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 a

[a]Email: beigel@cis.temple.edu. Department 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 Russian Foundation for Basic Research (grants N 04-01-00427, N 02-01-22001) and Council on Grants for Scientific Schools.

[h]Email: fstephan@comp.nus.edu.sg. School of Computing and Department of Mathematics, National University of Singapore, 3 Science Drive 2, Singapore 117543, Republic of Singapore. Research was supported by the Deutsche Forschungsgemeinschaft (DFG), Heisenberg grant Ste 967/1-1 while F. Stephan was working at the Universität Heidelberg until August 2003. From August 2003 until June 2004, F. Stephan was working at the National ICT Australia LTD which is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence Program. Since July 2004, F. Stephan is working at the National University of Singapore and partially supported by NUS grant number R252–000–212–112.

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

$k(n)$-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:

- 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 Kolmogorov 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 Canetti and independently 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 a polynomial time 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 a polynomial time 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, quantum information processing and complexity theory. Li and Vitányi [23] provide a detailed discussion of many of these directions.

The Kolmogorov complexity is not computable [11, 19, 29], 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 [33] 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 one cannot compute the value of the function exactly, perhaps it would be possible to output several candidates for the value of the function, one of which is the actual value. 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, 6, 8, 9, 15, 22]. 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 unbounded 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 downward under Turing reduction and it has several other natural characterizations [25].

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 $\mathrm{S}_2^{\mathrm{p}}$ [10, 27] in terms of bounded truth-table and truth-table reductions to strong 2-enumerators. We show that P = 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 when 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* [23], *is given as $C(x|y) = \min\{|p| : U(p,y) = x\}$ where p stands for a program and U for a universal machine, that is, for a binary partial-recursive function satisfying that for every other binary partial-recursive function V there is a constant c such that whenever $V(p,y)$ is defined then there is a $q \in \{0,1\}^*$ with $|q| \leq |p| + c$ and $U(q,y) = V(p,y)$. We also use a 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 [16] defined a time-bounded version of Kolmogorov complexity, but resource-bounded versions of Kolmogorov complexity date back as far as 1968 [3], see also [23]. The space bounded version C_{space} is used in many results where $C_{space}(x|y)$ is defined as the length of the shortest program p telling how to compute x from input y with a space bound polynomial in $|p| + |x| + |y|$. Note that the polynomial in this bound is in the same way part of the underlying convention as the choice of the universal machine.

Intuitively, a computable k-enumerator for a function h enumerates on any input x up to $k(|x|)$ possible values such that $h(x)$ 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 [14] 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 C, then one can without loss of generality assume the following additional properties of f:

- *For all x, $f_1^x > f_2^x > \ldots > f_{b_x}^x$.*
- *There is a constant c with $f_1^x \leq |x| + c$ for all 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 notation 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.

The reader might note that enumerators appear outside the field of frequency computation also in the context of the study of recursively traceable and r.e. traceable oracles where Terwijn and Zambella [31] called a set A recursive traceable iff there is a recursive function k such that every $g \leq_T A$ has a strong $k(n)$-enumerator; A is r.e. traceable iff there is a recursive function k such that every $g \leq_T A$ has a $k(n)$-enumerator.

3 The Complexity to Compute Enumerators

The fact that the Kolmogorov function itself is hard for r.e. sets is a folk theorem with a relatively easy proof. Kummer [21] 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.

Note that not only C itself is Turing hard for r.e. sets, but also every nontrivial lower bound for C is. Li and Vitanyi [23] considered the function

m given as $m(x) = \min\{C(x) : x \in \{0,1\}^* \wedge |x| \geq n\}$ which is the largest nondecreasing lower bound of the Kolmogorov function. This function m – as well as any further nondecreasing unbounded function majorized by m – is Turing hard for r.e. sets; one can exactly take the proof in [33]: The function $s(n) = \min\{n : m(n) > 2e + c\}$ majorizes $\min\{t : \varphi_{e,t}(e)\downarrow\}$ for some constant c and all $e \in K$ since the first t where $\varphi_{e,t}(e)$ is defined can be computed from e. Thus $e \in K$ iff $\varphi_{e,s(n)}(e)$ is defined.

This paper is not about the complexity of the Kolmogorov function itself, but about enumerations of the Kolmogorov function. The question is: how hard are these enumerations? Clearly, the complexity depends on the number of outputs of such a function. On one hand, the Kolmogorov function can be directly computed from a 1-enumerator. 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. The constant c used here is the one given in Definition 2.1. In this paper we determine how the complexity depends on the number of values that are output by the enumerator and 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 recursive $k(n)$-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. Let f be any enumerator for C which satisfies all conditions from Remark 2.3. The goal is now to get the desired lower bound for k given as $k(n) = \max\{b_x : x \in \{0,1\}^n\}$.

In order to use in the considerations for the lower bound the index e of f as a partial-recursive function, the considered input is chosen such that e can be computed from it: the inputs are taken from sets $X_{n,e} = \{0\}^e \cdot \{1\}^1 \cdot \{0,1\}^{n-e-1}$.

Now consider any $n \geq 2^e$ and all inputs from $X_{n,e}$. In the following, let m be the maximal number such that there is an $x \in X_{n,e}$ with f^x_m being defined; note that $m \leq k(n)$ and $C(x) = f^x_m$ for this x. The lower bound for $k(n)$ will be established in the following case-distinction.

(a) There is an $x \in X_{n,e}$ with f^x_m being defined and $f^x_m \geq n/2 - e$. Given m, n, e, one can search for an $x \in X_{n,e}$ such that $f^x_m = \varphi_e(x, m)$ is defined and $f^x_m \geq n/2 - e$. 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 $n/2 - e \leq c_1 + 4 \cdot \log(n)$

and thus case (a) holds only for finitely many n.

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

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

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 $f^x_{m'+1}$ is defined there is a z where the computation of $f^z_{m'+1}$ terminates last. There is a $d \geq 1$ such that $f^z_{m'+d} = C(z)$. There is a description y of length $f^z_{m'+d}$ such that $U(y) = z$. By choice of m', $C(z) < n - (m'+d) \cdot b/2 - e$. Now it is shown that given y, b, d, one can compute an x' of length n with $C(x') \geq C(z) + d \cdot b/2$.

1. Compute $U(y)$ – the result is z.

2. Compute the number of 0s 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 is the set of those $x \in X_{n,e}$ where $f^x_{m'+1}$ is defined at all.

7. Search for an $x' \in X_{n,e} - Y$ such that $f^{x'}_{m'} \geq C(z) + d \cdot b/2$ – for this x', $f^{x'}_{m'} = C(x')$ and $f^{x'}_{m'+1}$ is undefined.

Note that, in the seventh step, x' exists since by the choice of m': there is an element x'' in $X_{n,e}$ such that $f^{x''}_{m'}$ is defined and $C(x'') \geq n - m' \cdot b/2 - e$. Furthermore, m' is the maximal such value and therefore either $f^{x''}_{m'+1}$ is undefined or $C(x'') < n - (m'+1) \cdot b/2 - e$; as that second condition cannot hold, the first one is true and $x'' \notin Y$. Since $C(z) < n - (m'+d) \cdot b/2 - e$, one has $C(x'') \geq n - m' \cdot b/2 - e > C(z) + d \cdot b/2$ and x'' is a possible choice for x'.

Now 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 $a = \max\{b' + 2 : b'$ 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 above constants c_1, c_2, a are independent of e and that the construction goes through whenever the e-th partial-recursive function is an enumerator for C with the properties from Remark 2.3. □

Remark 3.2 *The result above is 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.*

The concept of enumerators can be relativized. A function g is $k(n)$-enumerable relative to A if there is an $k(n)$-enumerator f such that $x, t \to f_t^x$ is a partial A-recursive function with $g(x) \in f(x) = \{f_1^x, \ldots, f_{b_x}^x\}$ for all x. The next result provides that – for a constant k – the Kolmogorov function is k-enumerable relative to A iff A is Turing-hard for r.e. sets. For the proof of this result, Π_1^0-classes are important.

Remark 3.3 (Π_1^0-classes) *A Π_1^0-class is the complement of a Σ_1^0-class. A Σ_1^0-class can be described by an oracle machine (with void input) which on oracles inside the class reads some data and eventually terminates in an accepting state while on oracles outside the class it either never halts or terminates in a rejecting state. Since the latter can be avoided by going into an artificially nonterminating loop, one can characterize any given Π_1^0-class as the class of all oracles for which a suitable oracle machine (with void input) does not halt. That is, the Π_1^0-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_1^0-class is the set of infinite branches of a binary recursive tree T; here $\sigma \in T$ iff M has not yet halted with oracle σ quickly, that is, the computation M^σ does not terminate within $|\sigma|$ steps.

One can relativize the concept of a Π_1^0-class to an oracle A and adapt the cone-avoidance result of Jockusch and Soare [18, Theorem 2.5] *to the following. Given a set $A \not\geq_T K$, the relativized Π_1^0-class*

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

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

Theorem 3.4 *Let $k \in \{1, 2, 3, \ldots\}$ be a constant. The Kolmogorov function is k-enumerable relative to a set A iff $A \geq_T K$.*

Proof: Since $C \leq_T K$, the Kolmogorov function C is k-enumerable relative to any $A \geq_T K$ by taking $f(x) = \{C(x)\}$ for all x. So the main task is to show that this is impossible for $A \not\geq_T K$.

Let a constant $k \in \{1, 2, 3, \ldots\}$ and an oracle $A \not\geq_T K$ be given. Assume by way of contradiction that there is a partial A-recursive k-enumerator f for C, that is, the function $x, t \to f_t^x$ is partial A-recursive, $b_x \in \{1, 2, \ldots, k\}$ and $f_{b_x}^x = C(x)$ for all x.

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 intersection of all nonempty $X_{n,i}$ contains then – for each n – strings where C is difficult to k-enumerate. The further outline is that a set B_F selecting such strings for each n will be chosen such that on one hand $A \oplus B_F \not\geq_T K$ and on the other hand one can compute a function dominating c_K using k-enumerator at the strings selected by B_F, a contradiction.

The algorithm to enumerate the $X_{n,i}$ is uniform in n and has a variable i whose value is always the largest index such that $X_{n,0}, X_{n,1}, \ldots$ is not empty. For each of these sets there is at most one stage s where elements are enumerated into it.

- 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 an $x \in X_{n,i-1} - X_{n,i}$ for which $C_s(x)$ is greatest and enumerate x into $X_{n,i}$.

Note that for every n, the maximal value of the variable i is 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_1^0-class since the conditions to be checked are uniformly Π_1^0: 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 nonemptiness is the function F which is defined on input n as follows: Let i be the maximal number where $X_{n,i} \neq \emptyset$ and let $F(n)$ is the lexicographically first 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 an $(A \oplus B_F)$-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, that is, $j_n = b_{F(n)}$ and $f_{j_n}^{F(n)} = C(F(n))$. 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 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 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, one can compute $F(m)$ from m, i and the number of elements which go into $X_{m,i}$ before $F(m)$. 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. □

Considering a slowly increasing, recursive and unbounded functions $k(n)$ instead of a constant k, the Theorem 3.7 below shows that one can find an r.e. incomplete degree relative to which the Kolmogorov function is $k(n)$-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; Muchnik presented the first such construction 1999 in a seminar in Moscow and included the publication of the result into this work. Nies [25] showed that the following two notions are both equivalent to saying that E is low for prefix-free Kolmogorov complexity:

- The notion "Low for Random" introduced by Zambella [32] and proven to exist outside the class of the recursive sets by Kučera and Terwijn [20]: E is low for random iff every Martin-Löf random set is also Martin-Löf random relative to E.
- The notion "H-Trivial", also called "K-trivial", introduced by Chaitin [12] and proven to exist outside the class of the recursive sets by Solovay [30]: E is H-trivial iff $\exists c \forall n \, (H(E(0)E(1)\ldots E(n)) \leq H(n) + c)$.

Downey, Hirschfeldt, Nies and Stephan [13] provide an overview on this field and indicate with reference to Muchnik that Theorem 3.6 can be obtained by modifying some proof of a related result.

Definition 3.5 *A set E is* low for prefix-free Kolmogorov complexity *if* $\exists c \forall x \, (H^E(x) \leq H(x) + c)$.

Theorem 3.6 *There is a recursively enumerable and nonrecursive set which is low for prefix-free Kolmogorov complexity.*

Proof: The set E is constructed to satisfy the following two conditions:

- E is simple;
- There is a constant c such that $\forall x \, (H(x) \leq H^E(x) + c)$.

In order to deal with H^E properly, the universal machine U on which the Kolmogorov complexity is based is chosen such that it makes use of an oracle and satisfies the following conditions:

- U^A is a universal prefix-free machine defining H^A for all oracles A;
- For all oracles A, the sum over all $2^{-|p|}$ where $U^A(p)$ is defined is at most $1/2$.

The set E is enumerated in stages with E_0 being initialized as $\emptyset$. In step s one searches for the least $e \leq s$ such that there is an $x \leq s$ to satisfy the following four conditions where the first three conditions guarantee that E will be a simple set and the fourth is the Kraft-Chaitin condition:

- first, $x \geq 2e$;
- second, $x \in W_{e,s}$;
- third, no element is already enumerated into the intersection of W_e and E within s steps, that is, $W_{e,s} \cap E_s = \emptyset$;
- fourth, $r_s < 2^{-e-2}$ where r_s is the sum of all $2^{-|p|}$ such that the computation $U^{E_s}(p)$ has terminated within s computation steps and queried the oracle E_s at x.

If an e is found and x is the least witness for e to satisfy the given four conditions then $E_{s+1} = E_s \cup \{x\}$ else $E_{s+1} = E_s$.

Now let G be the set of all triples (p, y, s) such that $U^{E_{s+1}}(p)$ is defined and outputs y within $s + 1$ computation steps and the computation does either not halt in s steps or there is a number queried by the computation which is in E_{s+1} but not in E_s. Let G_1 be the subset of all $(p, y, s) \in G$ such that no element queried by the computation of $U^{E_{s+1}}(p)$ is in $E - E_{s+1}$; note that all elements queried are in the set $\{0, 1, \ldots, s\}$ but it might be that only some of the members of this set are queried. Let $G_2 = G - G_1$. Now it is shown that the sum of all $2^{-|p|}$ over all $(p, y, s) \in G_a$ is bounded by $1/2$ for $a = 1, 2$ which then gives that the complete sum over all $(p, y, s) \in G$ is bounded by 1.

First consider G_1. Note that $(p, y, s) \in G$ iff $U^E(p) = y$ and s is the first stage such that $U^E(p)$ halts within $s + 1$ steps and no element queried by the computation is in $E - E_{s+1}$. Note that no computation is considered to halt in 0 steps and therefore it is safe to work with $s + 1$ and not with s. The implication $(p, y, s) \in G \Rightarrow U^E(p) = y$ and the fact that for every (p, y) with $U^E(p) = y$ there is at most one s with $(p, y, s) \in G_1$ imply that one can transfer the bound on the halting probability of U^E to a bound for the sum over all members of G_1:

$$\sum_{(p,y,s)\in G_1} 2^{-|p|} \leq \sum_{p:U^E(p)\downarrow} 2^{-|p|} < 1/2.$$

Second consider G_2. If $(p, y, s) \in G_2$ then there is a $t > s$ such that an element queried by the computation of $U^{E_{s+1}}(p)$ is in $E_{t+1} - E_t$. In particular, $2^{-|p|}$ contributes to the sum r_t considered in the fourth condition of the choice of e and x in the enumeration algorithm for E. Thus one gets the inequality

$$\sum_{(p,y,s)\in G_2} 2^{-|p|} < r_0 + r_1 + \ldots < 1/2$$

where the second part $r_0 + r_1 + \ldots < 1/2$ is due to the fact that every stage deals either with one e uniquely assigned to it or does not change the oracle: if $E_{s+1} = E_s$ then $r_s = 0$ and if $E_{s+1} = E_s \cup \{x\}$ for some $x \notin E_s$ then there is a parameter e with $x \in W_{e,s+1} \cap E_{s+1}$, $W_{e,s} \cap E_s = \emptyset$ and $r_t < 2^{-e-2}$. So the sum of all r_t can be bounded by the sum of all 2^{-e-2} which is $2^{-2} + 2^{-3} + 2^{-4} + \ldots = 1/2$. Using $G = G_1 \cup G_2$, the inequality

$$\sum_{(p,y,s)\in G} 2^{-|p|} < 1$$

holds. Thus one can build by the Kraft-Chaitin Theorem a prefix-free machine V such that for every $(p, y, s) \in G$ there is a $q \in \{0,1\}^{|p|}$ with $V(q) = y$. For every y there is a shortest description $p \in \{0,1\}^{H^E(y)}$ with $U^E(p) = y$. For these y, p there is an s with $(y, p, s) \in G_1$ and therefore there is a $q \in \{0,1\}^{|p|}$ with $V(q) = y$. Thus $H^E(y) \leq \min\{|q| : V(q) = y\}$. It follows that there is a constant c with $\forall y\, (H^E(y) \leq H(y) + c)$ and E is low for prefix-free Kolmogorov complexity.

It remains to show that E is simple. So let e be an index of an infinite set. Let $x \in W_e$ be so large that $x \geq 2e$ and

$$\sum_{(p,y,s)\in G \wedge s \geq x} 2^{-|p|} < 2^{-e-2}.$$

Such an x exists since the sum of all $2^{-|p|}$ over all $(p, y, s) \in G$ converges and one can pick an x from the infinite set W_e which is larger than the parameter s in those first finitely many addends which are needed to get all of the sum but a missing subsum of less than 2^{-e-2}. Now let t be so large that $t > x$, $x \in W_{e,t}$ and $W_{e',t}$ meets E_t whenever $e' < e$ and $W_{e'}$ meets E. It follows that no $e' < e$ qualifies at stage t but either $W_{e,t}$ meets E_t or e qualifies by satisfying all four conditions; the last one is met by arguing that r_t is bounded by all the sum of $2^{-|p|}$ over all the $(p, y, s) \in G$ with $s \geq x$. It follows that either $W_{e,t}$ meets E_t or an element of W_e is enumerated into E at stage t, so W_e and E are not disjoint. That E is coinfinite is enforced by the first condition which implies that there are at most e numbers x smaller than $2e$ enumerated into E. That E is recursively enumerable is easy seen from the fact that every step in the given enumeration procedure of E is done by the algorithm in finite time. This completes the proof. $\square$

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

Proof: One can generalize Theorem 3.6 such that there is an oracle machine which enumerates relative to an oracle B a set E^B with the following properties:

- E^B is simple relative to B: it is coinfinite, recursively enumerable relative to B and intersects with every infinite B-r.e. set.
- $B \leq_T E^B$.
- E^B is low for prefix-free Kolmogorov-complexity relative to B, that is, $\exists c \forall x (H^B(x) \leq H^{E^B}(x) + c)$.

As the anonymous referee of the Journal of Symbolic Logic pointed out, the relativization of Theorem 3.6 to this result is standard; nevertheless, a direct proof of this result is found in the technical report version of this paper [5]. Jockusch and Shore [17] showed that the second condition $\forall B\, (B \leq_T E^B)$ implies that there is an r.e. set A for which E^A has the Turing degree of the halting problem. Fix from now on A with this property. Note that even $A <_T E^A \equiv_T K$ since E^A is simple relative to A. Nies [25, Theorem 6.3(II)] mentioned this construction of such a set A first, details are given here.

Let U^B be the universal function for prefix-free Kolmogorov complexity; that is, for every oracle B, the complexity H^B relative to B is defined using U^B. 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 $|p| \leq 3\log(k(n))$.

Note that due to U^A being prefix-free, $\tilde{f}$ is already a $k(n)$-enumerator. Now one constructs from $\tilde{f}$ a $k(n)$-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; b_x is as in Remark 2.3.

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 bounded by $2\log(r)$ for almost all 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, $|q_{r(x)}| \leq 3\log(k(|x|))$ 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, 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 Strong 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. As a corollary one can say that every strong k-enumerator for C computes the halting-problem, but the proof does not provide an algorithm to do this for all strong enumerators in a uniform way. Therefore we ask in this section which information can be retrieved uniformly from a given strong 2-enumerator for C. Later we will ask the same question in the complexity-theoretic context and show that the complexity-class PSPACE can be characterized this way. In the present section, the next result reveals that enumerators uniformly carry some information although the further results show that it is not possible to compute a concrete nonrecursive set uniformly from every strong enumerator of the Kolmogorov function. Indeed, 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. Note that any extension f of ψ has PA-complete Turing degree if ψ is taken to be

$$\psi(x) = \begin{cases} \varphi_x(x) & \text{if } \varphi_x(x) \downarrow \in \{0,1\}; \\ \uparrow & \text{otherwise.} \end{cases}$$

That is, such an f can compute a complete and consistent extension of Peano Arithmetic in this case.

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 $U(p10^{l+l'}1) = \tilde{U}(p)$ if $\tilde{U}(p)$ is defined, $|p| + 2 + l \equiv 0$ 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. □

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 strong 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 strong enumerator. Furthermore, in the 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 function 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. □

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 b be a recursive bound for g, more precisely, choose b such that b is recursive and $g(x) \in \{0, 1, \ldots, b(x)\}$ for all x. For input x, compute the number h of queries made by the reduction φ_e and the h places $q_1, \ldots, q_h$ of these queries. Furthermore, compute the maximal possible values $b(q_1), \ldots, b(q_h)$ of g at these places. Now search for y, $n_1 \in \{0, \ldots, b(q_1)\}$, $\ldots$, $n_m \in \{0, \ldots, b(q_m)\}$ such that for all $m_1 \in \{0, \ldots, b(q_1)\}$, $\ldots$, $m_h \in \{0, \ldots, b(q_h)\}$, the reduction φ_e returns for input x the value y provided that it receives the answers $\{n_1, m_1\}, \ldots, \{n_h, m_h\}$ for the queries to the strong 2-enumerator at the places $q_1, \ldots, q_h$. The verification of the correctness and existence of this answer y is analogous to the verification in the proof of Theorem 4.2. □

The condition that g is recursively bounded is necessary. Recall the nonrecursive 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 strong 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 has 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. $\square$

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: This result is obtained by combining the methods from Theorem 4.4 with a finite extension argument. 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. □

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 partial-recursive function 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 the prefix-free Kolmogorov complexity 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 $3 + 2\log(m)$ bits 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. This gives 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, nondecreasing and unbounded function k with $k(0) = 1$ the prefix-free Kolmogorov function is $k(n)$-enumerable relative to A.*

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 not matter whether g is C, is H or is something else.

6 Resource-Bounded Reductions to 2-Enumerators

From now on, we consider the resource-bounded world. In particular, we consider polynomial time reductions to 2-enumerators of the conditional polynomial space Kolmogorov complexity and other functions. We characterize some well-known complexity classes as follows: A set A is in this class iff there is a reduction of a specific type computing A relative to any 2-enumerator of a suitable function. In the case of PSPACE, this function can be the space bounded conditional Kolmogorov complexity. Besides PSPACE, we will be able to characterize the class S_2^p which was introduced by Canetti [10] and independently Russel and Sundaram [27]. 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 exist a polynomial p and a polynomial time computable ternary predicate Q such that*

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

Note that for general languages in $\Sigma_2^{\mathrm{p}} \cap \Pi_2^{\mathrm{p}}$ the second occurrence of Q could be replaced by an arbitrary polynomial time predicate Q'. It is not known whether $\mathrm{S}_2^{\mathrm{p}} = \Sigma_2^{\mathrm{p}} \cap \Pi_2^{\mathrm{p}}$. The class $\mathrm{S}_2^{\mathrm{p}}$ 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. The following result is included as it deals with reductions to enumerators and is a model for the latter characterization of PSPACE but the result itself has no connection to Kolmogorov complexity.

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

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

(b) *There are a fixed polynomial time 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 polynomial time 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 polynomial time 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 values with one of them being $g(q(x))$ from any strong 2-enumerator for h; this computation will be realized as a tt-reduction. 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 $i, j \in \{0, 1, \dots, |x|^c\}$ 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 can 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 $i, j \in \{0, 1, \dots, |x|^c\}$ 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 lengths 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 $v = h'(u_1)h'(u_2) \ldots h'(u_{p(x)})$;
- $Q(x, v, w)$ is false if $x \notin A$ and $w = h'(u_1)h'(u_2) \ldots h'(u_{p(x)})$.

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

Definition 6.3 *Let U be a fixed space bounded machine with two inputs p, w which respects the space bound $2(|p|+|x|+|w|)$ and is universal for this space bound in the sense that for every V respecting the same space bound there is a constant c such that whenever $V(p, w)$ is defined then there is a program q of length up to $|p| + c$ with $U(q, w) = V(p, w)$. 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.*

Note that one could define the space bound also with respect to some other underlying polynomial in $|p| + |x| + |w|$ instead of the given $n \to 2 \cdot n$; one just has to fix any reasonable choice in the same way as a universal machine is fixed. The universal machine is keeping track of its space use and checks before outputting x whether the bound is kept; if the bound is not kept then it just does not halt.

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

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

where $a_1, b_1, a_2, b_2, \ldots, a_n, b_n$ are Boolean variables and ϕ has no bound variables. The parameter n is not a constant but denotes the half of the number of free variables occurring in ϕ. If $n = 0$ then ϕ is just a Boolean combination of the Boolean constants "false" and "true". 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 any given constants c, k there is a constant ℓ such that one can find, for every w, a string $u \in \{0,1\}^\ell$ with $C_{space}(u|\ell, w) > c$ by querying, for all $v \in \{0,1\}^\ell$, the values $f_1^{v,\ell,w}, \ldots, f_k^{v,\ell,w}$ given by a strong k-enumerator f for $C_{space}(v|\ell, w)$.*

The time and space complexity to find u is independent of w except at the place where the queries to f occur and v, ℓ, w have to be copied onto the oracle tape to query f.

Proof: In the following, let $\{f_1^{v,\ell,w}, \ldots, f_k^{v,\ell,w}\}$ be the output of f on input v, ℓ, w and let $f_1^{v,\ell,w} < \ldots < f_k^{v,\ell,w}$; one of these k values is $C_{space}(v|\ell, w)$.

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 ℓ, every interval $I \subseteq \{0,1\}^\ell$, every $u, u' \in I$ and all w, $C_{space}(u'|\ell, w) \leq C_{space}(u|\ell, w) + c' + 2 \cdot \log(\|I\|)$ where $\|I\|$ is the cardinality of I.

Now let $c_1 = c$ and inductively $c_{i+1} = 3c_i + c' + 2$ for $i = 1, 2, \ldots, k$. Let $\ell = c_{k+1}$ and let $L_i = \{v \in \{0,1\}^\ell : f_i^{v,\ell,w} \leq c_i\}$ for $i = 1, 2, \ldots, k$.

Now fix w. On one hand, there are less than 2^{c_k+1} strings with conditional complexity up to c_k for the given w; since $f_k^{v,\ell,w}$ 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 at least 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 and I.

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

If $i > 1$ then split I canonically into $2^{c_i - c_{i-1}}$ disjoint subintervals J of length $2^{c_{i-1}}$. Each such interval J contains an element of L_{i-1} by choice of i and I, so let v_J be the least element of the intersection $L_{i-1} \cap J$. Note that $f_{i-1}^{v_J,\ell,w} \leq c_{i-1} < c_i < f_i^{v_J,\ell,w}$ for all J. If J, J' are neighbouring subintervals of I and $C_{space}(v_J|\ell, w) \leq c_{i-1}$ then $C_{space}(v_{J'}|\ell, w) \leq C_{space}(v_J|\ell, w) + c' + 2\log(2 \cdot 2^{c_{i-1}}) \leq 3c_{i-1} + c' + 2 = c_i < f_i^{v_{J'},\ell,w}$. Hence $C_{space}(v_{J'}|\ell, w) \leq f_{i-1}^{v_{J'},\ell,w} \leq c_{i-1}$. By induction we get that $C_{space}(v_J|\ell, w) \leq c_{i-1}$ either for all or for none of the considered subintervals J of I. Since there are $2^{c_i - c_{i-1}}$ such intervals J, $2^{c_i - c_{i-1}} \geq 2^{c_{i-1}+1}$ and the number of J with $C_{space}(v_J|\ell, w) \leq c_{i-1}$ is at most $2^{c_{i-1}+1} - 1$, the relation $C_{space}(v_J|\ell, w) > c_{i-1}$ holds for all J. Taking J to be the first subinterval of I of length $2^{c_{i-1}}$ and letting $u = v_J$, we have $C_{space}(u|\ell, w) \geq c_{i-1} \geq c$. □

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. Recall that all QBF-instances considered have an even number of variables.

The algorithm consists of a loop which is entered with a set $\{x, y_1, \ldots, y_m\}$ of instances and a set $V \subseteq \{0,1\}^{m+1}$ of binary vectors such that, for a constant j to be determined later, the following constraints hold and steps are done:

1. $|V| \leq j$ and $m \leq j^2$;

2. V consists of binary vectors representing functions from $\{x, y_1, \ldots, y_m\}$ to $\{0,1\}$ such that $(QBF(x), QBF(y_1), \ldots, QBF(y_m)) \in V$;

3. in each round, instances $z_1, \ldots, z_{4m}$ are selected by fixing the first two variables in $y_1, \ldots, y_m$ in all four possibilities and a set $W \subseteq V \times \{0,1\}^{4m}$ is selected such that every vector $w \in W$ is consistent with the self-reduction of QBF on $\{y_1, \ldots, y_m\}$ to QBF on $\{z_1, \ldots, z_{4m}\}$;

4. queries to the k-enumerator for the space bounded conditional Kolmogorov complexity are used in order to remove vectors different from QBF on $x, y_1, \ldots, y_m, z_1, \ldots, z_{4m}$ until at most j vectors are left in W;

5. up to j^2 instances $y'_1, \ldots, y'_{m'} \in \{z_1, \ldots, z_{4m}\}$ are selected and the projection V' of W onto the coordinates belonging to $x, y'_1, \ldots, y'_{m'}$ for running all above steps in the next round;

6. the algorithm halts when all variables are removed or the value of $QBF(x)$ can be determined otherwise.

In the following fix k to be the constant from the k-enumerator for C_{space}. Now some constants and functions are introduced which are needed for the step to reduce the number of vectors in Step 4. For any vector $r = (a_0, a_1, \ldots, a_h)$ of QBF-instances, let

$$g(r) = (QBF(a_0), QBF(a_1), \ldots, QBF(a_h)).$$

For r and further inputs $\ell', u_1, \ldots, u_{j'}$ with $j' = 2^{\ell'} - 1$, let $g'(r, \ell', u_1, \ldots, u_{j'})$ be that string in $\{0,1\}^{\ell'}$ which represents an i such that

- i is the first number with $u_i = g(r)$ if $g(r) \in \{u_1, \ldots, u_{j'}\}$;
- $i = 0$ if $g(r) \notin \{u_1, \ldots, u_{j'}\}$.

Now choose a constant c such that

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

for all $r, \ell', u_1, \ldots, u_{j'}$ of the form as above. This constant c exists since g, g' are computable in linear space. Let ℓ depend on c, k as in Proposition 6.5 and let $j = 2^\ell - 1$.

The just outlined algorithm is now presented in detail. 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; the numbers of steps in the verification below refer to this algorithm and no longer to the overview at the beginning of this proof.

1. 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. Recall that $g(x, y_1, \ldots, y_m, z_1, \ldots, z_{4m})$ is the $5m+1$-fold characteristic function

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

   and $r = (x, y_1, \ldots, y_m, z_1, \ldots, z_{4m})$.

2. Now let W be the set of all strings $w \in \{0,1\}^{5m+1}$ such that

   - w is an extension of a $v \in V$;
   - 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.

3. While $|W| \geq j$, use the oracle f to find an i such that $u_i \neq g(r)$ where $u_1, \ldots, u_j$ are the first j members of W and remove u_i from W. This i can be found by searching for a number where the binary representation $b_1 \ldots b_\ell$ satisfies

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

   Such an index can be found by Proposition 6.5.

4. If all $w \in W$ give the same value for $QBF(x)$ then output this value and halt. (We say that the algorithm halts with output.)

5. For all different $w, w' \in W$ select one $z_{d(w,w')}$ such that the components of w, w' at $z_{d(w,w')}$ representing $QBF(z_{d(w,w')})$ are different. If for some such w, w', $z_{d(w,w')}$ does not exist or does not contain at least two variables, then the algorithm halts without output. Otherwise, replace $y_1, \ldots, y_m$ by a new sequence of instances $y'_1, \ldots, y'_{m'}$ which are these selected $z_{d(w,w')}$.

6. 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 $x, y'_1, \ldots, y'_{m'}$.

After each third step of every round, W contains at most j binary vectors. In each fifth step of every round, 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 with output, 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 the same statement 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 W and never removed from W 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 the algorithm always halts with output, that is, that it never halts without output what can happen only in the fifth step. Note that whenever an instance $z_{d(w,w')}$ is selected such that w differs from w' on $z_{d(w,w')}$ then it has an even number of bound quantified variables which could be easily verified by induction and this even quantity is not 0 since otherwise w, w' would have to be consistent with $QBF(z_{d(w,w')})$ due to the consistency check in the second step. So the only cause could be that $z_{d(w,w')}$ is not found and w, w' coincide on $z_1, \ldots, z_{4m}$. Let v, v' be the restrictions of w, w' to the components for the instances $x, y_1, \ldots, y_m$. If it is the first round then one of the vectors v, v' is $(0, 0)$ and the other one $(1, 1)$ so that they differ on the instance y_1. Otherwise there is a previous round and $y_1, \ldots, y_m$ had been selected in the fifth step of the previous round such that v, v' differ on some y_k. But now one obtains a contradiction since the consistency check in the second step enforces that one can compute w, w' on $y_1, \ldots, y_m$ from w, w' on $z_1, \ldots, z_{4m}$ via the given self-reduction and thus w, w' would coincide on $y_1, \ldots, y_m$. Therefore the instance $z_{d(w,w')}$ exists and the algorithm does not halt without output. □

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

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

(b) *A is polynomial time Turing reducible by a fixed Turing 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 polynomial time 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 with 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. $\square$

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 [26, 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 ties exist);
- 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. A complete proof is omitted, but a sketch is given by presenting the basic concepts and intermediate results.

- SCV is the succinct circuit value problem considered by Papadimitriou [26, Section 20.1]. SCV is the set of all succinctly coded circuits which are evaluated to 1. Formally, an 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 have space bound s on all legal inputs.

  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 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 $SCV \in \text{PSPACE}^f$ 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 components 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 be stated with any constant $k \geq 2$. A detailed analysis shows that even a slowly growing non-constant function k is possible. But $2^{\ell(n)}$ always has to be polynomially bounded, thus only functions where $k(n) \leq \log\log(n)$ can be considered.

And even this bound requires that the algorithm in Theorem 6.7 and the g' there be 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 of what statements can be made for faster growing k.

7 Space Bounded Kolmogorov Complexity

In this section we show how to solve any PSPACE-complete problem given nondeterministic access to an $O(\log n)$ enumerator for the conditional space bounded Kolmogorov complexity.

Theorem 7.1 *If $k(n) \in O(\log n)$ and f is a strong $k(n)$-enumerator for $C_{space}(x \mid w, |x|)$ then* PSPACE $\subseteq$ NP^f*.*

Our proof builds on the work of Buhrman and Torenvliet [7] who show that the interactive proof (IP) protocol for PSPACE [24, 28] 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 space-bounded complexity greater than some fixed constant that are used to generate the proof in the IP protocol.

Theorem 7.2 (Buhrman-Torenvliet) *There is a constant m such that* PSPACE $\subseteq$ NP^g *for all functions $g(w, 1^n)$ that outputs a string x of length at least n with $C_{space}(x \mid w, |x|) > m$.*

Given a strong $k(n)$-enumerator f for the space bounded conditional Kolmogorov function we show how to create a g that nondeterministically reduces to f and fulfills the conditions of Theorem 7.2. Theorem 7.1 immediately follows.

Lemma 7.3 *If $k(n) \in O(\log n)$, f a strong $k(n)$-enumerator for $C_{space}(x \mid w, |x|)$ and m is a constant then there is an* NP *oracle machine M such that M^f on input $\langle 1^n, w\rangle$ can guess and verify a string x of length at least n such that $C_{space}(x \mid w, n) \geq m$.*

Proof of Lemma 7.3: Without loss of generality we assume m is greater than the constant-size programs described in this proof.

Fix w and a large n and let $k = k(n)$. We assume the enumerator f on input x will give us a list $f_1^x, \ldots, f_k^x$ ordered such that $f_i^x < f_j^x$ whenever $i < j$.

We create our machine M^f that acts as follows:

- Guess an integer ℓ, $0 \leq \ell < k$.
- If $\ell = 0$, guess a string x of length n such that $f_1^x > m$. If so output x and halt.
- Otherwise nondeterministically guess an a set $S \subset \{0,1\}^n$ of size 2^{lm-m} such that for every x in S, $f_\ell^x < 2\ell m - m$ and $f_{\ell+1}^x \geq 2\ell m + m$. Let z be the concatenation of all of the strings of S and output z and halt. Note the length of z is bounded by a polynomial in n.
- If no such string or set is found then halt and reject.

We need to show that M^f always has an output path and that every possible output is a string with $C(x|w, |x|) \geq m$.

If M^f does not have any output paths then there are at most a polynomial number of strings x of length n with $f_k^x \geq 2km - m$. But this contradicts the fact that at least 2^{n-1} strings must have $f_k^x \geq C_{space}(x \mid w, |x|) \geq n - 1 > 2km - m$ for large n.

If M^f accepts with $\ell = 0$ we have $C_{space}(x \mid w, |x|) \geq f_1^x \geq m$.

Suppose M^f accepts with $\ell > 1$ and $C_{space}(z \mid w, |z|) < m$. Since there are not enough small programs, one string x in S must have $C_{space}(x \mid w, |x|) > 2\ell m - m = f_\ell^x$ and thus $C_{space}(x \mid w, |x|) \geq f_{\ell+1}^x \geq 2\ell m + m$. We can describe m by z and the index of x in z, a total of $2\ell m + O(1)$ bits, a contradiction for sufficiently large m. $\square$

We can try to extend the above results by

1. Strengthening the reduction type in Theorem 7.2 to deterministic polynomial time
2. Weakening 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 the proof of Theorem 7.2 by Buhrman and Torenvliet [7] depends on IP = PSPACE and thus does not relativize.

Theorem 7.4 *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 $\text{PSPACE}^A \neq \text{P}^A$ 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 $\text{PSPACE}^A \neq \text{P}^A$. $\square$

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. Ravi Kumar provided reference [10] and Paul Vitányi reference [33].

References

[1] Andris Ambainis, Harry Buhrman, William Gasarch, Bela Kalyanasundaram and Leen Torenvliet. The communication complexity of enumeration, elimination and selection. In *Proceedings 15th IEEE Conference 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. Bārzdiņš. Complexity of programs to determine 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, Harry Buhrman, Peter A. Fejer, Lance Fortnow, Piotr Grabowski, Luc Longpré, Andrej A. Muchnik, Frank Stephan and Leen Torenvliet. Enumerations of the Kolmogorov function. *Electronic Colloquium on Computational Complexity* (*ECCC*), (015), 2004.

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

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

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

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

[10] Ran Canetti. More on BPP and the polynomial-time hierarchy. *Information Processing Letters*, 57(5):237–241, 1996.

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

[12] Gregory Chaitin. Information-theoretical characterizations of recursive infinite strings. *Theoretical Computer Science*, 2:45–48, 1976.

[13] Rod Downey, Denis Hirschfeldt, André Nies and Frank Stephan. Trivial reals. In *Proceedings of the 7th and 8th Asian Logic Conferences*, pages 103–131. World Scientific, River Edge, 2003.

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

[15] 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.

[16] 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.

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

[18] 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.

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

[20] Antonín Kučera and Sebastiaan A. Terwijn. Lowness for the class of random sets. *The Journal of Symbolic Logic*, 64:1396–1402, 1999.

[21] 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.

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

[23] 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.

[24] 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.

[25] André Nies. Lowness properties of reals and randomness. *Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland*, CDMTCS 201, 2002. *Advances in Mathematics*, to appear.

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

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

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

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

[30] Robert Solovay. Draft of a paper on Chaitin's work. Manuscript, 215 pages, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1975.

[31] Sebastiaan A. Terwijn and Domenico Zambella. Algorithmic randomness and lowness. *The Journal of Symbolic Logic*, 66:1199–1205, 2001.

[32] Domenico Zambella. On sequences with simple initial segments. Technical Report ILLC Technical Report ML 1990-05, University of Amsterdam, 1990.

[33] 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.