Reductions to the set of random strings: the resource-bounded case

Eric Allender[1], Harry Buhrman[2], Luke Friedman[1], and Bruno Loff[3]

[1] Department of Computer Science, Rutgers University, Piscataway, NJ 08855, USA
{allender/lbfried}@cs.rutgers.edu
[2] CWI and University of Amsterdam buhrman@cwi.nl
[3] CWI bruno.loff@gmail.com

Abstract. This paper is motivated by a conjecture [1, 4] that BPP can be characterized in terms of polynomial-time nonadaptive reductions to the set of Kolmogorov-random strings. In this paper we show that an approach laid out in [4] to settle this conjecture cannot succeed without significant alteration, but that it does bear fruit if we consider time-bounded Kolmogorov complexity instead.
We show that if a set A is reducible in polynomial time to the set of time-t-bounded Kolmogorov-random strings (for all large enough time bounds t), then A is in P/poly, and that if in addition such a reduction exists for any universal Turing machine one uses in the definition of Kolmogorov complexity, then A is in PSPACE.

1 Introduction

The roots of this investigation stretch back to the discovery that PSPACE $\subseteq$ P^R and NEXP $\subseteq$ NP^R, where R is the set of Kolmogorov-random strings [3, 2]. Later, it was shown that BPP $\subseteq$ P$^R_{tt}$ [7], where P$^A_{tt}$ denotes the class of problems reducible to A via polynomial-time *nonadaptive* (or *truth-table*) reductions.

There is evidence indicating that some of these inclusions are in some sense optimal. The inclusions mentioned in the preceding paragraph hold for the two most-common versions of Kolmogorov complexity (the plain complexity C and the prefix-free complexity K), and they also hold no matter which universal Turing machine one uses when defining the measures K and C.

Let R_{K_U} denote the set of random strings according to the prefix-free measure K given by the universal machine U: $R_{K_U} = \{x : K_U(x) \geq |x|\}$. Last year, it was shown that the class of decidable sets that are polynomial-time truth-table reducible to R_{K_U} for every U is contained in PSPACE [5]. That is, although P$^{R_{K_U}}_{tt}$ contains arbitrarily complex decidable sets, an extremely complex set can only be there because of characteristics of R_{K_U} that are fragile with respect to the choice of U.

This motivates the following definition: DTTR is the class of all decidable problems that are polynomial-time truth-table reducible to R_{K_U} for every choice of universal prefix-free Turing machine U. Thus it was proven that

$$\text{BPP} \subseteq \text{DTTR} \subseteq \text{PSPACE} \subseteq \text{P}^{R_K}. \quad (1)$$

So we naturally come upon the following.

Research question: *Does* DTTR *sit closer to* BPP*, or closer to* PSPACE*?*

A conjecture by various authors [4, 1] is that DTTR actually characterizes BPP exactly. Part of the intuition is that a non-adaptive reduction cannot make use of queries to R_K larger than $O(\log n)$ to solve a decidable problem. If indeed true we could use the strings of length at most $O(\log n)$ as advice and answer the larger queries with NO, to show that these sets are in P/poly. The rest of the intuition is that the smaller strings can only be used as a source for pseudo-randomness. If we are able to prove this conjecture, then we can make use of the tools of Kolmogorov complexity to study various questions about the class BPP. Because of the inclusions listed in (1) above, this now amounts to understanding the relative power of Turing reductions *vs.* truth-table reductions to R_K.

In an attempt to tackle this question, it was conjectured in [4, 1] that the $\mathsf{DTTR} \subseteq$ PSPACE upper bound can be improved to PSPACE $\cap$ P/poly, and an approach was suggested, based on the above mentioned intuition in connection with formal systems of arithmetic. In this paper, we show that this approach must fail, or at least requires significant changes. Interestingly, we can also prove that this intuition — that the large queries can be answered with NO — *can* be used in the resource-bounded setting to show an analogue of the P/poly inclusion. While demonstrating this discrepancy we show several other ways in which reductions to R_K and R_{K^t} are actually very different; in particular, we construct a counter-intuitive example of a polynomial-time non-adaptive reduction that distinguishes R_K from R_{K^t}, for any sufficiently large time-bound t.

To investigate the resource-bounded setting we define a class TTRT as a time-bounded analog of DTTR; informally, TTRT is the class of problems that are polynomial-time truth-table reducible to R_{K^t} for every sufficiently fast-growing time-bound t, and every "time-efficient" universal Turing machine used to define K^t. We prove that, for all monotone nondecreasing computable functions $\alpha(n) = \omega(1)$,

$$\text{BPP} \subseteq \mathsf{TTRT} \subseteq \text{PSPACE}/\alpha(n) \cap \text{P/poly}.$$

Here, PSPACE$/\alpha(n)$ is a "slightly non-uniform" version of PSPACE. We believe that this indicates that TTRT is "closer" to BPP than it is to PSPACE.

It would be more appealing to avoid the advice function, and we are able to do so, although this depends on a fine point in the definition of time-efficient prefix-free Kolmogorov complexity. This point involves a subtle technical distinction, and will be left for the appropriate section. To summarize:

- In Section 3 we prove that $\mathsf{TTRT} \subseteq$ P/poly, by using the same basic idea of [4, 1]. We further show, however, that this approach will not work to prove $\mathsf{DTTR} \subseteq$ P/poly, and by reversing the logic connection of [4, 1], this will give us an independence result in certain extensions of Peano arithmetic.
- Then in section 4 we prove that $\mathsf{TTRT} \subseteq$ PSPACE$/\alpha(n)$, which is a non-trivial adaptation of the techniques from [5]. In section 5 we show how to get the result without the super-constant advice term.

In the final section we discuss prospects for future work.

2 Preliminaries

We assume the reader is familiar with basic complexity theory [6] and Kolmogorov complexity [11]. We use $\leq_T^p$ and P^A when referring to polynomial-time Turing reductions, and $\leq_{tt}^p$ and P_{tt}^A for polynomial-time truth-table (or *non-adaptive*) reductions. For example, $M : A \leq_T^p B$ means that M is a Turing reduction from A to B. For a set A of strings, $A^{\leq n}$ denotes the set of all strings of length at most n in A.

We let K_U denote Kolmogorov complexity with respect to prefix machine U, i.e., $K_U(x) = \min\{|p| : U(p) = x\}$. We use R_{K_U} to denote the set of K_U*-random* strings $\{x|K_U(x) \geq |x|\}$. In this paper, a function $t : \mathbb{N} \to \mathbb{N}$ is called a "time-bound" if it is non-decreasing and time-constructible. We use the following time-bounded version of Kolmogorov complexity: for a prefix machine U and a time-bound t, $K_U^t(x)$ is the length of the smallest string p such that $U(p)$ outputs x and halts in fewer than $t(|x|)$ time steps. Then $R_{K_U^t}$ is the set of K_U^t*-random* strings $\{x|K_U^t(x) \geq |x|\}$. Let us define what it means for a machine to be "universal" in the time-bounded setting:

Definition 2.1. *A prefix machine U is a time-efficient universal prefix machine if there exist constants c and c_M for each prefix machine M, such that*

1. $\forall x$, $K_U(x) \leq K_M(x) + c_M$
2. $\forall x$, $K_U^t(x) \leq K_M^{t'}(x) + c_M$ *for all* $t > t'^c$

We will sometimes omit U in the notation K_U, R_{K_U}, K_U^t, $R_{K_U^t}$, in which case we mean $U = U_0$, for some arbitrary choice of a time-efficient universal prefix machine U_0. Now we can formally define the time-bounded analogue of DTTR:

Definition 2.2. TTRT *is the class of languages L such that there exists a time bound t_0 (depending on L) such that for all time-efficient universal prefix machines U and $t \geq t_0$, $L \leq_{tt}^p R_{K_U^t}$.*

Corollary 12 from [7] says that, if $t \geq t_0 = 2^{2^{2n}}$, then BPP $\leq_{tt}^p R_{K_U^t}$, for any time-efficient universal U. This implies:

Theorem 2.3 **([7]).** BPP $\subseteq$ TTRT.

Now we prove some basic fats about time-bounded prefix-free Kolmogorov complexity.

Proposition 2.4. *For any machine M and $t'(|x|) > 2^{|x|}t(|x|)$, the query $x \in R_{K_M^t}$? can be computed in time t'.*

Proof. Simulate the machine M on every string of length less than x for $t(|x|)$ steps. Because there are fewer than $2^{|x|}$ such strings, the bound follows. □

Proposition 2.5. *Let $L \leq_{tt}^p R_{K_U^t}$ for some time-bound t. Then there exists a constant k such that the language L can be computed in $t_L(n) = 2^{n^k}t(n^k)$ time.*

Proof. Let M be a machine that computes L by running the polynomial-time truth-table reduction from L to $R_{K_U^t}$, and computing by brute-force the answer to any queries from the reduction. Using Proposition 2.4, we have that for large enough k, M runs in at most $t_L(n) = 2^{n^k} t(n^k)$ time, so L is decidable within this time-bound.

It is the ability to compute R_{K^t} for short strings that makes the time-bounded case different from the ordinary case. This will be seen in proofs throughout the paper.

3 How and why to distinguish R_K from R_{K^t}

At first glance, it seems reasonable to guess that a polynomial-time reduction would have difficulty telling the difference between an oracle for R_K and an oracle for R_{K^t}, for large enough t. Indeed $R_K \subseteq R_{K^t}$ and in the limit for $t \to \infty$ they coincide.

One might even suspect that a polynomial-time reduction must behave the same way with R_{K^t} and R_K as oracle, already for modest time bounds t. However, this intuition is wrong. Here is an example for adaptive polynomial-time reductions.

Observation 3.1 *There is a polynomial-time algorithm which, given oracle access to R_K and input 1^n, outputs a K-random string of length n. However, for any time-bound t such that $t(n+1) \gg 2^n t(n)$, there is no polynomial-time algorithm which, given oracle access to R_{K^t} and input 1^n, outputs a K^t-random string of length n.*

For the algorithm, see [8]; roughly, we start with a small random string and then use [8, Theorem 15] (described later) to get a successively larger random string. But in the time-bounded case in [9] it is shown that on input 1^n, no polynomial-time machine M can query (or output) any K^t-random string of length n: in fact, $M(1^n)$ is the same for both oracles R_{K^t} and $R' = R_{K^t}^{\leq n-1}$. This is proven as follows: since R' can be computed in time $t(n)$ (by Proposition 2.4), then any query of length $\geq n$ made by $M^{R'}(1^n)$ is described by a pointer of length $O(\log n)$ in time $t(n)$, and hence is not in R_{K^t}.

3.1 Small circuits for sets reducible to R_{K^t}

We now prove that TTRT is a subset of P/poly. Actually, we will prove that this holds even for Turing reduction to R_{K_U} for a single universal Turing machine U:

Theorem 3.2. *Suppose $A \in \mathsf{DTIME}(t_1)$ and $M : A \leq_T^p R_{K^t}$, for some time-bounds t, t_1 with $t(n+1) \geq 2^n t(n) + 2^{2^n} t_1(2^n)$.*[4] *Then $A \in$ P/poly; in fact, if M runs in time n^c, and $R' = R_{K^t}^{\leq \lceil (c+1)\log n \rceil}$, then $\forall x \in \{0,1\}^n$ $M^{R'}(x) = A(x)$.*

Proof. Let $\ell(n) = \lceil (c+1)\log n \rceil$, $R'(n) = R_{K^t}^{\leq \ell(n)}$, and suppose that $M^{R'(n)}(x) \neq A(x)$ for some x of length n. Then we may find the first such x in time $2^{\ell(n)} t(\ell(n)) + 2^{n+1}(t_1(n) + O(n^c))$ (cf. Proposition 2.4), and each query made by $M^{R'(n)}(x)$ can

[4] For example, if $A \in$ EXP, then t can be doubly-exponential. If A is elementary-time computable, then t can be an exponential tower.

be output by a program of length $c \log n + O(1)$, running in the same time bound. But since $A(x) \neq M^{R'(n)}(x)$, it must be that, with $R'(n)$ as oracle, M makes some query q of size $m \geq \ell(n) + 1$ which is random for t-bounded Kolmogorov complexity (because both small and non-random queries are answered correctly when using R' instead of R_{K^t}). Hence we have both that q is supposed to be random, and that q can be output by a program of length $< \ell(n)$ in time $2^{\ell(n)} t(\ell(n)) + 2^{n+1}(t_1(n) + O(n^c)) \ll 2^{\ell(n)} t(\ell(n)) + 2^{2^{\ell(n)}} t_1(2^{\ell(n)}) \leq t(\ell(n)+1) \leq t(m)$, which is a contradiction. □

Corollary 3.3. $\mathsf{TTRT} \subseteq$ P/poly

Proof. Let $L \in \mathsf{TTRT}$. By the definition of TTRT, $L \leq^p_{tt} R_{K^{t_0}}$. Using Proposition 2.5, we then have that L is decidable in time $t_L(n) = 2^{n^k} t_0(n^k)$ for some constant k. Choose a time-bound t such that $t(n+1) \geq 2^n t(n) + 2^{2^n} t_L(2^n)$. By the definition of TTRT, since $t > t_0$, we have that $L \leq^p_{tt} R_{K^t_{U_0}}$, from which by Theorem 3.2 it follows that $L \in$ P/poly.

PSPACE $\leq^p_T R_K$ [3], but Theorem 3.2 implies that PSPACE $\not\leq^p_T R_{K^t}$ for sufficiently-large t, unless PSPACE $\subseteq$ P/poly. This highlights the difference between the time-bounded and ordinary Kolmogorov complexity, and how this comes to the surface when working with reductions to the corresponding sets of random strings.

3.2 A reduction distinguishing R_K from R_{K^t}, and an incorrect conjecture

Theorem 3.2 shows that a polynomial-time truth-table reduction to R_{K^t} for sufficiently-large t will work just as well if only the logarithmically-short queries are answered correctly, and all of the other queries are simply answered "no".

The authors of [4] conjectured that a similar situation would hold if the oracle were R_K instead of R_{K^t}. More precisely, they proposed a proof-theoretic approach towards proving that DTTR is in P/poly: Let PA$_0$ denote Peano Arithmetic, and for $k > 0$ let PA$_k$ denote PA$_{k-1}$ augmented with the axiom "PA$_{k-1}$ is consistent". In [4] it is shown that, for any polynomial-time truth-table reduction M reducing a decidable set A to R_K, one can construct a true statement of the form $\forall n \forall j \forall k \Psi(n, j, k)$ (which is provable in a theory such as Zermelo-Frankel), with the property that if, for each fixed (**n**,**j**,**k**) there is some k' such that PA$_{k'}$ proves ψ(**n**,**j**, **k**), then $\mathsf{DTTR} \subseteq$ P/poly. Furthermore, if these statements were provable in the given extensions of PA, it would follow that, for each input length n, there is a finite subset $R' \subseteq R_K$ consisting of strings having length at most $O(\log n)$, such that $M^{R'}(x) = A(x)$ for all strings x of length n.

Thus the authors of [4] implicitly conjectured that, for any polynomial-time truth-table reduction of a decidable set to R_K, and for any n, there would be some setting of the short queries so that the reduction would still work on inputs of length n, when all of the long queries are answered "no". While we have just seen that this is precisely the case for the time-bounded situation, the next theorem shows that this does not hold for R_K, even if "short" is interpreted as meaning "of length $< n$". (It follows that infinitely many of the statements ψ(**n**,**j**, **k**) of [4] are independent of every PA$_{k'}$.)

Theorem 3.4. *There is a truth-table reduction $M : \{0,1\}^* \leq_{tt}^p R_K$, such that, for all large enough n:*

$$\forall R' \subseteq \{0,1\}^{\leq n-1} \exists x \in \{0,1\}^n \; M^{R'}(x) \neq 1.$$

Proof. Theorem 15 of [8] presents a polynomial-time procedure which, given a string z of even length $n-2$, will output a list of constantly-many strings $z_1, \ldots, z_c$ of length n, such that at least one of them will be K-random if z is. We use this to define our reduction M as follows: on input $x = 00\ldots0z$ of length n having even $|z|$, we query each of z, $z_1, \ldots, z_c$, and every string of length at most $\log n$. If there are no strings of length at most $\log n$ in the oracle, we reject. Else, if z is in the oracle but none of the z_i are, we reject. On all other cases we accept.

By [8, Theorem 15], and since R_K has strings at every length, it is clear that M accepts every string with oracle R_K, and rejects every string if $R' = \varnothing$. However, for any non-empty set $R' \subseteq \{0,1\}^{\leq n-1}$, let $\ell \leq n-1$ be the highest even length for which $R'^{=\ell} \neq \varnothing$, and pick $z \in R'^{=\ell}$. Then we will have $z \in R'^{=\ell}$ but every $z_i \notin R^{=\ell+2}$, hence $M^{R'}(00\ldots0z)$ rejects. □

In fact, if we let $R' = R_{K^t}^{\leq n-1}$, for even n, then for the first $x = 00z$ such that $M^{R'}(x) = 0$, we will have $z \in R' \subseteq R_{K^t}$, but each z_i can be given by a small pointer in time $O(2^{n-1}t(n-1))$ (again we use Proposition 2.4), and hence $z_i \notin R_{K^t}$ for suitably fast-growing t. Thus $M^{R_{K^t}}(x) = 0 \neq M^{R_K}(x)$, and we conclude:

Observation 3.5 *If $t(n+1) \gg 2^n t(n)$, then the non-adaptive reduction M above behaves differently on the oracles R_K and R_{K^t}.*

4 Polynomial Space with Advice

Our single goal for this section is proving the following:

Theorem 4.1. *For any computable unbounded function $\alpha(n) = \omega(1)$,*

$$\mathsf{TTRT} \subseteq \text{PSPACE}/\alpha(n).$$

The proof of this theorem is patterned closely on related arguments in [5], although a number of complications arise in the time-bounded case. Because of space limitations, the presentation here will not be self-contained; readers will often be referred to [5]. Before proving the theorem we present several supporting propositions.

Proposition 4.2 (Analogue to Coding Theorem). *Let f be a function such that*

1. $\sum_{x \in \{0,1\}^*} 2^{-f(x)} \leq 1$
2. *There is a machine M computing $f(x)$ in time $t(|x|)$*

Let $t'(|x|) > 2^{2|x|}t(|x|)$. Then for some M', $K_{M'}^{t'}(x) = f(x) + 2$.

Proof. The proof is similar to the proof of Proposition 5 from [5]. Let

$$E = \langle x_0, f(x_0)\rangle, \langle x_1, f(x_1)\rangle \ldots$$

be an enumeration of the function f ordered lexicographically by the strings x_i.

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 element $\langle x_i, f(x_i)\rangle$ from the enumeration E with a subinterval $I_i \subseteq S$ as follows:

$I_0 = [0, 2^{-f(x_0)})$, and for $i \geq 1$, $I_i = [\sum_{k<i} 2^{-f(x_k)}, \sum_{k \leq i} 2^{-f(x_k)})$. That is, I_i is the half-open interval of length $2^{-f(x_i)}$ that occurs immediately after the interval corresponding to the element $\langle x_{i-1}, f(x_{i-1})\rangle$ that appeared just prior to $\langle x_i, f(x_i)\rangle$ in the enumeration E.

Since $\sum_{i \geq 0} 2^{-f(x_i)} \leq 1$, 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 element $\langle x_i, f(x_i)\rangle$, there must exist a lexicographically first string z_i of length $f(x_i) + 2$ such that $\Gamma_{z_i} \subseteq I_i$. Observe that, since the intervals I_i are disjoint, no string z_i is a prefix of any other.

Let M' be the following machine. On input z, M' runs M to compute the enumeration E until it finds an element $\langle x_i, f(x_i)\rangle$ that certifies that $z = z_i$. If it finds such an element then M' outputs x_i.

Suppose that M' outputs x_i on input z, and let $\langle x_i, f(x_i)\rangle$ be the element of E corresponding to x_i. Before outputting x_i, M' must compute $|\langle x_j, f(x_j)\rangle|$ for every string x_j such that $x_j < x_i$ (under the lexicographical ordering). There are at most $2^{|x_i|+1}$ strings x_j such that $x_j < x_i$, so overall this will take less than $2^{2|x_i|} t(|x_i|)$ time.

M' will be a prefix machine, and we have that $K^{t'}_{M'}(x) = f(x) + 2$.

Proposition 4.3 (Analogue to Proposition 6 from [5]). *Let U be a time-efficient universal prefix Turing machine and M be any prefix Turing machine. Suppose that t, t', and t'' are time bounds and f, g are two time-constructible increasing functions, such that f is upper bounded by a polynomial, and $t''(|x|) = f(t(|x|)) = g(t'(|x|))$.*

Then there is a time-efficient universal prefix machine U' such that

$$K^{t''}_{U'}(x) = \min(K^t_U(x), K^{t'}_M(x)) + 1$$

.

Proof. On input $0y$, U' runs U on input y. If U would output string x on y after s steps, then U' outputs string x after $f(s)$ steps. Similarly, on input $1y$, U' runs M on input y. If M would output string x on y after s steps, then U' outputs string x after $g(s)$ steps.

Note that because U is an efficient universal prefix machine, U' will be an efficient universal prefix machine as well. □

Proposition 4.4 (Analogue of Proposition 7 from [5]). *Given any time-efficient universal prefix machine U, time bound t, and constant $c \geq 0$, there is a time-efficient universal prefix machine U' such that $K^t_{U'}(x) = K^t_U(x) + c$.*

Proof. On input $0^c x$, M' runs M on input x, and doesn't halt on other inputs. □

Proof (of Theorem 4.1). Fix α, and suppose for contradiction that $L \in \mathsf{TTRT} - \mathrm{PSPACE}/\alpha(n)$. Let t_0 be the time bound given in the definition of TTRT, and let U_0 be some arbitrary time-efficient universal prefix machine. By the definition of TTRT, $L \leq_{tt}^{p} R_{K_{U_0}^{t_0}}$. Therefore, by Proposition 2.5, L is decidable in time $t_L(n) = 2^{n^k} t_0(n^k)$ for some constant k.

Let $t^*(n)$ be an extremely fast-growing function, so that for any constant d, we have $t^*(\log(\alpha(n))) > 2^{n^d} t_L(n)$ for all large n. To get our contradiction, we will show that there exists a time-efficient universal prefix machine U such that $L \not\leq_{tt}^{p} R_{K_U^{t^{*3}}}$. Note that because $t^* > t_0$, this is a contradiction to the fact that $L \in \mathsf{TTRT}$.

For any function $f : \{0,1\}^* \to \mathbb{N}$, define $R_f = \{x : f(x) \geq |x|\}$. We will construct a function $F : \{0,1\}^* \to \mathbb{N}$ and use it to form a function $H : \{0,1\}^* \to \mathbb{N}$ such that:

1. F is a total function and $F(x)$ is computable in time ${t^*}^2(|x|)$ by a machine M.
2. $H(x) = \min(K_{U_0}^{t^*}(x) + 5, F(x) + 3)$.
3. $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq 1/8$
4. $L \not\leq_{tt}^{p} R_H$

Claim (Analogue of Claim 1 from [5]). Given the above properties $H = K_U^{t^{*3}}$ for some efficient universal prefix machine U.

By Property 4 this ensures that the theorem holds.

Proof. By Property 3 we have that $\sum_{x \in \{0,1\}^*} 2^{-(F(x)+3)} \leq 1/8$. Hence $\sum_{x \in \{0,1\}^*} 2^{F(x)} \leq 1$. Using this along with Property 1, we then have by Proposition 4.2 that $K_{M'}^{t^{*3}} = F + 2$ for some prefix machine M'. By Proposition 4.4 we have that $K_{U'}^{t^*} = K_{U_0}^{t^*} + 4$ for some efficient universal prefix machine U'. Therefore, by Proposition 4.3, with $f(n) = n^3, g(n) = n$, we find that $H(x) = \min(K_{U_0}^{t^*}(x) + 5, F(x) + 3) = \min(K_{U'}^{t^*}(x), K_{M'}^{t^{*3}}) + 1$ is $K_U^{t^{*3}}$ for some efficient universal prefix machine U. □

We now need to show that, for our given language L, we can always construct functions H and F with the desired properties. As part of this construction we will set up and play a number of games. Our moves in the game will define the function F. Potentially during one of these games, we will play a move forcing a string z to be in the complement of R_H. To do this we will set $F(z) = |z| - 4$. Therefore, a machine M can compute $F(z)$ by running our construction, looking for the first time during the construction that $F(z)$ is set to $|z| - 4$, and outputting $|z| - 4$. If a certain amount of time elapses during the construction without $F(z)$ ever being set to $|z| - 4$, then the machine M outputs the default value $2|z|$.

As in [5], to ensure that $L \not\leq_{tt}^{p} R_H$, we need to satisfy an infinite list of requirements of the form

R_e : γ_e is not a polynomial-time truth-table reduction of L to R_H.

In contrast to the situation in [5], we do not need to worry about playing different games simultaneously or dealing with requirements in an unpredictable order; we will first satisfy R_1, then R_2, etc. To satisfy R_e we will set up a game $\mathcal{G}_{e,x}$ for an appropriate string x of our choice, and then play out the game in its entirety. We will choose x so that we can win the game $\mathcal{G}_{e,x}$, which will ensure that R_e is satisfied. If the K player cheats on game $\mathcal{G}_{e,x}$, then we play $\mathcal{G}_{e,x'}$ for some x'. For the same reasons as in [5] the K player cannot cheat infinitely often on games for a particular e, so eventually R_e will be satisfied.

A game $\mathcal{G}_{e,x}$ will be played as follows:

First we calculate the circuit $\gamma_{e,x}$, which represents the reduction γ_e on input x. Let F^* be the function F as it is at this point of the construction when the game $\mathcal{G}_{e,x}$ is about to be played. For any query z_i that is an input of this circuit such that $|z_i| \leq \log(\alpha(|x|)) - 1$, we calculate $r_i = \min(K^{t^*}_{U_0}(z_i) + 5, F^*(z_i) + 3)$. If $r_i < |z_i|$ we substitute FALSE in for the query, and simplify the circuit accordingly, otherwise we substitute TRUE in for the query, and simplify the circuit accordingly. (We will refer to this as the "pregame preprocessing phase".)

The remaining queries z_i are then ordered by increasing length. There are two players, the F player (whose moves will be played by us during the construction), and the K player (whose moves will be determined by K_{U_0}). As in [5], in each game the F player will either be playing on the YES side (trying to make the final value of the circuit equal TRUE), or the NO side (trying to make the final value of the circuit equal FALSE).

Let S_1 be the set of queries from $\gamma_{e,x}$ of smallest length, let S_2 be the set of queries that have the second smallest length, etc. So we can think of the queries being grouped into an ordered set $\mathcal{S} = (S_1, S_2, \ldots, S_r)$ for some r.

The scoring for the game is similar to that in [5]; originally each player has a score of 0 and a player loses if his score exceeds some threshold ϵ. When playing a game $\mathcal{G}_{e,x}$, we set $\epsilon = 2^{-e-6}$.

In round one of the game, the K player makes some (potentially empty) subset Z_1 of the queries from S_1 nonrandom. For any $Z_1 \subseteq S_1$ that he chooses to make nonrandom, $\sum_{z \in Z_1} 2^{-(|z|-6)} - 2^{-2|z|}$ is added to his score. As in [5], a player can only legally make a move if doing so will not cause his score to exceed ϵ.

Let us provide some explanation of how to interpret this score. Originally the function H is set so that for all z, $H(z) = 2|z|$. Because $H = \min(K^{t^*}_{U_0} + 5, F + 3)$, if $K^{t^*}_{U_0}(z) \leq |z| - 6$ then this ensures that z will be non-random according to H. It would be sub-optimal for the K player to set $K^{t^*}_{U_0}(z)$ to a value lower than $|z| - 6$, because this would add more to his score without any additional benefit. Therefore we assume without loss of generality that when the K player makes a move he does so in exactly this way. Thus the amount that is added to the score of player K corresponds to the amount by which K is changing the probability assigned to each string z (viewing K as a probability function). As in [5], for the case of analyzing the games and determining who has a winning strategy, we assume that the K player is an adversary playing optimally, even though in reality his moves will be based on an enumeration that knows nothing of these games.

After the K player makes his move in round 1, the F player responds, by making some subset Y_1 of the queries from $S_1 - Z_1$ nonrandom. After the F player moves, $\sum_{z \in Y_1} 2^{-(|z|-4)} - 2^{-2|z|}$ is added to his score.

This is the end of round one. Then we continue on to round two, played in the same way. The K player goes first and makes some subset of the queries from S_2 nonrandom (which makes his score go up accordingly), and then the F player responds by making some subset of the remaining queries from S_2 nonrandom. Note that if a query from S_i is not made nonrandom by either the K player or the F player in round i, it cannot be made nonrandom by either player for the remainder of the game.

After r rounds are finished the game is done and we see who wins, by evaluating the circuit $\gamma_{e,x}$ using the answers to the queries that have been established by the play of the game. If the circuit evaluates to TRUE (FALSE) and the F player is playing as the YES (NO) player, then the F player wins, otherwise the K player wins.

Note that the game is asymmetric between the F player and the K player; the F player has an advantage due to the fact that he plays second in each round and can make an identical move for fewer points than the K player. Because the game is asymmetric, it is possible that F can have a winning strategy playing on *both* the YES and NO sides. Thus we define a set $val(\mathcal{G}_{e,x'}) \subseteq \{0,1\}$ as follows: $0 \in val(\mathcal{G}_{e,x'})$ if the F player has a winning strategy playing on the NO side in $\mathcal{G}_{e,x'}$, and $1 \in val(\mathcal{G}_{e,x'})$ if the F player has a winning strategy playing on the YES side in $\mathcal{G}_{e,x'}$.

Now we describe the construction. Suppose s time steps have elapsed during the construction up to this point, and we are getting ready to construct a new game in order to satisfy requirement R_e. (Either because we just finished satisfying requirement R_{e-1}, or because K cheated on some game $\mathcal{G}_{e,x}$, so we have to start a new game $\mathcal{G}_{e,x'}$).

Starting with the string $0^{t^{*4}(s)}$ (i.e. the string of $t^{*4}(s)$ zeros), we search strings in lexicographical order (as we do in [5]) until we find an x' such that $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. (Here, L denotes the characteristic function of the set L.)

Once we find such a string x' (which we will prove we always can), then we play out the game $\mathcal{G}_{e,x'}$ with the F player (us) playing on the YES side if $L(x') = 0$ and the NO side if $L(x') = 1$. To determine the K player's move in the ith round, we let $Z_i \subseteq S_i$ be those queries $z \in S_i$ for which $K^{t^*}_{U_0}(z) + 5 < |z|$. Our moves are determined by our winning strategy, and are played as in [5]. (These determine the function F; as in [5] initially $F(x) = 2|x|$ for all x). If the game is completed without the K player cheating, then we will have won the game, and R_e will be satisfied and will stay satisfied for the rest of the construction.

Note that when a game $\mathcal{G}_{e,x}$ is played, x is always chosen large enough so that any query that is not fixed during the pregame preprocessing has not appeared in any game that was played previously, so the games will never conflict with each other.

The analysis for why Properties 3 and 4 hold is basically identical to [5].

To wrap up the proof of the theorem, we need to prove a couple of claims.

Claim (Analogue of Claim 4 from [5]). During the construction, for any requirement R_e, we can always find a witness x with the needed properties to construct $\mathcal{G}_{e,x}$.

Proof. Suppose for some requirement R_e, our lexicographical search goes on forever without finding an x such that $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. Then $L \in$ PSPACE/$\alpha(n)$, which is a contradiction.

Here is the PSPACE algorithm to decide L. Hardcode all the answers for the initial sequence of strings up to the point where we got stuck in the construction. Let F^* be the function F up to that point in the construction. On a general input x, construct $\gamma_{e,x}$. The advice function $\alpha(n)$ will give the truth-table of $\min(K^{t^*}_{U_0}(z) + 5, F^*(z) + 3)$ for all queries z such that $|z| \leq \log(\alpha(|x|)) - 1$. For any query z of $\gamma_{e,x}$ such that $|z| \leq \log(\alpha(|x|)) - 1$, fix the answer to the query according to the advice.

If the F player had a winning strategy for both the YES and NO player on game $\mathcal{G}_{e,x}$, then we wouldn't have gotten stuck on R_e. Also the F player must have a winning strategy for either the YES or the NO player, since he always has an advantage over the K player when playing the game. Therefore, because we got stuck, it must be that the F player has a winning strategy for the YES player if and only if $L(x) = 1$. Once the small queries have been fixed, finding which side (YES or NO) the F player has a winning strategy for on $\mathcal{G}_{e,x}$, and hence whether $L(x) = 1$ or $L(x) = 0$, can be done in PSPACE.[5] □

Claim. $F(z)$ is computable in time ${t^*}^2(|z|)$

Proof. The function F is determined by the moves we play in games during the construction. In order to prove the claim, we must show that if during the construction we as the F player make a move that involves setting a string z to be non-random, then fewer than ${t^*}^2(|z|)$ time steps have elapsed during the construction up to that point. The machine M that computes F will on input z run the construction for ${t^*}^2(|z|)$ steps. If at some point before this during the construction we as the F player make z non-random, then M outputs $|z| - 4$. Otherwise M outputs $2|z|$.

Suppose during the construction that we as the F player make a move that sets a query z to be non-random during a game $\mathcal{G}_{e,x}$. Note that $|z| \geq \log(\alpha(|x|))$, otherwise z would have been fixed during the preprocessing stage of the game.

There are at most $2^{|x|+1}$ strings x' that we could have considered during our lexicographic search to find a game for which we had a winning strategy before finally finding x. Let s be the number of time steps that have elapsed during the construction before this search began.

Let us first bound the amount of time it takes to reject each of these strings x'. To compute the circuit $\gamma_{e,x'}$ takes at most $|x'|^k$ time for some constant k. For each query y such that $|y| \leq \log(\alpha(|x'|)) - 1$ we compute $\min(K^{t^*}_{U_0}(y) + 5, F^*(y) + 3)$. To calculate $F^*(y)$ it suffices to rerun the construction up to this point and check whether a move had been previously made on the string y. To do this takes s time steps, and by construction we have that $t^*(|z|) \geq t^*(\log \alpha(|x|)) > |x'| \geq {t^*}^4(s)$, so $s < |z|$. By Proposition 2.4, to compute $K^{t^*}_{U_0}(y)$ takes at most $2^{|y|}t^*(|y|) \leq 2^{|z|}t^*(|z|)$ times steps. Therefore, since there can be at most $|x'|^k$ such queries, altogether computing $\min(K^{t^*}_{U_0}(y) + 5, F^*(y) + 3)$ for all these y will take fewer than $|x'|^k 2^{|z|} t^*(|z|)$ time steps.

Then we must compute $L(x')$, and check whether $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. Computing $L(x')$ takes $t_L(|x'|)$ time. By Claim 4, once the small queries have been fixed appropriately, computing $val(\mathcal{G}_{e,x'})$ can be done in PSPACE, so it takes at most $2^{|x'|^d}$ time for some constant d.

Compiling all this information, and using the fact that for each of these x' we have that $|x'| \leq |x|$, we get that the total number of timesteps needed to reject all of these x' is less than $2^{|x|^{d'}} 2^{|z|} t_L(|x|) t^*(|z|)$ for some constant d'.

[5] This follows from [5], as these games are a restricted case of the games from that paper. The point is that we can write the predicate "The F player has a winning strategy as the YES player on $\mathcal{G}_{e,x}$" as a simple quantified boolean formula.

During the actual game $\mathcal{G}_{e,x}$, before z is made non-random the construction might have to compute $K^{t^*}_{U_0}(y) + 5$ for all queries of $\gamma_{e,x}$ for which $|y| \leq |z|$. By Proposition 2.4 this takes at most $|x|^k 2^{|z|} t^*(|z|)$ time.

Therefore, overall, for some constant d'' the total amount of time steps elapsed before z is made non random in the construction is at most

$$T = 2^{|x|^{d''}} 2^{|z|} t_L(|x|) t^*(|z|) + s < {t^*}^2(|z|).$$

Here the inequality follows from the fact that $t^*(\log(\alpha(|x|))) > 2^{|x|^d} t_L(|x|)$ for any constant d, and that $|z| \geq \log(\alpha(|x|))$. □

5 Removing the Advice

With the plain Kolmogorov complexity function C, it is fairly clear what is meant by a "time-efficient" universal Turing machine. Namely, U is a time-efficient universal Turing machine if, for every Turing machine M, there is a constant c so that, for every x, if there is a description d for which $M(d) = x$ in t steps, then there is a description d' of length $\leq |d| + c$ for which $U(d') = x$ in at most $ct \log t$ steps. However, with prefix-free Kolmogorov complexity, the situation is more complicated. The easiest way to define universal Turing machines for the prefix-free Kolmogorov complexity function K is in terms of *self-delimiting Turing machines.* These are machines that have one-way access to their input tape; x is a valid input for such a machine if the machine halts while scanning the last symbol of x. For such machines, the notion of time-efficiency carries over essentially unchanged. However, there are several other ways of characterizing K (such as in terms of partial-recursive functions whose domains form a prefix code, or in terms of prefix-free entropy functions). The running times of the machines that give short descriptions of x using some of these other conventions can be substantially less than the running times of the corresponding self-delimiting Turing machines. This issue has been explored in detail by Juedes and Lutz [10], in connection with the P versus NP problem. Given that there is some uncertainty about how best to define the notion of time-efficient universal Turing machine for K^t-complexity, one possible response is simply to allow much more leeway in the time-efficiency requirement.

If we do this, we are able to get rid of the small amount of non-uniformity in our PSPACE upper bound.

Definition 5.1. *A prefix machine U is an f-efficient universal prefix machine if there exist constants c_M for each prefix machine M, such that*

1. *$\forall x$, $K_U(x) \leq K_M(x) + c_M$*
2. *$\forall x$, $K^t_U(x) \leq K^{t'}_M(x) + c_M$ for all $t(n) > f(t'(n))$*

In Definition 2.1 we defined a time-efficient universal prefix machine to be any poly(n)-efficient universal prefix machine.

Definition 5.2. *Define TTRT' to be the class of languages L such that for all computable f there exists t_0 such that for all f-efficient universal prefix machines U and $t \geq t_0$, $L \leq^p_{tt} R_{K^t_U}$.*

Theorem 5.3. BPP $\subseteq \mathsf{TTRT}' \subseteq$ PSPACE $\cap$ P/poly.

Note that $\mathsf{TTRT}' \subseteq \mathsf{TTRT}$, so from Theorem 3.2 we get $\mathsf{TTRT}' \subseteq$ P/poly. Also, the proofs in [7] can be adapted to show that BPP $\subseteq \mathsf{TTRT}'$. So all we need to show is the PSPACE inclusion.

Proof (of Theorem 5.3).
The proof is similar to the proof of Theorem 4.1, with some minor technical modifications. Let L be an arbitrary language from TTRT'. Because $\mathsf{TTRT}' \subseteq \mathsf{TTRT}$, as in the proof of Theorem 4.1 we have that L is decidable in time $t_L < 2^{n^k} t'(n^k)$ for some fixed time bound t' and constant k.

Define f to be a fast enough growing function that $f(n) > 2^{(t_L(n^d))^d}$ for any constant d. Let t_0 be the time bound given in the definition of TTRT' for language L and function f. Let $t^*(n)$ be a time bound such that for all n, $t^*(n) > f(n)$ and $t^*(n) > t_0(n)$. To get our contradiction, we will show that there exists an f-efficient universal prefix machine U and constant $c > 1$ such that $L \not\leq^p_{tt} R_{K^v_U}$, where $v(|x|) = 2^{(t_L(t^*(|x|)))^c} > t_0(|x|)$.

We will make use of the following revised proposition:

Proposition 5.4 (Revised Proposition 4.3). *Let U and M be an n^c-efficient universal prefix Turing machine and a prefix Turing machine respectively. Let t, t' be time bounds and f, g be two time-constructible increasing functions, such that $g(n^c) < f(n)$. Let $t''(|x|) = g(t(|x|)) = h(t'(|x|))$. Then there is an f-efficient universal prefix machine U' such that*

$$K^{t''}_{U'}(x) = \min(K^t_U(x), K^{t'}_M(x)) + 1.$$

Proof. Almost identical to before: On input $0y$, U' runs U on input y. If U would output string x on y after s steps, then U' outputs string x after $g(s)$ steps. Similarly, on input $1y$, U' runs M on input y. If M would output string x on y after s steps, then U' outputs string x after $h(s)$ steps.

Note that because U is an n^c-efficient universal prefix machine, U' will be an f-efficient universal prefix machine. □

We will construct functions F and H such that

1. F is a total function such that for all x, $F(x) \leq 2|x|$, and $F(x)$ is computable in time $2^{(t_L(t^*(|x|)))^d}$ by a machine M for some constant d.
2. $H(x) = \min(K^{t^*}_{U_0} + 5, F(x) + 3)$.
3. $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq 1/8$
4. $L \not\leq^p_{tt} R_H$

Claim (Revised Claim 4). Given the above properties $H = K^v_U$ for some f-efficient universal prefix machine U (which by Property 4 ensures that the theorem holds)

Proof. By Property 3 we have that $\sum_{x \in \{0,1\}^*} 2^{-F(x)+3} \leq 1/8$. Therefore it holds that $\sum_{x \in \{0,1\}^*} 2^{F(x)} \leq 1$. Using this along with Property 1, we then have by Proposition

4.2 that $K^u_{M'} = F + 2$ for some prefix machine M' and constant d', where $u(x) = 2^{(t_L(t^*(|x|)))^{d'}}$. By Proposition 4.4 we have that $K^{t^*}_{U'} = K^{t^*}_{U_0} + 4$ for some $n^{c'}$-efficient universal prefix machine U'. Therefore, by Proposition 5.4, $H(x) = \min(K^{t^*}_{U_0}(x) + 5, F(x) + 3) = \min(K^{t^*}_{U'}(x), K^u_{M'}(x)) + 1$ is K^v_U for some f-efficient universal prefix machine U and constant $c > 1$, where $v(|x|) = 2^{(t_L(t^*(|x|)))^c}$. (In this last step we are using the fact that $f(n) > 2^{(t_L(n^k))^k}$ for any constant k to ensure that U is an f-efficient universal prefix machine by Proposition 5.4). □

The construction is virtually the same as in Theorem 4.1.

There is one change from Theorem 4.1 in how the games are played. During the preprocessing step of a game $\mathcal{G}_{e,x}$, all queries z such that $t^*(|z|) \leq |x|$ are fixed according to $\min(K^{t^*}_{U_0}(z) + 5, F^*(z) + 3)$.

If we get stuck during our lexicographical search to find a suitable x' to play the game $\mathcal{G}_{e,x'}$, then this implies that the language L is in PSPACE, since by Proposition 2.4, for some constant k fixing all queries z such that $t^*(|z|) \leq |x|$ according to $\min(K^{t^*}_{U_0}(z) + 5, F^*(z) + 3)$ can be done in $|x|^k 2^{|z|} t^*(|z|) \leq |x|^k t^*(|z|)^2 \leq |x|^{k+2}$ time (and then it is a PSPACE computation to determine which side the F player has a winning strategy for).

It remains to prove the following claim.

Claim. $F(z)$ is computable in time $2^{(t_L(t^*(|z|)))^d}$ for some constant d.

Proof. Suppose during the construction we as the F player make a move that sets a query z to be non-random during a game $\mathcal{G}_{e,x}$. Note that $t^*(|z|) > |x|$, otherwise z would have been fixed during the preprocessing stage of the game.

As in the proof of Claim 4, we can bound the total amount of time steps elapsed before z is made non random in the construction to be at most

$$T = 2^{|x|^d} 2^{|z|} t_L(|x|) t^*(|z|) + s < 2^{(t_L(t^*(|z|)))^d} \square$$

6 Conclusion

We have made some progress towards settling our research question in the case of time-bounded Kolmogorov complexity, but we have also discovered that this situation is substantially different from the ordinary Kolmogorov complexity. Solving this latter case will likely prove to be much harder.

We would like to prove an exact characterization, such as BPP $=$ DTTR (or the time-bounded analogue thereof), but there seems to be no naive way of doing this. It has been shown in [7] that the initial segment $R_K^{\leq \log n}$, a string of length n, requires circuits of size n/c, for some $c > 1$ and all large n; it is this fact that is used to simulate BPP. However, much stronger circuit lower bounds for the initial segment do not seem to hold (cf. Theorems 4–9 of [7]), suggesting that R_K has some structure. This structure can actually be detected — the reduction M of Theorem 3.4 can be adapted to distinguish R_K from a random oracle w.h.p. — but we still don't know of any way of using R_K non-adaptively, other than as a pseudo-random string. A new idea will be needed in order to either prove or disprove the BPP $=$ DTTR conjecture.

Acknowledgments

The first and third authors acknowledge NSF Grants CCF-0832787 and CCF-1064785.

References

1. E. Allender. Curiouser and curiouser: The link between incompressibility and complexity. In *Proc. Computability in Europe (CiE)*, LNCS. Springer, 2012. To appear.
2. 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.
3. E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35:1467–1493, 2006.
4. E. Allender, G. Davie, L. Friedman, S. B. Hopkins, and I. Tzameret. Kolmogorov complexity, circuits, and the strength of formal theories of arithmetic. Technical Report TR12-028, Electronic Colloquium on Computational Complexity, 2012. Submitted for publication.
5. E. Allender, L. Friedman, and W. Gasarch. Limits on the computational power of random strings. *Information and Computation*, 2012. To appear; special issue on ICALP 2011. See also ECCC TR10-139.
6. José Luis Balcázar, Josep Días, and Joaquim Gabarró. *Structural Complexity I*. Springer-Verlag, 1988.
7. 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, 2010.
8. H. Buhrman, L. Fortnow, I. Newman, and N. Vereshchagin. Increasing Kolmogorov complexity. In Volker Diekert and Bruno Durand, editors, *STACS 2005*, volume 3404 of *Lecture Notes in Computer Science*, pages 412–421. Springer Berlin / Heidelberg, 2005.
9. Harry Buhrman and Elvira Mayordomo. An excursion to the kolmogorov random strings. *J. Comput. Syst. Sci.*, 54(3):393–399, 1997.
10. David W. Juedes and Jack H. Lutz. Modeling time-bounded prefix Kolmogorov complexity. *Theory of Computing Systems*, 33(2):111–123, 2000.
11. M. Li and P. Vitanyi. *Introduction to Kolmogorov Complexity and its Applications*. Springer, third edition, 2008.