Electronic Colloquium on Computational Complexity, Report No. 15 (2004)

Enumerations of the Kolmogorov Function

Richard Beigel[a] Harry Buhrman[b] Peter Fejer[c]
Lance Fortnow[d] Piotr Grabowski[e] Luc Longpré[f]
Andrej Muchnik[g] Frank Stephan[h] Leen Torenvliet[i]

Abstract

A recursive enumerator for a function h is an algorithm f which enumerates for an input x finitely many elements including $h(x)$. f is an $k(n)$-enumerator if for every input x of length n, $h(x)$ is among the first $k(n)$ elements enumerated by f. If there is a $k(n)$-enumerator for h then h is called $k(n)$-enumerable. We also consider enumerators which are only A-recursive for some oracle A.

We determine exactly how hard it is to enumerate the Kolmogorov function, which assigns to each string x its Kolmogorov complexity:

[a]Email: beigel@cis.temple.edu. Deptartment of Computer and Information Sciences, Temple University, 1805 North Broad Street, Philadelphia PA 19122, USA. Research performed in part at NEC and the Institute for Advanced Study. Supported in part by a State of New Jersey grant and by the National Science Foundation under grants CCR-0049019 and CCR-9877150.

[b]Email: buhrman@cwi.nl. CWI, Kruislaan 413, 1098SJ Amsterdam, The Netherlands. Partially supported by the EU through the 5th framework program FET.

[c]Email: fejer@cs.umb.edu. Department of Computer Science, University of Massachusetts Boston, Boston, MA 02125, USA.

[d]Email: fortnow@cs.uchicago.edu. Department of Computer Science, University of Chicago, 1100 East 58th Street, Chicago, IL 60637, USA. Research performed in part at NEC Research Institute.

[e]Email: pgrabows@ix.urz.uni-heidelberg.de. Institut für Informatik, Im Neuenheimer Feld 294, 69120 Heidelberg, Germany.

[f]Email: longpre@cs.utep.edu. Computer Science Department, UTEP, El Paso, TX 79968, USA.

[g]Email: muchnik@lpcs.math.msu.ru. Institute of New Techologies, Nizhnyaya Radishevskaya, 10, Moscow, 109004, Russia. The work was partially supported by the Russian Foundation for Basic Research grants 01-01-00505 and 02-01-10904.

[h]Email: fstephan@cse.unsw.edu.au. National ICT Australia LTD, Sydney Research Laboratory at Kensington, The University of New South Wales, Sydney NSW 2052, Australia. Research supported by the Deutsche Forschungsgemeinschaft (DFG), Heisenberg grant Ste 967/1-1 while previously working at the Universität Heidelberg.

[i]Email: leen@science.uva.nl. Institute for Language Logic and Computation, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands.

ISSN 1433-8092

- For every underlying universal machine U, there is a constant a such that C is $k(n)$-enumerable only if $k(n) \geq n/a$ for almost all n.
- For any given constant k, the Kolmogorov function is k-enumerable relative to an oracle A if and only if A is at least as hard as the halting problem.
- There exists an r.e., Turing-incomplete set A such for every non-decreasing and unbounded recursive function k, the Kolmogorov function is $k(n)$-enumerable relative to A.

The last result is obtained by using a relativizable construction for a nonrecursive set A relative to which the prefix-free Kolmogorov complexity differs only by a constant from the unrelativized prefix-free Komogorov complexity.

Although every 2-enumerator for C is Turing hard for K, we show that reductions must depend on the specific choice of the 2-enumerator and there is no bound on the quantity of their queries. We show our negative results even for strong 2-enumerators as an oracle where the querying machine for any x gets directly an explicit list of all hypotheses of the enumerator for this input. The limitations are very general and we show them for any recursively bounded function g:

- For every Turing reduction M and every non-recursive set B, there is a strong 2-enumerator f for g such that M does not Turing reduce B to f.
- For every non-recursive set B, there is a strong 2-enumerator f for g such that B is not wtt-reducible to f.

Furthermore, we deal with the resource-bounded case and give characterizations for the class $\mathrm{S}_2^{\mathrm{p}}$ introduced by Russell and Sundaram and the classes PSPACE, EXP.

- $\mathrm{S}_2^{\mathrm{p}}$ is the class of all sets A for which there is a polynomially bounded function g such that there is one tt-reduction which reduces A to every strong 2-enumerator for g.
- PSPACE is the class of all sets A for which there is a polynomially bounded function g such that there is one Turing reduction which reduces A to every strong 2-enumerator for g. Interestingly, g can be taken to be the Kolmogorov function for the conditional space-bounded Kolmogorov complexity.
- EXP is the class of all sets A for which there is a polynomially bounded function g and a machine M which witnesses $A \in \mathrm{PSPACE}^f$ for all strong 2-enumerators f for g.

Finally, we show that any strong $O(\log n)$-enumerator for the conditional space-bounded Kolmogorov function must be PSPACE-hard if $\mathrm{P} = \mathrm{NP}$.

1 Introduction

The Kolmogorov complexity of a binary string x, $C(x)$, is the size of the smallest program that outputs x. Kolmogorov complexity has its roots in the study of randomness: it is one way to measure randomness in a string. It has had a vast area of applications, including information theory, combinatorics, analysis of algorithms, distributed computing, statistical properties of long finite and of infinite sequences, learning theory, and even quantum information processing. It proved to be an invaluable tool in proving or simplifying the proofs of a large number of lower bounds. See Li and Vitányi [19] for a discussion of many of these directions.

The Kolmogorov complexity is not computable [9, 16, 26], it is even hard for every r.e. set. When a set or function is not computable or intractable, one often turns to the complexity of approximations. For example, would it be possible to approximate the Kolmogorov function to within reasonable bounds? Kolmogorov [27] showed that the Kolmogorov complexity function can be approximated from above: there is a total recursive function $\tilde{C}$ such that $C(x) = \min\{\tilde{C}(t, x) : t = 1, 2, 3, \ldots\}$.

A different approach to approximation is that if we cannot compute the result of the function exactly, perhaps it would be possible to output several candidates for the result of the function, one of which is the actual result. Traditional approximations are a special case of this, in which the set of candidates is the set of numbers in a given range. This kind of approximation has been called enumeration complexity, see [1, 2, 4, 5, 7, 8, 12, 18]. Bill Gasarch suggested the natural question whether we can approximate the Kolmogorov function in this sense, or more precisely, how many values does a Turing machine need to output before it is guaranteed that one of the values is $C(x)$.

By a simple table-lookup argument, $C(x)$ can be $(n - a)$-enumerable for every a, where $n = |x|$. For every constant a there is even a programming system such that $C(x)$ is n/a-enumerated, see Remark 3.2. However, we show that for every programming system and enumeration of the resulting C there is another constant c such that for every length n there is an $x \in \{0, 1\}^n$ for which the enumeration outputs more than n/c many hypotheses.

Next we look at how much extra power a Turing machine needs before it can compute an $O(1)$-enumeration of the Kolmogorov function. We show that such a machine must be powerful enough to compute the Kolmogorov function directly. That is, for constant k, the Kolmogorov function is k-enumerable relative to an oracle A if and only if the halting problem is Turing-reducible to A. However we show in Theorem 3.7 that for some very slowly growing function k, the Kolmogorov function is $k(n)$-enumerable relative to an oracle for an incomplete r.e. set.

The proof of Theorem 3.7 is based on a result which is more than just a tool. It shows that there are non-recursive oracles A relative to which the prefix-free Kolmogorov complexity is up to a constant identical with the non-relativized one. This class of oracles is obviously closed under Turing reduction and it has several other natural characterizations [21].

Then we investigate the computational power provided by an oracle for a k-enumerator. We show that a single query to a strong 2-enumerator for the Kolmogorov function allows us to extend a partial recursive function to a total recursive function. However, even unlimited access to a strong 2-enumerator provides essentially no help in computing sets.

Our results have some nice complexity theoretic counterparts. In Section 6, we characterize the class $\mathrm{S}_2^{\mathrm{p}}$ [24] in terms of bounded truth-table and truth-table reductions to strong 2-enumerators. We show that $\mathrm{P} = \mathrm{PSPACE}$ if the polynomial space-bounded Kolmogorov function has a polynomial-time strong 2-enumerator. This result makes use of the theorem that the sets in the polynomial hierarchy are Turing reducible to any strong 2-enumerator for the Kolmogorov function independent of the actual choice of the enumerator. This contrasts to the recursion theoretic case where no non-recursive set is Turing reducible to an arbitrary enumerator by a fixed reduction. Finally we show that every strong $O(\log n)$-enumerator of the polynomial-space bounded Kolmogorov function is hard for PSPACE under nondeterministic polynomial time reductions.

2 Preliminaries

We assume the reader is familiar with basic notions in computational complexity theory. Fix a universal Turing machine U. Except as explicitly stated, all our results are independent of the particular choice of the universal machine. Strings are denoted as lower case letters $x, y, u, v, \ldots$ and are all elements of $\{0, 1\}^*$. We use the 1-1 correspondence between strings and binary numbers and have numbers sometimes appear as arguments to functions on strings and vice versa. Also we let functions defined on strings sometimes act on numbers where the length of the number is the logarithm of its value.

Definition 2.1 *The conditional Kolmogorov complexity function $C(x|y)$, see [19], is given as $C(x|y) = \min\{|p| : U(p, y) = x\}$ where p stands for a program and U for a universal acceptable numbering of the functions $y \rightarrow U(p, y)$. We also use an unary version $U(p)$ as an abbreviation for $U(p, \lambda)$ and let $C(x) = \min\{|p| : U(p) = x\}$ be the unconditional Kolmogorov complexity. In* *most places, we will just work with the unary U and unconditional complexity C. U and C have recursive approximations U_s and C_s such that*

- *There is a q such that $U_0(qx) = x$;*
- *$U(p) = x \Leftrightarrow (\exists s)\,[U_s(p) = x]$;*
- *$C_s = \min\{|p| : U_s(p) = x\}$;*
- *The function $s, x \to C_s(x)$ is total and recursive in both parameters.*

The first condition guarantees that not only $C(x) \leq |x| + c$ but also, for each s, $C_s(x) \leq |x| + c$ for the constant $c = |q|$.

Hartmanis [13] defined a time-bounded version of Kolmogorov complexity, but resource-bounded versions of Kolmogorov complexity date back as far as 1968 [3], see also [19]. So C_{space} may be defined as space and time-bounded versions of C in the usual way.

Intuitively, a computable k-enumerator for a function h enumerates on any input x up to $k(|x|)$ possibilities such that the value $h(x)$ is among these values. Formal definitions are given below.

Definition 2.2 ($k(n)$-Enumerable) *A recursive $k(n)$-enumerator f for a function h is an algorithm which enumerates on input x a finite set, denoted by $f(x)$, such that*

- *$h(x)$ is among the enumerated values: $h(x) \in f(x)$;*
- *the cardinality is at most $k(n)$: $|f(x)| \leq k(n)$ where n is the length of x.*

If a function h has a recursive $k(n)$-enumerator f then h is called $k(n)$-enumerable.

If f is a recursive $k(n)$-enumerator for h and if in addition $x \to |f(x)|$ is a computable function then f is called a strong $k(n)$-enumerator for h and h is said to be strongly $k(n)$-enumerable.

For an oracle A, an A-recursive enumerator is an enumeration algorithm using the oracle A; furthermore, a strong A-recursive enumerator is an A-recursive enumerator where the function $x \to |f(x)|$ is also A-recursive. If it is not important, relative to which oracle A an A-recursive (strong) enumerator f is computed, then f is just called a (strong) enumerator.

In Section 4, strong enumerators are also used as oracles themselves; the query protocol is that a query is made at a place x and an explicit list of the elements of the set $f(x)$ is returned.

If one would query a recursive enumerator as an oracle by the protocol given above, it might be that one would retrieve information that cannot be computed. In contrast to that, a recursive strong enumerator does not give away any nonrecursive information. Therefore the above protocol of access to enumerators as oracles is indeed more adequate for strong enumerators. So we consider only strong enumerators as oracles. Friedberg and Rogers [11] introduced the notion of enumeration-reducibility, which is often abbreviated as e-reducibility. This notion would give an adequate environment to deal with computations relative to enumerators (and not only strong enumerators). But since most of our results using an enumerator as an oracle are negative results which even hold for accessing strong enumerators, it would not pay off to formalize our results within the framework of e-reducibility. Instead we limit ourselves to querying strong enumerators. In addition we consider reductions to sets A relative to which an A-recursive enumerator for C has certain properties (like, for example, being a 2-enumerator).

Remark 2.3 *In the following, one denotes by f_t^x the t-th element enumerated by f on input x. Thus the following connections hold between enumerators and the partial function $x, t \to f_t^x$:*

- *For every enumerator f and every x there is a bound b_x such that f_t^x is defined iff $t \in \{1, 2, \ldots, b_x\}$. Furthermore, $f(x) = \{f_t^x : f_t^x$ is defined$\} = \{f_1^x, f_2^x, \ldots, f_{b_x}^x\}$.*
- *f is an A-recursive enumerator iff the function $x, t \to f_t^x$ is a partial A-recursive function.*
- *f is a strong A-recursive enumerator iff the function $x, t \to f_t^x$ is a partial A-recursive function and has an A-recursive domain.*

If f is a (not necessarily recursive) enumerator for the Kolmogorov function, then one can without loss of generality assume the following additional properties of f:

- *For all x, $f_1^x > f_2^x > \ldots > f_{b_x}^x$.*
- *Assume that Kolmogorov complexity is defined with respect to the universal Turing machine U. For every x, t where f_t^x is defined there is a $p \in \{0, 1\}^{f_t^x}$ such that $U(p) = x$.*

Note that these conditions imply that $C(x) = f_{b_x}^x$. Thus one cannot compute b_x from x.

In order to keep the notion simple, k depends only on the length of x. We denote the length of x by n and might refer to $k(n)$-enumerators. When k

is constant, we will speak of 2-enumerators, 3-enumerators, k-enumerators and so on.

A set A is *weak-truth-table reducible* (wtt-reducible) to set B if there is a Turing reduction M from A to B such that there is a recursive function g which is an upper bound for the length of all queries of M: $M^B(x)$ never asks queries to B of length greater than $g(x)$. Furthermore, A is *truth-table reducible* (tt-reducible) to B if there is a Turing reduction M from A to B such that $M^C(x)$ is defined for all oracles C and inputs x. Note that for tt-reductions there also exists the upper bound g on the length of the queries of M.

3 Bounds for Turing Reducibility

The fact that the Kolmogorov function itself is hard for r.e. sets is a folk theorem with a relatively easy proof. Kummer [17] showed that the halting problem, K, is even tt-reducible to C. Conversely, with K as an oracle we can easily decide the Kolmogorov complexity of any string. So, loosely speaking, C and K have the same hardness.

Li and Vitanyi [19] considered functions m with the following properties: m is nondecreasing, m is unbounded and $m(n) \leq C(x)$ for all n and $x \in \{0,1\}^*$ with $|x| \geq n$. The proof of the hardness of the Kolmogorov function can be adapted to show that m is also hard for r.e.; one can even exactly take the proof in [27]: $\min\{n : m(n) > 2e\}$ must be larger than $\min\{t : \varphi_{e,t}(e) \downarrow\}$ if $e \in K$. For if $e \in K$ and $\varphi_{e,t'}(e) \downarrow$ only for $t' > \min\{n : m(n) > 2e\}$, then $C(0^{\min\{t' : \varphi_{e,t'}(e)\downarrow\}}) > 2e$. Yet it is described by e. This can only be true for finitely many e.

This paper is not about the complexity of the Kolmogorov function itself, but about enumerations of the Kolmogorov function. We ask how hard are these enumerations? Clearly, the complexity depends on the number of outputs of such a function. The Kolmogorov function can be seen as a 1-enumerator for itself and vice versa. Thus every 1-enumerator for C is hard for r.e. sets. On the other hand, for any constant a, we can compute an $(n-a)$-enumerator for C, essentially a Turing machine that outputs all values v with $c + a < v \leq n + c$ on inputs of length n except on the finite number of strings that have $C(x) \leq c+a$; there $C(x)$ is output explicitly. So the complexity somehow depends on the number of values that are output by the enumerator. We show first that no recursive function can do better than essentially enumerate all possible values of C.

Theorem 3.1 *There is a constant a depending only on the universal machine U defining the Kolmogorov complexity C such that every $k(n)$-enumerator f satisfies $k(n) \geq n/a$ for almost all n.*

Proof: Let C be the plain Kolmogorov complexity and U be the underlying unary universal machine. Furthermore, there is an enumeration of enumerators such that the following holds:

- The function $x, t \to f_t^x$ is partial-recursive.
- For each x there is a number b_x such that f_t^x is defined iff $1 \leq t \leq b_x$; note that no f_t^x is defined in the case that $b_x = 0$.
- For every x, t such that f_t^x is defined there is a $p \in \{0,1\}^{f_t^x}$ with $U(p) = x$.
- For every x, e, t such that 0^e1 is a prefix of x, the output of f and of the e-th enumerator are compatible in the following sense:
  - whenever f_t^x is defined, f_t^x occurs also in the output of the e-th enumerator at input x;
  - whenever the e-th enumerator at input x enumerates $C(x)$, there is a t with $f_t^x = C(x)$.

Informally, the last condition says that whenever the e-th enumerator is an $k(n)$-enumerator for C then f restricted to the inputs x with prefix 0^e1 is also a $k(n)$-enumerator for C on this restricted domain. These properties are obtained by adapting Remark 2.3 to a list of operators.

Let e be an index of an $k(n)$-enumerator and consider any $n \geq 2^e$. Let $X_{n,e} = \{0\}^e \cdot \{1\}^1 \cdot \{0,1\}^{n-e}$. Let in the following m be the maximal number such that there is an $x \in X_{n,e}$ with f_m^x being defined; note that $m \leq k(n)$ and $C(x) = f_m^x$ for this x. There are two cases.

(a) There is an $x \in X_{n,e}$ with f_m^x being defined and $f_m^x \geq n/2 - e$. A program knowing m, n, e can simulate the function f until an x is found such that f_m^x is defined and $f_m^x \geq n/2$. So one can describe x with $c_1 + 4 \cdot \log(n)$ many bits for a constant c_1 being independent of n, m, e. It follows that $n/2 \leq c_1 + 4 \cdot \log(n)$ and thus case (a) holds only for finitely many n.

(b) If f_m^x is defined for an $x \in X_{n,e}$ then $f_m^x < n/2 - e$. Let

$$m' = \max\{t \leq m : (\exists x \in X_{n,e})\, [f_t^x \text{ is defined and } C(x) \geq n - t \cdot b/2 - e]\}$$

where b is the greatest lower integer bound for n/m. Note that $m' < m$ since otherwise case (a) would hold. Among the $x \in X_{n,e}$ where $f_{m'+1}^x$ is defined there is an z where the computation of $f_{m'+1}^z$ terminates last. There is a

$d \geq 1$ such that $f^z_{m'+d} = C(z)$. There is a program y of length $f^z_{m'+d}$ which computes z. By choice of m', $C(z) < n - (m'+d) \cdot b/2$. Now it is shown that given y, b, d, one can compute an x' of length n with $C(x') \geq C(z) + d \cdot b/2 - e$.

1. Compute $U(y)$ – the result is z.
2. Compute the number of 0 before the first occurrence of a 1 in z – the result is e.
3. Compute the length of y – the result is $C(z)$.
4. Compute the length of z – the result is n.
5. Find the number m' such that $f^z_{m'+d} = C(z)$ – the value m' is unique and the same as above.
6. Determine the number s of steps to compute $f^z_{m'+1}$ and let
$$Y = \{x \in X_{n,e} : f^x_{m'+1} \text{ is computed in up to } s \text{ steps}\}$$
– by choice of z, Y contains exactly all $x \in X_{n,e}$ where $f^x_{m'+1}$ is defined at all.
7. Search for an $x' \in X_{n,e} - Y$ such that $f^{x'}_{m'} \geq C(z) + d \cdot b/2$ – this x' exists by the choice of m' and $f^{x'}_{m'} = C(x')$ by $f^{x'}_{m'+1}$ being undefined.

So one has that $C(z) + d \cdot b/2 \leq C(x')$. Furthermore, x' was computed from b, d and y, thus $C(x') \leq C(z) + 2 \cdot (\log(b) + \log(d)) + c_2$ for some constant c_2. Recall that $\log(b) + \log(d) = \log(b \cdot d)$. So, $b \cdot d \leq 4 \cdot \log(b \cdot d) + 2 \cdot c_2$. The value $a = \max\{b' + 2 : b' \text{ is a natural number and } b' \leq 4 \cdot \log(b') + 2 \cdot c_2\}$ is an upper bound for $b+1$ and thus for n/m.

So, putting all things together, case (b) applies to all sufficiently large n and $k(n) \geq n/a$ for the a defined in this case. Note that the constants c_1, c_2 are independent from e and that the above construction goes through whenever the e-th enumerator is an enumerator for C. □

Remark 3.2 *The result above tight: For every positive integer a there is a universal machine U such that every program's length is divisible by a. Then $C(x)$ is strongly n/a-enumerable for this fixed a.*

In order to prove the Turing-hardness of k-enumerating the Kolmogorov function, we state some properties of Π^0_1 classes which are used in the proof of this result.

Remark 3.3 **(Π^0_1-classes)** *An important ingredient for the next proof is the concept of Π^0_1-classes and the fact that such a class has a member which is not above a given non-recursive set.*

A Π^0_1-class is a class of sets such that there is a nullary oracle-machine M which either never halts or rejects the given set. As accepting an infinite set is an infinite process, it is adequate to define acceptance in this case by the absence of a (rejecting) halting state of M. So the Π^0_1-class S defined by M is given as

$$S = \{B \subseteq \mathbb{N} : M^B \text{ never halts}\}.$$

Equivalently, one can define that a Π^0_1-class is the set of infinite branches of a binary recursive tree T; here $\sigma \in T$ iff M has not yet halted and rejected with oracle σ: M^σ is undefined.

One can relativize the concept of a Π^0_1-class to an oracle A and adapt a result of Jockusch and Soare [15, Theorem 2.5] *to the following. Given a set $A \not\geq_T K$, the relativized Π^0_1-class*

$$S = \{B \subseteq \mathbb{N} : M^{A \oplus B} \text{ never halts}\}$$

is either empty or contains a set B such that $A \oplus B \not\geq_T K$.

Theorem 3.4 *Let k be a constant. If the Kolmogorov function is k-enumerable relative to a set A then $A \geq_T K$.*

Proof: Assume by way of contradiction that $A \not\geq_T K$ and the Kolmogorov function C is k-enumerable relative to A via f. In particular, $x, t \to f^x_t$ is a partial A-recursive function.

Let K_s be a recursive enumeration of K in the sense that K_s contains those s elements which are enumerated into K first; $K_0 = \emptyset$. Recall from Definition 2.1 that C can be approximated monotonically from above by the function $s, x \to C_s(x)$.

We uniformly in n, i recursively enumerate sets $X_{n,i}$. The algorithm is given below. It has n as a parameter and keeps track of the other parameter i; for each n, i there is at most one stage where elements are enumerated into the set $X_{n,i}$.

- In stage 0 let $i = 0$ and $X_{n,0} = \{0,1\}^{(n+1)(n+1)}$. Furthermore, initialize $X_{n,1}, X_{n,2}, \ldots$ as empty sets.
- In stage $s = 1, 2, \ldots$, check whether

  $$\{0, 1, \ldots, n\} \cap K_s \neq \{0, 1, \ldots, n\} \cap K_{s-1}.$$

  If so, do the following:

- Update $i = i + 1$.
- While $|X_{n,i}| < 2^{(n+1)(n+1-i)}$, select that $x \in X_{n,i-1} - X_{n,i}$ for which $C_s(x)$ is greatest and enumerate x into $X_{n,i}$.

Note that for every n, the parameter i is in the limit the number of stages s such that some $m \leq n$ goes into K at s.

Consider the class S of all functions F mapping each n into $X_{n,0}$ such that, for all n and $i \in \{0, 1, \ldots, n+1\}$, either $X_{n,i} = \emptyset$ or $F(n) \in X_{n,i}$. Note that formally S does not contain functions F but coded versions B_F of these F where each B_F has the characteristic function $F(0) \cdot F(1) \cdot F(2) \cdot \ldots$ being obtained from F by concatenating the strings of the values of F; B_F and F can easily be calculated from each other since $|F(n)| = (n+1)^2$ for all n.

The class S is a Π^0_1-class since the conditions to be checked are uniformly Π^0_1: Whenever some element is enumerated into $X_{n,i}$ then exactly $2^{(n+1)(n+1-i)}$ many elements are enumerated into $X_{n,i}$. So whenever one discovers that $X_{n,i}$ is not empty, one can check explicitly whether $F(n) \in X_{n,i}$.

Furthermore, S is not the empty class. A witness for the nonemptyness is the function F which is defined on input n as follows: Take i be the largest number with $X_{n,i} \neq \emptyset$ and let then $F(n)$ be the lexicographically minimal element of $X_{n,i}$. Since $X_{n,i} \subseteq \ldots \subseteq X_{n,1} \subseteq X_{n,0}$, one has that $F(n) \in X_{n,0}, X_{n,1}, \ldots, X_{n,i}$ and B_F is a member of S.

Using Remark 3.3, we fix F such that $B_F \in S$ and $A \oplus B_F \not\geq_T K$.

Now we construct a $(A \oplus B_F)$-computable function g which dominates the function c_K given by

$$c_K(n) = \max\{s : s = 0 \vee \{0, 1, \ldots, n\} \cap K_s \neq \{0, 1, \ldots, n\} \cap K_{s-1}\}$$

Then, for almost all n, $n \in K \Leftrightarrow n \in K_{g(n)}$. This gives then $A \oplus B_F \geq_T K$ in contradiction to the assumption on A and the choice of B_F.

For each n let j_n be the maximal t such that $f_t^{F(n)}$ is defined. Now let j be the limit superior of the j_n for $n \to \infty$ and let

> $g(n) = s$ for the first stage s for which there is an $m \geq n$ such that $f_j^{F(m)}$ is defined within s steps and $C_s(F(m)) = f_j^{F(m)}$.

Now it is verified that g dominates c_K.

Given any such n, let m be the value of the variable of the same name in the algorithm g. Furthermore, denote by $s_1, \ldots, s_i$ the stages where the elements of $X_{m,1}, \ldots, X_{m,i}$ are enumerated into these sets. Without loss of generality, $0 \in K$ and $i, s_i > 0$. So i is the largest index of a set $X_{m,i}$ which is not empty and $s_1 < s_2 < \ldots < s_i$. Furthermore, $F(m) \in X_{m,i}$ and $s_i = c_K(m)$.

On the one hand $F(m)$ enters $X_{m,i}$ at stage s_i and it follows from the definition of the $X_{m,i}$ that $C_{s_i}(F(m)) \geq C_{s_i}(x)$ for at least half of the members x of $X_{m,i-1}$. Since $X_{m,i-1}$ has $2^{(m+1)(m+2-i)}$ members, $C_{s_i}(F(m)) \geq (m+1)(m+2-i)-1$.

On the other hand, if one knows m, i and the number of elements which go into $X_{m,i}$ before $F(m)$ then one can compute $F(m)$. Since $i \leq m+1$, a prefix-free coding of the numbers m, i can be done using $3\log(m)+4$ bits. Furthermore, $X_{m,i}$ has $2^{(m+1)(m+1-i)}$ many members. Thus there is a constant c with $C(F(m)) \leq (m+1)(m+1-i) + 3\log(m) + c$.

If n is sufficiently large, then one can use $m \geq n$ to conclude that $j_m = j$, $C(F(m)) = f_j^{F(m)}$ and $C_s(F(m)) = C(F(m)) \leq (m+1)(m+1-i) + 3\log(m)+c < (m+1)(m+2-i)-1 \leq C_{s_i}(F(m))$. Since C is approximated from above, it follows that $s > s_i$ and $g(n) = s > s_i = c_K(m) \geq c_K(n)$. So g dominates c_K although g is computable relative to $A \oplus B_F$. This contradiction gives that the assumption $A \not\geq_T K$ is false. So C is k-enumerable only relative to those oracles which are hard for the halting-problem. □

If k is no longer bounded, but a recursive increasing function, then Theorem 3.7 below shows that one can find an r.e. incomplete degree relative to which the Kolmogorov function is k-enumerable. Theorem 3.7 uses the existence of a relativized construction giving a set which is low for prefix-free Kolmogorov complexity. Such sets are of independent interest and play a major role in the field of algorithmic randomness.

Definition 3.5 *A set $B \not\leq_T A$ is* low for prefix-free Kolmogorov complexity relative to A *if* $(\exists c)\,(\forall x)\,[H^A(x) \leq H^B(x) + c]$.

An enumeration operator $W : A \to W^A$ is given by a recursive subset $\tilde{W}$ of $\{0,1\}^* \times \{0,1\}^*$ for which one defines that

$$x \in W^A \Leftrightarrow (\exists \text{ prefix } \sigma \text{ of } A)\,[(x,\sigma) \in \tilde{W}].$$

Note that W^A is recursively enumerable relative to A for every oracle A. One can adapt the notion of a universal prefix-free machine in order to get that it is universal for every oracle A: Starting with an enumeration $V_0^A, V_1^A, \ldots$ of all prefix-free partial A-recursive machines, let $U^A(0^e1p) = V_e^A(p)$ and $U^A(0^e)$ be undefined for all e, p. For U^A there is a partial-recursive function $\tilde{U}$ such that its domain is a recursive subset of $\{0,1\}^* \times \{0,1\}^*$ and

$$U^A(p) = x \Leftrightarrow (\exists \text{ prefix } \sigma \text{ of } A)\,[\tilde{U}(p,\sigma) = x].$$

Without loss of generality $\tilde{U}(p,\eta) = \tilde{U}(p,\sigma)$ whenever $\tilde{U}(p,\sigma)$ is defined and σ is a prefix of η.

Theorem 3.6 *There is an enumeration operator E such that for every oracle A, $A \oplus E^A$ is low for prefix-free Kolmogorov complexity relative to A.*

Proof: One constructs E satisfying the following two conditions.

- E^A is simple relative to A.
- There is a constant c such that $(\forall x)\,[H^A(x) \leq H^{A\oplus E^A}(x) + c]$.

The construction is a modification of ideas of Post's construction of a simple set [22, Theorem III.2.11]. This modification takes care of two aspects: first the construction is made uniform in all oracles, second only those elements are enumerated which do not violate the Kraft-Chaitin condition explained below.

The sets $\tilde{E}$ and $\tilde{G}$ are actually recursive and one defines by induction over σ (with respect to length-lexicographic ordering) whether a tuple with last component σ goes in and if so, which one. In this definition the oracle B considered has to agree with σ on the domain of σ. At the other places the value of B does not matter since only computations lasting $|\sigma|$ steps are taken into account and they do not inspect B outside the domain of σ. So one can take the oracle B to have the characteristic function $\sigma 0^\infty$.

The step σ for $\tilde{E}$ searches for the first $e \leq |\sigma|$ and first $x \leq |\sigma|$ to satisfy the following conditions. If (e, x) are found, one puts (x, σ) into $\tilde{E}$ otherwise no pair (y, σ) goes into $\tilde{E}$. The first two conditions are from the construction of Post's simple set, the third is the Kraft-Chaitin condition.

- The first condition is $x \geq 2e$.
- Note that the elements enumerated into the set E^B within $|\sigma|$ steps are precisely those y such that there is a proper prefix η of σ with $(y, \eta) \in \tilde{E}$. Let $E^B_{|\sigma|}$ denote this set. The second condition is that no element is enumerated into the intersection of W^B_e and E^B within $|\sigma|$ steps, that is, the sets $W^B_{e,|\sigma|}$ and $E^B_{|\sigma|}$ are disjoint.
- Let $r^B_{|\sigma|}$ be the sum of all $2^{-|p|}$ where $(p, y, \eta) \in \tilde{G}$ for some y, η with $|\eta| \geq x$ and η being a proper prefix of σ; the sum of 0 sumands is 0. The third condition is that $r^B_{|\sigma|} < 2^{-e}$.

At the step σ for $\tilde{G}$, one searches for the first p of length up to $|\sigma|$ such that p and the prefix η of $B \oplus E^B_{|\sigma|}$ of length $2|\sigma|$ satisfy

- $\tilde{U}(p, \eta)$ is defined and takes a value u;
- $(p, u, \tau) \notin \tilde{G}$ for any proper prefix τ of σ.

If p, u are found then (p, u, σ) goes into $\tilde{G}$ else no triple with last component σ goes into $\tilde{G}$. This completes the step to define which tuples with last component σ go into $\tilde{E}$ and $\tilde{G}$.

For the verification of the construction, let E^A be the set of all x such that $(x, \sigma) \in \tilde{E}$ for a prefix σ of A and G^A be the set of all (p, y) such that

$(p, y, \tau) \in \tilde{G}$ for some prefix τ of A. Furthermore let s_u be the sum over all $2^{-|p|}$ where $(p, u) \in G^A$ and s be the sum over all s_u. It holds that $s \leq 3$ by the following reason: The sum over all $2^{-|p|}$ such that $U^{A \oplus E^A}(p) = u$ is bounded by 1. So consider now any $(p, u) \in G^A$ such that $U^{A \oplus E^A}(p)$ is either undefined or different from u. Let τ be the prefix of A with $(p, u, \tau) \in \tilde{G}$. Then there must be some prefix σ of A longer than τ where an element bounded by $|\tau|$ goes into E^A. Then the corresponding e of this stage satisfies that $r^B_{|\sigma|} < 2^{-e}$ and $2^{-|p|}$ contributes to the sum $r^B_{|\sigma|}$. Since each e is dealt with in only one step, the sum of $2^{-|p|}$ going over all $(p, u) \in G^A$ with $U^{A \oplus E^A}(p) \neq u$ is bounded by $2^{-0} + 2^{-1} + \ldots = 2$. It follows that s is bounded by 3.

The set E^A is coinfinite by the first condition. Now consider an infinite set W_e^A and denote by q_x the sum of all $2^{-|p|}$ where there is a $(p, y, \sigma) \in \tilde{G}$ with $x \leq |\sigma|$. Since $s \leq 3$, these sums q_x go to 0. Thus there is an $x \in W_e^A$ with $x \geq 2e$ and $q_x < 2^{-e}$. In the case that E^A and W_e^A are disjoint, all three conditions would be satisfied for e, x and almost all prefixes σ of A. This would give the contradiction that x goes into E^A eventually. Thus the set E^A is simple relative to A.

Recall that the set G^A is recursively enumerable relative to A and that the sum s over $2^{-|p|}$ for all $(p, u) \in G^A$ is bounded by 3. Then the Kraft-Chaitin Theorem [10] says that there is a prefix-free partial A-recursive function V^A and a constant c such that for every u there is a program q with $V^A(q) = u$ and $s_u \leq 2^c \cdot 2^{-|q|}$. In particular, for every program p where $U^{A \oplus E^A}(p)$ is defined, there is a q such that $V^A(q) = U^{A \oplus E^A}(p)$ and $2^{-|p|} \leq 2^c \cdot 2^{-|q|}$. It follows that $|q| \leq |p| + c$. Therefore, V^A is one possible choice for a universal machine for A and witnesses that all u satisfy $H^A(u) \leq H^{A \oplus E^A}(u) + c$. So $A \oplus E^A$ is low for prefix-free Kolmogorov complexity relative to A. $\square$

Theorem 3.7 *There is an r.e., Turing-incomplete set A such that for any recursive, monotone nondecreasing and unbounded function k with $k(0) = 1$ the Kolmogorov function is $k(n)$-enumerable relative to A.*

Proof: Let E be the enumeration operator from Theorem 3.6. Jockusch and Shore [14] showed that for any enumeration operator satisfying $(\forall B)\, [W^B \geq_T B]$ there is an r.e. set A such that W^A has the Turing degree of the halting problem. This holds of course for the operator $W^A = A \oplus E^A$ from Theorem 3.6 and so one can pick an r.e. set A with $W^A \equiv_T K$. Let U^A be the universal function for H^A relative to A. Note that $A <_T W^A$ since E^A is simple relative to A.

Given the recursive function k, one defines an A-recursive $k(n)$-enumerator $\tilde{f}$ which does for input x of length n the following:

> $\tilde{f}(x)$ enumerates the component v_x of each vector v such that v is so long that the component v_x exists and $v = U^A(p)$ for a program p with $|p| \leq \log(k(n))$.

Note that due to U^A being prefix-free, $\tilde{f}$ is already a $k(n)$-enumerator. Now one constructs from $\tilde{f}$ an $k(n)$-enumerator *for C* by

$$f_t^x = \begin{cases} C(x) & \text{if } t = 1 \text{ and } C(x) \notin \{\tilde{f}_1^x, \ldots, \tilde{f}_{b_x}^x\}; \\ \tilde{f}_t^x & \text{otherwise.} \end{cases}$$

where f_t^x is undefined whenever $t \neq 1$ and $\tilde{f}_t^x$ is undefined.

It remains to show that f is also an A-recursive enumerator for C. This is done by showing that $f, \tilde{f}$ are finite variants. The proof of this uses the following property of the universal K-recursive machine U^K as a tool: For every number r there is a program p_r such that $U^K(p_r)$ computes the vector

$$(C(\lambda), C(0), C(1), C(00), C(01), \ldots, C(1^m))$$

where m is the first length such that $k(m) \geq 2^{r+1}$. Without loss of generality, p_r is the shortest program for this vector. Note that m can be computed from r since k is recursive. Thus the length of p_r is logarithmic in the parameter r.

Now let $r(x) = \log(k(|x|))$ for any x; more precisely, let $n = |x|$ and $r(x)$ be the unique integer with $2^{r(x)} \leq k(n) < 2^{r(x)+1}$. Then $m > n$ for the m computed from $r(x)$ above. So $U^K(p_{r(x)})$ outputs a vector v such that v_x exists and is equal to $C(x)$.

Due to the choice of A, there is a program $q_{r(x)}$ with $U^A(q_{r(x)}) = U^K(p_{r(x)})$ which is only by a constant longer than $p_{r(x)}$. So, for almost all x, $|q_{r(x)}| \leq \log(k(|x|))$ and $U^A(q_{r(x)})$ is taken into account by $\tilde{f}(x)$. Therefore, for almost all x, the function $\tilde{f}(x)$ enumerates $C(x)$ relative to A. The enumerators $\tilde{f}$ and f are finite variants and f is an A-recursive $k(n)$-enumerator for C. $\square$

We now use Theorem 3.7 to extend our result to strong enumerations as follows.

Theorem 3.8 *Let k be a strictly positive, monotone nondecreasing recursive function. Then there exists a set B not above K such that the Kolmogorov function has a strong B-recursive k-enumerator.*

Proof: Take the set A from Theorem 3.7 and consider the partial A-recursive function $i, x \to f_i^x$ equal to the ith element enumerated by the enumeration algorithm in the proof of Theorem 3.7. Recall that $A \not\geq_T K$. Using the fact that $i, x \to f_i^x$ is recursively bounded and Remark 3.3, there is an extension $\overline{f}$ of f such that its domain is $\{(i, x) : 1 \leq i \leq k(|x|)\}$ and its graph $B = \{(i, x, y) : y = \overline{f}_i^x \wedge 1 \leq i \leq k(|x|)\}$ satisfies $A \oplus B \not\geq_T K$. In particular, $\overline{f}$ is a strong B-recursive $k(n)$-enumerator for C. $\square$

4 Enumerators as Oracles

We saw in the previous section that for constant k, a k-enumerator for the Kolmogorov function cannot be computed without access to an oracle that is already as hard as K. That construction was not uniform in the given enumerator. Therefore we ask in this section which information can be retrieved uniformly from a given strong 2-enumerator for C. The following result shows that, under certain assumptions about the choice of the universal Turing machine used in defining Kolmogorov complexity, one can extend any $\{0,1\}$-valued partial-recursive function uniformly using any strong 2-enumerator for the Kolmogorov function as an oracle.

Theorem 4.1 *Let ψ be any given partial-recursive $\{0,1\}$-valued function. One can choose the universal machine U on which the Kolmogorov complexity is based in such a way that there is a fixed program e such that, given any strong 2-enumerator f for the Kolmogorov function, φ_e^f computes a total extension of ψ with only one query to f.*

Proof: One chooses U such that $C(x) \equiv \psi(x)$ modulo 3 whenever $\psi(x)$ is defined and $C(x) \equiv 2$ modulo 3 otherwise. This is obtained by starting with an arbitrary universal machine $\tilde{U}$ and defining that $U(p10^{l+l'}) = \tilde{U}(p)$ if $\tilde{U}(p)$ is defined, $|p|+1+l \equiv 0$ modulo 3 and either $l' = 2$ or $l' = \psi(\tilde{U}(p))$. For those q where $U(q)$ cannot be defined by this method, $U(q)$ remains undefined.

Now define the program e taking the first case to apply from the below case distinction where f_1^x, f_2^x are the two values given by any strong 2-enumerator f for C queried exactly at x:

$$\varphi_e^f(x) = \begin{cases} \psi(x) & \text{if } f_1^x \not\equiv 2 \wedge f_2^x \not\equiv 2 \text{ modulo 3;} \\ 0 & \text{if } f_1^x \not\equiv 1 \wedge f_2^x \not\equiv 1 \text{ modulo 3;} \\ 1 & \text{if } f_1^x \not\equiv 0 \wedge f_2^x \not\equiv 0 \text{ modulo 3.} \end{cases}$$

Since there are only two values f_1^x, f_2^x, it is clear that at least one of these conditions holds. Furthermore, if both, f_1^x and f_2^x, are different from 2 modulo 3, then $\psi(x)$ is defined. Thus, φ_e^f is total and $\{0,1\}$-valued. If $\psi(x) \downarrow= b$ then one of f_1^x and f_2^x must be b, so the case for $\varphi_e^f(x) = 1 - b$ does not apply and $\varphi_e^f(x)$ is correct by either taking the case $\varphi_e^f(x) = \psi(x)$ or the case $\varphi_e^f(x) = b$. So, the program e works with every strong 2-enumerator supplied as oracle f to e. □

Note that the above construction goes also through if you take any strong 2-emumerator for $C(x)$ modulo 3 as an oracle. This is no longer true for 2-enumerators itself, since there is a recursive 2-enumerator f with this property: $f_1^x = 2$ and $f_2^x = \psi(x)$ whenever that is defined. Then one has that there is for every x a t with f_t^x being defined and equivalent to $C(x)$ modulo 3. So this construction needs strong 2-enumerators f for which f_1^x and f_2^x are always both defined.

Since C is as hard as the halting problem, it is natural to ask whether Theorem 4.1 also holds for computing a fixed set, for example K, instead of just finding an arbitrary extension depending on the queried enumerator. Theorem 4.2 answers this question negatively. We extend the negative result in two directions: For every fixed Turing reduction e, every nonrecursive set A and every recursively bounded function g there is a strong 2-enumerator for g such that the reduction e does not compute A relative to the given enumerator. Furthermore, in case of weak-truth-table reducibility, we can consider all reductions instead of a fixed one. Corollary 4.5 shows that, given A and g as above, there is a strong 2-enumerator for g which is not wtt-hard for A.

Theorem 4.2 *Assume that A can be computed by a fixed reduction making one query relative to any strong 2-enumerator of C. Then A is recursive.*

Proof: Assume that the hypothesis of the theorem holds. That is, there is a program e and a recursive functions h such that for every strong 2-enumerator f of C,

$$\forall x \; A(x) = \varphi_e^{\{f_1^{h(x)}, f_2^{h(x)}\}}(x).$$

Now it is shown that A is recursive.

Let c be a constant with $C(z) \leq |z| + c$ for all z. Let y be the query generated by e on input x. Let $\varphi_e^f(x)$ denote the outcome of the computation e on input x using oracle f and let $\varphi_e^{\{n_1,n_2\}}(x)$ denote the outcome of the program e run on input x, but with the numbers n_1, n_2 substituted for the answer to the query y. We determine whether $x \in A$ as follows: Compute $\varphi_e^{\{n_1,n_2\}}(x)$ for all possible $n_1, n_2 \in \{0, 1, \ldots, |y| + c\}$. Clearly if there are an n_1 and an $i \in \{0, 1\}$ such that $(\forall n_2 \in \{0, 1, \ldots, |y| + c\})\, [\varphi_e^{\{n_1,n_2\}}(x) = i]$, then also $\varphi_e^f(x) = i$ since $C(y)$ is one of these values. It remains to argue that such an n_1 exists. However $n_1 = C(y)$ meets this condition. $\square$

This theorem easily generalizes in two directions: to more general reductions and to more general functions.

Corollary 4.3 *Let g be any recursively bounded function. Suppose that there is a fixed reduction φ_e that weak-truth-table reduces a set A to all possible strong 2-enumerators for g. Then A is recursive.*

Proof: Let $g'(x) = \langle g(q_1), \ldots, g(q_m) \rangle$, where $q_1, \ldots, q_m$ are the queries made by the reduction φ_e on input x. The reduction φ_e must compute

$A(x)$ correctly if, for each q_i, it is given a pair of answers (a_i, b_i) such that $g(q_i) \in \{a_i, b_i\}$. *A fortiori*, φ_e must compute $A(x)$ correctly if it is given a pair of sequences $(\langle a_1, \ldots, a_m\rangle, \langle b_1, \ldots, b_m\rangle)$ such that $g'(x) \in \{\langle a_1, \ldots, a_m\rangle, \langle b_1, \ldots, b_m\rangle\}$. It follows that there is a fixed reduction $\varphi_{e'}$ from A to all possible strong 2-enumerators for g'. Since g' is recursively bounded, A is recursive by the same argument as in the proof of Theorem 4.2. $\Box$

The condition that g is recursively bounded is necessary. Recall the non-recursive convergence-modulus c_K of K from the proof of Theorem 3.4: c_K can be computed with one query relative to any strong $k(n)$-enumerator f for c_K. Querying f at input x, one receives a set $f(x)$ containing $c_K(x)$ and knows that $c_K(x) = s$ for the maximal $s \in f(x)$ such that $K_s \cap \{0, 1, \ldots, x\} \neq K_{s-1} \cap \{0, 1, \ldots, x\}$. Furthermore, the halting problem K itself is computable relative to any enumerator for c_K.

With a bit more care, the proof of Theorem 4.2 even extends to Turing reductions.

Theorem 4.4 *Let g be any recursively bounded function. Suppose that there is a fixed reduction φ_e that Turing reduces a set A to all possible strong 2-enumerators for g. Then A is recursive.*

Proof: Let $b(x)$ be a recursive function such that $0 \leq g(x) \leq b(x)$ for all x.

A full query tree of e on input x of the following form: At each internal node we have labelled the query q and a possible answer $y_q \leq b(q)$. There is a branch for each $z \leq b(q)$ representing (y_q, z) as the strong 2-enumeration for $g(q)$. A full query tree has finite size, every leaf has the computation halting and the answers at all leaves agree.

First Claim: A full query tree for x exists.

Simply consider the tree with $y_q = g(q)$ at every internal node. All leaves must give the same (correct) answer. If the tree is not finite, by König's lemma it must have an infinite path which defines a strong 2-enumerator that e fails to reduce to.

Second Claim: Any full query tree gives the correct answer on all leaves.

Consider a path such that either $y_q = g(q)$ or $z = g(q)$ for all queries q on that path. Since φ_e reduces A to all strong 2-enumerators for g, this leaf must give the correct answer. Since all leaves give the same answer, all leaves give the correct answer.

The recursive algorithm for A just searches for a full query tree and outputs the answer that all leaves agree to. $\Box$

Corollary 4.5 *For any nonrecursive set A and any recursively bounded function g there exists a strong 2-enumerator f of g such that $A \not\leq_{wtt} f$.*

Proof: Use a finite extension argument on Theorem 4.4. Start with σ_0 having the domain $\emptyset$. For every i there is an extension f_i of σ_i and an x_i such that f_i is a strong 2-enumerator for g and the i-th wtt-reduction φ_{e_i} fails to compute $A(x_i)$ from f_i. Since the reduction queries f_i at only finitely many places, one can take an upper bound u_i on the length of σ_i and the queried places. Let σ_{i+1} be the restriction of f_i to the domain $\{0, 1, \ldots, u_i\}$. σ_{i+1} is a strong 2-enumerator for g on this domain and the wtt-reductions $\varphi_{e_0}, \varphi_{e_1}, \ldots, \varphi_{e_i}$ do not wtt-reduce A to any extension of σ_{i+1}. Repeating this argument inductively, one obtains that the limit f of all σ_i is a strong 2-enumerator for g to which A is not wtt-reducible. $\square$

5 Prefix-Free Kolmogorov Complexity

In this section it is shown that the results for C also hold for the prefix-free complexity H: H is based on a unary numbering U such that for all distinct programs p, p' in the domain of U it holds that neither p is a prefix of p' nor p' a prefix of p. Furthermore, U has to be universal among all these numberings.

With minor modifications, the proof of Theorem 3.1 works also for prefix-free Kolmogorov complexity. The corresponding result is then the following.

Theorem 5.1 *There is a constant a depending only on the universal machine U defining the prefix-free Kolmogorov complexity H such that every $k(n)$-enumerator f satisfies $k(n) \geq n/a$ for almost all n.*

Furthermore, one can also transfer the hardness-result Theorem 3.4 to prefix-free Kolmogorov complexity to H. Here of course one defines the $X_{n,i}$ with respect to approximations to H instead of approximations to C. The most important ingredient for transferring the proof is that one can build a prefix-free machine which codes every $x \in X_{m,i}$ with $3\log(m)+4+(m+1)(m+1-i)$ many input bits by coding first in $3 + 2\log(m)$ bit the number m in a prefix free way, than using $\log(m)+1$ bits to code i and code with $(m+1)(m+1-i)$ bits how many numbers go into $X_{m,i}$ before x. Thus the upper bound on $C(F(m))$ is actually an upper bound on $H(F(m))$. Furthermore, the lower bound on $C_{s_i}(F(m))$ from the proof of Theorem 3.4 can directly be taken as a lower bound of $H_{s_i}(F(m))$ in this proof. The rest of the proof transfers directly. Thus one has the following result.

Theorem 5.2 *Let k be a constant. If the prefix-free Kolmogorov function H is k-enumerable relative to a set A then $A \geq_T K$.*

The proof of Theorem 3.7 does not use any property of C besides the fact that C is a total K-recursive function. This clearly also holds for H and thus the result transfers immediately.

Theorem 5.3 *There is an r.e., Turing-incomplete set A such that for any recursive, monotone nondecreasing and unbounded function k with $k(0) = 1$ the prefix-free Kolmogorov function is $k(n)$-enumerable relative to A.*

Note that the proof does not use any other property of C and H as that they are approximable from above. Thus the result holds for all functions which are approximable from above.

The choice of the underlying universal function U in Theorem 4.1 works also for a universal function defining the prefix-free Kolmogorov complexity.

Theorem 5.4 *Let ψ be any given partial-recursive $\{0,1\}$-valued function. One can choose the universal machine U on which prefix-free Kolmogorov complexity is based in such a way that there is a program e which computes a total extension φ_e^f of ψ with one query to any strong 2-enumerator f for the prefix-free Kolmogorov function.*

The further results of Section 4 state that the following holds for every given function g.

- No non-recursive set is Turing reducible to all strong 2-enumerators for g by the same reduction;
- No non-recursive set is wtt-reducible to all strong 2-enumerators of g.

For these results it does of course not matter whether g is C, is H or is something else.

6 Ressource-Bounded 2-Enumerators

We will be able to characterize the class S_2^p which was introduced by Russel and Sundaram [24] and the class PSPACE in terms of reductions to strong 2-enumerators for some functions. Note that $\mathrm{NP} \subseteq \mathrm{S}_2^\mathrm{p} \subseteq \Sigma_2^\mathrm{p} \cap \Pi_2^\mathrm{p}$.

Definition 6.1 *A set A is in S_2^p if and only if there exist a polynomial p and a polynomial time computable ternary predicate Q such that*

- $x \in A$ *if* $\exists v \in \{0,1\}^{p(|x|)}$ *such that* $(\forall w \in \{0,1\}^{p(|x|)})\,[Q(x,v,w)]$;
- $x \notin A$ *if* $\exists w \in \{0,1\}^{p(|x|)}$ *such that* $(\forall v \in \{0,1\}^{p(|x|)})\,[\neg Q(x,v,w)]$.

Note that for general languages in $\Sigma_2^{\rm p} \cap \Pi_2^{\rm p}$ the second occurrence of Q could be replaced by an arbitrary polynomial time predicate Q'. It is not known whether $\mathrm{S}_2^{\rm p} = \Sigma_2^{\rm p} \cap \Pi_2^{\rm p}$. The class $\mathrm{S}_2^{\rm p}$ can be characterized in terms of reductions to strong 2-enumerators for a bounded function g where g is bounded iff there is a polynomial p with $|g(x)| \leq |p(|x|)|$ for all x.

Theorem 6.2 *The following statements are equivalent for any set A.*

(a) *$A \in \mathrm{S}_2^{\rm p}$;*

(b) *There are a fixed btt(1)-reduction M and a polynomially bounded function g such that M computes A relative to any strong 2-enumerator of g;*

(c) *There is a fixed tt-reduction N and a $\{0,1,2\}$-valued function h such that N tt-reduces A to any strong 2-enumerator of h.*

(d) *There is a fixed tt-reduction N' and a polynomially bounded function h' such that N' tt-reduces A to any strong 2-enumerator of h'.*

Proof: (a) $\Rightarrow$ (b): Given A, let p, Q as in Definition 6.1. Furthermore, let g be a function such that

- if $x \in A$ then $g(x) = (v, 1)$ where v is the leftmost witness in $\{0,1\}^{p(|x|)}$ for $x \in A$;
- if $x \notin A$ then $g(x) = (w, 0)$ where w is the leftmost witness in $\{0,1\}^{p(|x|)}$ for $x \notin A$.

A strong 2-enumerator for g produces for input x two candidates (u, a) and (u', a'). If $a = a'$, then one knows that $A(x) = a$. If $a = 0$ and $a' = 1$ then $A(x) = Q(x, u', u)$. If $a = 1$ and $a' = 0$ then $A(x) = Q(x, u, u')$.

(b) $\Rightarrow$ (c): Without loss of generality, one can assume that M on input x computes a position $q(x)$ such that $g(q(x))$ is a number between 0 and $2^{|x|^c}$ for some constant c. The idea is to define a function h and a reducibility N from A to strong 2-enumerators of h which can simulate the reduction M from A to any strong 2-enumerator for g.

In order to simulate M, one considers for given input x the place $q(x)$ and codes $q(x)$ at polynomially many places into h such that one can compute two possible values for $g(q(x))$ with one being correct from a tt-reduction to any strong 2-enumerator for h. For input x, i, j, a, b, consider the following two statements:

- The ith bit of $g(q(x))$ is a;

- The jth bit of $g(q(x))$ is b.

Now let $h(x, i, j, a, b)$ be the number of those statements, which are correct. The function h is $\{0, 1, 2\}$-valued. Furthermore, for fixed x, only the $i, j \in \{0, 1, \ldots, c \cdot \log(|x|)\}$ and $a, b \in \{0, 1\}$ are relevant, so one has to query a given strong 2-enumerator of h only at polynomially many places and these queries an be done in parallel.

There are no three different numbers y_1, y_2, y_3 which are consistent with all answers to h: There is a position i such that the ith digit of one number, say y_3, is a while the ith digit of the other two numbers y_1, y_2 differ from a. Furthermore, there is a position j where the digits of y_1, y_2 differ. Say, y_2 and y_3 have the same jth digit b and y_1 not. It follows that

$$h(x, i, j, a, b) = \begin{cases} 0 & \text{if } g(x) = y_1; \\ 1 & \text{if } g(x) = y_2; \\ 2 & \text{if } g(x) = y_3. \end{cases}$$

So at most 2 numbers are consistent with all the outputs of the strong 2-enumerator for h on those inputs x, i, j, a, b which satisfy $i, j \in \{0, 1, \ldots, c \cdot \log(|x|)\}$ and $a, b \in \{0, 1\}$.

One can determine these two numbers modulo 2^m by just considering the last m binary digits. Thus one can construct two candidates for $g(q(x))$ step by step with the search space always having only at most 2 candidates modulo 2^m before m is incremented; after m is incremented and before the conditions on the new digit are taken into account, the number of candidates is at most 4. So the search-space to construct the two possible vectors for $g(q(x))$ contains in every step only up to 4 candidates and the search is performed in polynomial time. Thus one can turn the btt(1)-reduction M from A to strong 2-enumerators for g into a tt-reduction N from A to strong 2-enumerators for h.

(c) $\Rightarrow$ (d) holds by definition since every $\{0, 1, 2\}$-valued function is polynomially bounded.

(d) $\Rightarrow$ (a): Let N' compute the tt-reduction from A to strong 2-enumerators for h'. Without loss of generality there is a polynomial p_1 such that N' queries the strong 2-enumerator at $p_1(|x|)$ many places at input x and the length of each of the places is bounded by $p_1(|x|)$. Furthermore, there is a polynomial p_2 bounding the length of h'; $|h'(u)| \leq p_2(|u|)$ for all u. Without loss of generality every considered strong 2-enumerator f for h outputs on input u a pair (f_1^u, f_2^u) such that both strings have at most the length $p_2(|u|)$. Now one defines the predicate Q such that $Q(x, v, w)$ is the output of N querying f if v, w are lists of strings such that v contains the values f_1^u and w contains the values f_2^u where $u = u_1, u_2, \ldots, u_{p_1(|x|)}$ runs over the places queried by

N'. Note that the length of v, w are bounded by $2 \cdot p_1(|x|) \cdot p_2(p_1(|x|))$. The predicate Q has the following properties:

- $Q(x, v, w)$ is true if $x \in A$ and $v = h(u_1)h(u_2)\dots h(u_{p(x)})$;
- $Q(x, v, w)$ is false if $x \notin A$ and $w = h(u_1)h(u_2)\dots h(u_{p(x)})$.

These properties witness that $A \in \mathrm{S}_2^{\mathrm{p}}$. $\square$

Definition 6.3 *Let U be a fixed universal space-bounded machine with two inputs p, w which respects the space bound $2(|p| + |w|)$. In the following, $C_{space}(x|w)$ denotes the size of the smallest program p such that $U(p, w) = x$. $C_{space}(x|w)$ is called the space-bounded conditional Kolmogorov complexity.*

Remark 6.4 *Recall that QBF is the set of all true formulas of the form*

$$\exists a_1 \forall b_1 \exists a_2 \forall b_2 \dots \exists a_n \forall b_n \phi(a_1, b_1, a_2, b_2, \dots, a_n, b_n)$$

where $a_1, b_1, a_2, b_2, \dots, a_n, b_n$ are Boolean variables and ϕ is variable-free. The parameter n is not a constant. The set QBF is PSPACE-complete. Formulas of this type are called QBF-instances or just instances.

An important property of QBF is self-reducibility. Given a QBF-instance as above, one can write it as

$$\exists a_1 \forall b_1 \psi(a_1, b_1)$$

where the other quantifiers and their variables are moved into the formula ψ. Then QBF is self-reducible by the following formula:

$$\exists a_1 \forall b_1 \psi(a_1, b_1) \Leftrightarrow (\psi(0,0) \wedge \psi(0,1)) \vee (\psi(1,0) \wedge \psi(1,1)).$$

So every instance with $2n$ variables, $n > 0$, can be reduced to four smaller instances with $2n - 2$ variables.

Proposition 6.5 *For every given constants c, k there is a constant ℓ such that one can find with queries to a strong k-enumerator f at the places $f(v|\ell, w)$ for all $v \in \{0,1\}^\ell$ a vector $u \in \{0,1\}^\ell$ with $C_{space}(u|\ell, w) > c$. The time and space complexity to find u is independent of w except at the place where the queries to $f(v|\ell, w)$ occur and w has to be copied onto the oracle tape to query f.*

Proof: In the following, let $f(v) = \{f_1^v, \dots, f_m^v\}$ be the output of f on input w and let $f_1^v < \dots < f_k^v$.

Call a set I of strings of the same length an interval iff the binary values of the strings in I form an interval in the natural numbers. Let c' be a constant such that $c' \geq 2$ and for every interval I and every $u, u' \in I$ and all w hold that $C_{space}(u'|\ell, w) \leq C_{space}(u|\ell, w) + c' + \log(\|I\|)$ where $\|I\|$ is the cardinality of I.

Now let $c_1 = c$ and inductively $c_{i+1} = 2c_i + c'$ for $i = 1, 2, \ldots, k$. Let $\ell = c_{k+1}$ and let $L_i = \{v \in \{0,1\}^\ell : f_i^v \leq c_i\}$ for $i = 1, 2, \ldots, k$.

On the one hand, there are less than 2^{c_k+1} strings with conditional complexity c_k for the given w; since f_k^v is an upper bound for this complexity the cardinality of L_k is less than 2^{c_k+1}. On the other hand, there are 4^{c_k+2} strings in $\{0,1\}^\ell$. So there is an interval of length 2^{c_k} not containing any element of L_k.

Thus there is a smallest i such that an interval $I \subseteq \{0,1\}^\ell$ of length 2^{c_i} does not contain any string from L_i. Fix this I.

If $i = 1$, one can just pick and output any $u \in I$ since $C_{space}(u|\ell, w) \geq f_1^u > c_1 \geq c$.

If $i > 1$ then there are $2^{c_{i-1}+1}$ disjoint subintervals J of length $2^{c_{i-1}}$ each of them containing an element of L_{i-1}. If there is a $u \in I \cap L_{i-1}$ satisfying $C_{space}(u|\ell, w) \leq f_{i-1}^u$, then $C_{space}(v|\ell, w) \leq f_{i-1}^v + c' \leq c_{i-1} + c'$ for all $v \in I \cap L_{i-1}$. But all these v satisfy that $f_{i-1}^v \leq c_{i-1}$ and $f_i^v > c_i \geq c_{i-1} + c'$ and thus it would hold that $C_{space}(v|\ell, w) \leq c_{i-1}$ in contradiction to $\|L_{i-1} \cap I\| \geq 2^{c_{i-1}+1}$. Thus $C_{space}(u|\ell, w) \geq c_i > c$ for every $u \in I \cap L_{i-1}$ and one can pick and output any such u. □

Theorem 6.6 *For every constant k, there is a fixed polynomial time Turing reduction M such that $QBF = M^f$ for all strong k-enumerators f of the space-bounded conditional Kolmogorov function C_{space}.*

Proof: In the following, x is the input to M and represents a QBF-instance.

At the beginning let $m = 1$ and $y_1 = x$. Initialize the set V of possible characteristic functions $(QBF(x), QBF(y_1))$ as $\{(0,0),(1,1)\}$. Now M runs the following loop until the algorithm halts.

- For given m and instances $y_1, \ldots, y_m$, M chooses $4m$ instances $z_1, \ldots, z_{4m}$ by replacing in each formula the first two Boolean variables by 0 and 1, respectively.
- Let $g(x, y_1, \ldots, y_m, z_1, \ldots, z_{4m})$ be the $5m+1$-fold characteristic function

  $$(QBF(x), QBF(y_1), \ldots, QBF(y_m), QBF(z_1), \ldots, QBF(z_{4m}))$$

  and let $r = (x, y_1, \ldots, y_m, z_1, \ldots, z_{4m})$.
- Let $g'(\ell', r, u_1, \ldots, u_{j'})$ with $j' = 2^{\ell'} - 1$ be that string in $\{0,1\}^{\ell'}$ which represents an i such that

– i is the first number with $u_i = g(w)$ if $g(r) \in \{u_1, \ldots, u_{j'}\}$;

– $i = 0$ if $g(w) \notin \{u_1, \ldots, u_{j'}\}$.

Now choose c such that

$$\begin{array}{rcll} C_{space}(g'(\ell', r, u_1, \ldots, u_{j'})|r, \ell', u_1, \ldots, u_{j'}) & \leq & c & \text{and} \\ C_{space}(0^{\ell'}|\ell', r, u_1, \ldots, u_{j'}) & \leq & c & \end{array}$$

for all ℓ' and j' with $j' = 2^{\ell'}$. This constant c exists since g, g' are computable in linear space. Let ℓ depend on c, k as in Proposition 6.5.

- Let $L = \{0,1\}^{5m+1}$ and $j = 2^\ell - 1$.
- While $|L| \geq j$, use the oracle f to find an i such that $u_i \neq g(w)$ where $u_1, \ldots, u_j$ are the first j members of L and remove u_i from L. This i can be found by searching for a number where the binary representation $b_1 \ldots b_{m'}$ satisfies

$$C_{space}(b_1 \ldots b_{m'}|r, m', u_1, \ldots, u_j) > c.$$

  Such an index can be found by Proposition 6.5.
- Now let W be the set of all strings $w \in \{0,1\}^{5m+1}$ such that
  – w has never been removed from L;
  – w is an extension of an $v \in V$;
  – w is consistent with the self-reduction from QBF on $y_1, \ldots, y_m$ to QBF on $z_1, \ldots, z_{4m}$;
  – w is consistent with $QBF(z_l)$ whenever z_l does not contain any variables.
- If all $w \in W$ give the same value for $QBF(x)$ then output this value and halt.
- For each different $w, w' \in W$ select one $z_{d(w,w')}$ such that w, w' contain different entries for $QBF(z_{d(w,w')})$. Replace $y_1, \ldots, y_m$ by a new sequence of instances $y'_1, \ldots, y'_{m'}$ being these selected $z_{d(w,w')}$.
- Repeat the loop with the instances $x, y'_1, \ldots, y'_{m'}$ and V being the restriction of the vectors in W to these instances $x, y'_1, \ldots, y'_{m'}$.

Note that the constants c, ℓ, j are the same in every round of the algorithm, they are just introduced where needed for the first time, so that it is easier to understand how they are defined. In every step, one selects at most j^2 many

$z_{d(w,w')}$ and thus m', m are both bounded by 4^ℓ. Due to this constant bound, the operations above can in each single step be carried out in polynomial time with polynomially many queries to f.

Furthermore, it is easy to verify by induction, that whenever the algorithm halts, then the output is $QBF(x)$: the reason is that it is sufficient to show that the vector $v = (QBF(x), QBF(y_1), \ldots, QBF(y_m))$ is in V whenever the new loop is started. This is true for the initialization. Assume that same now as induction hypothesis at the beginning of the loop. Then the vector $w = (QBF(x), QBF(y_1), \ldots, QBF(y_m), QBF(z_1), \ldots, QBF(z_{4m}))$ is in the list L and never removed from L because $C_{space}(w|r, u_1, \ldots, u_j) \leq c$ whenever $w \in \{u_1, \ldots, u_j\}$. Since $v \in V$ and w extends v, the vector w is also in W. Its projection $(QBF(x), QBF(y'_1), \ldots, QBF(y'_{m'}))$ goes into the set V for the next iteration of the loop.

So it remains to show that algorithm halts. This is done by showing the following invariant: every $v' \in V$ different from v differs from v on some instances $y_1, \ldots, y_m$. So consider now any $w' \in W$ which assigns to x a value different from $QBF(x)$. w' extends some $v' \in V$. This v' is different from v on some $y_1, \ldots, y_m$ as well. Since the values of w' on $y_1, \ldots, y_m$ are self-reduced to those on $z_1, \ldots, z_{4m}$, there is a $z_{d(w,w')}$ such that w and w' differ on $z_{d(w,w')}$. By the consistency condition it follows that this can only occur while the $z_{d(w,w')}$ still has some variable. This $z_{d(w,w')}$ or some other is selected into the new vector for the next round and therefore also the new V satisfies the invariant given. Since all variables are eventually eliminated from all instances (except x), the remaining vectors in W after that step assign $QBF(x)$ to x and so the algorithm halts. $\square$

Theorem 6.7 *Let A be any set. Then the following statements are equivalent for A.*

(a) *A is in* PSPACE*;*

(b) *A is Turing reducible by a fixed reduction to every strong* 2*-enumerator for the conditional space-bounded Kolmogorov function;*

(c) *There is a polynomially bounded function g such that A is Turing reducible by a fixed reduction to any strong* 2*-enumerator of g.*

Proof: (a) $\Rightarrow$ (b) by Theorem 6.6 and the fact that QBF is PSPACE-complete. (b) $\Rightarrow$ (c) is trivial. For (c) $\Rightarrow$ (a) consider a set A and a Turing reduction M which computes $A(x)$ relative to any strong 2-enumerator f for g. Now consider the following game with two players Anke and Boris associated to the reduction at an instance x.

On input x, the computation of M relative to f is simulated. Whenever M asks for the values f_1^y, f_2^y, Anke and Boris each supply one of them. Anke wins the game iff M halts with output 0 and Boris wins the game iff M halts with output 1. If $x \notin A$ then Anke has a winning strategy by always supplying $g(y)$: the oracle-answers to every query y contain two values with $g(y)$ being among them and thus the reduction M behaves as if these answers are from a strong 2-enumerator f for g. So M returns the value $A(x)$ which is 0 in this case. If $x \in A$ then Boris has the same winning strategy by also always supplying $g(y)$, this case is symmetric to the previous one and M returns at the end the value $A(x)$ which now is 1.

So one has for every x a game uniform in x which is played polynomially many rounds and each round is played in polynomial time. It is well-known that the problem of which player has a winning strategy for such a game is in PSPACE. Thus A is in PSPACE. □

Recall that the class EXP contains all sets which have a decision procedure using time $2^{p(n)}$ for some polynomial p where n is the size of the input. This class can also be characterized as the class of sets A computable in alternating polynomial space [23, Section 20.1]. Such a characterization is equivalent to saying that there is a game for A with the following properties:

- $x \in A$ iff Anke has a wining strategy for the game starting with a configuration $c(x)$;
- Every game terminates and either Anke or Boris wins (no tie exists);
- Every configuration in the game starting with $c(x)$ is coded in space $p(|x|)$ where p is a polynomial;
- For any two configurations (y, z) it can be decided in polynomial space which of the players Anke or Boris has the right to move at y and whether this player can move to z.

One can adapt Proposition 6.5 and Theorem 6.6 to the exponential-time computable conditional Kolmogorov function and instances of the EXP-complete set SCV, using the following concepts and intermediate results.

- SCV is the succinct circuit value problem considered by Papadimitriou [23, Section 20.1]. SCV is the set of all succinctly coded circuits which are evaluated to 1. Formally, a SCV-instance is a quintuple $x = (e_1, e_2, 0^a, 0^b, 0^s)$ satisfying the following:
  - The instance x represents an exponentially sized circuit with 2^a gates and 2^b input-bits.
  - The program e_1 computes for any gate at position $u \in \{0,1\}^a$ two positions $v, w \in \{0,1\}^a$ lexicographically before u, an input-position $z \in \{0,1\}^b$ and a formula ϕ such that the value of the

gate at u is $\phi(v', w', z')$ where v', w' are the values of the gates at v, w and z' is the value of the zth input-bit.

  - The program e_2 computes z' from z for every $z \in \{0,1\}^b$.
  - The programs e_1, e_2 keep on all legal inputs the space-bound s.

  SCV is now the set of all SCV-instances where the gate with position $11\ldots1$ is evaluated to 1.

- Let g compute for any SCV-instance, for any m and for any m positions in the circuit a vector in $\{0,1\}^m$ which contains the values of these gates of the circuit in this instance. There is a polynomial p such that the function g can be computed in time $2^{p(n+m)}$ where n is the size of the instance and m the number of the gates given in the input.

- With queries to any strong 2-enumerator for the exponential-time computable conditional Kolmogorov function one can compute in polynomial space a list L such that the number of members of L is polynomial in m and that L contains the value of g at the given input. The same can be done for the function g' constructed from g as in Theorem 6.6.

- One can adapt Theorem 6.6 to show that $SCV \in \mathrm{PSPACE}^f$ whenever f is any strong 2-enumerator of the exponential-time computable conditional Kolmogorov function. The number of rounds is exponential and not polynomial as in Theorem 6.6 but the number m of coordinates in the considered vectors is again bounded by a constant. Therefore the polynomial bound on the space usage is kept.

These results can be put together to yield the following theorem.

Theorem 6.8 *The following statements are equivalent for every set A.*

- *A is in* EXP*;*

- *A is in* PSPACE^f *by a fixed machine where f is any strong* 2*-enumerator for the exponential-time computable conditional Kolmogorov function;*

- *There is a polynomially bounded function g and a fixed machine witnessing that A is in* PSPACE^f *where f is any strong* 2*-enumerator for g.*

Note that Theorems 6.7 and 6.8 can also stated with any constant $k \geq 2$. A detailed analysis shows that even a slowly growing non-constant function

k is possible. But $2^{\ell(n)}$ has always to be polynomially bounded, thus only functions where $k(n) \leq \log\log(n)$ can be considered.

And even this bound needs that the algorithm in Theorem 6.7 and the g' there is adapted: instead of eliminating single possible solutions from L, one has to eliminate intervals so that g' takes an interval of strings and not a single string as its value. Then one would have to apply an iterated elimination of a non-desired interval and splitting of the largest remaining one until only $2^\ell - 1$ many intervals of size 1 are left.

The next section deals with the question, what statements can be made for faster growing k.

7 Space-Bounded Kolmogorov Complexity

Buhrman and Torenvliet [6] show that the interactive proof (IP) protocol for PSPACE [20, 25] can be simulated by an NP-oracle machine that has access to a set of space-bounded Kolmogorov random strings. The NP machine in the proof guesses a sequence of polynomials and a sequence of numbers of high space-bounded complexity, that are used to generate the proof in the IP protocol. The fact that these numbers have high complexity is enough to guarantee correctness, since intersections of known polynomials have low complexity. For the proof of that theorem it suffices to generate numbers that have complexity that exceeds some constant. Below, we show how a strong $\log n$-enumerator can be used by an NP machine to generate numbers that have complexity that exceeds a given constant. As a consequence, enumerating $O(\log n)$ values of the space-bounded Kolmogorov function is hard for PSPACE under NP-reductions.

Theorem 7.1 *Let k and m be such that $k(n)m(n) \in O(\log n)$. Let f be a strong k-enumerator for $C_{space}(x \mid w, |x|)$, and assume that $C(m(n) \mid n) \in O(1)$. Then there is an* NP *oracle machine M such that M^f on input $\langle n, w\rangle$ can guess and verify a string x of length n such that $C_{space}(x \mid w, |x|) \geq m(n) - O(1)$.*

By taking m to be some high enough constant and using Buhrman and Torenvliet [6, Theorem 5.10], one can obtain the following corollary.

Corollary 7.2 *Let f be a strong $k(n)$-enumerator for $C_{space}(x \mid w, |x|)$ and let $k(n) \in O(\log n)$. Then* PSPACE = NP^f.

Proof of Theorem 7.1: Fix n and w, consider strings of length n, that is, for the remainder of this proof a string with name x will have $|x| = n$. Again, let f on input $\langle x, w\rangle$ output $f_1^x, \ldots, f_{k(n)}^x$ without loss of generality ordered such that $f_i^x < f_j^x$ whenever $i < j$. Let c be some constant depending on the choice of universal machine and this conversation. We prove that M can guess and verify a string of space bounded complexity $m(n) - c$. The proof consists of $k(n) + 1$ cases, depending on the values of the enumerator on strings of length n. We present case 0, then case 1, then generalize case 1 to case ℓ for $1 < \ell < k(n)$ and finally case $k(n)$.

case 0: If there is a string x of length n such that $f_1^x > m(n) - c$ then guess this string, verify $f_1^x > m(n) - c$ by querying $f(x)$ and output x.

case 1: We are not in case 0, so none of the strings x of length n has $f_1^x > m(n) - c$. Let $S = \{x : f_2^x > 2m(n) - c\}$ and suppose $\|S\| > 2^{m(n)-c}$. Let z be the concatenation of the first $2^{m(n)-c} + 1$ strings in S. Note that one of these strings x must have $C(x \mid w, n) \geq f_2^x > 2m(n) - c$ because we are not in case 0 and not all strings x' in S can have $C(x \mid w, n) \leq m(n) - c$. We claim that $C(z \mid w, n) \geq m(n) - c$. Suppose not then let P be a program of at most $m(n) - c - 1$ bits that prints z, and let s be a number that reveals the position of the first substring x in z with $C(x \mid w, n) > 2m(n) - c$. Without loss of generality $|P|$ is exactly $m(n)-c-1$. Note that $|s| \leq m(n)-c+1$. Let Q be a program that: 1. unpacks P and s from Ps (by first computing $m(n) - c - 1$ from n). 2. runs P to obtain z. 3. prints the sth thru $s + n - 1$st bit of z. Without loss of generality Q is self delimited and of size $d < c/2$. Then QPs is a string of less than $2m(n) - c$ bits that describes a string of complexity greater than $2m(n) - c$.

case ℓ: We are not in case $\ell - 1$, so there are at least $2^n - 2^{(\ell-1)m(n)-c} - 1$ strings x with $f_\ell^x \leq \ell m(n) - c$. Let $S = \{x \mid f_\ell^x \leq \ell m(n) - c \wedge f_{\ell+1}^x > (\ell+1)m(n) - c\}$. Suppose that $\|S\| > 2^{\ell m(n)-c}$. Note that one of these strings x must have $C(x \mid w, n) \geq f_\ell^x > (\ell+1)m(n) - c$ because we are not in case $\ell - 1$ and not all strings x' in S can have $C(x \mid w, n) \leq \ell m(n) - c$. Let z be the concatenation of the first $2^{\ell m(n)} - c + 1$ strings in S. We claim that $C(z \mid w, n) \geq m(n) - c$. Suppose not, then let P be a program of at most $m(n) - 1 - c$ bits that prints z and let s be a number that reveals the position of the first string x in z with $C(x \mid w, n) > (\ell+1)m(n) - c$. Note that $|s| \leq \ell m(n) - c + 1$. Without loss of generality P is a string of *exactly* $m(n) - c - 1$ bits. Let Q be a program that: 1. unpacks P and s from Ps (by first computing $m(n) - c - 1$ from n). 2. runs P to obtain z. 3. prints the sth thru $s + n - 1$st bit of z. Again without loss of generality, Q is self delimited and of size $d < c/2$. Then QPs is a string of less than $(\ell+1)m(n)$ bits that describes a string of complexity greater than $(\ell+1)m(n)$.

case $k(n)$: We are not in case $k(n)-1$. There are at least $2^n - 2^{(k(n)-1)m(n)-c} - 1$ strings x with $f^x_{k(n)} \leq k(n)m(n) - c$. However there is a constant b such that there are $2^n/b$ random strings, so this case cannot occur.

The NP machine can now on input x and w given f as an oracle guess a number $\ell < k(|x|)$ (the case). If $\ell = 0$ it guesses a y of length n such that $f^y_1 > m(n) - c$. Otherwise it guesses $S' \subseteq S = \{x \mid f^x_\ell \leq \ell m(n) - c \wedge f^x_{\ell+1} > (\ell+1)m(n)-c\}$ such that $\|S'\| = 2^{\ell m(n)-c}+1$ and lets z be the concatenation of all strings in S'. By the first part of the proof $C(z \mid w) > m(n) - c$. $\square$

Obvious extensions of the above theorem are: (1) strengthen the reduction type in Corollary 7.2 to deterministic polynomial time and (2) weaken the enumerator to $f(n)$ where $f(n)$ is some function between $\log n$ and n. In the following theorem we create a relativized world that may indicate that these extensions maybe hard to find. However, note that Corollary 7.2 has a non-relativizing proof.

Theorem 7.3 *There is a relativized world where C_{space} is polynomial-time $\log^2(n)$-enumerable but not polynomial-time computable; here $n = |x| + |w|$ when $C_{space}(x \mid w)$ has to be computed.*

Proof: Start with a relativized world where P = PSPACE. Let $b_0 = 1$ and $b_{m+1} = 2^{b_m}$ for all m. Now add an oracle A satisfying the following conditions:

- A is in EXP relative to the given world;
- If $n \notin \{b_0, b_1, \ldots\}$ then $A \cap \{0,1\}^n$ is empty;
- For every m, the intersection $A \cap \{0,1\}^{b_m}$ contains exactly one element x_m;
- Only the last $h(b_m)$ bits of x_m can be different from 0 where $h(n) = \frac{\log(n)\cdot\log(n)}{\log\log(n)}$;
- There is no sparse set in PSPACE which contains infinitely many x_m.

Note that $\mathrm{PSPACE}^A \neq \mathrm{P}^A$ in the given relativized world since one can compute the partial function $0^{b_m} \to x_m$ in PSPACE^A but not in P^A.

Since P = PSPACE (without oracle A) one can compute C_{space}. Of course $C^A_{space}(x \mid w) \leq C_{space}(x \mid w) + c_1$ for some constant c_1. On the other hand, if $U^A(p,w) = x$ one knows that there is a further machine V with $V(vp,w) = U^A(p,w)$ where v is the last $h(b_m)$ bits of x_m for the largest m where $|x_m| \leq 2 \cdot (|p| + |w|)$. Note that the computation of $U^A(p,w)$ cannot access the oracle A at strings longer than $2 \cdot (|p| + |w|)$. Then one knows that a program p' with $U(p', w) = V(vp, w)$ is at most c_2 bits longer than $|vp|$ and $C_{space}(x \mid w) \leq C^A_{space}(x \mid w) + c_2 + h(2 \cdot (|x| + |w|)) \leq C^A_{space}(x \mid w) + c_3 + 3 \cdot h(|x| + |w|)$ for a suitable constant c_3. This gives that

$$C_{space}(x \mid w) - c_3 - 3 \cdot h(|x| + |w|) \leq C^A_{space}(x \mid w) \leq C_{space}(x \mid w) + c_1$$

and C^A_{space} is $(3 \cdot h(n) + c_1 + c_3 + 1)$-enumerable. If n is sufficiently large, this expression is below $\log^2(n)$. If n is not large enough, one can get $C^A_{space}(x \mid w)$ from some table. Thus C^A_{space} has a P^A-computable strong $\log^2(n)$-enumerator although $\mathrm{PSPACE}^A \neq \mathrm{P}^A$. $\square$

Acknowledgments

We would like to thank William Gasarch, André Nies, Jim Owings, Sebastiaan A. Terwijn and Theodore A. Slaman for proofreading and interesting discussions on the subject; Gasarch also suggested to us the investigation of enumerations of the Kolmogorov function as a research topic. Paul Vitányi provided reference [27].

References

[1] Andris Ambainis, Harry Buhrman, William Gasarch, Bela Kalyanasundaram, and Leen Torenvliet. The communication complexity of enumeration, elimination and selection. In *Proc. 15th IEEE Conf. on Computational Complexity*, pages 44–53, 2000.

[2] Amihood Amir, Richard Beigel, and William Gasarch. Some connections between bounded query classes and nonuniform complexity. In *Proceedings of the 5th Annual Conference on Structure in Complexity Theory*, pages 232–243, 1990.

[3] Janis M. Barzdin. Complexity of programs to detemine whether natural numbers not greater than n belong to a recursively enumerable set. *Soviet Mathematics Doklady*, 9:1251–1254, 1968.

[4] Richard Beigel. *Query-limited Reducibilities.* PhD thesis, Department of Computer Science, Stanford University, 1987.

[5] Richard Beigel, William Gasarch, John Gill, and Jim Owings Jr. Terse, superterse and verbose sets. *Information and Computation*, 103:68–85, 1993.

[6] Harry Buhrman and Leen Torenvliet. Randomness is hard. *SIAM Journal on Computing*, 30(5):1485–1501, 2000.

[7] Jin-Yi Cai and Lane Hemachandra. Enumerative counting is hard. *Information and Computation*, 82(1):34–44, 1989.

[8] Jin-Yi Cai and Lane Hemachandra. A note on enumerative counting. *Information Processing Letters*, 38(4):215–219, 1991.

[9] Gregory Chaitin. On the length of programs for computing finite binary sequences. *Journal of the Association for Computing Machinery*, 13:547–569, 1966.

[10] Gregory Chaitin. A theory of program size formally identical to information theory. *Journal of the Association for Computing Machinery*, 22:329–340, 1975.

[11] Richard Friedberg and Hartley Rogers. Reducibilities and completeness for sets of integers. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 5:117–125, 1959.

[12] William Gasarch and Frank Stephan. A techniques oriented survey of bounded queries. In Cooper and Truss, editors, *Models and Computability: Invited Papers from Logic Colloquium 1997 – European Meeting of the Association for Symbolic Logic,* Leeds, July 1997, pages 117–156. Cambridge University Press, 1999.

[13] Juris Hartmanis. Generalized Kolmogorov complexity and the structure of feasible computations. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 439–445, 1983.

[14] Carl Jockusch and Richard Shore. Pseudo-jump operators I: the r.e. case. *Transactions of the American Mathematical Society*, 275:599–609, 1983.

[15] Carl G. Jockusch Jr. and Robert I. Soare. Π^0_1 classes and degrees of theories. *Transactions of the American Mathematical Society*, 173:33–56, 1972.

[16] Andrei Kolmogorov. Three approaches for defining the concept of information quantity. *Problems of Information Transmission*, 1:1–7, 1965.

[17] Martin Kummer. On the complexity of random strings. In *Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 1996.

[18] Martin Kummer and Frank Stephan. Effective search problems. *Mathematical Logic Quarterly*, 40:224–236, 1994.

[19] Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Graduate Texts in Computer Science. Springer-Verlag, second edition, 1997.

[20] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the Association for Computing Machinery*, 39(4):859–868, October 1992.

[21] André Nies. Lowness properties and randomness. *Manuscript*, 2003.

[22] Piergiorgio Odifreddi. *Classical Recursion Theory*. North-Holland, 1989.

[23] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[24] Alexander Russel and Ravi Sundaram. Symmetric alternation capturers BPP. *Computational Complexity*, 7:152–162, 1998.

[25] Adi Shamir. IP=PSPACE. *Journal of the Association for Computing Machinery*, 39(4):869–877, October 1992.

[26] Ray Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Information and Control*, 7:1–22, 224–254, 1964.

[27] Alexander K. Zvonkin and Leonid A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25:83–124, 1970.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'