

# Expanded and improved proof of the relation between description complexity and algorithmic probability

Peter Gács  
Boston University

November 30, 2008

## Abstract

This manuscript defines monotonic description complexity and algorithmic probability, and then gives a proof that the former is not within an additive constant of the logarithm of the latter. It is a more detailed exposition of the proof given in the original paper, and also incorporates Adam Day's ideas that led to an improved lower bound for the binary case.

## 1 Introduction

This manuscript defines monotonic description complexity and algorithmic probability, and then gives a proof that the former is not within an additive constant of the logarithm of the latter. It is a more detailed exposition of the proof given in the paper [3], and also incorporates Adam Day's ideas from [2] that led to an improved lower bound for the binary case. On the other hand, the reader is sent to the original paper and its references for the motivation of the chosen definitions of complexity and algorithmic probability.

**Notation 1.1.** All logarithms and exponentials in this paper are to the base 2. Probabilities will be denoted by  $\mathbf{P}[\cdot]$ .

Let  $\mathbb{Q}$  denote the set of rational numbers, let  $\mathbb{N}$  be the set of natural numbers, and  $\mathbb{B} = \{0, 1\}$ . Let  $\Lambda$  denote the empty string. For a set  $A$  let  $A^n$  denote the set of strings  $(x(1), \dots, x(n))$  with elements in  $A$ ,  $A^0 = \{\Lambda\}$ ,  $A^* = \bigcup_{i=0}^{\infty} A^i$ . Let  $l(x)$  denote the length of a finite or infinite sequence  $x$ , and let  $x(\leq n) = (x(1), \dots, x(n))$ . Normally, we will write the *concatenation* of two strings  $x, y$  just as  $xy$ . If this leads to misunderstanding we will write  $x \cdot y$ . Similarly, if  $x \in A^*$  and  $a \in A$  then  $xa$  will denote the string obtained by appending  $a$  to  $x$ .

For strings  $\mathbf{x} \in \mathbb{N}^*$ ,  $\mathbf{y} \in \mathbb{N}^* \cup \mathbb{N}^{\mathbb{N}}$ , we write  $\mathbf{x} \sqsubseteq \mathbf{y}$  to denote that  $\mathbf{x}$  is a prefix of  $\mathbf{y}$ . For  $(\mathbf{p}_i, \mathbf{x}_i) \in \mathbb{N}^* \times \mathbb{N}^*$  let  $(\mathbf{p}_1, \mathbf{x}_1) \sqsubseteq (\mathbf{p}_2, \mathbf{x}_2)$  mean  $\mathbf{p}_1 \sqsubseteq \mathbf{p}_2$  and  $\mathbf{x}_1 \sqsubseteq \mathbf{x}_2$ . Two objects  $\mathbf{x}, \mathbf{y}$  are

*compatible* if there is a  $\mathbf{z}$  with  $\mathbf{x} \sqsubseteq \mathbf{z}$ ,  $\mathbf{y} \sqsubseteq \mathbf{z}$ . If the elements of a set  $A \subset \mathbb{N}^*$  are pairwise incompatible, this set will also be called *prefix-free*.

We say that two sets  $A, B \subseteq \mathbb{N}^*$  are mutually incompatible if every pair  $\mathbf{a} \in A$ ,  $\mathbf{b} \in B$  is.

For string  $\mathbf{x} \in \mathbb{N}^*$  let  $\mathbf{x}\mathbb{N}^{\mathbb{N}} = \{\eta \in \mathbb{N}^{\mathbb{N}} : \mathbf{x} \sqsubseteq \eta\}$ . For a set  $A \subseteq \mathbb{N}^*$  let  $A\mathbb{N}^{\mathbb{N}} = \bigcup_{\mathbf{a} \in A} \mathbf{a}\mathbb{N}^{\mathbb{N}}$ .

For an infinite binary sequence  $\boldsymbol{\pi} = (p(1), p(2), \dots) \in \mathbb{B}^{\mathbb{N}}$  let  $[\boldsymbol{\pi}]$  denote the real number  $0.p(1)p(2)\dots$  in the interval  $[0, 1]$  in binary notation. (The function  $[\cdot]$  is not a one-to-one correspondence between real numbers and infinite strings, since binary rational numbers have two different representation. But this need not concern us.) For  $\mathbf{p} \in \mathbb{B}^*$  let  $[\mathbf{p}] = \{[\eta] : \eta \in \mathbf{p}\mathbb{B}^{\mathbb{N}}\}$ . For example,  $[101]$  is the interval  $[5/8, 6/8]$ . For  $A \subseteq \mathbb{B}^*$  let  $[A] = \bigcup_{\mathbf{p} \in A} [\mathbf{p}]$ . Note that  $A, B$  are mutually incompatible if and only if  $[A]$  and  $[B]$  are disjoint.

For a set  $E \subseteq \mathbb{N}^*$ , we define its *bottom* as

$$E' = \{\mathbf{x} \in E : \forall \mathbf{y} \in E \mathbf{y} \sqsubseteq \mathbf{x} \Rightarrow \mathbf{y} = \mathbf{x}\}.$$

Note that  $E'$  is prefix-free and still  $E'\mathbb{N}^{\mathbb{N}} = E\mathbb{N}^{\mathbb{N}}$ .

For two sets  $E, F \subseteq \mathbb{N}^*$  we will write  $E \leq F$  if for all  $\mathbf{x} \in E$  there is a  $\mathbf{y} \in F$  with  $\mathbf{y} \sqsubseteq \mathbf{x}$ . The relation  $E \leq F$  implies  $E\mathbb{N}^{\mathbb{N}} \subseteq F\mathbb{N}^{\mathbb{N}}$ , but the converse is not true.

Let  $[\![\cdot]\!] : \mathbb{N}^* \rightarrow \mathbb{N}$  be some standard one-to-one encoding function with partial inverses  $[\![\cdot]\!]_i$ . For example if  $z = [\![x, y]\!]$  then  $y = [\![z]\!]_2$ .

The relations  $f \overset{*}{<} g$ ,  $f \overset{+}{<} g$  denote  $f = O(g)$ , and  $f = g + O(1)$  respectively. We write  $f \overset{\pm}{=} g$  if both  $f \overset{+}{<} g$  and  $f \overset{*}{>} g$  holds. ┘

The following proposition is immediate from the definitions.

**Proposition 1.1.** *For binary strings  $\mathbf{p}, \mathbf{q}$  we have  $\mathbf{p} \sqsubseteq \mathbf{q}$  if and only if  $[\mathbf{p}] \supseteq [\mathbf{q}]$ . And  $\mathbf{p}, \mathbf{q}$  are incompatible if and only if  $[\mathbf{p}]$  and  $[\mathbf{q}]$  are disjoint.*

Let us define the description complexity  $K(\mathbf{y} | \mathbf{x})$  of a string  $\mathbf{y} \in \mathbb{N}^*$ , conditional on a string  $\mathbf{x} \in \mathbb{N}^*$  as the length of the shortest program  $\mathbf{p}$  telling a certain universal Turing machine  $T$  to output  $\mathbf{y}$ , when  $\mathbf{x}$  is also given as argument. For technical reasons of the characterization of random sequences, it is advantageous to require the output  $\mathbf{y}$  of machine in question to depend on the program  $\mathbf{p}$  in a monotonic way: an extension of  $\mathbf{p}$  should give an extension of  $\mathbf{y}$ . A somewhat more general version of the monotonic machine is formulated in the definition below.

**Definition 1.2** (Monotonic interpreter). A r.e. (recursively enumerable) set  $A \subseteq \mathbb{B}^* \times \mathbb{N}^* \times \mathbb{N}^*$  is called a *monotonic interpreter* if the following holds: whenever  $(\mathbf{p}_1, \mathbf{x}_1, \mathbf{y}_1)$  and  $(\mathbf{p}_2, \mathbf{x}_2, \mathbf{y}_2)$  are in  $A$  and  $(\mathbf{p}_1, \mathbf{x}_1)$  is compatible with  $(\mathbf{p}_2, \mathbf{x}_2)$  then  $\mathbf{y}_1$  is compatible with  $\mathbf{y}_2$ . For a monotonic interpreter,  $A$ , let us fix a particular enumeration of the set  $A$  and let  $A_t$  be the subset enumerated in  $t$  steps. Of course, the finite set  $A_t$  is also a monotonic interpreter.

An interpreter defines a monotonic (in  $\sqsubseteq$ ) function

$$A : (\mathbb{B}^* \cup \mathbb{B}^{\mathbb{N}}) \times (\mathbb{N}^* \cup \mathbb{N}^{\mathbb{N}}) \rightarrow (\mathbb{N}^* \cup \mathbb{N}^{\mathbb{N}})$$

by

$$A(\mathbf{p}, \mathbf{x}) = \sup_{\sqsubseteq} \{ \mathbf{y} \in \mathbb{N}^* : \exists (\mathbf{p}', \mathbf{x}') \sqsubseteq (\mathbf{p}, \mathbf{x}) (\mathbf{p}', \mathbf{x}', \mathbf{y}) \in A \},$$

$$A(\mathbf{p}) = A(\mathbf{p}, \Lambda).$$

A partial inverse is also worth defining:

$$A^{-1}(\mathbf{x}, \mathbf{y}) = \{ \mathbf{p} \in \mathbb{B}^* : \exists \mathbf{p}' \sqsubseteq \mathbf{p}, \mathbf{y}' \sqsupseteq \mathbf{y} (\mathbf{p}', \mathbf{x}, \mathbf{y}') \in A \},$$

$$A^{-1}(\mathbf{y}) = A^{-1}(\Lambda, \mathbf{y}).$$

┘

**Definition 1.3** (Self-delimiting interpreter). A monotonic interpreter is called *self-delimiting* if each of the sets  $A^{-1}(\mathbf{x}, \mathbf{y})$  is prefix-free. ┘

**Definition 1.4** (Monotonic complexity). For a monotonic interpreter  $A$  and  $\mathbf{x}, \mathbf{y} \in \mathbb{N}^* \cup \mathbb{N}^{\mathbb{N}}$ , we define by

$$K_A(\mathbf{y} | \mathbf{x}) = \min_{\mathbf{p} \in A^{-1}(\mathbf{x}, \mathbf{y})} l(\mathbf{p})$$

the *monotonic complexity* of string  $\mathbf{y}$  conditionally on  $\mathbf{x}$ , relative to the interpreter  $A$ . ┘

Standard arguments give the following:

**Proposition 1.2** (Optimal interpreter). (a) *There is an optimal monotonic interpreter  $U$  with respect to which complexity is minimal to within an additive constant. In other words, for any monotonic interpreter  $A$  there is a constant  $c_A$  such that for all  $x, y$  we have*

$$K_U(\mathbf{y} | \mathbf{x}) \leq K_A(\mathbf{y} | \mathbf{x}) + c_A.$$

(b) *Similarly, there is an optimal prefix-free interpreter  $V$ .*

**Definition 1.5.** Let us fix an optimal interpreter  $U$  and write

$$K(\mathbf{y} | \mathbf{x}) = K_U(\mathbf{y} | \mathbf{x}), \quad K(\mathbf{x}) = K(\mathbf{x} | \Lambda).$$

Let us also fix an optimal prefix-free interpreter  $V$  and write  $KP(\mathbf{y} | \mathbf{x}) = K_V(\mathbf{y} | \mathbf{x})$ . ┘

Clearly we have  $KP(\mathbf{y} | \mathbf{x}) \stackrel{+}{\geq} K(\mathbf{y} | \mathbf{x})$ . The more exact relation

$$KP(\mathbf{y} | \mathbf{x}) \stackrel{\pm}{=} K(\llbracket \mathbf{y} \rrbracket | \mathbf{x})$$

will be established by the Coding Theorem 1.1 below.

Here are some typical orders of magnitude. For natural numbers  $n$  (sequences of length 1) we have  $K(n) \stackrel{+}{\leq} 2 \log n$ , moreover

$$K(n) \stackrel{+}{\leq} \log n + K(\lceil \log n \rceil). \tag{1.1}$$

It can be shown that this last estimate is sharp for most numbers  $k \leq n$ .

Let us define algorithmic probability now.

**Definition 1.6** (Algorithmic probability). Let  $Z = (Z(1), Z(2), \dots)$  be an infinite sequence of identically distributed random variables with  $\mathbf{P}[Z(i) = 0] = \mathbf{P}[Z(i) = 1] = 1/2$ . The number

$$M(\mathbf{x}) = \mathbf{P}[U(Z) \supseteq \mathbf{x}]$$

is called the *algorithmic probability* of  $\mathbf{x}$ . is the probability that our optimal interpreter outputs a continuation of  $\mathbf{x}$  when given an infinite coin-tossing sequence as input.

Let  $H(\mathbf{x}) = -\log M(\mathbf{x})$ . ┘

We can express this probability in more elementary notation and contrast it with complexity:

$$\begin{aligned} M(\mathbf{x}) &= \sum_{\mathbf{p} \in (A^{-1}(\mathbf{x}))'} 2^{-l(\mathbf{p})}, \\ 2^{-K(\mathbf{x})} &= \max_{\mathbf{p} \in (A^{-1}(\mathbf{x}))'} 2^{-l(\mathbf{p})} = \max_{\mathbf{p} \in A^{-1}(\mathbf{x})} 2^{-l(\mathbf{p})}. \end{aligned} \tag{1.2}$$

The function  $M(\mathbf{x})$  also has a certain optimality property. To express it we introduce the notion of semimeasure.

**Definition 1.7.** A nonnegative real function  $\nu : \mathbb{N}^* \rightarrow \mathbb{R}$  is a *semimeasure* if  $\nu(\Lambda) \leq 1$  and for all  $\mathbf{x} \in \mathbb{N}^*$  we have

$$\sum_{n \geq 0} \nu(\mathbf{x}n) \leq \nu(\mathbf{x}). \tag{1.3}$$

A semimeasure is a *measure* if equality holds here, a *probability measure* if also  $\nu(\Lambda) = 1$ .

When  $\nu(\mathbf{x}) = 0$  on sequences of length  $> 1$  then the requirement (1.3) simplifies to  $\sum_n \nu(n) \leq 1$ .

The Lebesgue measure over  $\mathbb{B}^*$  is defined by  $\lambda(\mathbf{p}) = 2^{-l(\mathbf{p})}$ . Over subsets of the real line, it retains the usual meaning. ┘

It is easy to check that the function  $M(\mathbf{x})$  is a semimeasure. It is not computable, but has a certain weaker computability property.

**Definition 1.8.** A real function  $f : \mathbb{N}^* \rightarrow \mathbb{R}$  is said to be *lower semicomputable* if there exists a recursive function  $g : \mathbb{N}^* \times \mathbb{N} \rightarrow \mathbb{Q}$  monotonically increasing in its second argument, such that  $f(\mathbf{x}) = \lim_{t \rightarrow \infty} g(\mathbf{x}, t)$ .

A real function  $f$  is *upper semicomputable* if  $-f$  is lower semicomputable. It is *computable* if it is both lower and upper semicomputable. ┘

**Definition 1.9.** Let the interpreter  $U_t$  be defined as the one obtained from the first  $t$  steps of approximation of the optimal interpreter  $U$ . ┘

The algorithmic probability is semicomputable since it can be written as  $M(\mathbf{x}) = \lim_t M_t(\mathbf{x})$ , with

$$M_t(\mathbf{x}) = \mathbf{P}[U_t(Z) \supseteq \mathbf{x}].$$

The following is easy to see.

**Proposition 1.3.** *If a probability measure is lower semicomputable then it is computable.*

Of course, for each lower semicomputable semimeasure there is a program  $e$  enumerating it. Moreover, the following is also known (and not difficult to prove).

**Proposition 1.4.** *There is a lower semicomputable function  $\psi : \mathbb{N} \times \mathbb{N}^* \rightarrow \mathbb{R}$  such that for each fixed  $e \in \mathbb{N}$  the function*

$$\psi_e(\mathbf{x}) = \psi(e, \mathbf{x})$$

*is a semimeasure. The algorithmic probability  $M(\mathbf{x})$  majorizes all lower semicomputable semimeasures within a multiplicative constant, moreover the following more exact relation holds:*

$$\psi_e(\mathbf{x}) \leq^* M(\mathbf{x})/M(e).$$

Thus, for semimeasure  $\psi_e$  the multiplicative constant of majorization is  $O(1/M(e))$ . In additive notation the relation says

$$H(\mathbf{x}) \leq^+ -\log \psi_e(\mathbf{x}) - H(e). \quad (1.4)$$

The paper [3] (relying on several earlier papers of Levin) lists several reasons why  $H(x)$  can be viewed as a kind of complexity. Note first of all that the relation

$$H(\mathbf{x}) \leq K(\mathbf{x})$$

follows immediately from the expressions (1.2). We are interested in the sharpness of this inequality. The above papers list also the important subsets of  $\mathbb{N}^*$  on which these two quantities differ only within an additive constant. In particular, the following result is equivalent to the so-called “coding theorem” of algorithmic information theory (see [4]).

**Theorem 1.1** (Coding). *For  $e \in \mathbb{N}$  let  $W_e$  be the recursively enumerable prefix-free set of  $\mathbb{N}^*$  given with a “program”  $e$ . Then on the elements  $x$  of  $W_e$  the following relation holds:*

$$KP(\mathbf{x}) \leq^+ K(\mathbf{x}) \leq^+ H(\mathbf{x}) + K(e).$$

**Corollary 1.5.** *We have  $KP(\mathbf{x}) \leq^+ KP(\llbracket \mathbf{x} \rrbracket) \leq^+ K(\llbracket \mathbf{x} \rrbracket)$ .*

*Proof.* The first relation holds since  $\mathbf{x} \mapsto \llbracket \mathbf{x} \rrbracket$  is a recursive invertible function. The second relation holds since  $\llbracket \mathbf{x} \rrbracket$  is in the set of one-element strings, which is a prefix-free set.  $\square$

As a consequence, we have the following bound.

**Corollary 1.6.**

$$H(\mathbf{x}) \leq^+ K(\mathbf{x}) \leq^+ H(\mathbf{x}) + K(l(\mathbf{x})). \quad (1.5)$$

*Proof.* Indeed, for each  $n$  there is a program with complexity  $\overset{+}{<} K(n)$  listing all strings of length  $n$ .  $\square$

From (1.5) and (1.1) we have for example for  $\mathbf{x}$  of length  $n$ :

$$K(\mathbf{x}) \overset{+}{<} H(\mathbf{x}) + \log n + 2 \log \log n.$$

As an even simpler consequence, restricting attention to sequences of length 1, that is natural numbers, we get

$$K(n) \overset{+}{<} H(n). \tag{1.6}$$

Here is an easy reformulation of this result in a form that we need:

**Proposition 1.7.** *For an upper semicomputable function  $g : \mathbb{N} \rightarrow \mathbb{N}$  the following properties are equivalent:*

- (a)  $\sum_n 2^{-g(n)} < \infty$ .
- (b)  $\sum_n 2^{-g(n)} \overset{*}{<} 1$ .
- (c)  $\exists c \forall n g(n) \geq K(n) - c$ .
- (d)  $\exists c \forall n g(n) \geq H(n) - c$ .

In view of the above proposition let us use the following terminology.

**Definition 1.10.** An upper semicomputable function  $g : \mathbb{N} \rightarrow \mathbb{N}$  is said to be *sub- $K$* , if  $\sum_n 2^{-g(n)} = \infty$ .  $\lrcorner$

The main theorem of this paper shows that the bound (1.5) is in some sense sharp.

**Theorem 1.2.** *For any upper semicomputable function  $g : \mathbb{N} \rightarrow \mathbb{N}$  satisfying for all  $x$*

$$K(x) - H(x) \leq g(l(x))$$

*there is a constant  $c_g$  with the property  $g(n) \geq K(n) - c_g$  for all  $n$ .*

In view of the proposition above, the following corollary is an equivalent reformulation of the theorem.

**Corollary 1.8.** *For any upper semicomputable function  $g : \mathbb{N} \rightarrow \mathbb{N}$  with  $\sum_n 2^{-g(n)} = \infty$  there is an  $x$  with*

$$K(x) > H(x) + g(l(x)).$$

In particular, since for each  $c$  the function  $g(n) = \lceil \log n \rceil + c$  is upper semicomputable with  $\sum_n 2^{-g(n)} = \infty$  it follows that for each  $c$  there is an  $x$  with

$$K(\mathbf{x}) > H(\mathbf{x}) + \log(l(\mathbf{x})) + c.$$

The original proof of the theorem gave only a poor lower bound for binary sequences. Using the work of Adam Day from [2], we also prove the following.

**Theorem 1.3.** *For every  $\varepsilon > 0$ , binary sequences the inequality*

$$K(x) - H(x) \geq (1 - \varepsilon) \log \log l(x)$$

*holds for infinitely many strings  $x$ .*

## 2 The coding theorem

Theorem 1.2 essentially states that the assertion corresponding to the Coding Theorem 1.1 is false when  $\mathbf{x}$  is allowed to run over all strings in  $\mathbb{N}^*$ . We will first review a proof of the Coding Theorem in order to understand what problems prevent its application in the general case. In the presentation of the proof we will not strive for conciseness and elegance, since we want to introduce the elements of later constructions.

It is easy to see that the general proposition is equivalent to its special case (1.6). Let  $a : \mathbb{N} \rightarrow \mathbb{R}$  be a lower semicomputable semimeasure. We will prove the inequality  $KP(x) \stackrel{+}{<} -\log a(x)$  by defining a self-delimiting interpreter  $A$  with the property

$$K_A(x) \stackrel{+}{<} -\log a(x).$$

Since the semimeasure  $a(x)$  is lower semicomputable, it can be represented as

$$a(x) = \lim_{t \rightarrow \infty} a(x, t)$$

where  $a(x, t) \in \{0\} \cup \{2^{-k} : k \in \mathbb{N}\}$ , and  $(x, t) \mapsto a(x, t)$  is recursive and increasing in  $t$ . This motivates the following definition.

**Definition 2.1** (Accumulation sequence). Let  $X$  be a set of natural numbers. A sequence  $a : X \times \mathbb{N} \rightarrow \mathbb{Q}_+$  is called a *accumulation sequence* over  $X$  if it has the following properties.

- (a) For each  $t$  the function  $a(\cdot, t)$  is a semimeasure.
- (b)  $a(x, t)$  is nondecreasing in  $t$ .

For any set  $S \subseteq X$  let

$$a(S, t) = \sum_{x \in S} a(x, t).$$

If  $X = \mathbb{N}$  then we may omit mentioning it. ┘

**Definition 2.2.** An accumulation sequence will be called *systematic* if it also has the property that for each  $t$  there is an element  $x_t$  such that  $a(x, t + 1) = a(x, t)$  for all  $x \neq x_t$ , and  $a(x_t, t) \leq 2a(x_t, t)$ .  $\lrcorner$

Now the following is easy to show.

**Proposition 2.1.** *A lower semicomputable semimeasure can be represented as the limit of a computable systematic accumulation sequence.*

Using the characterization (1.2)—which holds for arbitrary  $K_A$ —and going to multiplicative notation, we want to find an algorithm  $A$  with the property

$$\max_{\mathbf{p} \in A^{-1}(x)} 2^{-l(\mathbf{p})} > a(x).$$

It is convenient to work with the function  $x \mapsto A^{-1}(x)$  instead of the function  $\mathbf{p} \mapsto A(\mathbf{p})$ . The function

$$B(x) = A^{-1}(x)$$

has certain properties worth analyzing.

**Definition 2.3** (Allocation over  $\mathbb{N}$ ). Let us call a function

$$x \in \mathbb{N} \mapsto B(x) \subseteq \mathbb{B}^*$$

an *allocation function on  $X$*  it has the following properties.

- (i) The sets  $B(x)$  are prefix-free.
- (ii) For  $x \neq y$ , the sets  $B(x), B(y)$  are mutually incompatible, that is  $[B(x)]$  and  $[B(y)]$  are disjoint.

If the set  $\{(\mathbf{p}, x) : \mathbf{p} \in B(x)\}$  is recursively enumerable then the allocation function will be called *lower semicomputable*.

If  $X = \mathbb{N}$  then we may omit mentioning it.  $\lrcorner$

The following observation is immediate.

**Proposition 2.2.** *If  $A$  is a self-delimiting interpreter then  $B(x) = A^{-1}(x)$  defines a lower semicomputable allocation function.*

*Conversely, to each lower semicomputable allocation function  $x \mapsto B(x)$  the set  $A = \{(\mathbf{p}, x) : \mathbf{p} \in B(x)\}$  defines a self-delimiting interpreter with  $B(x) = A^{-1}(x)$ .*

We will define the sets  $B(x)$  as a limit

$$B(x) = \bigcup_t B(x, t),$$

where the function  $(x, t) \mapsto B(x, t)$  is recursive and nondecreasing in  $t$ . This gives rise to the following definition.

**Definition 2.4.** A function  $B(x, t)$  will be called a *monotonic allocation sequence* on  $X$  if it satisfies the following conditions.

- (i) For each  $t$  the function  $B(\cdot, t)$  is an allocation function on  $X$ .
- (ii) It is monotonic in  $t$ :  $B(x, t) \subseteq B(x, t + 1)$ .
- (iii) The set  $\bigcup_x B(x, t)$  is finite for each  $t$ .

For the given sequence  $B(x, t)$  define

$$\begin{aligned} B(X, t) &= \bigcup_{x \in X} B(x, t), \\ b(x, t) &= \lambda(B(x, t)), \\ b(X, t) &= \lambda(B(X, t)) = \sum_{x \in X} b(x, t). \end{aligned}$$

┘

Now the above observation can be formulated by saying that a lower semicomputable allocation function is the limit of a computable monotonic allocation sequence.

Instead of binary sequences  $\mathbf{p}$ , it is helpful to think of the binary subintervals  $[\mathbf{p}]$  of the interval  $[0, 1]$  that they correspond to. Let us introduce some terminology.

**Definition 2.5.** For a binary sequence  $\mathbf{p}$  of length  $r$ , the binary interval  $[\mathbf{p}]$  will be called an *r-bin*. We will say that an *r-bin*  $[\mathbf{p}]$  is *allocated to*  $x \in \mathbb{N}$  at time  $t$  if  $[\mathbf{p}] \subseteq [B(x, t)]$ , else it is *free* (they it is also disjoint from  $[B(x, t)]$ ). The allocatedness relation is monotonic in  $t$ , since  $B(x, t)$  is increasing.

┘

**Definition 2.6.** Assume that both an accumulation sequence  $a(x, t)$  and a monotonic allocation sequence  $B(x, t)$  are given. Let

$$\begin{aligned} c(x, t) &= \max_{\mathbf{p} \in B(x, t)} \lambda([\mathbf{p}]), \\ C(x, t) &= \bigcup_{\substack{\mathbf{p} \in B(x, t) \\ l(\mathbf{p}) = c(x, t)}} [\mathbf{p}], \\ C(X, t) &= \bigcup_{x \in X} C(x, t), \\ c(X, t) &= \lambda(C(X, t)). \end{aligned}$$

┘

Now the Coding Theorem is implied from the following.

**Theorem 2.1.** For every computable accumulation sequence  $a(x, t)$  there is a computable monotonic allocation sequence  $B(x, t)$  with  $c(x, t) \geq a(x, t)/4$  for all  $x, t$ .

With the above definitions, an algorithm for  $B(x, t)$  is given in Algorithm 2.1.

**Algorithm 2.1.** Construction for the Coding Theorem

**input** : An accumulation sequence  $(x, t) \mapsto a(x, t)$ . Only the part for time  $\leq t$  is given at time  $t$ .  
**output** : A monotonic allocation function  $(x, t) \mapsto B(x, t)$ .  
 $B(x, 0) \leftarrow \emptyset$   
**for**  $t = 0$  **to**  $\infty$  **do**  
    **foreach**  $x \neq x_t$  **do**  
         $B(x, t+1) \leftarrow B(x, t)$   
         $r \leftarrow \lfloor -\log a(x_t, t) \rfloor$   
        **if**  $\lfloor -\log a(x_t, t+1) \rfloor = r$  **then**  
             $B(x_t, t+1) \leftarrow B(x_t, t)$   
            (In all other cases  $\lfloor -\log a(x_t, t+1) \rfloor = r - 1$ .)  
        **else if** there are free  $r$ -bins at time  $t$  **then**  
             $[p] \leftarrow$  the first one  
             $B(x_t, t+1) \leftarrow B(x_t, t) \cup \{p\}$   
        **else**  
            give up

If it never gives up, this algorithm guarantees

$$K_A(x) \leq -\log a(x) + 2.$$

Indeed, with  $s = \lfloor -\log a(x) \rfloor$  there is a  $t$  with  $\lfloor -\log a(x, t) \rfloor \leq s + 1$ , so  $B(x, t)$  will contain an  $(s + 2)$ -bin, and hence  $K_A(x) \leq s + 2 \leq -\log a(x) + 2$ .

Let us show that it indeed never gives up. The following lemma is immediate from the definition of the algorithm.

**Lemma 2.3.** *Suppose that  $b(x, t)$  is constructed by Algorithm 2.1. For all  $x, t, r$  if  $a(x, t) < 2^{-(r-1)}$  then  $b(x, t) < 2^{-r}$ . In particular,  $b(x, t) < a(x, t)$ .*

*Proof.* For each  $s < r$  we increased  $b(x, t)$  by  $2^s$  when  $a(x, t)$  exceeded  $2^{-(s-1)}$ . The sum of all these increases is  $< 2^{-r}$ .  $\square$

This lemma implies that at the time  $t$  when the algorithm is looking for an  $r$ -bin, we have

$$\begin{aligned} \sum_{x \neq x_t} b(x, t) &< \sum_{x \neq x_t} a(x, t) = \sum_{x \neq x_t} a(x+1), \\ 2^{-r} + b(x, t) &< 2^{-(r-1)} < a(x_t, t+1), \\ 2^{-r} + b(\mathbb{N}, t) &\leq a(\mathbb{N}, t) \leq 1. \end{aligned}$$

The following lemma provides the required free bin.

**Lemma 2.4** (Pippenger, see [1]). *The union of all free bins at time  $t$  can be written as a finite union  $\cup_{i=1}^n [\mathbf{p}_i]$  where  $[\mathbf{p}_i]$  appears after  $[\mathbf{p}_{i-1}]$  in the interval  $[0, 1]$  and the length of strings  $\mathbf{p}_i$  is strictly decreasing. Therefore  $1 - b(\mathbb{N}, t) \geq 2^{-r}$  implies that there is a free  $r$ -bin.*

*Proof.* Immediate from the observation of the algorithm. □

This concludes the proof of the coding theorem.

### 3 A memory-allocation game on integers

In the above proof of the Coding Theorem, the allocation function  $B(x, t)$  was defined sequentially, depending on the definition of the function  $a(x, t)$ . You can view it as a strategy followed in a certain game.

**Definition 3.1** (Game 1). Let  $g > 0$  be a real number called the *redundancy*. Game<sub>1</sub>( $g$ ) has two players, *Client* and *Server*, or Csilla and Sandor (so that we can refer to them by “she” and “he”). In each step  $t = 1, 2, \dots$ , it is Client’s step first and Server’s next. In step  $t$ , Client defines  $a(\cdot, t)$  for a systematic accumulation sequence  $a(x, t)$ , after which Server defines  $B(\cdot, t)$  for a monotonic allocation sequence  $B(x, t)$  over natural numbers.

As is usual in games, we allow the choice of  $a(x, t)$  to depend on the values of  $B(\cdot, u)$  for all  $u < t$ , and the choice of  $B(x, t)$  to depend on the values of  $a(\cdot, u)$  for all  $u \leq t$ .

Server loses unless for each  $r, x, t$ , an  $r$ -bin will be allocated to  $x$  at time  $t$  if  $a(x, t) \geq 2^{-r+g}$ , after he made his step. ┘

The following simple observation is immediate.

**Proposition 3.1.** *Server obeys a redundancy requirement given by  $g$  if and only if we have*

$$c(x, t) > 2^{-g-1} a(x, t)$$

for all  $x, t$ .

So, Server’s goal is that if the measure  $a(x, t)$  is large then  $B(x, t)$  should contain a large bin, while Client is trying to cause as much *fragmentation* for Server as possible.

The proof of the Coding Theorem gives Server a strategy to survive in Game 1 if the redundancy is allowed to be at least 2. We will give a lower bound on redundancy needed for Server. What we will use later is not the theorem itself but its proof technique, in particular Client’s strategy and the main lemma.

**Theorem 3.1.** *For every  $g < \log \frac{3}{2}$ , Server loses in Game<sub>1</sub>( $g$ ).*

To prepare the proof, we define a strategy for Client. For at least some  $x$  and  $r$ , she wants to prevent Server from slowly filling in an  $r$ -bin as  $a(x, t)$  increases. When Server started filling in an  $r$ -bin, we can say he has “reserved” it. Let us formalize this.

**Definition 3.2** (Reservation). For any allocation function  $B(\cdot)$ , we will say that it *reserves* a bin  $[\mathbf{p}]$  for number  $x$  if  $[\mathbf{p}]$  intersects  $[B(x)]$  but is disjoint from all other  $[B(y)]$  for  $y \neq x$ . We will say that an allocation sequence  $B(\cdot, \cdot)$  reserves  $[\mathbf{p}]$  at time  $t$  for  $x$  if  $B(\cdot, t)$  reserves it for  $x$ . If the sequence  $B(\cdot, \cdot)$  is clear from the context then we will just say that  $[\mathbf{p}]$  is *reserved* for  $x$  at time  $t$ . Let

$$B_r(x, t)$$

denote the union of  $r$ -bins reserved for  $x$  at time  $t$ , with

$$b_r(x, t) = \lambda(B_r(x, t)).$$

┘

Note that  $B_r(x, t)$  is not necessarily monotonic in  $t$ .

*Remark 3.2.* In what follows when we speak about a *strategy* of Client, this may be like a “procedure” in a computer program: we allow it to have some parameters and to start at some intermediate stage of the game, under some conditions. ┘

The following idea will provide a strategy for Client:

1. Fix some  $r$ .
2. Repeatedly, find the first object  $x$  that has no  $r$ -bin reserved for it, and increase  $a(x, t)$  by a tiny amount.
3. Stop at some time when you made Server waste enough space.

The following lemma implements this idea.

**Lemma 3.3.** *Suppose that we are given a set  $X \subseteq \mathbb{N}$ , and some parameters as follows:  $g > 0$ ,  $0 < \varepsilon < 1/2$ ,  $0 < \rho \leq 1$ , and an integer  $r$  with*

$$2^{-r} \leq \varepsilon^2 \rho. \tag{3.1}$$

*Then in  $\text{Game}_1(g)$ , Client has a strategy*

$$\text{inflate-ints}(X, r, g, \rho, \varepsilon)$$

*that, when started at a time  $t_0$  with  $a(\mathbb{N}, t_0) \leq 1 - \rho$  and  $a(X, t_0) = 0$ , provides the following.*

- (a) *The accumulation  $a(x, t)$  will only be increased for elements  $x$  of  $X$  and each increase is by an amount in the range  $[\varepsilon^2 2^{-r}, \varepsilon 2^{-r}]$ .*
- (b) *The strategy stops at a time  $t' > t_0$  with*

$$\begin{aligned} \varepsilon \rho &\leq a(X, t') \leq \rho, \\ \lambda(C(X, t') \cup B_r(X, t')) &\geq (3/2 - 5\varepsilon) 2^{-g} a(X, t'). \end{aligned} \tag{3.2}$$

**Definition 3.3.** Denote by

$$\text{stopcond-ints}(t', X, r, g, \rho, \varepsilon).$$

the stopping condition (3.2) above. ┘

We postpone the proof of Lemma 3.3, but define the strategy for Client in Algorithm 3.1 that will be used in it.

**Algorithm 3.1.** Client's strategy  $\text{inflate-ints}(X, r, g, \rho, \varepsilon)$

```

 $r' = r'(r, \varepsilon) \leftarrow r + g + \lceil \log(1/\varepsilon) \rceil$ 
 $m' \leftarrow \lfloor \rho 2^{r'} / \varepsilon^2 \rfloor$ 
for  $t = t_0$  to  $m'$  do
  if  $\text{stopcond-ints}(t, X, r, g, \rho, \varepsilon)$  then
    return
     $x_t \leftarrow$  the smallest integer in  $X$  for which no  $r$ -bin has been reserved
     $a(x_t, t + 1) \leftarrow a(x_t, t) + 2^{-r'+g}$ 

```

It is easy to check that  $r'$  satisfies  $\varepsilon^2 2^{-r} \leq 2^{-r'+g} \leq \varepsilon 2^{-r}$  and is therefore an allowed increment according to requirement (a) in Lemma 3.3.

Before proving the lemma, let us use it to prove Theorem 3.1.

*Proof of Theorem 3.1.* Choose some parameters  $\rho, \varepsilon, r$  to be used in Lemma 3.3.

$$m = \left\lfloor \frac{1}{\varepsilon \rho} \right\rfloor.$$

We will define a strategy for Client that applies the strategy  $\text{inflate-ints}()$  repeatedly, on a sequence

$$X_0, X_2, \dots, X_{m-1}$$

of disjoint infinite sets of integers, to lowerbound on the volume of the sets

$$B_i = C(X_i, t'_i) \cup B_{r_i}(X_i, t'_i).$$

There is a problem with this application, though, that is at the core of the inefficiency of our later client strategies. In order to derive a lower bound on the volume of  $B_0 \cup \dots \cup B_{m-1}$  conveniently, we would need these sets to be disjoint. But reservation is not monotonic in time: in stage  $i + 1$  Server might allocate some of the set  $C(X_{i+1}, t)$  inside  $B_{r_i}(X_i, t'_i)$ . To prevent this, we will make the sequence  $r_i$  decreasing, in such a way that for any element in stage  $i + 1$  the allocation will require a bin that is bigger than  $2^{-r_i}$ . This way the unallocated parts of  $r_i$ -bins reserved in stage  $i$  cannot be allocated in any useful way in stage  $i + 1$ . We will set

$$r_i = r'(r_{i+1}, \varepsilon)$$

where the function  $r'(r, \varepsilon)$  was introduced in the strategy of Algorithm 3.1. Define now the strategy of Client in 3.2.

**Algorithm 3.2.** Client's strategy in Game 1

```

for  $i = 0$  to  $m - 1$  do
  (we are at time  $t_i$ )
  if  $a(\mathbb{N}, t_i) > 1 - \rho$  then
    return
     $r_i \leftarrow r + (m - i)(g + \lceil \log(1/\varepsilon) \rceil)$ 
    run strategy inflate-ints( $X_i, r_i, g, \rho, \varepsilon$ ) until it stops at some time  $t_{i+1}$ 

```

After the end of part  $i$ , at time  $t'_i$ , Lemma 3.3 gives

$$\lambda(B_i) \geq (3/2 - 5\varepsilon)2^{-g} a(X_i, t'_i).$$

where we used the abbreviation  $B_i = C(X_i, t'_i) \cup B_{r_i}(X_i, t'_i)$ .

**Claim 3.4.** For  $i < j$  the sets  $B_i$  and  $B_j$  are disjoint.

*Proof.* The reserved set  $B_r(X_j, t'_j)$  cannot intersect  $B_i$ , since the latter is allocated or reserved to elements of set  $X_i$  which is disjoint from  $X_j$ .

The allocated set  $C(X_j, t'_j)$  cannot intersect  $C(X_i, t'_i)$  for the same reason.

The allocated set  $C(X_j, t'_j)$  cannot intersect  $B_{r_i}(X_i, t'_i)$  either. Indeed, for  $x \in X_j$  we have

$$a(x, t'_j) = 2^{-r'(r_j, \varepsilon) + g} = 2^{-r_{j-1} + g},$$

hence  $C(X_j, t'_j)$  consists of  $s$ -bins of some  $s \leq r_{j-1} \leq r_i$  allocated to  $X_j$ . □

It follows, with  $n'$  as the iteration  $i$  at which the algorithm returns:

$$\begin{aligned}
1 &\geq \lambda\left(\bigcup_i B_i\right) = \sum_i \lambda(B_i) \geq 2^{-g} (3/2 - 5\varepsilon) \sum_i a(X_i, t'_i) \\
&= 2^{-g} (3/2 - 5\varepsilon) a(\mathbb{N}, t'_{n'}) > 2^{-g} (3/2 - 5\varepsilon) (1 - \varepsilon) && \text{since Client stopped,} \\
2^g &\geq (3/2 - 5\varepsilon) (1 - \varepsilon).
\end{aligned}$$

If  $2^g < 3/2$  we get a contradiction with  $\varepsilon$  chosen sufficiently small. □

In the proof of the lemma, for later applications, will also use a parameter  $\gamma$  which now we set to

$$\gamma = 2^{-g}.$$

The proof of Lemma 3.3 will be divided into the following lemmas.

**Lemma 3.5.** There is a time  $t'$  during the application of strategy `inflate-ints`( $\cdot$ ) with  $\varepsilon\rho \leq a(X, t') \leq \rho$  and

$$b_r(X, t') \geq (1 - 4\varepsilon)2^{-g} a(X, t').$$

**Lemma 3.6.** *The following holds for all  $t$  during the application of strategy `inflate-ints(·)`:*

$$\lambda(C(X, t) \cup B_r(X, t)) \geq (\gamma/2)a(X, t) + (1 - \varepsilon)b_r(X, t). \quad (3.3)$$

*Proof of Lemma 3.3.* Using the two claims gives

$$\begin{aligned} \lambda(B(X, t') \cup B_r(X, t')) &\geq a(X, t')(\gamma/2 + 2^{-g}(1 - \varepsilon)(1 - 4\varepsilon)) \\ &\geq a(X, t')(\gamma/2 + 2^{-g}(1 - 5\varepsilon)). \end{aligned}$$

But  $\gamma$  was defined as  $2^{-g}$ , so this finishes the proof.  $\square$

*Proof of Lemma 3.5.* Let us start with a simple observation.

**Claim 3.7.** *During the algorithm `inflate-ints(·)` we have  $a(x, t) \leq 2^{-r}(2^g + \varepsilon)$  for all  $x \in X$ .*

*Proof.* If the algorithm increases  $a(x, t)$  then it does not have an  $r$ -bin reserved, so it does not have an  $r$ -bin allocated either. Since Server must satisfy the redundancy condition we have still  $a(x, t) \leq 2^{-r+\varepsilon}$ . The right-hand side grows by at most  $\varepsilon 2^{-r}$  in the very step after which  $x$  gets an  $r$ -bin allocated.  $\square$

Let  $u$  be the first step  $t$  at which  $a(X, t) > \varepsilon\rho$ , and let  $v$  be the step  $t$  in which  $a(X, t)$  would exceed the bound  $\rho$  if Client did not halt before. Define

$$m = \max_{t \in [u, v]} |B_r(X, t)|,$$

and let  $X = \{y_1, y_2, \dots\}$  in increasing order. For  $u \leq t \leq v$  Client always increases some  $a(y_i, t)$  for  $i \leq m + 1$ . We have hence

$$\begin{aligned} a(X, v) - a(X, u) &\leq (m + 1) \max_{i \leq m+1} a(y_i, v) \\ &\leq (m + 1)2^{-r}(2^g + \varepsilon) && \text{by Claim 3.7,} \\ m2^{-r} &\geq \frac{2^{-g}(a(X, v) - a(X, u))}{1 + \varepsilon} - 2^{-r}. \end{aligned} \quad (3.4)$$

The maximum  $m$  is achieved for some  $t' \in [u, v]$ . We can estimate

$$\begin{aligned} \frac{a(X, v) - a(X, u)}{a(X, v)} &\geq 1 - \frac{\varepsilon\rho}{\rho - \varepsilon 2^{-r}} > 1 - \frac{\varepsilon}{1 - \varepsilon^3} && \text{by (3.1),} \\ &> 1 - 2\varepsilon. \end{aligned}$$

This can be used for  $a(X, v) - a(X, u) \geq (1 - 2\varepsilon)a(X, t')$ , allowing with (3.4) to conclude

$$\begin{aligned} b_r(X, t') &= m2^{-r} \geq \frac{2^{-g}(1 - 2\varepsilon)a(X, t')}{1 + \varepsilon} - 2^{-r} \\ &\geq 2^{-g}a(X, t') \left( \frac{1 - 2\varepsilon}{1 + \varepsilon} - \frac{2^{-r}}{\varepsilon\rho} \right) && \text{by } a(X, t') \geq \varepsilon\rho, \\ &\geq 2^{-g}a(X, t') \left( \frac{1 - 2\varepsilon}{1 + \varepsilon} - \varepsilon \right) && \text{by (3.1),} \\ &\geq 2^{-g}a(X, t')(1 - 4\varepsilon). \end{aligned}$$

$\square$

*Proof of Lemma 3.6.* For each  $x \in X$ , if no  $r$ -bin is reserved for  $x$  at time  $t$  then  $B_r(x, t) = \emptyset$  and  $C(x, t) \cup B_r(x, t) = C(x, t)$ . In this case, the right-hand side above is  $(\gamma/2)a(x, t)$  and the inequality follows from Proposition 3.1.

Suppose that there are some  $r$ -bins reserved for  $x$  at time  $t$ , namely  $n$  of them, and let  $u \leq t$  be the earliest time one of them has been reserved. Then  $a(x, t) = a(x, u)$ . We have

$$\lambda(C(x, t) \cup B_r(x, t)) \geq \lambda(C(x, u-1) \cup B_r(x, t)).$$

The set  $B_r(x, t)$  of new  $r$ -bins reserved by time  $t$  for  $x$  is disjoint from  $B(x, u-1)$ , so we have

$$\lambda(C(x, u-1) \cup B_r(x, t)) \geq c(x, u-1) + n2^{-r} \geq (\gamma/2)a(x, u-1) + n2^{-r} \quad (3.5)$$

by Proposition 3.1. Now  $a(x, u-1) \geq a(x, u) - \varepsilon 2^{-r} = a(x, t) - \varepsilon 2^{-r}$ , the right-hand side of (3.5) is therefore greater than

$$\gamma(a(x, t) - \varepsilon 2^{-r}) + n2^{-r} \geq \gamma a(x, t) + n2^{-r}(1 - \varepsilon).$$

Summing over  $x \in X$  concludes the proof. □

## 4 Game on strings

Let us return to the problem of semimeasures and description complexity on strings. The following definition is a straightforward generalization for the one over integers.

**Definition 4.1** (Accumulation sequence on strings). A sequence  $a : \mathbb{N}^* \times \mathbb{N} \rightarrow \mathbb{Q}_+$  is called a *accumulation sequence* if it has the following properties.

- (a) For each  $t$  the function  $a(\cdot, t)$  is a semimeasure over  $\mathbb{N}^*$ .
- (b)  $a(\mathbf{x}, t)$  is nondecreasing in  $t$ .

┘

**Definition 4.2** (Allocation over  $\mathbb{N}^*$ ). Let us call a function

$$x \in \mathbf{w}\mathbb{N}^* \mapsto B(x) \subseteq \mathbb{B}^*$$

an *allocation function on  $\mathbf{w}\mathbb{N}^*$*  if it has the following properties.

- (i) For  $\mathbf{x} \sqsubseteq \mathbf{y}$  we have  $B(\mathbf{x}) \leq B(\mathbf{y})$ .
- (ii) For incompatible strings  $\mathbf{x}, \mathbf{y}$ , the sets  $B(\mathbf{x}), B(\mathbf{y})$  are mutually incompatible, that is  $[B(\mathbf{x})]$  and  $[B(\mathbf{y})]$  are disjoint.

If  $\mathbf{w}$  is the empty string we may omit mentioning it.

If the set  $\{(\mathbf{p}, \mathbf{x}) : \mathbf{p} \in B(\mathbf{x})\}$  is recursively enumerable then the allocation function  $B(\cdot)$  will be called *lower semicomputable*. ┘

The following observation is immediate.

**Proposition 4.1.** *If  $A$  is a monotonic interpreter then  $B(\mathbf{x}) = A^{-1}(\mathbf{x})$  defines a lower semicomputable allocation function over  $\mathbb{N}^*$ .*

*Conversely, to each lower semicomputable allocation function  $x \mapsto B(\mathbf{x})$  over  $\mathbb{N}^*$ , the set  $A = \{(\mathbf{p}, \mathbf{x}) : \mathbf{p} \in B(\mathbf{x})\}$  defines a monotonic interpreter with  $B(\mathbf{x}) = A^{-1}(\mathbf{x})$ .*

We will define the sets  $B(\mathbf{x})$  as a limit

$$B(\mathbf{x}) = \bigcup_t B(\mathbf{x}, t),$$

where the function  $(\mathbf{x}, t) \mapsto B(\mathbf{x}, t)$  is recursive and nondecreasing in  $t$ . This gives rise to the following definition.

**Definition 4.3.** For a string  $\mathbf{w}$ , a function  $B(\mathbf{x}, t)$  will be called a *monotonic allocation sequence* on  $\mathbf{w}\mathbb{N}^*$  if it satisfies the following conditions.

- (i) For each  $t$  the function  $B(\cdot, t)$  is an allocation function over  $\mathbf{w}\mathbb{N}^*$ .
- (ii) It is monotonic in  $t$ :  $B(\mathbf{x}, t) \subseteq B(\mathbf{x}, t+1)$ .
- (iii) The set  $\bigcup_{\mathbf{x}} B(\mathbf{x}, t)$  is finite for each  $t$ .

For the given sequence  $B(\mathbf{x}, t)$  define  $b(\mathbf{x}, t) = \lambda(B(\mathbf{x}, t))$ . ┘

Let us define now the game needed for the proof of Theorem 1.2.

**Definition 4.4** (Game 2). A sequence  $g : \mathbb{N} \rightarrow \mathbb{R}_+$ ,  $n \mapsto g(n)$  is called the *redundancy sequence*.

For a redundancy sequence  $g(\cdot)$ , the game  $\text{Game}_2(g(\cdot))$  has two players, Client and Server as the earlier game, and the same alternation of steps. In step  $t$ , Client defines  $a(\cdot, t)$  for an accumulation sequence  $a(x, t)$  over  $\mathbb{N}^*$ . Server defines a monotonic allocation sequence  $B(\mathbf{x}, t)$  over  $\mathbb{N}^*$ .

Server loses in step  $t$  unless the inequality

$$c(\mathbf{x}, t) > 2^{-g(l(\mathbf{x})) - 1} a(\mathbf{x}, t)$$

holds for all  $x$  after he made his step. ┘

Suppose that Server has a strategy to survive  $\text{Game}_2(g(\cdot))$ . Then, just as from Game 1 the coding theorem, we could conclude

$$K(\mathbf{x}) \stackrel{+}{<} H(\mathbf{x}) + g(l(\mathbf{x})).$$

It is less obvious whether a corresponding statement follows from a strategy of Client, but this is what the next proposition shows.

**Definition 4.5.** Recall Definition 1.10. We say that Client has a strategy against all sub- $K$  redundancy sequences if there is a recursive function

$$S : \mathbb{N} \rightarrow \mathbb{N}$$

with the following property. Whenever  $e$  can be interpreted (in a standard way) as the program of a sub- $K$  sequence  $g_e(n)$  then the integer  $S(e)$  can be interpreted as the program of a recursive strategy of Client that defeats Server in the game using the redundancy sequence  $g_e(n)$ .  $\lrcorner$

Our main tool is the following:

**Theorem 4.1.** *Client has a strategy against all sub- $K$  redundancy sequences.*

It will be much work to prove this theorem, so let us show first how it helps achieve our original goal.

**Proposition 4.2.** *If Client has a strategy against all sub- $K$  redundancy sequences then Theorem 1.2 holds.*

*Proof.* In view of Proposition 1.7, it is sufficient to show that for all sub- $K$  redundancy sequences  $g(n)$  there is an  $x$  with  $K(x) > H(x) + g(l(x))$ .

Let the computable monotonic allocation sequence  $B(x, t)$  be defined as

$$B(x, t) = \{\mathbf{p} : \mathbf{x} \sqsubseteq U_t(\mathbf{p})\}$$

where the interpreter  $U_t$  was defined in Definition 1.9 above.

Let  $g$  be a sub- $K$  redundancy sequence and define  $g_k(n) = g(n) + k$ . Then  $g_k$  is also a sub- $K$  redundancy sequence. If Client has a strategy against all sub- $K$  redundancy sequences then there is a recursive function  $S_0(k)$  that assigns to each  $k$  the program of a recursive strategy of Client defeating Server on redundancy function  $g_k$ .

Strategy  $S_0(k)$  of Client computes from Server's strategy  $B(x, t)$  a recursive accumulation sequence  $a_k(x, t)$  on which Server fails with redundancy sequence  $g_k(n)$ : in other words, there is an  $\mathbf{x}_k$  with

$$K(\mathbf{x}) > -\log a_k(\mathbf{x}_k) + g_k(l(\mathbf{x}_k)) = -\log a_k(\mathbf{x}_k) + g(l(\mathbf{x}_k)) + k.$$

Let us use (1.4) noting  $a_k(\mathbf{x}) = \psi_{f(k)}(\mathbf{x})$  for a recursive function  $f(k)$ . It gives

$$\begin{aligned} H(\mathbf{x}_k) &\stackrel{+}{<} -\log a_k(\mathbf{x}_k) + H(f(k)) \stackrel{+}{<} -\log a_k(\mathbf{x}_k) + H(k), \\ -\log a_k(\mathbf{x}_k) &\stackrel{+}{>} H(\mathbf{x}_k) - H(k), \\ K(\mathbf{x}_k) &\stackrel{+}{>} H(\mathbf{x}_k) + g(l(\mathbf{x}_k)) + k - H(k). \end{aligned}$$

For sufficiently large  $k$  this implies  $K(\mathbf{x}_k) > H(\mathbf{x}_k) + g(l(\mathbf{x}_k))$ .  $\square$

Theorem (3.1) suggests that, at least, a certain naive application of Server's strategy of Game 1 will increase redundancy too much. Suppose that in Game 2, Server proceeds as follows.

1. On strings of form  $\mathbf{x} = x(1)\cdots x(n)$ , Client applies the strategy of Algorithm 3.2 to the accumulation sequence  $a(x(1), t)$ .
2. At the same time, for each value  $z_1$  of  $x(1)$  he applies the same strategy to the accumulation sequence  $a(z_1x(2), t)$ , but of course allocating only in  $B(z_1, t)$ .
3. At the same time, for each value  $z_2$  of  $x(2)$  he applies the same strategy to the accumulation sequence  $a(z_1z_2x(3), t)$ , but of course allocating only in  $B(z_1z_2, t)$ .

And so on. A naive calculation suggests that the redundancies of the different levels add up (we lose a factor  $2^{-g}$  on each level). Since by Theorem 3.1 Server needs redundancy of at least  $\log \frac{3}{2}$ , if the game is played on  $n$  levels we would need redundancy proportional to  $n$  (the factors  $2^{-g}$  would multiply to  $2^{-ng}$ ).

This reasoning is misleading, however, since we know from Corollary 1.6 that the actual redundancy is bounded by  $K(n)$  and not  $n$ .

We will nevertheless use a version of Client's strategy  $\text{inflate-ints}(\cdot)$  and an appropriate variant of Lemma 3.3. Note that the lemma's conclusion is by itself quite weak, since it is a lower bound on (at least partly) *reservation*, not on allocation.

For future use, let us define reservation on strings: it is completely analogous to that on integers, but we add a variant dependent on time.

**Definition 4.6** (Reservation on strings). For any allocation function  $B(\cdot)$ , we will say that it *reserves* an  $r$ -bin  $[\mathbf{p}]$  for string  $\mathbf{x}$  if  $[\mathbf{p}]$  intersects  $[B(\mathbf{x})]$  but is disjoint from all other  $[B(\mathbf{y})]$  for  $\mathbf{y}$  incompatible with  $\mathbf{x}$ .

We will say that an allocation sequence  $B(\cdot, \cdot)$  reserves  $[\mathbf{p}]$  at time  $t$  for  $x$  if  $B(\cdot, t)$  reserves it for  $x$ . If the sequence  $B(\cdot, \cdot)$  is clear from the context then we will just say that  $[\mathbf{p}]$  is *reserved* for  $x$  at time  $t$ . Let again

$$B_r(\mathbf{x}, t)$$

denote the union of  $r$ -bins reserved for  $\mathbf{x}$  at time  $t$ , and

$$B_r(W, t) = \bigcup_{w \in W} B_r(w, t)$$

for a prefix-free set of strings  $W$ . For  $s_1 \leq s_2$ ,  $t_1 \leq t_2$  let

$$\overline{B}_{s_1}^{s_2}(\mathbf{x}, t_1, t_2) = \bigcup_{\substack{t_1 \leq u < t_2 \\ s_1 \leq r \leq s_2}} B_r(\mathbf{x}, u)$$

denote the union of  $r$ -bins with  $s_1 \leq r \leq s_2$  which have been reserved for  $\mathbf{x}$  during the time interval  $(t_1, t_2]$ . (They may or may not be still reserved for  $\mathbf{x}$  at time  $t_2$ .) Define the same way  $\overline{B}_{s_1}^{s_2}(W, t_1, t_2)$  for a prefix-free set of strings  $W$ . Denote

$$\overline{b}_r^s(\mathbf{x}, t, u) = \lambda(\overline{B}_r^s(\mathbf{x}, t, u)),$$

with a similar notation for prefix-free sets. ┘

Though  $\overline{B}_r^s(\mathbf{x}, t, u)$  is monotonic in  $u$ , the sets  $\overline{B}_{r_1}^{s_1}(\mathbf{x}, t_1, u_1)$  and  $\overline{B}_{r_2}^{s_2}(\mathbf{y}, t_2, u_2)$  are not necessarily disjoint for incompatible  $\mathbf{x}, \mathbf{y}$ . However, the following holds.

**Lemma 4.3.** *Let  $t_2 \geq u_1$ ,  $r_1 \geq s_2$  and let  $W$  be a prefix-free set of strings. We have*

$$\overline{B}_{r_1}^{s_1}(W, t_1, u_1) \cap \overline{B}_{r_2}^{s_2}(W, t_2, u_2) = \emptyset.$$

*Proof.* An  $r$ -bin once reserved for string  $\mathbf{x}$ , even if then abandoned, cannot be reserved later for any string incompatible with  $\mathbf{x}$ .  $\square$

## 5 Proof of the main theorem

We will prove that if  $g(\cdot)$  is a sub- $K$  redundancy sequence then Client has a winning strategy. So, we know  $\sum_{n=0}^{\infty} 2^{-g(n)} = \infty$ . Assume without loss of generality

$$\sum_{i=0}^{\infty} 2^{-g(2i+1)} = \infty.$$

Then from any  $M$  one can compute a lower approximation  $g_M(\cdot)$  of  $g(\cdot)$  such that

$$\sum_{i=0}^{\infty} 2^{-g_M(2i+1)} > M.$$

To simplify of notation from now on we assume that  $g(\cdot)$  is given as such a  $g_M(\cdot)$ , and return to this at the end of the proof.

**Notation 5.1.** Denote

$$\gamma_n(m) = 2^{-g(2n+1)} + 2^{-3} \sum_{i=n-m}^{n-1} 2^{-g(2i+1)}.$$

┘

Lemma 3.3 needs reformulation in order to become usable for multiple levels. We want to use its conclusion (“output”) as “input” in a renewed application of itself on a higher level (on shorter sequences). For this, we will recast it into a form where not only the conclusion speaks about reservation but also the assumption. Also, it will be about Game 2. From now on assume that a redundancy sequence  $g(\cdot)$  has been fixed.

The following definition defines the strings and times when Client has a chance to do harm:

**Definition 5.2.** We say that string  $\mathbf{w}$  is a  $(r, r')$ -*virgin* at time  $t$  if the following holds:

$$\begin{aligned} a(\mathbf{w}, t) &= 0, \\ \overline{B}_r^{r'}(\mathbf{w}, t) &= \emptyset. \end{aligned}$$

┘

Here is what the recursive strategy of Client needs to achieve.

**Definition 5.3.** Let  $0 \leq m \leq n$  be integers, where  $n$  will be fixed. For each  $m$ , we say that Client has an  $(n, m)$ -*expanding* strategy if there is a strategy

$$\text{inflate-seqs}(W, r, m, n),$$

a computable function

$$r'(r, m, n)$$

with  $r' = r'(r, m, n)$  and all  $\mathbf{w}$  a subset

$$\begin{aligned} \hat{B}_r^{r'}(\mathbf{w}, \tau | W, m) &\subseteq \overline{B}_r^{r'}(\mathbf{w}, \tau_0, \tau) && \text{with} \\ \hat{b}_r^{r'}(\mathbf{w}, \tau | W, m) &= \lambda(\hat{B}_r^{r'}(\mathbf{w}, \tau | W, m)), \end{aligned}$$

such that defining for any prefix-free set  $S$

$$\begin{aligned} \hat{B}_r^{r'}(S, \tau | W, m) &= \bigcup_{\mathbf{w} \in S} \hat{B}_r^{r'}(\mathbf{w}, \tau | W, m), \\ \hat{b}_r^{r'}(S, \tau | W, m) &= \lambda(\hat{B}_r^{r'}(S, \tau | W, m)) \end{aligned}$$

we have the following properties. Let

$$\eta = 8. \tag{5.1}$$

- (a) If an  $s$ -bin is contained in  $\hat{B}_r^{r'}(\mathbf{w}, \tau | W, m)$  then it is contained also in all  $\hat{B}_r^{r'}(\mathbf{w}', \tau | W, m)$  for any  $\mathbf{w}' \sqsubseteq \mathbf{w}$ .
- (b) Let  $W \subseteq \mathbb{N}^{2(n-m)}$ , let us further have some integer  $r \geq 0$  with  $r' = r'(r, m, n)$ . Suppose that all elements  $\mathbf{w} \in W$  are  $(r, r')$ -virgins at time  $\tau_0$ , and we have enough weight to spend:  $a(W, \tau_0) \leq 1 - |W|2^{-r}$ . Then Client can carry out the strategy. At its end, at time  $\tau''$ , we will have the following, for all  $\mathbf{w} \in W$ :

$$2^{-r-1} \leq a(\mathbf{w}, \tau'') \leq 2^{-r}, \tag{5.2}$$

$$\hat{b}_r^{r'}(W, \tau'' | W, m) \geq \gamma_n(m) a(W, \tau''), \tag{5.3}$$

$$\hat{b}_r^{r'}(\mathbf{w}, \tau'' | W, m) \leq \eta \gamma_n(m) a(\mathbf{w}, \tau''). \tag{5.4}$$

┘

Let us show that Definition 5.3 is not empty:

**Lemma 5.1.** *For every  $n$ , Client has an  $(n, 0)$ -expanding strategy.*

*Proof.* Let

$$r' = r'(r, 0, n) = r + g(2n + 1).$$

Let us define the  $(n, 0)$ -expanding strategy in Algorithm 5.1.

**Algorithm 5.1.** Client's strategy  $\text{inflate-seqs}(W, r, 0, n)$  for  $W \subseteq \mathbb{N}^{2n}$

**for**  $v \in W \setminus \{1, \dots, 2^{g(2n+1)}\}$  **do**  
 $a(v) \leftarrow 2^{-r'}$

This strategy increases the weight of the string  $\mathbf{w}0j$  to  $2^{-r'}$  and that of  $\mathbf{w}0$  to at least  $2^{-r'+g(2n+1)}$ , forcing Server to allocate an  $r'$ -bin to  $\mathbf{w}0$ . With starttime  $\tau_0$  and endtime  $\tau'' = \tau_0 + 1$  we define  $\hat{B}_r^{r'}(\mathbf{w}0, \tau'' | W, 0)$  as just the first  $r'$ -bin reserved for  $\mathbf{w}0$  at time  $\tau''$ . Then we have

$$\begin{aligned} \hat{b}_r^{r'}(\mathbf{w}, \tau'' | W, 0) &= 2^{-r'} = 2^{-g(2n+1)} a(\mathbf{w}, \tau'') = \gamma_n(0) a(\mathbf{w}, \tau''), \\ \hat{b}_r^{r'}(W, \tau'' | W, 0) &= |W| 2^{-r'} = \gamma_n(0) a(W, \tau''). \end{aligned} \tag{5.5}$$

□

**Lemma 5.2 (Main).** *If Client has an  $(n, m)$ -expanding strategy with  $m < n$  then she also has an  $(n, m + 1)$ -expanding strategy.*

Before proving the lemma let us use it to prove Theorem 4.1.

*Proof of Theorem 4.1.* Let us fix some  $n > 0$ . Now, Lemma 5.1 gives us an  $(n, 0)$ -expanding strategy. Applying the Main Lemma 5.2 inductively gives us an  $(n, n)$ -expanding strategy. Starting the strategy at time 0 gives us

$$1 \geq \bar{b}_r^{r'}(\Lambda, 0, t_1) \geq \gamma_n(n) a(\Lambda, t_1) \geq \gamma_n(n) 2^{-r-1} \geq 2^{-4-r} \sum_{i=0}^n 2^{-g_M(2i+1)}.$$

Now recall that for any  $M$ , Client can choose a computable  $g_M(\cdot)$  and an  $n$  with  $\sum_{i=0}^n 2^{-g_M(2i+1)} > M$ . With  $M = 17$ ,  $r = 0$  (and the strategy derived from  $g_M(\cdot)$ ), Client would cause contradiction in the last inequality and thus defeat Server. □

From now on, denote

$$\begin{aligned} \gamma &= \gamma_n(m), \\ g &= g(2(n - m) - 1). \end{aligned}$$

For the proof of Lemma 5.2, we define the bin sizes (and other quantities) for  $m + 1$  first (assuming they have been defined for  $m$ ):

$$\begin{aligned}
\sigma &= 2^{-1}\eta^{-1}(\gamma^{-1} \wedge 1), \\
h = h(m, n) &= \lceil 2^{g+4}/\sigma \rceil, \\
s_{h+4} &= r, \\
s_t &= r'(s_{t+1}, m, n) \quad \text{for } 0 \leq t < h + 4, \\
p &= s_h, \\
r'(r, m + 1, n) &= s_0, \\
L &= 2^{p-r-g-1}.
\end{aligned} \tag{5.6}$$

This defines the function  $r'(r, m, n)$  recursively. Now we define the strategy  $\text{inflate-seqs}(W, r, n)$  in Algorithm 5.2 for a set of strings  $W$  of even length  $< 2n$ . This algorithm incorporates Adam Day's ideas from [2]:

- Adding a “spend” phase to the “advantage” phase, to guarantee that Server's wasted space is identifiable at the end of the procedure.
- Using the added predictability for inflating not just a simple sequence  $\mathbf{w}$  but a set of sequences  $W$ .

**Notation 5.4.** Denote  $\mathbb{N}_i = \{0, \dots, i - 1\}$ . ┘

**Algorithm 5.2.** Client's strategy  $\text{inflate-seqs}(W, r, m + 1, n)$  for  $W \subseteq \mathbb{N}^{2(n-m-1)}$

```

for  $t = 0$  to  $h - 1$  do # advantage phase
   $V_0 = V_0(\tau_t) \leftarrow \{\mathbf{v} \in W\mathbb{N}_L : B_p(\mathbf{v}) \neq \emptyset\}$ 
  if  $|V_0| \geq |W|L/2$  then break from the loop
  for  $\mathbf{v} \in W\mathbb{N}_L \setminus V_0$  do
     $Y(\mathbf{v}) \leftarrow$  the set of smallest  $\leq \sigma 2^{s_t - p}$  integers  $y$  with  $a(\mathbf{v}y) = 0$ 
     $V_1 = V_1(\tau_t) \leftarrow \bigcup_{\mathbf{v} \in V_0} \mathbf{v}Y(\mathbf{v})$ 
    run  $\text{inflate-seqs}(V_1, s_t, m, n)$ 
   $t_1 \leftarrow t$ 
  for  $t = t_1$  to  $t_1 + 3$  do # spend phase
     $W' \leftarrow \{\mathbf{w} \in W : a(\mathbf{w}) < 2^{-r-1}\}$ 
    if  $W' = \emptyset$  then break from the loop
    for  $\mathbf{w} \in W'$  do
       $V_2(\mathbf{w}) \leftarrow$  the smallest  $2^{s_t - r - 1}$  strings in  $\mathbf{w}(L + t)\mathbb{N}$ 
     $V_1 = V_1(\tau_t) \leftarrow \bigcup_{\mathbf{w} \in W'} V_2(\mathbf{w})$ 
    run  $\text{inflate-seqs}(V_1, s_t, m, n)$ 
   $t_2 \leftarrow t$ 

```

Let

$$\tau' = \tau_{t_1}, \quad \tau'' = \tau_{t_2}$$

be the end of the advantage phase and the spend phase respectively. Define the set  $\hat{B}_r^r(\mathbf{w}, \tau | W, m+1)$  as follows. Let

$$\hat{B}_p = \bigcup_{\mathbf{v} \in V_0(\tau')} u(\mathbf{v}), \quad (5.7)$$

where  $u(\mathbf{v})$  is the first  $p$ -bin in  $B_p(\mathbf{v}, \tau')$ . We further define for all  $\tau \leq \tau'$

$$\begin{aligned} \hat{B}_p(\mathbf{w}\mathbb{N}_L, \tau) &= B_p(\mathbf{w}\mathbb{N}_L, \tau) \cap \hat{B}_p, \\ \hat{b}_p(\mathbf{w}\mathbb{N}_L, \tau) &= \lambda(\hat{B}_p(\mathbf{w}\mathbb{N}_L, \tau)). \end{aligned} \quad (5.8)$$

Let

$$\hat{B}_{s_t}^{r'}(\mathbf{w}\mathbb{N}_L\mathbb{N}, \tau_t) = \bigcup_{u=0}^{t-1} \hat{B}_{s_{u+1}}^{s_u}(\mathbf{w}\mathbb{N}_L\mathbb{N}, \tau_{u+1} | V_1(\tau_u), m), \quad (5.9)$$

$$\hat{B}_r^{r'}(\mathbf{w}\mathbb{N}_L, \tau'' | W, m+1) = \hat{B}_r^{r'}(\mathbf{w}, \tau'') \cup \hat{B}_p(\mathbf{w}, \tau'). \quad (5.10)$$

This concludes the definition of the strategy.

Assume that strategy  $\text{inflate-seqs}(W, r, m, n)$  is  $(n, m)$ -expanding. We prove that the strategy  $\text{inflate-seqs}(W, r, m+1, n)$  is  $(n, m+1)$ -expanding. In order to show the main inequality (5.5), we need to show that the strategy  $\text{inflate-seqs}(\cdot)$  ends at some time  $\tau''$  with

$$\hat{b}_r^{r'}(W, \tau'' | W, m+1) \geq a(W, \tau'')(\gamma + 2^{-g-3}).$$

The following observation is similar to the case of integers:

**Lemma 5.3.** *During the advantage phase we have*

$$a(\mathbf{w}x) \leq 2^{-p}(2^g + \sigma), \quad (5.11)$$

$$a(\mathbf{w}) \leq 2^{-r} \quad (5.12)$$

for all  $\mathbf{w} \in W$ ,  $x \in \mathbb{N}$

*Proof.* When the algorithm increases  $a(\mathbf{w}x)$  then  $\mathbf{w}x$  does not have a  $p$ -bin reserved, so it does not have a  $p$ -bin allocated either. Since Server must satisfy the redundancy condition we have still  $a(\mathbf{w}x) \leq 2^{-p+g}$ . The right-hand side grows by at most

$$2^{s_t-p} \sigma 2^{-s_t} = 2^{-p} \sigma$$

in the very step after which  $\mathbf{w}x$  gets a  $p$ -bin allocated.

We obtain the second inequality multiplying the first one by  $L$  and using  $g \geq 0$  and  $\sigma \leq 1$ :

$$2^{p-r-g-1} 2^{-p}(2^g + \sigma) = 2^{-r-1}(1 + \sigma 2^{-g}) \leq 2^{-r}.$$

□

The following lemma corresponds to Lemma 3.5 of the integers case.

**Lemma 5.4.** *At the end of the advantage phase, we have  $|V_0| \geq L|W|/2$  and thus in the definition of  $\hat{B}_r^{r'}(\mathbf{w}, \tau''|W, m+1)$  in (5.10) the contribution of (5.8) is at least*

$$2^{-p-1}L|W| = 2^{-r-g-2}|W|.$$

*Proof.* Unless the advantage phase ends with  $|V_0| \geq L|W|/2$ , its “for” loop runs to the end. Each iteration pours a weight of at least

$$(|W|L/2)2^{s_t-p}\sigma 2^{-s_t-1} = \sigma|W|2^{-r-g-3}$$

into  $a(\text{WN}_L)$ . The total weight is then at least  $h|W|\sigma 2^{-r-g-3}$  which, combined with inequality (5.12), gives

$$\begin{aligned} h|W|\sigma 2^{-r-g-3} &\leq |W| \cdot 2^{-r}, \\ h\sigma 2^{-g-3} &\leq 1, \end{aligned}$$

contradicting the definition of  $h$ . □

The following lemma corresponds to Lemma 3.6 of the integers case.

**Lemma 5.5.** *The following holds for all stages  $\tau_t$  during the application of strategy inflate-seqs( $\cdot$ ) in the advantage phase:*

$$\lambda\left(\hat{B}_{s_t}^{r'}(\text{WN}_L\mathbb{N}, \tau_t) \cup \hat{B}_p(\text{WN}_L, \tau_t)\right) \geq \gamma a(W, \tau_t) + 2^{-1}\hat{b}_p(\text{WN}_L, \tau_t), \quad (5.13)$$

$$\lambda\left(\hat{B}_{s_t}^{r'}(\mathbf{w}, \tau_t) \cup \hat{B}_p(\mathbf{w}, \tau_t)\right) \leq \eta\gamma a(\mathbf{w}, \tau_t) + \hat{b}_p(\mathbf{w}, \tau_t). \quad (5.14)$$

*In particular, we have at the end of this phase:*

$$\begin{aligned} \hat{b}_p^{r'}(\text{WN}_L\mathbb{N}, \tau') &\geq \gamma a(W, \tau') + 2^{-1}\hat{b}_p(\text{WN}_L, \tau'), \\ \hat{b}_p^{r'}(\mathbf{w}, \tau') &\leq \eta\gamma a(\mathbf{w}, \tau') + \hat{b}_p(\mathbf{w}, \tau'). \end{aligned}$$

*Proof.* The set  $\hat{B}_p(\mathbf{w}, \tau_t)$  defined in (5.8) is the union of those  $p$ -bins in the selected set  $\hat{B}_p$  that have already been reserved at time  $\tau_t$ . As such, it is monotonically increasing in  $t \leq t_1$ . Recall the set  $\hat{B}_{s_t}^{r'}(\mathbf{w}, \tau_t)$  from (5.9). We prove the inequalities by induction on  $t$ . Both inequalities hold for  $t = 0$ , when both sides are 0. Suppose they hold for  $t$ , we prove them for  $t + 1$ .

1. Consider the contribution of  $s$ -bins with  $s_{t+1} \leq s < s_t$  to the left side of (5.13). By choice of the extensions in the advantage stage, the set  $\bar{B}_{s_{t+1}}^{s_t}(\text{WN}_L\mathbb{N}, \tau_t, \tau_{t+1})$  is disjoint from  $B_p(\text{WN}, \tau_t)$ . The fact that the sequence  $s_i$  is nonincreasing and that all used extension strings are incompatible, implies via Lemma 4.3 that the set is also disjoint from  $\bar{B}_{s_t}^{r'}(W, \tau_0, \tau_t)$ , so it is disjoint from  $\hat{B}_{s_t}^{r'}(W, \tau_t) \cup \hat{B}_p(W, \tau_t)$ . Therefore the contribution of  $s$ -bins with  $s_{t+1} \leq s < s_t$  to the left side of (5.13) is

$$\hat{b}_{s_{t+1}}^{s_t}(V_1(\tau_t), \tau_{t+1}|V_1(\tau_t), m),$$

which is  $\geq \gamma a(V_1(\tau_t), \tau_{t+1})$  due to the  $(n, m)$ -expansion of the strategy that is applied between times  $\tau_t$  and  $\tau_{t+1}$  to the set  $V_1(\tau_t)$ .

For an upper bound, the same expansion property gives for each  $\mathbf{v} \in V_1(\tau_t)$ :

$$\hat{b}_{s_{t+1}}^{s_u}(\mathbf{v}, \tau_{t+1} | V_1(\tau_t), m) \leq \eta \gamma a(\mathbf{v}, \tau_{t+1}).$$

2. Now consider the contribution of the  $p$ -bins. Take any  $\mathbf{v} \in \mathbf{w}\mathbb{N}_L$  for which  $a(\mathbf{v}, \tau_{t+1}) - a(\mathbf{v}, \tau_t) \neq 0$ , and  $B_p(\mathbf{v}, \tau_{t+1}) \cap \hat{B}_p \neq \emptyset$ . By definition (5.7), the latter set consists of a single  $p$ -bin. It is disjoint from  $\overline{B}_{s_t}^r(W, \tau_0, \tau_t)$ , therefore would increase the left-hand side of (5.13) by  $2^{-p}$ . However, this contribution overlaps with the already counted contribution of the  $s$ -bins with  $s_{t+1} \leq s < s_t$ , which by (5.9) is

$$\hat{B}_{s_{t+1}}^{s_t}(\mathbf{v}\mathbb{N}, \tau_{t+1} | V_1(\tau_t), m).$$

The inductive assumptions' condition (5.4) provides an upper bound for this overlap:

$$\hat{b}_{s_t}^{s_{t+1}}(\mathbf{v}, \tau_{t+1}) \leq \eta \gamma (a(\mathbf{v}, \tau_{t+1}) - a(\mathbf{v}, \tau_t)) \leq \eta \gamma \sigma 2^{-p}.$$

So the total additional contribution of the  $p$ -bin newly reserved for  $\mathbf{v}$  to the left-hand side of (5.13) is, using the definition of  $\eta$  in (5.1) and the definition of  $\sigma$  in (5.6):

$$\geq (1 - \eta \gamma \sigma) 2^{-p} \geq 2^{-p-1} = 2^{-1} (\hat{b}_p(\mathbf{v}, \tau_{t+1}) - \hat{b}_p(\mathbf{v}, \tau_t)).$$

For the upper bound of the same contribution we can use

$$2^{-p} = \hat{b}_p(\mathbf{v}, \tau_{t+1}) - \hat{b}_p(\mathbf{v}, \tau_t).$$

3. Combining the contributions of the first and second kind shows that the increase on the left-hand side of (5.13) is at least

$$\begin{aligned} & \gamma (a(W, \tau_{t+1}) - a(W, \tau_t)) + 2^{-1} \sum_{\mathbf{v} \in \mathbf{w}\mathbb{N}_L} \hat{b}_p(\mathbf{v}, \tau_{t+1}) - \hat{b}_p(\mathbf{v}, \tau_t) \\ &= \gamma (a(W, \tau_{t+1}) - a(W, \tau_t)) + 2^{-1} (\hat{b}_p(W, \tau_{t+1}) - \hat{b}_p(W, \tau_t)), \end{aligned}$$

proving the induction step for the inequality (5.13). With a similar combination, the upper bound for the increase of the left-hand side of (5.14) gives

$$\begin{aligned} & \eta \gamma (a(\mathbf{w}, \tau_{t+1}) - a(\mathbf{w}, \tau_t)) + \sum_{\mathbf{v} \in \mathbf{w}\mathbb{N}_L} \hat{b}_p(\mathbf{v}, \tau_{t+1}) - \hat{b}_p(\mathbf{v}, \tau_t) \\ &= \eta \gamma (a(\mathbf{w}, \tau_{t+1}) - a(\mathbf{w}, \tau_t)) + \hat{b}_p(\mathbf{w}, \tau_{t+1}) - \hat{b}_p(\mathbf{w}, \tau_t), \end{aligned}$$

proving the induction step for the inequality (5.14). □

Consider the spend phase now.

**Lemma 5.6.** *For all  $\mathbf{w} \in W$ , at the end of the spend phase we have  $2^{-r-1} \leq a(\mathbf{w}) \leq 2^{-r}$ .*

*Proof.* Lemma 5.3 shows that the upper bound holds at the start of the spend phase. For any particular  $\mathbf{w} \in W$ , suppose that  $\mathbf{w}$  leaves the set  $W'$  at a certain point of the iteration in the spend phase. Then we have  $a(\mathbf{w}) < 2^{-r-1}$  before the iteration, which adds at most  $2^{-r-1}$  to the weight.

Since the iteration adds at least  $2^{-r-2}$  to the weight of each  $\mathbf{w}$ , in at most 4 iterations each string  $\mathbf{w}$  reaches the desired weight.  $\square$

*Proof of Lemma 5.2.* Lemmas 5.5 and 5.4 imply for the advantage phase

$$\hat{b}_p^{r'}(\mathbb{W}\mathbb{N}_L, \tau') \geq \gamma a(W, \tau') + 2^{-1} \hat{b}_p(\mathbb{W}\mathbb{N}_L, \tau') \geq \gamma a(W, \tau') + 2^{-r-g-3} |W|, \quad (5.15)$$

$$\hat{b}_p^{r'}(\mathbf{w}, \tau') \leq \eta \gamma a(\mathbf{w}, \tau') + \hat{b}_p(\mathbf{w}, \tau') \leq \eta \gamma a(\mathbf{w}, \tau') + 2^{-p} L = \eta \gamma a(\mathbf{w}, \tau') + 2^{-r-g-1}. \quad (5.16)$$

The contribution of the spend phase to  $\hat{b}_r^{r'}(W, \tau'')$  is disjoint from that of the advantage phase:

$$\hat{b}_r^{r'}(\mathbb{W}\mathbb{N}_L\mathbb{N}, \tau'') = \hat{b}_p^{r'}(\mathbb{W}\mathbb{N}_L\mathbb{N}, \tau') + \sum_{u=t_1}^{t_2-1} \hat{b}_{s_{u+1}}^{s_u}(\mathbb{W}\mathbb{N}_L\mathbb{N}, \tau_{u+1} | V_1(\tau_u), m).$$

The inductive assumption gives for the sum the lower bound  $\gamma(a(W, \tau'') - a(W, \tau'))$ . Combining this with (5.15) and the bound of Lemma 5.6 gives

$$\hat{b}_r^{r'}(W, \tau'') \geq \gamma a(W, \tau'') + 2^{-r-g-3} |W| \geq a(W, \tau'')(\gamma + 2^{-g-3}),$$

proving (5.3) for  $m + 1$ .

The upper bound is obtained similarly. Lemma 5.5 gives

$$\hat{b}_p^{r'}(\mathbf{w}, \tau') \leq \eta \gamma a(\mathbf{w}, \tau') + 2^{-r-g-1}.$$

The spend phase adds at most

$$\sum_{u=t_1}^{t_2-1} \hat{b}_{s_{u+1}}^{s_u}(\mathbf{w}, \tau_{u+1} | V_1(\tau_u), m).$$

The inductive assumption gives for the sum the upper bound  $\eta \gamma (a(\mathbf{w}, \tau'') - a(\mathbf{w}, \tau'))$ . Combining this with (5.16), the bound  $2^{-r} \leq 2a(\mathbf{w}, \tau'')$  of Lemma 5.6 and the definition of  $\eta$  in (5.1) gives

$$\begin{aligned} \hat{b}_r^{r'}(\mathbf{w}, \tau'') &\leq \eta \gamma a(\mathbf{w}, \tau'') + 2^{-r-g-1} \leq a(\mathbf{w}, \tau'')(\eta \gamma + 2^{-g}) \\ &\leq \eta a(\mathbf{w}, \tau'')(\gamma + 2^{-g-3}) \end{aligned}$$

proving (5.4) for  $m + 1$ .  $\square$

## 6 The lower bound for binary sequences

The original proof of Theorem 1.2 used a recursion that forced Client's strategy to introduce very large numbers, but using Adam Day's idea this is no longer the case. What results is an improved lower bound for binary sequences in Theorem 1.3.

*Proof of Theorem 1.3.* Let us upperbound the number of sequences of length  $2n$  used by Client's strategy  $\text{inflate-seqs}(\Lambda, 0, n)$ . Define the sequence  $r_m$  recursively by  $r_0 = 0$ ,

$$r_{m+1} = r'(r_m, m, n).$$

If we assume (since we can) for an appropriate  $\varepsilon$ :

$$g(2n + 1) \leq (1 + \varepsilon) \log n, \quad (6.1)$$

$$\gamma_n(m) \leq \log n, \quad (6.2)$$

then for

$$k = \max_m h(m, n) + 4,$$

following estimates hold.

$$\begin{aligned} r'(r, 0, n) &= r + g(2n + 1) \leq r + (1 + \varepsilon) \log n, \\ r'(r, 1, n) &\leq r + (1 + \varepsilon)k \log n, \\ r'(r, 2, n) &\leq r + (1 + \varepsilon)k^2 \log n, \\ &\dots \\ r'(r, m, n) &\leq r + (1 + \varepsilon)k^m \log n. \end{aligned}$$

where, using (6.1):

$$\begin{aligned} k &\leq 2^{(1+\varepsilon)\log n + 4} / \sigma + 6 \leq 32n^{1+\varepsilon} \log n + 6, \\ k^n \log n &\leq 2^{(1+2\varepsilon)n^{1+\varepsilon}} \end{aligned}$$

for large  $n$ . Strategy  $\text{inflate-seqs}(W, r, n)$  with  $W \subseteq \mathbb{N}^{2(n-m-1)}$  uses recursive calls on at most

$$2^{r'-r} \leq 2^{(1+\varepsilon)k^m \log n}$$

sequences of length  $2(n - m)$ . The total number of sequences of length  $2n$  used is, for large  $n$  therefore at most

$$2^{(1+\varepsilon)\log n(k^0 + k^1 + \dots + k^{n-1})} \leq 2^{(1+2\varepsilon)k^n \log n} \leq 2^{(1+2\varepsilon)2^{(1+2\varepsilon)n^{1+\varepsilon}}}$$

for large  $n$ . It follows that we can encode these strings with binary strings of length  $L = (1 + 2\varepsilon)2^{(1+2\varepsilon)n^{1+\varepsilon}}$ , thus  $n \leq (\log L)^{1-2\varepsilon}$  for large  $n$ . On such strings then, the redundancy bound cannot be less than  $\log n$  for sufficiently large  $n$ .

For the difference  $K(x) - H(x)$ , this gives for infinitely many binary strings of length  $L$  a lower bound of  $(1 - 2\varepsilon) \log \log L$ .  $\square$

## References

- [1] G. Chaitin, A theory of program size formally identical to information theory. *J. Assoc. Comput. Mach.*, 22:329–340, 1975. [2.4](#)
- [2] Adam Day, Increasing the gap between Descriptive Complexity and Algorithmic Probability. See [http://homepages.mcs.vuw.ac.nz/~adam/papers/day\\_monotone\\_a\\_priori.pdf](http://homepages.mcs.vuw.ac.nz/~adam/papers/day_monotone_a_priori.pdf). [1](#), [1](#), [5](#)
- [3] Peter Gacs, *On the relation between descriptive complexity and algorithmic probability*, *Theoretical Computer Science* **22** (1983), 71–93, Short version: Proc. 22nd IEEE FOCS (1981) 296-303. [1](#), [1](#)
- [4] Peter Gács, *Lecture notes on descriptive complexity and randomness*, Tech. report, Boston University, Computer Science Dept., Boston, MA 02215, 1988. See the new version in [www.cs.bu.edu/~gacs/papers/ait-notes.pdf](http://www.cs.bu.edu/~gacs/papers/ait-notes.pdf). [1](#)