George Barmpalias, Andrew Lewis-Pye

Optimal redundancy in computations from random oracles

Article (Accepted version)
(Refereed)

Original citation:
Barmpalias, George and Lewis-Pye, Andrew (2017) *Optimal redundancy in computations from random oracles.* Journal of Computer and System Sciences. ISSN 0022-0000

DOI: 10.1016/j.jcss.2017.06.009

Optimal redundancy in computations from random oracles *

George Barmpalias Andrew Lewis-Pye

June 13, 2017

Abstract. It is a classic result in algorithmic information theory that every infinite binary sequence is computable from an infinite binary sequence which is random in the sense of Martin-Löf. Proved independently by Kučera [Kuč85] and Gács [Gác86], this result answered a question by Charles Bennett and has seen numerous applications in the last 30 years. The optimal redundancy in such a coding process has, however, remained unknown. If the computation of the first n bits of a sequence requires $n + g(n)$ bits of the random oracle, then g is the *redundancy* of the computation. Kučera implicitly achieved redundancy $n \log n$ while Gács used a more elaborate block-coding procedure which achieved redundancy $\sqrt{n} \log n$. Merkle and Mihailović [MM04] provided a different presentation of Gács' approach, without improving his redundancy bound. In this paper we devise a new coding method that achieves optimal logarithmic redundancy. For any computable non-decreasing function g such that $\sum_i 2^{-g(i)}$ is bounded we show that there is a coding process that codes any given infinite binary sequence into a Martin-Löf random infinite binary sequence with redundancy g. This redundancy bound is exponentially smaller than the previous bound of $\sqrt{n} \log n$ and is known to be the best possible by recent work [BLPT16], where it was shown that if $\sum_i 2^{-g(i)}$ diverges then there exists an infinite binary sequence X which cannot be computed by any Martin-Löf random infinite binary sequence with redundancy g. It follows that redundancy $\epsilon \cdot \log n$ in computation from a random oracle is possible for every infinite binary sequence, if and only if $\epsilon > 1$.

George Barmpalias

State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China.
E-mail: barmpalias@gmail.com. *Web:* http://barmpalias.net

Andrew Lewis-Pye

Department of Mathematics, Columbia House, London School of Economics, Houghton Street, London, WC2A 2AE, United Kingdom.
E-mail: A.Lewis7@lse.ac.uk. *Web:* http://aemlewis.co.uk

*Barmpalias was supported by the 1000 Talents Program for Young Scholars from the Chinese Government, grant no. D1101130. Additional support was received by the Chinese Academy of Sciences (CAS) and the Institute of Software of the CAS. Lewis-Pye was supported by a Royal Society University Research Fellowship.

1 Introduction

If an infinite binary sequence is algorithmically random, one does not expect to be able to extract any useful information from it. Although this reasonable intuition can be verified in many formal contexts[1], it fails for the most accepted and robust notion of algorithmic randomness, which is Martin-Löf randomness [ML66], also formulated as incompressibility in terms of Kolmogorov complexity by Chaitin [Cha75] and Levin [Lev73]. Indeed, Kučera [Kuč85], and independently Gács [Gác86], showed that any infinite binary sequence is computable from a Martin-Löf random sequence. Both authors constructed a uniform process that codes every infinite binary sequence into some Martin-Löf random infinite binary sequence. The Kučera-Gács theorem, as it is known in algorithmic information theory, has been studied and extended in numerous ways in the last 30 years[2] and has become a standard prominent topic in most textbooks and presentations of this area.[3] In the context of Martin-Löf randomness, this result says that:

> Any type of information that can be coded into an infinite binary sequence, no matter how structured that might be, can be obfuscated into an algorithmically random infinite binary sequence, from which it is effectively recoverable.

Here *information* could be the solution to a problem of interest, such as the halting problem, the word problem for finite groups, or any of the numerous and often algorithmically unsolvable problems whose solutions can be represented as a set of integers. *Effectively recoverable* means computable by means of a Turing reduction, without any restrictions on time or memory. As we discuss below, however, the coding constructed in both [Kuč85] and [Gác86] gives a Turing reduction with a computable upper bound on the length of the initial segment of the oracle that is used in the computation on any given argument—the *oracle use*.[4]

It is hardly surprising that such a coding process occasionally introduces an overhead on the codes of the initial segments of certain infinite binary sequences. More specifically, if we code an infinite binary sequence X into a Martin-Löf random infinite binary sequence Y, then it is very possible that for some n, in order to recover the first n bits of X (denoted $X\upharpoonright_n$), we need $Y\upharpoonright_{n+g(n)}$, i.e. $g(n)$ more bits of Y. Such a function g that bounds from above the number of extra bits needed in the decoding process is known as the *redundancy* in the computation of X from Y. For example, it is known from [BLPT16] that certain infinite binary sequences X are not computable from any Martin-Löf random infinite binary sequence with redundancy $\log n$. Such restrictions can be intuitively understood if one considers that information introduces structure, and in order to obfuscate the structure of a given X into a random Y, extra bits amplifying the complexity of the code might be necessary.

In the context of information theory, it is important to:

---

[1] If an infinite binary sequence is *arithmetically random* in the sense that it avoids all null sets of reals which are arithmetically definable, then it cannot compute any noncomputable infinite binary sequence that is definable in arithmetic. Similarly, if an infinite binary sequence is Martin-Löf random relative to the halting problem, then it cannot compute any noncomputable infinite binary sequence which is definable in arithmetic with one unbounded quantifier. A less trivial example is a fact from Stephan [Ste06] that incomplete Martin-Löf random infinite binary sequences cannot compute any complete extensions of Peano Arithmetic, and its extensions in Levin [Lev02, Lev13].

[2] For example see [Kuč89, Boo94, Her97, MM04, Dot06, DM06, BDN11].

[3] For example consider the standard textbooks [LV97, Cal94, Nie09, DH10] and the surveys [DHNT06, MN06].

[4] In the terminology of computability theory, every infinite binary sequence is weak-truth-table computable from a Martin-Löf random infinite binary sequence. Bennett [Ben88] observed that this is no longer true for truth-table computations, and used this fact in order to define *logical depth* for infinite binary sequences. Other refined reducibilities were considered by Book [Boo94].

(a) Determine the optimal redundancy in coding into Martin-Löf random infinite binary sequences;

(b) Construct a coding process that achieves the optimal redundancy.

The original work in [Kuč85, Gác86] did not achieve these goals, nor did the subsequent work of Merkle and Mihailović [MM04] and Doty [Dot06]. The goal of the present work is to give a definitive answer to challenges (a) and (b).

1.1 Previous, directly relevant work

Kučera [Kuč85] did not show an interest in optimising the redundancy of his coding, other than observing that it can be computably bounded. An examination of his argument (see the survey [BLP17] for such a discussion) shows that his method produces redundancy $n \log n$. Gács [Gác86], on the other hand, has a clear interest in minimising the redundancy of his coding, which he bounds by $\sqrt{n} \log n$ by means of a more sophisticated block-coding, with carefully chosen block-lengths.[5] Merkle and Mihailović [MM04] give an interpretation of Gács' coding in terms of effective martingales, instead of the effective closed sets approach employed in the original argument. Although the latter analysis is rather elegant and geared toward obtaining a small redundancy $\mathbf{o}(n)$, the resulting upper bound is identical with Gács' bound of $\sqrt{n} \log n$.

Doty [Dot06] showed how to reduce the oracle-use when coding an infinite binary sequence X into a Martin-Löf random Y, based on suitable bounds on the constructive dimension of X. Partially extending previous work by Ryabko [Rya86], he showed that the asymptotic ratio between the optimal oracle-use in computing $X \upharpoonright_n$ and n is directly related to the constructive dimension of X. Unfortunately, the arguments developed in this latter work do not shed light on our main goal, which asks for the *actual* optimal redundancy in coding into Martin-Löf random infinite binary sequences, and not the mere asymptotic behavior of the oracle-use in such a reduction. For infinite binary sequences X of dimension 1, for example, the work in [Dot06] merely shows that they can be computed by a Martin-Löf random infinite binary sequence Y with oracle-use ℓ_n such that $\liminf_n(\ell_n/n) = 1$. From the latter we cannot even deduce Gács upper bound $n + \sqrt{n} \log n$ on the oracle-use.

1.2 Our results

Our main contribution is a coding process that codes an arbitrary infinite binary sequence into a Martin-Löf random infinite binary sequence, with optimal redundancy which is exponentially smaller[6] than the previous known bound of $\sqrt{n} \log n$. In the following statement, 'uniformly computable' means that there is a single coding process that works for all infinite binary sequences, i.e. a single Turing functional that provides the promised reduction of each given infinite binary sequence to some Martin-Löf random infinite binary sequence.

Theorem 1.1. *If (ℓ_i) is a computable increasing function such that $\sum_i 2^{-\ell_i+i} < 1$ then every infinite binary sequence is uniformly computable from a Martin-Löf random infinite binary sequence with oracle-use (ℓ_i).*

[5] In actual fact, Gács [Gác86] achieves redundancy $3\sqrt{n} \log n$, but a careful examination of his argument (as this is discussed in the survey [BLP17]) shows that it can be reduced to $\sqrt{n} \log n$.

[6] When f and g are unbounded, we say g is exponentially smaller than f if there exists a constant c such that $2^{c \cdot g(n)} < f(n)$ for all n.

If we require only that $\sum_i 2^{-\ell_i+i}$ is bounded in the statement of Theorem 1.1, then of course the same conclusion will hold so long as either: (a) we allow oracle use $(\ell_i)+c$ for some constant c, or (b) we drop the requirement of uniformity. In fact, Theorem 1.1 is part of the following slightly more general fact that we prove, regarding coding into effectively closed sets of positive measure.

Lemma 1.2. *Let (ℓ_i) be an increasing computable sequence and let $\mathcal{P}$ be a Π^0_1 class. If $\sum_i 2^{-\ell_i+i} < \mu(\mathcal{P})$ then every infinite binary sequence is uniformly computable from some member of $\mathcal{P}$ with oracle-use (ℓ_i).*

Note that Theorem 1.1 follows directly from Lemma 1.2 since the class of Martin-Löf random infinite binary sequences is a Σ^0_2 set of measure 1. We are also able to establish the optimality of these two results. In [BLPT16] it was shown that if the sum in Theorem 1.1 is not bounded, then there exists an infinite binary sequence which is not computable by any Martin-Löf random infinite binary sequence with oracle-use (ℓ_i). If we combine this with Theorem 1.1 we get the following characterization.

Corollary 1.3. *Let g be a nondecreasing computable function. Then the following are equivalent:*

(i) every infinite binary sequence is computable from a Martin-Löf random infinite binary sequence with redundancy g;

(ii) $\sum_i 2^{-g(i)} < \infty$.

Note that in clause (i) for Corollary 1.3 we can replace 'computable' with 'uniformly computable' so long as the sum in (ii) is strictly bounded by 1.

1.3 Terminology, methodology and novelty

1.3.1 Terminology

The Cantor space 2^ω is the class of all infinite binary sequences. Unless explicitly stated otherwise, it is to be assumed that binary strings are finite, so that strings are finite objects. We let $|\sigma|$ denote the length of the string σ. If Q is a computably enumerable set of binary strings, we let $[\![Q]\!]$ denote the class of infinite binary sequences which are prefixed by some string in Q. If $Q = \{\nu\}$, then we simply write $[\![\nu]\!]$ for $[\![Q]\!]$. In this way Σ^0_1 subsets of 2^ω can be represented by c.e. sets of strings Q. The Lebesgue measure of $[\![Q]\!]$ may be denoted simply by $\mu(Q)$. A tree T in the Cantor space is a downward closed set of binary strings with respect to the prefix relation. A branch of T is simply a string in T and a path through T is an infinite binary sequence for which all finite initial segments are branches of T. A Π^0_1 class in the Cantor space can be represented as a Π^0_1 tree or as $2^\omega - [\![Q]\!]$ for some c.e. set of strings Q. The *n-th level* of T consists of the strings in T of length n. A *leaf* of a tree is a branch of the tree with no proper extensions in the tree. For any set of strings T, we let $[T]$ denote the set of all infinite binary sequences with infinitely many prefixes in T. Note that when T is a tree, $[T]$ denotes the set of infinite paths through T.

Now suppose we are given the Π^0_1 class $\mathcal{P}$ and the increasing computable sequence (ℓ_i) of Lemma 1.2. Our task is to construct a Turing functional Φ with uniform oracle-use (ℓ_i) on all oracles, with the property that for every infinite binary sequence X there exists some $Y_X \in \mathcal{P}$ such that $X = \Phi^{Y_X}$. It is convenient to define Φ by assigning labels for strings of each length n to strings of each length ℓ_n. If σ is of length n, τ is of length ℓ_n and we assign the label x_σ to τ, then this is equivalent to defining $\Phi^\tau = \sigma$. Of course these assignments need to be consistent, in the sense that if $\tau \subseteq \tau'$, $\Phi^\tau = \sigma$ and $\Phi^{\tau'} = \sigma'$ then $\sigma \subseteq \sigma'$. In our analysis we thus present Φ as a *partially labelled tree*, by which we mean the full binary tree $2^{<\omega}$ along with a partial labelling of it. Given a partially labelled tree $\mathcal{T}$ and $\ell \in \mathbb{N}$, we let $\mathcal{T} \upharpoonright_\ell$ denote the restriction of $\mathcal{T}$ to the strings of length at most ℓ.

1.3.2 The departure from existing approaches

There are a number of different presentations of the Kučera-Gács theorem in the literature. Kučera [Kuč85] uses the recursion theorem and the universality properties of the class of Martin-Löf random infinite binary sequences. His coding method may be seen as being of the following inductive form. Working within a Π^0_1 class of Martin-Löf randoms $\mathcal{P}$, which is the set of all infinite paths through the computable tree $\mathcal{T}$, let us suppose that we have already determined 2^n strings of length ℓ_n in $\mathcal{T}$ which are extendable (i.e. have infinite extensions in $\mathcal{P}$), such that for each string σ of length n there is precisely one of these extendable strings τ for which we have defined $\Phi^\tau = \sigma$. From properties of the class $\mathcal{P}$, we are then able to determine a length ℓ_{n+1} such that each of these 2^n strings τ must have at least two incompatible and extendable extensions in $\mathcal{T}$ of length ℓ_{n+1}. If $\Phi^\tau = \sigma$, then for two of these extendable extensions τ' and τ'' of length ℓ_{n+1}, we can define $\Phi^{\tau'} = \sigma * 0$ and $\Phi^{\tau''} = \sigma * 1$. The coding may therefore be thought of as occurring *bit-by-bit*, and actually takes place inside a subclass $\mathcal{P}' \subseteq \mathcal{P}$ defined by the tree $\mathcal{T}'$ with the property that for all n:

$$\text{Every branch of } \mathcal{T}' \text{ at level } \ell_n \text{ has at least two extensions at level } \ell_{n+1} \text{ in } \mathcal{T}'. \tag{1}$$

As we proceed to code X, the manner in which we code $\sigma * i \subset X$ may also be seen to satisfy a strong *independence* property: our code for the initial segment of X which is $\sigma * i$ depends only on $\mathcal{P}$, i, and the code for σ (and not, for example, on $X(n)$ for $n > |\sigma * i|$).

In Gács' approach, he does not code bit-by-bit, but rather breaks the infinite binary sequences to be coded into finite blocks of appropriately chosen lengths, and then codes each block rather than each bit one at a time. Coding in blocks in this way allows for a substantial reduction in the redundancy. Nevertheless, it is easily seen that weaker versions of the independence property and condition (1) still hold. If the $(n + 1)$st block is of length m_n then (1) will hold with two replaced by 2^{m_n}. Similarly, the way in which we code the $(n+1)$st block will depend only on $\mathcal{P}$ and the coding of previous blocks. In order to achieve an exponentially smaller redundancy bound with our coding, we shall need to develop more general techniques, for which neither of these strong restrictions apply.

1.4 Background and organization

We assume a basic working knowledge of computability theory and its main concepts. Other than that, the proof of Lemma 1.2 is self-contained. In particular, knowledge of previous proofs of the Kučera-Gács theorem is not assumed. The reader who is interested in a more detailed analysis of the different approaches to the task of coding into random infinite binary sequences, is referred to the recent survey [BLP17]. For background on Martin-Löf randomness we refer to the textbooks Li and Vitanyi [LV97], Downey and Hirschfeldt [DH10] or Nies [Nie09]. The latter two books also contain background in computability theory.

As we discussed in Section 1.3.1, the promised reduction of Lemma 1.2 will be achieved by means of a labelling of the full binary tree. Section 2 is devoted to the construction of this labelling and the statement of its key properties. In Section 3 and Section 4 we verify the properties of the labelling construction and complete the proof of Lemma 1.2.

2 Partial labelling of the full binary tree

The reduction needed for the proof of Lemma 1.2 is constructed via the enumeration of a partially labelled tree $\mathcal{T}$ with certain properties, which we construct in this section. Recall that we are given a Π^0_1 class $\mathcal{P}$ and an increasing computable sequence (ℓ_i) such that:

$$\sum_i 2^{-\ell_i+i} < \mu(\mathcal{P}). \tag{2}$$

The partially labelled tree $\mathcal{T}$ will be determined as the limit of a computable sequence $(\mathcal{T}_s)$ of partially labelled trees. We call $(\mathcal{T}_s)$ a *labelling process* for $\mathcal{T}$. Let Q be a c.e. set of binary strings such that $\mathcal{P} = 2^\omega - [\![Q]\!]$. Let (Q_s) be a computable enumeration of Q. Before we give the construction of $(\mathcal{T}_s)$, we state a number of key properties that $(\mathcal{T}_s)$ will have and define some relevant notions.

2.1 Basic properties of the labelling

The partially labelled tree $\mathcal{T}$ that we construct will be structured in the following sense.

Definition 2.1 (Structured partially labelled trees)**.** *A partially labelled tree $\mathcal{T}$ is structured with respect to an increasing sequence (ℓ_i), if the following properties are met.*

(1) Restriction: *only strings at levels $\ell_i, i \in \mathbb{N}$ of $\mathcal{T}$ can have a label;*

(2) Layering: *the labels placed on the level ℓ_i of $\mathcal{T}$ are of the type x_σ where $|\sigma| = i$;*

(3) Completeness: *if label x_σ exists in $\mathcal{T}$ then all labels x_ρ, $\rho \in 2^{\leq|\sigma|}$ exist in $\mathcal{T}$;*

(4) Uniqueness: *each string in $\mathcal{T}$ can have at most one label;*

(5) Consistency: *if ρ of level ℓ_k in $\mathcal{T}$ has label x_σ then for each $i < k$, $\rho\upharpoonright_{\ell_i}$ has label $x_{\sigma\upharpoonright_i}$.*

The tree $\mathcal{T}$ will be determined as the limit of a computable labelling process $(\mathcal{T}_s)$ which is canonical with respect to the given (ℓ_i), in the following sense.

Definition 2.2 (Canonical labelling process)**.** *A labelling process $(\mathcal{T}_s)$ is canonical with respect to an increasing sequence (ℓ_i) if the following properties hold for all s.*

(1) Structure: *the tree $\mathcal{T}_s$ is structured with respect to (ℓ_i) ;*

(2) Finiteness: *only strings of length at most ℓ_s can have a label in $\mathcal{T}_s$;*

(3) Persistence: *if ρ has label x_σ in $\mathcal{T}_s$, then it has the same label in $\mathcal{T}_t$ for all $t > s$.*

Clearly a canonical labelling process $(\mathcal{T}_s)$ has a limit, which is a structured partially labelled tree. From now on we suppress the qualification 'with respect to an increasing sequence (ℓ_i)' when we use the notions of Definitions 2.1 and 2.2, and always assume the fixed sequence (ℓ_i) that is given in Lemma 1.2.

Note that Definition 2.1 and Definition 2.2 allow the possibility that a single label x_σ may have many copies at some level ℓ_k of some $\mathcal{T}_s$.

2.2 Definitions for the labelling construction

The following notation will be useful.

Definition 2.3 (Labelled subset and size)**.** *Given a structured partially labelled tree $\mathcal{T}$, let $\mathcal{T}^*$ denote the set which includes the empty string and all labelled strings in $\mathcal{T}$. The length of the longest σ such that a label x_σ has been placed on a string in $\mathcal{T}^*$ is denoted $\|\mathcal{T}^*\|$.*

The purpose of the labelling process is to ensure that for every string σ there is eventually a string ρ in $\mathcal{T}$ which is extendible in $\mathcal{P}$ and which has label x_σ. In this sense, the enumeration (Q_s) of Q is the main driver of the process, and determines the placement of additional copies of already existing labels. Timing is a crucial aspect of the labelling process, however, and for this reason we will not use the arbitrary enumeration (Q_s) directly in the construction. We use the following filtered version instead, which takes into account the existing labelling at each stage. Here and in the following discussions, a leaf of $\mathcal{T}_s^*$ is a string in $\mathcal{T}_s^*$ which does not have any labelled proper extensions.

Definition 2.4 (Filtered enumeration of Q)**.** *During the construction we define a c.e. set of strings $\mathcal{D}$ inductively. $\mathcal{D}_s$ denotes the set of strings enumerated into $\mathcal{D}$ by the end of stage s.*

- *At stage 0 let $\mathcal{D}_0 = \emptyset$;*
- *At stage $s+1$, if there exists a leaf of $\mathcal{T}_s^*$ which does not belong in $\mathcal{D}_s$ and has a prefix in Q_s, pick the lexicographically least such leaf and enumerate it into $\mathcal{D}$.*

Clearly $[\![\mathcal{D}_s]\!] \subseteq [\![Q_s]\!]$ while the converse is not generally true. Note that for a string ρ to enter $\mathcal{D}$ at stage $s+1$ it is not enough to have a prefix in Q_s. Hence $(\mathcal{D}_s)$ is a filtered version of (Q_s), in the sense that only *previously* labelled strings can be enumerated into $\mathcal{D}_s$.

As remarked previously, Definition 2.2 crucially allows for the possibility that a single label x_σ may have many copies at some level ℓ_n of some $\mathcal{T}_s$. Amongst all of the strings with the same label x_σ, however, we shall ensure that at any given time there is precisely one of these strings which is given the special status of being *active*. Roughly speaking, the active strings are those above which it presently seems there is still room for further coding at the next level. If $|\sigma| = n$ and the label x_σ is placed on τ, then while τ is active we may place labels for one element extensions of σ on the extensions of τ of level ℓ_{n+1}. We shall do so as the demands of the construction require, working from left to right. As each of these labels are placed, we do not have to be concerned initially as to whether they are placed on strings with prefixes in Q – we simply place the labels and then wait for the enumeration of $\mathcal{D}$ to subsequently alert us if we have placed labels on strings which do not have extensions in $\mathcal{P}$. Once labels have been placed on all extensions of τ of level ℓ_{n+1}, τ is said to be *saturated*. It should be noted that at any given point, if $\sigma \subset \sigma'$, τ and τ' have labels x_σ, $x_{\sigma'}$ respectively and are both active, it will not necessarily hold that $\tau \subset \tau'$. We make the following definitions.

Definition 2.5 (Active strings)**.** *Given a canonical labelling process $(\mathcal{T}_s)$, a string ρ in $\mathcal{T}_s^*$ is* active *if it has some label x_σ and ρ was the last string to receive this label in the approximations $\mathcal{T}_0, \dots, \mathcal{T}_s$.*

A string in $\mathcal{T}_s^*$ that is not active is called *inactive*.

Definition 2.6 (Saturated strings)**.** *A string ρ of level ℓ_k of a structured partially labelled tree $\mathcal{T}$ is saturated if all of its extensions at level ℓ_{k+1} of $\mathcal{T}$ are labelled.*

Note that if $(\mathcal{T}_s)$ is a canonical labelling process and a string of $\mathcal{T}_s$ is saturated, then the same string will also be saturated in $\mathcal{T}_t$ for all $t > s$. Similarly, by Definition 2.5, if a string in $\mathcal{T}_s^*$ is inactive then the same string will also be inactive in $\mathcal{T}_t^*$ for all $t > s$.

Each stage of the construction of $(\mathcal{T}_s)$ after stage 0, will be one of the following two kinds.

Definition 2.7 (Expansionary and adaptive stages). *A stage $s+1$ is called* expansionary *if $\|\mathcal{T}_{s+1}^*\| > \|\mathcal{T}_s^*\|$. Otherwise $s+1$ is called an* adaptive *stage.*

It will be immediate from the construction that $s+1$ is expansionary if and only if $\mathcal{D}_{s+1} = \mathcal{D}_s$.

In the labelling construction we will explicitly deactivate strings in order to emphasize the newly inactive strings. It will be evident that this is compatible with Definition 2.5.

Definition 2.8 (Cloning a branch). *Given $\delta, \beta \in \mathcal{T}_s^*$ such that δ is a leaf, suppose that:*

(i) if x_σ, x_τ are the labels of β, δ respectively then $\sigma \subset \tau$;

(ii) η is the leftmost string of length $|\delta|$ which extends β and $\eta \upharpoonright_{\ell_{k+1}}$ is not labelled.

Cloning δ above β *means to label $\eta \upharpoonright_{\ell_i}$ with the label of $\delta \upharpoonright_{\ell_i}$, for each i such that $\ell_i \in (|\beta|, |\delta|]$, making each of these strings active.*

In Definition 2.8, we allow the case that β is the empty string λ, in which case there is no label placed on β. Given a labelled string ρ in $\mathcal{T}_s$, the *active clone* of ρ in $\mathcal{T}_s$ is the unique active string in $\mathcal{T}_s$ which has the same label as ρ. Note that the active clone of an active string is the string itself. For uniformity, we define the active clone of the empty string λ to be λ.

2.3 The labelling construction

At stage 0 we place a label x_λ on the leftmost string of length ℓ_0 and make this string active. At stage $s+1$ suppose that the labelled tree $\mathcal{T}_s$ has been defined, and consider the following two cases:

Expansionary stage: If $\mathcal{D}_{s+1} = \mathcal{D}_s$ then let $\mathcal{T}_{s+1} \upharpoonright_{\ell_s} = \mathcal{T}_s \upharpoonright_{\ell_s}$ and for each active leaf ρ of $\mathcal{T}_s^*$ with label some x_σ, place labels $x_{\sigma*0}, x_{\sigma*1}$ on the leftmost and rightmost extensions of ρ of level ℓ_{s+1}, making these strings active, then end stage $s+1$.

Adaptive stage: If $\mathcal{D}_{s+1} \neq \mathcal{D}_s$ then let δ be the string in $\mathcal{D}_{s+1} - \mathcal{D}_s$ and let α_j, $j \le k$ be the empty or labelled initial segments of δ in order of magnitude, so that $\alpha_0 = \lambda$ and $\alpha_k = \delta$. Also let β_j, $j \le k$ be the active clones of α_j, $j \le k$ respectively in $\mathcal{T}_s$. Let j_0 be the largest number $j < k$ such that β_j is not saturated and

- deactivate β_j for each $j \in (j_0, k]$;
- clone δ above β_{j_0}.

If such j_0 does not exist, say that the construction *terminates* at stage $s+1$; otherwise end stage $s+1$.

3 Properties of the labelling algorithm

Note that since (ℓ_i) is increasing, each string of length ℓ_k has at least two distinct extensions of length ℓ_{k+1}. Hence the expansionary stages of the construction are well-defined. A straightforward induction on stages suffices to establish that $(\mathcal{T}_s)$ is a canonical labelling process, according to Definition 2.2. In particular, the placing of labels satisfies the consistency condition required in order to define a valid functional. While Definition 2.5 specifies the active strings at each stage, during the construction we have also directly deactivated strings, as well as activating them during the process of cloning and at expansionary stages. It is clear that at any stage the strings which have been activated and not directly deactivated by the construction, are precisely those which are active according to Definition 2.5, since it is precisely when we place a new version of a given label that we deactivate the previously active string with that label. It also follows by a straightforward induction on stages, that at the end of each stage s, any leaf of $\mathcal{T}_s^*$ is either active, or else has already been enumerated into $\mathcal{D}_s$. In particular, when δ is enumerated into $\mathcal{D}_{s+1}$ during stage $s+1$, it was previously active and is deactivated during this adaptive stage. This means, in the notation of Section 2.3, that when we deactivate β_k without requiring that it be saturated, in fact $\beta_k = \delta$, so that the only strings which are deactivated during an adaptive stage are strings which are enumerated into $\mathcal{D}$, or else are saturated:

$$\text{Inactive labelled strings in } \mathcal{T}_s \text{ are either saturated or else belong to } \mathcal{D}_s. \tag{3}$$

The following is also established easily by induction on stages:

$$\text{If } \delta \in \mathcal{D} \text{ then for all } s, \text{ no proper extension of } \delta \text{ is labelled in } \mathcal{T}_s. \tag{4}$$

3.1 Non-termination

In order to show that the labelling construction does not terminate (i.e. that we do not run out of room for coding), it suffices to establish that λ is never saturated (regarding λ as of level ℓ_{-1} in the definition of saturation). The following definition will be useful.

Definition 3.1 (Set of active strings)**.** *Let U_s be the set of active strings in $\mathcal{T}_s$. For each string ρ let $U_s(\rho)$ be the set of strings $\gamma \supseteq \rho$ which are active in $\mathcal{T}_s$.*

We are interested in the weight of the active strings, where the weight of a set of strings V is defined by:

$$\mathtt{wgt}\,(V) = \sum_{\eta \in V} 2^{-|\eta|}.$$

In order to show that λ is never saturated we shall first establish:

$$\mathtt{wgt}\,(U_s) + \mu(\mathcal{D}_s) < 1 \text{ for all stages } s. \tag{5}$$

The following claim will also be established by induction on stages:

$$\begin{array}{l}\text{Given any } s \text{ and any infinite binary sequence } Z \text{ which does not have a prefix in } \mathcal{D}_s,\\ \text{the largest labelled initial segment of } Z \text{ is active in } \mathcal{T}_s.\end{array} \tag{6}$$

Note that in this statement it is possible that Z does not have a labelled initial segment, in which case the assertion is trivially true. An immediate consequence of (6) is that

$$\text{For each } s \text{ and each } \nu \text{ which is labelled in } \mathcal{T}_s, \text{ we have } [\![\nu]\!] \subseteq [\![\mathcal{D}_s]\!] \cup [\![U_s(\nu)]\!]. \tag{7}$$

Now if λ is saturated at stage s then the entire Cantor space is covered by the labelled strings of length ℓ_0. Hence by (7) we have $2^\omega \subseteq [\![\mathcal{D}_s]\!] \cup [\![U_s]\!]$. Then $1 \leq \mu(\mathcal{D}_s) + \mathtt{wgt}\,(U_s)$, which contradicts (5).

It remains to establish (5) and (6). To see (5), note first that at each stage s and for each σ there is at most one active string in $\mathcal{T}_s$ with label x_σ. Since for each n there are only 2^n strings of length n, we have:

$$\texttt{wgt}\,(U_s) \leq \sum_{\rho\in U_s} 2^{-|\rho|} \leq \sum_n \left(\sum_{\eta\in U_s\cap 2^{\ell_n}} 2^{-|\eta|}\right) \leq \sum_n \left(2^n \cdot 2^{-\ell_n}\right) = \sum_n 2^{n-\ell_n}.$$

If we combine this with our hypothesis (2) we get $\texttt{wgt}\,(U_s) < \mu(\mathcal{P}) = 1 - \mu(Q_s)$. By the fact $[\![\mathcal{D}_s]\!] \subseteq [\![Q_s]\!]$ which we observed after Definition 2.4, we get $\texttt{wgt}\,(U_s) < 1 - \mu(\mathcal{D}_s)$, from which (5) follows.

It remains to prove (6) by induction on the stages of the labelling construction. At stage 0 we have $\mathcal{D}_0 = \emptyset$ and all labelled strings are active. It follows that in this case (6) holds. Inductively suppose that (6) holds at stage s. If stage $s+1$ is expansionary, then no string is deactivated, and any new labels are placed on strings that become active. So given an infinite binary sequence Z and the largest initial segment ν of Z that is labelled in $\mathcal{T}_{s+1}$, either the label of ν existed in $\mathcal{T}_s$ or it did not. In the first case we can use the inductive hypothesis to conclude that ν is active in $\mathcal{T}_{s+1}$. In the second case we can conclude the same, due to the fact that newly labelled strings in expansionary stages are active. Hence if $s+1$ is expansionary, (6) continues to holds at stage $s+1$.

Now suppose that $s+1$ is an adaptive stage and let δ be the unique element of $\mathcal{D}_{s+1} - \mathcal{D}_s$. Also let Z be an infinite binary sequence which has at least one labelled initial segment in $\mathcal{T}_s$, and let ν be the largest such initial segment. If $\nu = \delta$ there is nothing to prove, so assume otherwise. If no initial segment of Z is deactivated during stage $s+1$, the claim follows by the induction hypothesis. For the remaining case, let η be the largest labelled prefix of Z which is deactivated during stage $s+1$. If $|\eta| = \ell_k$, let $\eta' = Z\upharpoonright_{\ell_{k+1}}$. Since η was deactivated at $s+1$, it follows from (3) that it was saturated in $\mathcal{T}_s$. This means that η' must be labelled in $\mathcal{T}_s$. Hence the largest labelled initial segment of Z in $\mathcal{T}_s$, which is also the largest in $\mathcal{T}_{s+1}$, is active in $\mathcal{T}_{s+1}$, just as it was active in $\mathcal{T}_s$. Hence (6) holds for Z at stage $s+1$. Finally consider the case where Z did not have a labelled initial segment in $\mathcal{T}_s$, but it does in $\mathcal{T}_{s+1}$. Since all the newly labelled strings at stage $s+1$ are active in $\mathcal{T}_{s+1}$, in this case also we can conclude that (6) holds for Z at stage $s+1$.

This completes the induction step and the proof of (6).

3.2 Growth of the tree and the enumeration of $\mathcal{D}$

Now that we have proved the construction does not terminate, the rest of the verification is essentially routine. Note that at each stage s the set $\mathcal{T}^*_s$ is finite, and that in every adaptive stage some previously unlabelled strings receive labels. Since $(\mathcal{T}_s)$ is a canonical labelling process, it follows from the fact that the labelling construction does not terminate that there are infinitely many expansionary stages. Hence:

$$\lim_s \|\mathcal{T}_s\| = \infty. \tag{8}$$

Recall that $\mathcal{T}^*$ is the limit of all $\mathcal{T}^*_s$. We wish to show that:

$$\text{If } \tau \in \mathcal{T}^* \text{ extends a string in } Q \text{ then there exists } s \text{ with } [\![\tau]\!] \subseteq [\![\mathcal{D}_s]\!]. \tag{9}$$

This will follow once we establish the following fact:

$$\text{If } \tau \in \mathcal{T}^*_s \text{ is active, there is at least one leaf of } \mathcal{T}^*_s \text{ extending } \tau \text{ which is not in } \mathcal{D}_s. \tag{10}$$

In order to see that (9) follows from (10), suppose that s_0 is the least stage at which $\tau \in \mathcal{T}^*_{s_0}$ and there exists $\tau' \subseteq \tau$ with $\tau' \in \mathcal{Q}_{s_0}$. Let s_1 be the least expansionary stage $> s_0$. At the beginning of stage s_1, (10) implies that no string $\tau'' \supseteq \tau$ in $\mathcal{T}^*_{s_1}$ can be active, meaning that all such strings must either be saturated or else belong to $\mathcal{D}_{s_1}$, by (3). This implies that $[\![\tau]\!] \subseteq [\![\mathcal{D}_{s_1}]\!]$ as required.

We establish (10) by induction on stages. If s_0 is the first stage at which τ is active, then no extensions of τ are in $\mathcal{D}_{s_0}$. At any subsequent stage $s > s_0$ at which τ is still active, if a leaf δ extending τ is enumerated into $\mathcal{D}_s$, then that leaf will be cloned above some $\beta_{j_0} \supseteq \tau$, which completes the induction step.

4 The coding process and its verification

In this section we show how to determine the code Y of a given infinite binary sequence X, so that $X \upharpoonright_n$ can be uniformly computed by $Y \upharpoonright_{\ell_n}$. We define this reduction based on the labelling process of Section 2 and its properties. Note that a direct consequence of the labelling construction of Section 2.3 is that every leaf of $\mathcal{T}^*_s$ has the same length.

4.1 The coding process

For any given X consider the downwards closure of (i.e. the set of all initial segments of) strings that are labelled with a prefix of X. Since there are infinitely many expansionary stages, this set is an infinite tree, and so has an infinite path by König's Lemma. We let Y, the code for X, be any such infinite path.

4.2 The coding verification

We verify that the code Y of X determined by the above construction has the required properties.

► ***Y belongs to $\mathcal{P}$***

It suffices to show that Y does not have a prefix in $\mathcal{Q}$. This follows by (9) and (4).

► ***Y computes X with oracle use*** (ℓ_n)

We show that for each n we can compute $X \upharpoonright_n$ uniformly from $Y \upharpoonright_{\ell_n}$. Given n and $Y \upharpoonright_{\ell_n}$ we simply run the labelling construction until the first stage s_0 where the string $Y \upharpoonright_{\ell_n}$ is labelled in $\mathcal{T}_{s_0}$. Note that since $Y \in [\mathcal{T}^*]$ and $\mathcal{T}$ is a structured partially labelled tree, such a stage exists. If x_σ is the label of $Y \upharpoonright_{\ell_n}$ then $X \upharpoonright_n = \sigma$.

This concludes the verification of the coding process and the proof of Lemma 1.2.

References

[BDN11] George Barmpalias, Rod Downey, and Keng Meng Ng. Jump inversions inside effectively closed sets and applications to randomness. *J. Symb. Log.*, 76(2):491–518, 2011.

[Ben88] Charles H. Bennett. Logical depth and physical complexity. In R. Herken, editor, *The Universal Turing machine, a Half Century Survey*, pages 227–257. Oxford U.P., 1988.

[BLP17] George Barmpalias and Andrew Lewis-Pye. Coding into random reals. In *Post-proceedings volume of SEALS 2016 (South Eastern Logic Symposium) February 27, 2016 – February 28, 2016*. World Scientific, 2017.

[BLPT16] George Barmpalias, Andrew Lewis-Pye, and Jason Teutsch. Lower bounds on the redundancy in computations from random oracles via betting strategies with restricted wagers. *Inform. and Comput.*, 251:287–300, 2016.

[Boo94] Ronald V. Book. On languages reducible to algorithmically random languages. *SIAM Journal on Computing*, 23(6):1275–1282, 1994.

[Cal94] Cristian Calude. *Information and Randomness*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 1994. With forewords by Gregory J. Chaitin and Arto Salomaa.

[Cha75] Gregory J. Chaitin. A theory of program size formally identical to information theory. *J. Assoc. Comput. Mach.*, 22:329–340, 1975.

[DH10] Rod G. Downey and Denis Hirshfeldt. *Algorithmic Randomness and Complexity*. Springer, 2010.

[DHNT06] Rod G. Downey, Denis R. Hirschfeldt, André Nies, and Sebastiaan A. Terwijn. Calibrating randomness. *Bull. Symbolic Logic*, 12(3):411–491, 2006.

[DM06] Rod G. Downey and Joseph S. Miller. A basis theorem for Π^0_1 classes of positive measure and jump inversion for random reals. *Proc. Amer. Math. Soc.*, 134(1):283–288 (electronic), 2006.

[Dot06] David Doty. Every sequence is decompressible from a random one. In *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*, pages 153–162, 2006.

[Gác86] Péter Gács. Every sequence is reducible to a random one. *Inform. and Control*, 70(2-3):186–192, 1986.

[Her97] Peter Hertling. Surjective functions on computably growing Cantor sets. *J. UCS*, 3(11):1226–1240, 1997.

[Kuč85] Antonín Kučera. Measure, Π^0_1-classes and complete extensions of PA. In *Recursion theory week (Oberwolfach, 1984)*, volume 1141 of *Lecture Notes in Math.*, pages 245–259. Springer, Berlin, 1985.

[Kuč89] Antonin Kučera. On the use of diagonally nonrecursive functions. In *Logic Colloquium ’87 (Granada, 1987)*, volume 129 of *Stud. Logic Found. Math.*, pages 219–239. North-Holland, Amsterdam, 1989.

[Lev73] Leonid A. Levin. The concept of a random sequence. *Dokl. Akad. Nauk SSSR*, 212:548–550, 1973.

[Lev02] Leonid A. Levin. Forbidden information. In *FOCS*, page 761. IEEE Computer Society, 2002.

[Lev13] Leonid A. Levin. Forbidden information. *J. ACM*, 60(2):9:1–9:9, May 2013.

[LV97] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Graduate Texts in Computer Science. Springer-Verlag, New York, second edition, 1997.

[ML66] Per Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.

[MM04] Wolfgang Merkle and Nenad Mihailović. On the construction of effectively random sets. *J. Symb. Log.*, 69(3):862–878, 2004.

[MN06] Joseph S. Miller and André Nies. Randomness and computability: open questions. *Bull. Symbolic Logic*, 12(3):390–410, 2006.

[Nie09] André Nies. *Computability and Randomness*. Oxford University Press, 2009.

[Rya86] Boris Ya. Ryabko. Noiseless coding of combinatorial sources, Hausdorff dimension, and Kolmogorov complexity. *Problems Inform. Transmission*, 22:170–179, 1986.

[Ste06] Frank Stephan. Martin-Löf random and PA-complete sets. In *Logic Colloquium '02*, volume 27 of *Lect. Notes Log.*, pages 342–348. Assoc. Symbol. Logic, La Jolla, CA, 2006.