val(\mathcal{G}_{e,x}) = 0$ 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. The Formal Construction

We now present the formal details of the stage construction.

Stage 0:

- Let F^* initially be defined as $F^*(x) = 2|x| + 3$.
- Let K be the standard prefix complexity function, and K^* initially be the empty function.
- At all times throughout the construction, we have that $H^*(x) = \min(K^*(x) + 5, F^*(x) + 3)$. (In the case where $K^*(x)$ is undefined, let $H^*(x) = F^*(x) + 3$). We will define H_s to be 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 $\epsilon = 2^{-e-i_e-6}$ 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 $\epsilon = 2^{-e-i_e-6}$ 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: $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{8}$.

Proof of Claim 3:

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

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} = \sum_{x\in\{0,1\}^*} 2^{-H_0(x)} + \sum_{s\geq 1} \sum_{x\in\{0,1\}^*} (2^{-H_{s+1}(x)} - 2^{-H_s(x)}).$$

That is, we can bound the sum by bounding H_0 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 $H^*(x) = F^*(x) + 3 = 2|x| + 6$ for all x, so $\sum_{x\in\{0,1\}^*} 2^{-H_0(x)} = \frac{1}{32}$.

The total contribution that K can make to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ is bounded by the expression $\sum_{x\in\{0,1\}^*} 2^{-K(x)+5}$. Since K is a prefix free entropy function, this contribution is at most 1/32.

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 1/32.

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 $\epsilon = 2^{-e-i_e-6} = 2^{-e-6}$. On the second such game associated with R_e, $i_e = 1$, so F can change at most $\epsilon = 2^{-e-7}$. 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

$$\sum_{i=6}^{\infty} 2^{-e-i} = 2^{-e} \sum_{i=6}^{\infty} 2^{-i} = 2^{-e-5}$$

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 $\sum_{e=1}^{\infty} 2^{-e-5}$ = $\frac{1}{32}$.

Putting all this information together we have that

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{32} + \frac{1}{32} + \frac{1}{32} + \frac{1}{32} = \frac{1}{8}$$

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 $x \leq y$. Also hardwire in the function H^* at this moment in the construction and $X_{e'}$ for all $e' \leq e$. (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 $x \leq y$, 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, as follows:

Note that the number of moves in the game is bounded by a polynomial in $|x|$, since each move in the game involves lowering the value of $H^*(z)$ for one of the polynomially-many queries z. Thus, to determine if a player has a winning strategy from some game position C (represented by the circuit $\lambda_{e,x}$, along with the values of H^* restricted to the set X_e that is used in the game $\mathcal{G}_{e,x}$), it suffices to check if there *exists* a move for this player causing the circuit $\lambda_{e,x}$ to take on the desired value, and such that *for every* move of the opposing player, there *exists* a move of this player that again causes $\lambda_{e,x}$ to take on the desired value, such that ... until there are no more legal moves possible for the opposing player. 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 $b = n^{O(1)}$.) As mentioned above, 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 alternating polynomial time is equal to PSPACE [12], this contradicts the fact that $L \notin$ PSPACE.

End of Proof of Claim 4

∎

Theorem 9. $\Delta_1^0 \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{mtt} R_{K_U}\} \subseteq \mathrm{coNP} \cap \mathrm{P/poly}$

Proof: The containment in P/Poly comes from [1].

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 com-

plement 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 8. 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 10. $\Delta_1^0 \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq$ 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 8, but changing the time bound on the truth-table reductions from polynomial to exponential, immediately yields the EXPSPACE upper bound. ∎

4. Encoding in the Overgraph

We conjecture that our main results can be improved in several ways. In this section, we consider one type of improvement, and we present some reasons why a different proof strategy will be required, in order to obtain such an improvement.

Theorem 8 shows that, for every decidable set A outside PSPACE, there is *some* universal prefix machine U such that $A \not\leq^p_{tt} R_{K_U}$. It is natural to ask if there is a decidable set A such that, for *every* universal machine U, $A \not\leq^p_{tt} R_{K_U}$. We conjecture that this should hold for every decidable set A that is not in P/poly. (Some weaker results in this direction have been proved. It is known that if a decidable set A has high-enough non-uniform complexity, then for *any* universal machine U, any $\leq^{\mathrm{p}}_{tt}$ reduction from A to R_{C_U} must make at least $n/4 \log n$ queries [1]. Related questions were also explored by Hitchcock [15].)

However, the following theorem shows that no such improvement can carry over to the *overgraph* $ov(R_{K_U})$, which suggests that quite different techniques may be required than were employed in this paper. (Related observations occur in [20, Theorem 2.6].)

Theorem 11. *Let A be a computable set. Then there exists a universal (prefix) machine U such that $A \leq^{\mathrm{p}}_{tt} ov(C_U)$ ($A \leq^{\mathrm{p}}_{tt} ov(K_U)$, respectively).*

Proof: We present the proof for $ov(K_U)$. It is clear that the proof carries over also for the case of $ov(C_U)$.

Let M be the universal prefix machine that defines $K(x)$.

Consider the machine U that does not halt on any input in $00\{0,1\}^*$, and behaves as follows on inputs of the form $1d$ or $01d$ for any string d:

Simulate $M(d)$, and if it halts, let the output be x (so that $K(x) \leq |d|$).

Determine if $x \in A$ or not.

If $x \in A$ and $|d|$ is even, then $U(1d) = x$ and $U(01d) = \uparrow$.
If $x \in A$ and $|d|$ is odd, then $U(1d) = \uparrow$ and $U(01d) = x$.
If $x \notin A$ and $|d|$ is even, then $U(1d) = \uparrow$ and $U(01d) = x$.
If $x \notin A$ and $|d|$ is odd, then $U(1d) = x$ and $U(01d) = \uparrow$.

Note that U is a prefix machine, and that for all x $K_U(x) \leq K(x) + 2$.

Clearly, $x \in A$ if and only if $K_U(x)$ is odd. This can be determined by making a linear number of truth-table queries to $ov(K_U)$.

∎

5. Perspective and Open Problems

How should one interpret the theorems presented here?

Prior to this work, the inclusion NEXP $\subseteq$ NP^{R_K} was just a 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 computable 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 [20, 1], and the only machines for which the answer is known (positive and negative) are all decidedly "unnatural". 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 [13].

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 [1] to "factor out" the idiosyncrasies of individual universal machines is a more productive route to follow.

Alternatively, one can view our results as placing limits on what sets can be *proved* to be reducible to R_K, using only an "axiomatic" approach to Kolmogorov complexity. Let us expand on this view. The theory of Kolmogorov complexity (for instance, as developed in [19]) relies on the existence of universal machines U, in order to develop the measures K and C, but no properties are required of such machines U, other than that, for any other (prefix) machine M, $\exists c_M \forall x C_U(x) \leq C_M(x) + c_M$ (or $K_U(x) \leq K_M(x) + c_M$, respectively). The rest of the theory can proceed, using just this axiom about the machine U that defines C or K.

Our results show that, for any decidable set A outside of EXPSPACE, A cannot be proven to lie in NP^{R_K} without introducing additional axioms describing additional properties of the universal machine that defines K. One could pursue the same types of questions that we do in this paper using a more stringent definition of what constitutes a universal machine, but at the cost of adding additional axioms to the study of Kolmogorov complexity that in some sense are not strictly necessary.

It is natural to conjecture that our main theorems hold, even if "$\Delta_1^0\cap$" 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 computable.

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. Let us expand on this point. Our techniques build on the techniques that Muchnik and Positselsky [20] used, in order to show that, for some U, there is no truth-table reduction from the halting problem to $ov(K_U)$. It is impossible to generalize their result to the plain Kolmogorov complexity C, because Kummer showed [18] that there *is* a truth-table reduction from the halting problem to R_C (no matter which universal machine one uses to define C). Kummer actually presents a *disjunctive* truth-table reduction from the complement of the halting problem to R_C; it is known [1] that any such disjunctive truth-table reduction must require at least exponential time, but nothing is known for general truth-table reductions. That is, it remains unknown whether there is a c.e. set A and a universal machine U such that $A \not\leq_{tt}^{p} R_{C_U}$, and it is also unknown whether there is a universal machine U such that R_{C_U} is hard for the c.e. sets under $\leq_{tt}^{\mathrm{p}}$ reductions. We conjecture that the halting problem is *not* $\leq_{tt}^{\mathrm{p}}$ reducible to R_C or R_K (no matter which universal machine is used to define C and K).

The theorems presented here all relativize. For instance, for any computable 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 computable 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, Bruno Loff, Russell Impagliazzo, and Danny Gutfreund, Ken Regan, Sam Hopkins, Salil Vadhan, and the anonymous ICALP reviewers for helpful comments. The first two authors were supported in part by NSF Grants CCF-0830133, CCF-0832787, and CCF-1064785.

References

[1] 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.

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

[3] 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.

[4] E. Allender, L. Friedman, and W. Gasarch. Limits on the computational power of random strings. Technical Report TR10-139, Electronic Colloquium on Computational Complexity, 2010.

[5] E. Allender, L. Friedman, and W. Gasarch. Limits on the computational power of random strings. In *Proc. of International Conference on Automata, Languages, and Programming (ICALP)*, volume 6755 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 2011.

[6] E. Allender, M. Koucký, D. Ronneburger, and S. Roy. The pervasive reach of resource-bounded Kolmogorov complexity in computational complexity theory. *Journal of Computer and System Sciences*, 77:14–40, 2010.

[7] 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.

[8] R. V. Book. On languages reducible to algorithmically random languages. *SIAM Journal on Computing*, 23(6):1275–1282, 1994.

[9] R. V. Book, J. Lutz, and K. W. Wagner. An observation on probability versus randomness with applications to complexity classes. *Mathematical Systems Theory*, 27(3):201–209, 1994.

[10] R. V. Book and E. Mayordomo. On the robustness of ALMOST-*r*. *RAIRO Informatique Théorique et Applications*, 30(2):123–133, 1996.

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

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

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

[14] D. Gutfreund and S. P. Vadhan. Limitations of hardness vs. randomness under uniform reductions. In *APPROX-RANDOM*, volume 5171 of *Lecture Notes in Computer Science*, pages 469–482. Springer, 2008.

[15] 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.

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

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

[18] 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.

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

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

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

[22] O. Reingold, L. Trevisan, and S. P. Vadhan. Notions of reducibility between cryptographic primitives. In *Theory of Cryptography (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004.

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