TRANSACTIONS OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 363, Number 10, October 2011, Pages 5577–5604
S 0002-9947(2011)05315-8
Article electronically published on May 5, 2011

INCREASING THE GAP BETWEEN DESCRIPTIONAL COMPLEXITY AND ALGORITHMIC PROBABILITY

ADAM R. DAY

Abstract. The coding theorem is a fundamental result of algorithmic information theory. A well-known theorem of Gács shows that the analog of the coding theorem fails for continuous sample spaces. This means that descriptional monotonic complexity does not coincide within an additive constant with the negative logarithm of algorithmic probability. Gács's proof provided a lower bound on the difference between these values. He showed that for infinitely many finite binary strings, this difference was greater than a version of the inverse Ackermann function applied to string length. This paper establishes that this lower bound can be substantially improved. The inverse Ackermann function can be replaced with a function $O(\log(\log(x)))$. This shows that in continuous sample spaces, descriptional monotonic complexity and algorithmic probability are very different. While this proof builds on the original work by Gács, it does have a number of new features; in particular, the algorithm at the heart of the proof works on sets of strings as opposed to individual strings.

1. Introduction

In this paper we seek to analyze the relationship between two different ways of understanding the intrinsic algorithmic randomness of strings and real numbers. In the setting of discrete spaces, a cornerstone of the theory of prefix-free Kolmogorov complexity is the *coding theorem*.[1] This theorem says that the negative logarithm of the probability that a string is the output of a universal prefix-free machine coincides, within some additive constant, with the length of its shortest description. It has been suggested that this ties two fundamental principles together: Bayes' theorem and Occam's razor [4, 7]. Certainly, the result lends support to the belief that prefix-free complexity is a natural measure of the algorithmic randomness of strings.

In continuous sample spaces, the issue is less clear. Perhaps the most natural way of investigating complexity for real numbers in Cantor space is using monotone machines. One justification for this view is that the probability that a string is the output of a universal monotone machine (the *algorithmic probability*) is an optimal continuous semimeasure on Cantor space (see Theorem 4 due to Levin [6]). We will use $K_m(\sigma)$ to indicate the monotonic descriptional complexity of a string σ and $KM(\sigma)$ for the negative logarithm of its algorithmic probability. Levin conjectured that the natural analog of the coding theorem held in the setting of continuous

Received by the editors October 21, 2009 and, in revised form, February 10, 2010.
2000 *Mathematics Subject Classification.* Primary 68Q30.

[1]These statements will be made precise in the definitions and theorems to follow.

sample spaces as well, i.e. that K_m and KM would coincide within an additive constant. Gács, however, showed that this was not the case [4].

Theorem 1 (Gács)**.** *For any $d \in \mathbb{N}$, there exists a finite binary string σ such that*

$$K_m(\sigma) - KM(\sigma) > d.$$

One is naturally led to investigate the relationship between a given d and the corresponding σ. What Gács actually proved was that there exists some version of the inverse Ackermann function A^{-1}, such that for infinitely many strings σ, the difference between $K_m(\sigma)$ and $KM(\sigma)$ exceeded $A^{-1}(|\sigma|)$ (see Theorem 6). The inverse Ackermann function is *extremely* slow growing. If the difference between these two complexities was no more than A^{-1}, then the difference might be seen as essentially negligible. Since Gács's result, over twenty-five years ago, an important open question in algorithmic information theory has been whether this lower bound on the difference between K_m and KM can be improved. The goal of this paper is to show that this lower bound can be improved substantially from the inverse Ackermann function to $c \log\log |\sigma|$ for some constant c. The conclusion is that in continuous sample spaces these concepts are very different indeed.

Before giving a detailed statement of the theorem we will prove, we will review prefix-free machines and monotone machines. We will see how both these machines relate to certain types of measures.

1.1. **Basic definitions, results and conventions.** The set of all binary strings of length n, the set of all finite binary strings, and the set of all infinite binary strings will be denoted by $\{0,1\}^n$, $2^{<\omega}$, and 2^ω respectively. The empty string will be represented by λ. The relation $\preceq$ on $2^{<\omega} \times (2^{<\omega} \cup 2^\omega)$ is defined by $\sigma \preceq \tau$ if σ is an initial segment of τ. If $\sigma \preceq \tau$ or $\tau \preceq \sigma$, then σ and τ are said to be comparable. This will be written $\sigma \approx \tau$. Otherwise, σ and τ are incomparable and this will be written $\sigma \mid \tau$. The operation of appending a string τ to the end of a finite string σ will be represented by $\sigma\tau$. If $\Sigma, \Upsilon \subseteq 2^{<\omega}$, then $\Sigma\Upsilon = \{\sigma\upsilon : \sigma \in \Sigma \text{ and } \upsilon \in \Upsilon\}$.

Cantor space is the topology on 2^ω defined by taking $\{[\sigma] : \sigma \in 2^{<\omega}\}$, where $[\sigma] = \{\sigma\alpha : \alpha \in 2^\omega\}$, as a basis of open sets. If $X \subseteq 2^{<\omega}$, then $[X] = \bigcup_{\sigma \in X} [\sigma]$. The Lebesgue measure on Cantor space μ is the outer measure obtained by defining $\mu([\sigma]) = 2^{-|\sigma|}$ for all open sets $[\sigma]$ in the basis where $|\sigma|$ is the length of σ.

We need the following notion of effectiveness for the reals. The *dyadic rationals* are $\mathbb{Q}_2 = \{z2^{-n} : z \in \mathbb{Z}, n \in \mathbb{N}\}$. We say that a real α is *left-c.e.* if $\{q \in \mathbb{Q}_2 : q < \alpha\}$ is a computably enumerable set. Note that it can be shown that α is left-c.e. if and only if $\alpha = \lim_{n\to\infty} q_n$, for some nondecreasing computable sequence of $\{q_n\}_{n\in\mathbb{N}}$ of dyadic rationals [8]. A function $f : X \to \mathbb{R}$ is *uniformly left c.e.* if there is computable function $f : X \times \mathbb{N} \to \mathbb{Q}_2$, nondecreasing in the second variable such that for all $x \in X$, $f(x) = \lim_{n\to\infty} f(x,n)$. We call $f : X \times \mathbb{N} \to \mathbb{Q}_2$ an *enumeration* of $f : X \to \mathbb{R}$.

Logs used are all base 2 and are rounded up to the nearest integer value. By convention $\log 0 = 0$.

1.2. **Prefix-free machines and the coding theorem.** A subset $X \subseteq 2^{<\omega}$ is *prefix-free* if for all distinct $\sigma, \tau \in X$, $\sigma \mid \tau$. A *prefix-free machine* is a partial computable function $P : 2^{<\omega} \to 2^{<\omega}$ such that the domain of P is prefix-free. It is possible to enumerate all prefix-free machines $P^0, P^1, \ldots$ and then construct a

universal prefix-free machine U, by defining $U(0^e1\tau) = P^e(\tau)$. This allows us to define the prefix-free complexity K as follows.

Definition. The *prefix-free complexity* $K(\sigma) = \min\{|\tau| : U(\tau) = \sigma\}$.

One reason prefix-free complexity has provided significant insight into algorithmic randomness is the ease with which prefix-free machines can be created. Often there is no need to construct a prefix-free machine directly. A fundamental result of algorithmic information theory is the coding theorem. This theorem establishes a correspondence between prefix-free complexity and a class of measures. Rather than build a prefix-free machine directly, it is often easier to construct a measure from this class and invoke the coding theorem.

A *discrete probability measure* on $2^{<\omega}$ is a function $m^D : 2^{<\omega} \to \mathbb{R}^{\geq 0}$ such that $\sum_{\sigma\in 2^{<\omega}} m^D(\sigma) = 1$. If we relax the requirement on m^D to $\sum_{\sigma\in 2^{<\omega}} m^D(\sigma) \leq 1$, and require m^D to be uniformly left c.e., then we call m^D a *c.e. discrete semimeasure*. We can think of a prefix-free machine P, as giving rise to a c.e discrete semimeasure Q_P, where $Q_P(\sigma) = \sum_{P(\tau)=\sigma} 2^{-|\tau|}$. It turns out that this is the only way to create a c.e. discrete semimeasure because given any c.e. discrete semimeasure m^D, there is a prefix-free machine P, such that $m^D = Q_P$ [2]. This fact gives us the following theorem [7].

Theorem 2. *If U is a universal prefix-free machine, then Q_U majorizes all c.e. discrete semimeasures.*

Where majorizes means that if m^D is a c.e. discrete semimeasure, then there exists a positive constant $c \in \mathbb{R}$ such that for all $\sigma \in 2^{<\omega}$, $Q_U(\sigma) \geq c \cdot m^D(\sigma)$. The relationship between c.e. discrete semimeasures and prefix-free machines does not end here. The coding theorem tells us that K and $-\log Q_U$ agree within an additive constant [7].[2] Thus in discrete spaces, the length of the shortest description and the algorithmic probability are really different ways of looking at the same thing.

Theorem 3 (The coding theorem)**.** *For all $\sigma \in 2^{<\omega}$,*

$$K(\sigma) = -\log(Q_U(\sigma)) + O(1).$$

1.3. **Continuous semimeasures and monotone machines.** A natural way to view a real number is as an infinite binary sequence, i.e., an element of 2^ω. Hence real numbers can be thought of as points in Cantor space. To examine the randomness of reals, a natural tool to use would be a c.e. measure on Cantor space.

Definition. A *c.e. continuous semimeasure* is a uniformly left c.e. function $m : 2^{<\omega} \to \mathbb{R}^{\geq 0}$ such that

(i) $m(\lambda) \leq 1$, and
(ii) for all $\tau \in 2^{<\omega}$, $m(\tau) \geq m(\tau 0) + m(\tau 1)$.

We noted that prefix-free machines give rise to c.e. discrete semimeasures. For c.e. continuous semimeasures, the corresponding role is played by monotone machines.

Definition. A *monotone machine* L is a computably enumerable set of pairs of finite binary strings $\langle\tau, \sigma\rangle$ such that if $\langle\tau_1, \sigma_1\rangle, \langle\tau_2, \sigma_2\rangle \in L$ and $\tau_1 \preceq \tau_2$, then $\sigma_1 \approx \sigma_2$.

[2]This is the same as saying that the weight of the shortest description of a string σ, $2^{-K(\sigma)}$, and $Q_U(\sigma)$ agree within a multiplicative constant.

For example, given a computable real α, a monotone machine M could be created by enumerating $\langle \tau, \alpha \upharpoonright n \rangle$ into M at stage n. In this case, τ is a finite description of α. One reason it can be argued that monotone machines are more suited for characterizing the complexity of reals is that noncomputable reals can be considered as limits of computable reals rather than limits of strings [1].

If L is a monotone machine, we take an enumeration of L and define L_t to be the result of enumerating L for t steps. Again we can take an enumeration $L^0, L^1, \ldots$ of all monotone machines and define a universal monotone machine U as follows: we add $\langle 0^e 1\tau, \sigma \rangle$ to U_t if and only if $\langle \tau, \sigma \rangle \in L^e_t$ for some $e \leq t$.

Definition. The *monotonic descriptional complexity* of a binary string σ is

$$K_m(\sigma) = \min\{|\tau| : \exists \sigma' \in 2^{<\omega} \text{ such that } \langle \tau, \sigma' \rangle \in U \text{ and } \sigma \preceq \sigma'\}.$$

This definition is due to Levin [5]. It builds on earlier work by Levin and Zvonkin [6]. Schnorr independently developed similar ideas [9]. Note that if τ is a description of σ, then τ is a description of any initial segment of σ as well. We will need a computable approximation to K_m so we will take $K_{m,t}$ to be defined as above except with U_t in place of U.

We noted that any c.e. discrete semimeasure could be described by a prefix-free machine. The following theorem, due to Levin, shows that monotone machines play an analogous role for c.e. continuous semimeasures [6].

Theorem 4 (Levin)**.** (i) *If L is a monotone machine, then the function $M_L : 2^{<\omega} \to \mathbb{R}^{\geq 0}$ defined by $M_L(\sigma) = \mu([\{\tau : \exists \sigma' \succeq \sigma(\langle \tau, \sigma' \rangle \in L)\}])$ is a c.e. continuous semimeasure.*

(ii) *If m is a c.e continuous semimeasure, then there exists a monotone machine L such that $M_L = m$.*

(iii) *If U is a universal monotone machine, then M_U majorizes all c.e. continuous semimeasures.*

If U is a universal monotone machine, we call $M_U(\sigma)$ the *monotonic algorithmic probability*.[3] Now if the coding theorem could be adapted to monotone machines and c.e. continuous semimeasures as well, then we would have that $K_m(\sigma) = -\log M_U(\sigma) + O(1)$. Gács showed this is false (Theorem 1). Hence there are two types of complexity that arise from monotone machines. It is easier to consider $-\log M_U$, so we we make the following definition, again due to Levin [5].

Definition. $KM(\sigma) = -\log(M_U(\sigma))$.

It is clear that for any σ, $KM(\sigma) \leq K_m(\sigma)$ because if $K_m(\sigma) = n$, then there is some τ of length n and $\sigma' \succeq \sigma$ such that $\langle \tau, \sigma' \rangle \in U$, so $M_U(\sigma) \geq \mu([\tau]) = 2^{-n}$; thus $KM(\sigma) \leq -\log 2^{-n} = n$.

Gács proved that KM and K_m do not agree within an additive constant. The question is, how different are these two complexities? The following theorem gives an upper bound on the difference between the two complexities [4], [11]. First we inductively define $\log^k x$ by $\log^1 x = \log x$ and $\log^{n+1} x = \log \log^n x$.

Theorem 5. *For any $k \in \mathbb{N}$, $\epsilon \in \mathbb{R}^{>0}$, with $k \geq 1$:*

$$\begin{aligned} K_m(\sigma) &\leq KM(\sigma) + K(|\sigma|) + O(1) \\ &\leq KM(\sigma) + \log|\sigma| + \log\log|\sigma| + \ldots + (1+\epsilon)\log^k|\sigma| + O(1). \end{aligned}$$

[3]This is also known in the literature as the *a priori complexity*.

This theorem is still true if K_m is replaced by K [11]. It would be surprising if this upper bound was tight as it would imply a prefix-free machine was just as good at approximating KM as a monotone machine.

Now Gács's proof of Theorem 1 was set in the context of integer strings (as opposed to binary strings). However, out of the construction of the proof, Gács developed the following lower bound for binary strings [4].

Theorem 6 (Gács)**.** *There exists a computable unbounded function A^{-1}, where A^{-1} is some version of the inverse Ackermann function such that for infinitely many σ, $K_m(\sigma) > KM(\sigma) + A^{-1}(|\sigma|)$.*

The gap between the upper and lower bounds provided by Theorems 5 and 6 is enormous. In general, the upper and lower bounds between most Kolmogorov complexities are very close [7], [11].

The goal of this paper is to prove a very strong lower bound. Our objective is to prove the following theorem.

Theorem 7. *If $c \in \mathbb{R}$, and $c < 1$, then there exist infinitely many $\sigma \in 2^{<\omega}$ such that*

$$K_m(\sigma) > KM(\sigma) + c \log\log|\sigma|.$$

To show that this theorem holds, we will develop a proof that monotonic descriptional complexity and monotonic algorithmic probability do not agree within an additive constant. The improvement in the lower bound follows from the construction used in the proof. This proof builds on the original work of Gács but is more intricate. A main point of difference is that the algorithm at the heart of the new proof works on sets of strings as opposed to individual strings. It is fair to say that Gács's theorem is not well understood. We hope that the new proof, as well as improving the bound, clarifies the main ideas as to why K_m and KM are different.

We now give a proof of Theorem 4. Levin outlined a construction to prove part (ii) in [6]. However, as there does not seems to be any complete proof in the literature, it is worthwhile presenting a proof in full.

Proof of Theorem 4. (i) Clearly $M_L(\lambda) \leq \mu([2^{<\omega}]) = 1$. Secondly $[\{\tau : \exists\sigma' \succeq \sigma(\langle\tau,\sigma'\rangle \in L)\}] \supseteq [\{\tau : \exists\sigma' \succeq \sigma 0(\langle\tau,\sigma'\rangle \in L)\}] \cup [\{\tau : \exists\sigma' \succeq \sigma 1(\langle\tau,\sigma'\rangle \in L)\}]$, and $[\{\tau : \exists\sigma' \succeq \sigma 0(\langle\tau,\sigma'\rangle \in L)\}] \cap [\{\tau : \exists\sigma' \succeq \sigma 1(\langle\tau,\sigma'\rangle \in L)\}] = \emptyset$ (because L is a monotone machine). It follows that $M_L(\sigma) \geq M_L(\sigma 0) + M_L(\sigma 1)$. M_L is uniformly left c.e., as $M_L(\sigma) = \lim_{t\to\infty} M_{L_t}(\sigma)$, $M_{L_t}(\sigma) \in \mathbb{Q}_2^{\geq 0}$, and $M_{L_t}(\sigma)$ is clearly nondecreasing in t.

(ii) We will make use of the following lemma:

Lemma 1. *If T, S are two subsets of $2^{<\omega}$ such that $\forall\sigma \in T \cup S$, $|\sigma| \leq l$, then there exists a set $R \subseteq \{0,1\}^l$ such that $[R] = [T] \setminus [S]$.*

Proof. Let $R = \{\sigma \in \{0,1\}^l : [\sigma] \subseteq [T] \text{ and } [\sigma] \cap [S] = \emptyset\}$. It quickly follows that $[R] \subseteq [T]$, and $[R] \cap [S] = \emptyset$ so $[R] \subseteq [T] \setminus [S]$. Now if $\alpha \in [T] \setminus [S]$, $[\alpha \upharpoonright l] \subseteq [T]$ because there must be some string of length $\leq l$ in T that covers α. No string in S covers α and as all strings in S have length less than or equal to l, so $[\alpha \upharpoonright l] \cap [S] = \emptyset$. Hence $\alpha \upharpoonright l \in R$ and so $[T] \setminus [S] \subseteq [R]$. □

Let $m : 2^{<\omega} \to \mathbb{R}^{\geq 0}$ be any c.e. semimeasure. We can take an enumeration $m : 2^{<\omega} \times \mathbb{N} \to \mathbb{Q}_2$ of m such that for all $t \in \mathbb{N}$:

(a) For all $\sigma \in 2^{<\omega}$, if $|\sigma| \geq t$, then $m(\sigma, t) = 0$.
(b) There is at most one string σ such that $m(\sigma, t+1) \neq m(\sigma, t)$.
(c) If $m(\sigma, t+1) \neq m(\sigma, t)$, then $m(\sigma, t+1) = m(\sigma, t) + n2^{-(t+1)}$ for some $n \in \mathbb{N}$.
(d) For all $\sigma \in 2^{<\omega}$, $m(\sigma, t) \geq m(\sigma 0, t) + m(\sigma 1, t)$.

Given this enumeration of m, we construct a monotone machine L using an enumeration of L that has a certain set of properties. First define $D_t(\sigma) = \{\tau \in 2^{<\omega} : \langle \tau, \sigma \rangle \in L_t\}$. We will ensure that our enumeration of L has the following properties, for all $\sigma \in 2^{<\omega}$ and $t \in \mathbb{N}$:

(a) L_t is a monotone machine.
(b) $D_t(\sigma)$ is a prefix-free set.
(c) $\mu([D_t(\sigma)]) = m(\sigma, t)$.
(d) If $\tau \in D_t(\sigma)$, then $|\tau| \leq t$.
(e) If $\sigma \prec \sigma'$, then $[D_t(\sigma)] \supseteq [D_t(\sigma')]$.

The construction of L proceeds as follows. Set $L_0 = \emptyset$. L_0 has the desired properties trivially.

At stage $t+1$, if for all σ such that $|\sigma| < t$, $m(\sigma, t+1) = m(\sigma, t)$, then let $L_{t+1} = L_t$; i.e. if the semimeasure does not change, then do not change the machine. Otherwise, let σ be the unique string such that $m(\sigma, t+1) \neq m(\sigma, t)$. Let $n \in \mathbb{N}$ be such that $m(\sigma, t+1) = m(\sigma, t) + n2^{-(t+1)}$. If $\sigma \neq \lambda$, then let $\rho = \sigma \upharpoonright (|\sigma| - 1)$. By the properties of our enumeration of L it follows that

$$\begin{aligned} \mu([D_t(\rho)] \setminus ([D_t(\rho 0)] \cup [D_t(\rho 1)]) &= \mu([D_t(\rho)]) - \mu([D_t(\rho 0)]) - \mu([D_t(\rho 1)]) \\ &= m(\rho, t) - m(\rho 0, t) - m(\rho 1, t) \\ &\geq n2^{-(t+1)}. \end{aligned}$$

The first equality holds because $([D_t(\rho 0)] \cup [D_t(\rho 1)]) \subseteq [D_t(\rho)]$, and as L_s is a monotone machine, $[D_t(\rho 0)] \cap [D_t(\rho 1)] = \emptyset$. By Lemma 1, there is a set R of strings of length $t+1$ such that $[R] = [D_t(\rho)] \setminus ([D_t(\rho 0)] \cup [D_t(\rho 1)])$. So $\mu([R]) \geq n2^{-(t+1)}$. We choose n strings of length $t+1$ from R and let this set be T_t. If $\sigma = \lambda$, then we find a set $R \subseteq \{0,1\}^{t+1}$ such that $[R] = [\lambda] \setminus [D_t(\lambda)]$; take T_t to be n strings from this R.

Set $L_{t+1} = L_t \cup \{\langle \tau, \sigma \rangle : \tau \in T_t\}$.

We will show that the desired properties of our enumeration of L hold. L_{t+1} is a monotone machine. If $\sigma = \lambda$, this is trivial as λ is comparable with any string. If $\sigma \neq \lambda$, then take any $\tau \in T_t$ and consider any $\upsilon \prec \tau$ such that $\langle \upsilon, \pi \rangle \in L_t$. We will show that $\pi \prec \sigma$. First note, because of the way we selected τ, there exists some $\tau' \prec \tau$ with $\langle \tau', \rho \rangle \in L_s$. Hence $\tau' \approx \upsilon$ so $\pi \approx \rho$. If $\pi \succeq \rho 0$ or $\pi \succeq \rho 1$, then $[\upsilon] \subseteq ([D_t(\rho 0)] \cup [D_s(\rho 1)])$ and so τ would not have been chosen. Hence $\pi \preceq \rho \prec \sigma$.

$D_{t+1}(\sigma)$ is a prefix-free set because $D_t(\sigma)$ and T_t are prefix-free and none of the elements in T_t extend elements in $D_t(\sigma)$. $\mu([D_{t+1}(\sigma)]) = \mu([D_t(\sigma)]) + \mu([T_t]) = m(\sigma, t) + n2^{-(t+1)} = m(\sigma, t+1)$. The final two properties hold by construction.

As for all t, L_t is a monotone machine, it follows that L is a monotone machine. Note that $M_{L_t}(\sigma) = \mu([D_t(\sigma)])$, by property (e). This gives us that for all $\sigma \in 2^{<\omega}$,

$$\begin{aligned} M_L(\sigma) &= \lim_{t\to\infty} M_{L_t}(\sigma) \\ &= \lim_{t\to\infty} \mu([D_t(\sigma)]) \\ &= \lim_{t\to\infty} m(\sigma, t) = m(\sigma). \end{aligned}$$

(iii) Follows from (ii) and the universality of U. □

2. Overview of proof

From now on we will refer to a c.e. continuous semimeasure as just a c.e. semimeasure. We will also drop the word monotonic and just talk about descriptional complexity and algorithmic probability. We know that M_U majorizes all c.e. semimeasures. So, if it was true that $K_m(\sigma) = KM(\sigma) + O(1)$, then for any c.e. semimeasure m that we could construct, there would be some constant c such that $K_m(\sigma) \leq -\log(m(\sigma)) + c$. The general approach of the proof will be to build a c.e. semimeasure for which this is false. This means constructing a c.e. semimeasure m with the property that $\forall c \in \mathbb{N}$, $\exists \sigma \in 2^{<\omega}$ such that $K_M(\sigma) > -\log(m(\sigma)) + c$.

Of course we want to do more than this and prove Theorem 7. This result will come out of the construction used in the proof. For now we will just focus on showing that K_m and KM are different. We will leave the analysis of the size of the difference until later.

Another way to look at this problem is to consider whether for any $n \in \mathbb{N}$, we can construct a semimeasure m such that $m(\lambda) \leq 2^{-n}$ and there exists a σ such that $K_m(\sigma) > -\log(m(\sigma))$. Viewing the problem this way helps simplify the proof a little. This is because we can build a c.e. semimeasure $a(\sigma)$ such that:

(i) $a(1) \leq \frac{1}{2}, a(01) \leq \frac{1}{8}$, and for all $i \in \mathbb{N}, a(0^i1) \leq 2^{-2i-1}$,
(ii) for all $i \in \mathbb{N}$, $a(0^i) = \sum_{j\in\mathbb{N}} a(0^{i+j}1)$,
(iii) for all $i \in \mathbb{N}$ there exists a σ such that $K_m(0^i1\sigma) > -\log(a(0^i1\sigma))$.

Then we can *scale* the semimeasure a to construct a new semimeasure m with the desired properties.

Proposition 1. *If such a semimeasure exists, then Theorem* 1 *holds.*

Proof. From a we can define a c.e. semimeasure m by for all $\sigma \in 2^{<\omega}$, $i \in \mathbb{N}$: $m(0^i1\sigma) = 2^i a(0^i1\sigma)$ and $m(0^i) = \sum_{j\in\mathbb{N}} m(0^{i+j}1)$.

From this definition, for all σ, $m(\sigma) \geq m(\sigma 0) + m(\sigma 1)$. Also

$$m(\lambda) = \sum_{i\in\mathbb{N}} m(0^i1) = \sum_{i\in\mathbb{N}} 2^i a(0^i1) \leq \sum_{i\in\mathbb{N}} 2^i 2^{-2i-1} \leq 1,$$

so m is a c.e. semimeasure.

As M_U majorizes all c.e. semimeasures, there is some d, such that $M_U(\sigma) \geq 2^{-d}m(\sigma)$. Thus $KM(\sigma) \leq -\log(2^{-d} \cdot m(\sigma)) = -\log(m(\sigma)) + d$.

Now given any $c \in \mathbb{N}$, choose $i \geq c + d$. By our hypothesis on a, there is some σ such that

$$\begin{aligned} K_m(0^i 1\sigma) &> -\log(a(0^i 1\sigma)) \\ &= -\log(2^{-i} m(0^i 1\sigma)) \\ &= -\log(m(0^i 1\sigma)) + i \\ &\geq -\log(m(0^i 1\sigma)) + c + d \\ &\geq KM(0^i 1\sigma) + c. \end{aligned}$$

□

Note that we could use the recursion theorem to determine the constant d in the above proof. However, this is not necessary for our construction.

2.1. **The game: c.e. semimeasures vs. monotone machines.** a is c.e., so we can construct it in stages depending on some computable enumeration of the universal monotone machine U. We will start with $a(\sigma, 0) = 0$ for all $\sigma \in 2^{<\omega}$. At each stage $t + 1$, we have the option of choosing a single string ρ and a positive integer x such that $a(\lambda, t) + 2^{-x} \leq 1$, and defining $a(\sigma, t + 1)$ as follows:

$$a(\sigma, t+1) = \begin{cases} a(\sigma, t) + 2^{-x} & \text{if } \sigma \preceq \rho, \\ a(\sigma, t) & \text{otherwise.} \end{cases}$$

In the rest of the proof we will simply write $a(\rho, t + 1) = a(\rho, t) + 2^{-x}$ to refer to this process. If we select ρ at stage $t + 1$ and $a(\rho, t) = 0$ we will simplify the notation further and write $a(\rho, t+1) = 2^{-x}$. If no such ρ is chosen for a stage $t+1$, then this simply means that $a(\sigma, t + 1) = a(\sigma, t)$ for all σ.

Our objective is to construct a c.e. semimeasure a that meets the following requirements for all $i \in \mathbb{N}$:

$$R_i : a(0^i 1) \leq 2^{-2i-1} \text{ and } \exists \sigma \text{ such that } K_m(0^i 1\sigma) > -\log(a(0^i 1\sigma)).$$

For most of this proof we will focus on meeting a single requirement R_i for some arbitrary i. Once we can achieve a single requirement, it will be relatively easy to meet all requirements.

A natural way to view this construction is as a game between us constructing the semimeasure and an opponent constructing the universal monotone machine. For example, in order to meet the requirement that there is some string 1σ such that $K_m(1\sigma) > -\log(a(1\sigma))$ we could choose a string σ and set $a(1\sigma, 1) = 2^{-r}$. Then we could wait and see if a pair $\langle \tau, 1\sigma' \rangle$ is enumerated into U where $|\tau| \leq r$ and $\sigma \preceq \sigma'$. If such a pair is not enumerated into U, then the requirement is met because $K_m(1\sigma) > r = -\log(a(1\sigma))$. If such a pair is enumerated into U, then we can make another change to a at some future stage.

We need to develop a construction of a such that no matter how our opponent enumerates U, the requirements on a are met. We only have a finite amount of measure that we can use, and similarly, our opponent only has a finite number of descriptions of any length. Our goal is to make our opponent run out of suitable descriptions before we run out of measure. There are some fundamental concepts used by Gács in his proof of Theorem 6 that we will make use of. We will introduce one of these with the following example.

Example 1. We choose a string σ and set $a(\sigma, 1) = 2^{-4}$. Now our opponent must respond to this by ensuring that $K_m(\sigma) \leq 4$. To do this, the opponent must

enumerate some pair $\langle \tau, \sigma' \rangle$ into U, where $|\tau| \leq 4$ and $\sigma' \succeq \sigma$. Let us suppose that the opponent enumerates $\langle 0000, \sigma \rangle$ into U. Here comes the trick. Either it is possible for the opponent to add $\langle 000, \sigma \rangle$ into U or not. If not, we can set $a(\sigma, 2) = a(\sigma, 1) + 2^{-4} = 2^{-3}$. Now as it is not possible for the opponent to add $\langle 000, \sigma \rangle$ to U, so the opponent must find a new string, say 001, and add $\langle 001, \sigma \rangle$ to U. However, in this case, the opponent has used both 0000 and 001 to describe σ, a waste of resources. Alternatively, assume that the opponent can enumerate $\langle 000, \sigma \rangle$ into U. We can think of our opponent as holding 000 as a description in reserve for σ. Now we make this reservation useless. This is achieved by:

(i) Never again adding measure to a string comparable with σ.
(ii) Only using units of measure of size at least 2^{-3} from now on.

As we are using increments of measure of size 2^{-3} or more, our opponent must always respond with strings of length at most 3. However, our opponent cannot use 000 because 000 can only be used to describe a string comparable with σ. Hence, if we never increase the measure on a string comparable with σ our opponent can never use 000. This makes 0001 a *gap* in the domain of U as it cannot be used to respond to any of our increments in measure. Again this is a waste of our opponent's resources.

A proof of Theorem 1 can be developed by generalizing the approach of this example. A first step towards this generalization is the following. We take a string σ. We have two integer parameters r, g with $0 < r < g$. We want to increase the measure on σ in increments of 2^{-g}. Each time we increase the measure we want the opponent to respond. To do this, we do not increase the measure on σ directly but on some extension of σ. Let $\Xi = \{\sigma\}\{0,1\}^{g-r}$. We will take some $\xi \in \Xi$ and increase the measure on ξ instead. A first attempt at the algorithm we need is as follows:

Algorithm 1. *Let t be the current stage in the enumeration of U. Choose some $\xi \in \Xi$ that has not been used before (so $a(\xi, t) = 0$). If there is some description of a string σ' with $\sigma' \succeq \sigma$ in the domain of U_t that can be shortened to a description of length r, then terminate the algorithm. Otherwise, we increase the measure on ξ by 2^{-g}, and wait until a stage t' when the opponent describes ξ with a string of length at most g. If we have not used all elements of Ξ, then we let $t = t'$ and repeat. If we have used all elements of Ξ, then the measure on σ must be $|\Xi|2^{-g} = 2^{-r}$ so we wait until the opponent uses a string of length r to describe σ.*

If we apply this algorithm, then the opponent must either use a string of length r to describe σ, or have some string that describes a string comparable with σ that can be shortened to a string of length r. The crucial fact is this: at some point, we increased the measure on σ by *just* 2^{-g}, and the opponent had to respond by finding a *new* string of length r. The string is new in the sense that it is not just some initial segment of a string already used.

As it stands, this algorithm can only make a small gap between K_m and KM. The hard part is amplifying it. Amplification can be done by using the algorithm recursively. In the algorithm above, we act by setting $a(\xi, t) = 2^{-g}$ for some ξ at some stage t. However, rather than just directly increasing the measure on ξ, we could instead run the algorithm again on some *extension* of ξ. This would have the

effect of increasing the measure on ξ while forcing the opponent to waste even more resources. However, to achieve this, a number of issues need to be dealt with:

- We want to make sure our algorithm increases the measure enough. Algorithm 1 will increase the measure on σ somewhere between 2^{-g} and 2^{-r}. This is too great a range.
- We need to ensure that any gaps the algorithm makes in the domain of U are never used again. The algorithm above does not address this.
- Each time we use an extension of a string, we go deeper into the tree of binary strings. We would like to minimize the depth that we go to in order to get the best possible lower bound on the difference between K_m and KM.

The concepts in Example 1 and Algorithm 1 were used by Gács in his original proof of Theorem 6.[4] The approach of this paper is still based on these ideas but differs from that of Gács in how the problems listed above are addressed.

The algorithm that we will present will make decisions based on the domain of U. The question of whether there is a description of σ that can be shortened will be crucial (e.g. whether or not 0000 could be shortened to 000 in Example 1). First we will define certain subsets of the domain of U. We will prove some technical lemmas about how these sets change as the parameters defining them change. These lemmas will be crucial to the verification of the proof. Secondly, we will present the algorithm that establishes a gap between K_m and KM. Thirdly, we will verify that the algorithm works. Finally, we will analyze the algorithm to show how it improves the lower bound.

3. Examining the domain of U

At any stage t, U_t tells us the options our opponent has left. In particular, U_t tells us what our opponent can and cannot use certain strings for. We need a more exact notion for the idea of the opponent shortening an existing description. In his original proof, Gács came up with the notion of a string τ being *reserved* as a description for σ. The idea is not just that the opponent can use τ to describe σ, but additionally that the opponent cannot use τ to describe a string incomparable with σ. For τ to be reserved for σ, the opponent must have already used a string comparable with τ to describe an extension of σ.

There are really two ideas in this definition that are worth separating out and making more explicit. First we will say that a string τ is *fragmented* by σ if some string comparable to τ describes an extension of σ. This means the opponent cannot use τ to describe a string incomparable with σ (otherwise U would not be a monotone machine). Second, a string τ is *incompatible* with σ if some string comparable with τ describes a string incomparable with σ. This means that τ cannot be used to describe σ. Hence another way of saying that a string τ is reserved for σ is that τ is fragmented by σ but τ is not incompatible with σ. As we are not so much interested in particular descriptions, but descriptions of a certain length, we will make the following definitions.

Definition. (i) The strings of length r *fragmented* by σ at stage t are:

$$F_r(\sigma, t) = \{\tau \in \{0,1\}^r : \exists \langle \tau', \sigma' \rangle \in U_t((\tau' \approx \tau) \wedge (\sigma \preceq \sigma'))\}.$$

[4] An extended version of the original proof is available from Professor Gács's home page.

(ii) The strings of length r *incompatible* with σ at stage t are:

$$I_r(\sigma,t)=\{\tau\in\{0,1\}^r:\exists\langle\tau',\sigma'\rangle\in U_t((\tau'\approx\tau)\wedge(\sigma\mid\sigma'))\}.$$

(iii) The strings of length r *reserved* for σ at stage t are:

$$R_r(\sigma,t)=F_r(\sigma,t)\setminus I_r(\sigma,t).$$

Example 2. Assume at stage t, $U_t=\{\langle 00,11\rangle,\ \langle 000,1\rangle,\ \langle 000,11\rangle,\ \langle 001,11\rangle,\ \langle 011,01\rangle,\ \langle 100,01\rangle,\ \langle 110,011\rangle,\langle 111,1\rangle\}$.

If we consider the strings of length 2, working through the above definitions will establish that $F_2(01,t)=\{01,10,11\}$, $I_2(01,t)=\{00,11\}$, and $R_2(01,t)=\{01,10\}$.

The definitions given for F_r and I_r can be extended to sets of strings.

Definition. If Σ is a set of finite strings, then:

(i) $F_r(\Sigma,t)=\bigcup_{\sigma\in\Sigma}F_r(\sigma,t)$;
(ii) $I_r(\Sigma,t)=\bigcap_{\sigma\in\Sigma}I_r(\sigma,t)$.

We will explain later why we take the intersection in the definition of $I_r(\Sigma,t)$. The following lemmas show how the sets $F_r(\sigma,t)$, $I_r(\sigma,t)$, $F_r(\Sigma,t)$, and $I_r(\Sigma,t)$ change as the parameters that define them vary.

Lemma 2.

(i) *If* $r_0\le r_1$, *then* $[F_{r_0}(\sigma,t)]\supseteq[F_{r_1}(\sigma,t)]$ *and* $[I_{r_0}(\sigma,t)]\supseteq[I_{r_1}(\sigma,t)]$.
(ii) *If* $\sigma_0\preceq\sigma_1$, *then* $[F_r(\sigma_0,t)]\supseteq[F_r(\sigma_1,t)]$ *and* $[I_r(\sigma_0,t)]\subseteq[I_r(\sigma_1,t)]$.
(iii) *If* $t_0\le t_1$, *then* $[F_r(\sigma,t_0)]\subseteq[F_r(\sigma,t_1)]$ *and* $[I_r(\sigma,t_0)]\subseteq[I_r(\sigma,t_1)]$.

Proof. i) If $\alpha\in[F_{r_1}(\sigma,t)]$, then $\alpha\restriction r_1\in F_{r_1}(\sigma,t)$, so $\exists\tau\approx\alpha\restriction r_1$ and $\sigma'\succeq\sigma$ such that $\langle\tau,\sigma'\rangle\in U_t$. Now as $r_0\le r_1$, $\tau\approx\alpha\restriction r_0$, so $\alpha\restriction r_0\in F_{r_0}(\sigma,t)$ and hence $\alpha\in[F_{r_0}(\sigma,t)]$. Similarly $[I_{r_0}(\sigma,t)]\supseteq[I_{r_1}(\sigma,t)]$.

ii) If $\tau\in F_r(\sigma_1,t)$, then $\exists\tau'\approx\tau$ and $\sigma\succeq\sigma_1$ such that $\langle\tau',\sigma\rangle\in U_t$. As $\sigma\succeq\sigma_1\succeq\sigma_0$, so $\tau\in F_r(\sigma_0,t)$. If $\tau\in I_r(\sigma_0,t)$, then $\exists\tau'\approx\tau$ and $\rho\mid\sigma_0$ such that $\langle\tau',\rho\rangle\in U_t$. As $\sigma_1\succeq\sigma_0$, so $\sigma_1\mid\rho$ and thus $\tau\in I_r(\sigma_1,t)$.

iii) This follows because $U_{t_0}\subseteq U_{t_1}$. □

In summary $[F_r(\sigma,t)]$ and $[I_r(\sigma,t)]$ are both increasing in t and decreasing in r. In length of σ, $[F_r(\sigma,t)]$ is decreasing and $[I_r(\sigma,t)]$ is increasing. To establish similar results for $F_r(\Sigma,t)$, and $I_r(\Sigma,t)$, we need to define a relationship $\preceq$ between sets of strings.

Definition. If $\Sigma_0,\Sigma_1\subseteq 2^{<\omega}$, then $\Sigma_0\preceq\Sigma_1$ if for all $\sigma_1\in\Sigma_1$, there exists a $\sigma_0\in\Sigma_0$ such that $\sigma_0\preceq\sigma_1$.

Lemma 3. (i) *If* $r_0\le r_1$, *then* $[F_{r_0}(\Sigma,t)]\supseteq[F_{r_1}(\Sigma,t)]$ *and* $[I_{r_0}(\Sigma,t)]\supseteq[I_{r_1}(\Sigma,t)]$.
(ii) *If* $\Sigma_0\preceq\Sigma_1$, *then* $[F_r(\Sigma_0,t)]\supseteq[F_r(\Sigma_1,t)]$ *and* $[I_r(\Sigma_0,t)]\subseteq[I_r(\Sigma_1,t)]$.
(iii) *If* $t_0\le t_1$, *then* $[F_r(\Sigma,t_0)]\subseteq[F_r(\Sigma,t_1)]$ *and* $[I_r(\Sigma,t_0)]\subseteq[I_r(\Sigma,t_1)]$.

Proof. i) If $\tau\in F_{r_1}(\Sigma,t)$, then for some $\sigma\in\Sigma$, $\tau\in F_{r_1}(\sigma,t)$; thus by the previous lemma, $\tau\in F_{r_0}(\sigma,t)\subseteq F_{r_0}(\Sigma,t)$.

If $\tau\in I_{r_1}(\Sigma,t)$, then for all $\sigma\in\Sigma$, $\tau\in I_{r_1}(\sigma,t)$; thus by the previous lemma, $\tau\in\bigcap_{\sigma\in\Sigma}I_{r_0}(\sigma,t)=I_{r_0}(\Sigma,t)$.

ii) If $\alpha\in[F_r(\Sigma_1,t)]$, then for some $\sigma_1\in\Sigma_1$, $\alpha\in[F_r(\sigma_1,t)]$. There exists $\sigma_0\in\Sigma_0$ such that $\sigma_0\preceq\sigma_1$, so by the previous lemma, $\alpha\in[F_r(\sigma_0,t)]\subseteq[F_r(\Sigma_0,t)]$.

Consider if $\alpha \in [I_r(\Sigma_0, t)]$, then for all $\sigma_0 \in \Sigma_0$, $\alpha \in [I_r(\sigma_0, t)]$. Now if $\sigma_1 \in \Sigma_1$. then for some $\sigma_0 \in \Sigma_0$, $\sigma_0 \preceq \sigma_1$. Because $\alpha \in [I_r(\sigma_0, t)]$, so by the previous lemma, $\alpha \in [I_r(\sigma_1, t)]$. Thus $\alpha \in \bigcap_{\sigma_1 \in \Sigma_1} [I_r(\sigma_1, t)] = [I_r(\Sigma_1, t)]$.

iii) If $\tau \in F_r(\Sigma, t_0)$, then for some $\sigma \in \Sigma$, $\tau \in F_r(\sigma, t_0)$; thus by the previous lemma, $\tau \in F_r(\sigma, t_1) \subseteq F_r(\Sigma, t_1)$.

If $\tau \in I_r(\Sigma, t_0)$, then for all $\sigma \in \Sigma$, $\tau \in I_r(\sigma, t)$; thus by the previous lemma, $\tau \in \bigcap_{\sigma \in \Sigma} I_r(\sigma, t_1) = I_r(\Sigma, t_1)$. □

4. The main algorithm

We are going to describe an algorithm that can be used to prove Theorems 1 and 7. The algorithm will be allocated a certain amount of measure to spend on the construction of the c.e. semimeasure a. It will also be given a set of strings Σ, and it will only be able to increase the measure on extensions of these. Ultimately we want the algorithm to establish some difference between $K_m(\rho)$ and $-\log(a(\rho))$ for some string ρ that extends some element of Σ. We will argue that if this does not happen, then some contradiction must ensue. The basic idea is to make the opponent run out of short descriptions. Formalizing this idea is a little difficult. We want to be able to say if we spend 2^{-r} of measure on extensions of Σ, and we do not establish a difference between K_m and $-\log(a)$, then we can cause x amount of something to happen to U (where U is the universal machine controlled by the opponent). We will refer to this x, whatever it is, as the gain made by the algorithm.

Now the most obvious measurement to use would be $\mu([\Pi_1(U_t)])$ (where $\Pi_1(U_t)$ is the projection of U_t on the first coordinate) because this is analogous to the domain of U_t. However, remember that part of the idea is to leave gaps in U that the opponent cannot do anything useful with. The problem with this measurement is that it does not account for gaps. An alternative is $F_r(\Sigma, t)$, the strings of length r that are fragmented by elements of Σ. This will count gaps as well. This is almost what we need. The size of this set must be $\leq 2^r$. So if the algorithm ends at t_1, and we could ensure that $|F_r(\Sigma, t_1)| \geq k$ for an arbitrary k, we would be done. However, it is still more complicated than this. We will end up using the algorithm we define recursively. When verifying the algorithm we do not want to double-count any gain made. To make the verification possible, we will subtract $[I_{r'}(\Sigma, t_0)]$ from $[F_r(\Sigma, t_1)]$, where $r' \geq r$ and t_0 is the time the algorithm starts. This is what we will use to determine the gain made by the algorithm. The following definition, where G is for gain, codifies this idea.

Definition. If $\Sigma \subseteq 2^{<\omega}$; $r, r', t_0, t_1 \in \mathbb{N}$ such that $r \leq r'$ and $t_0 \leq t_1$, then:

$$G_r^{r'}(\Sigma; t_0, t_1) = [F_r(\Sigma, t_1)] \setminus [I_{r'}(\Sigma, t_0)].$$

$G_r^{r'}(\Sigma; t_0, t_1)$ can be thought of as the gain the algorithm obtains using the strings in Σ between stages t_0 and t_1, and using units of measure between r and r'. With this definition in hand, we can define what sort of algorithm is needed. Note that we describe an algorithm that works on a set of strings. The use of this type of algorithm is a new feature of this proof.

Definition. If $r, f \in \mathbb{N}$, and Σ is a finite prefix-free subset of $2^{<\omega}$, then an (r, f, Σ)-*strategy* is an algorithm that, if implemented at stage t_0, ensures that either:

(i) (a) There is some $\sigma \in \Sigma$ such that for some σ' extending σ, $K_m(\sigma') > -\log(a(\sigma'))$, and
(b) For all stages t, for all $\sigma \in \Sigma$, $a(\sigma, t) < \frac{5}{4}2^{-r}$; or

(ii) At some stage t_1, for some $r' \geq r$ computable from (r, f):
(a) For all $\sigma \in \Sigma$, $2^{-r} \leq a(\sigma, t_1) < \frac{5}{4}2^{-r}$, and
(b) $\mu(G_r^{r'}(\Sigma; t_0, t_1)) \geq \left(1 + \frac{f}{2}\right) a(\Sigma, t_1)$, and
(c) For all $\hat{\Sigma} \subseteq \Sigma$,
$\mu(G_r^{r'}(\hat{\Sigma}; t_0, t_1)) \geq \left(1 + \frac{f}{2}\right)\Big(a(\Sigma, t_1) - 2a(\Sigma \setminus \hat{\Sigma}, t_1)\Big)$,
where $a(\Sigma, t) = \sum_{\sigma \in \Sigma} a(\sigma, t)$.

This definition is complicated, so we will take some time to explain it. The input parameters for the strategy are r,f and Σ. Σ is a finite prefix-free set of binary strings. These are the strings that the algorithm will work on. The r parameter determines the measure that the algorithm should spend on *every* string in Σ. The f parameter determines the amount of gain that the algorithm should make. First we will establish the existence of an $(r, 0, \Sigma)$-strategy for any appropriate r; then we will inductively establish the existence of $(r, f + 1, \Sigma)$-strategies assuming the existence of (r, f, Σ)-strategies.

The definition gives two possible outcomes to a strategy. Outcome (i) is the preferred outcome. If outcome (i) occurs, then we have established a difference between K_m and KM. However, if we do not achieve this, then outcome (ii) is at least a step in the right direction. Provided $f > 0$, outcome (ii) ensures that some difference between $\mu(G_r^{r'}(\Sigma; t_0, t_1))$ and $a(\Sigma, t_1)$ is created.

In outcome (ii), condition (b) says that the gain on the set of strings Σ is bounded below. We do not want the gain to be too concentrated on some particular subset of Σ. The gain achieved need not be evenly distributed among the elements of Σ, but it is important that this distribution is not too skewed. This is the reason for condition (c). Condition (c) ensures that no individual string in Σ contributes too much to the overall gain made. We need condition (c) because when we use the algorithm recursively, we will not be able to count the gain that occurs on some individual elements of Σ. Hence we do not want an individual element of Σ providing too much of the overall gain. Note that (c) implies (b) by taking $\hat{\Sigma} = \Sigma$.

It is also important to observe that for all $\sigma \in \Sigma$, if the strategy is running at stage t, then $a(\sigma, t)$ is bounded above in outcomes (i) and (ii). In outcome (ii) there is also a lower bound for $a(\sigma, t_1)$. This lower bound is essential for using strategies recursively because we want these strategies to increase the measure on strings, as well as establish some gain.

If we can implement an (r, f, Σ)-strategy with an arbitrarily high f, then (ii) is not a possible outcome as the measure would be greater than 1. This would force outcome (i).

The other parameter in the above definition to discuss is r'. r' dictates where the strategy should make its gain. In practice, what it will mean is that $2^{-r'}$ will be the minimum sized increment in measure that the strategy will make to a. In other words, each time the strategy increases the measure on some string, the amount

increased will be at least $2^{-r'}$. The idea is that if any previous strategy has created a gap of measure less than $2^{-r'}$, then the current strategy will not interfere with it.

4.1. **A basic $f = 0$ strategy.** The idea behind the basic $(r, 0, \Sigma)$-strategy is simple. For every $\sigma \in \Sigma$ we set $a(\sigma) = 2^{-r}$. The opponent is then forced to respond by setting $K_m(\sigma) \leq r$. To do this, the opponent must find some string τ of length at most r and enumerate $\langle \tau, \sigma \rangle$ into U. The only difficulty is in showing that this basic idea meets our rather elaborate definition of a strategy.

Proposition 2. *If Σ is a finite prefix-free subset of $2^{<\omega}, r \in \mathbb{N}$, and t_0 is the current stage in the construction of a, such that $a(\Sigma, t_0) = 0$, and $|\Sigma|2^{-r} \leq 1$, we can implement an $(r, 0, \Sigma)$-strategy with $r' = r$.*

Proof. Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$. This strategy can be implemented by first for all $i \in \mathbb{N}$, $1 \leq i \leq n$ setting $a(\sigma_i, t_0 + i) = 2^{-r}$ (we know that $a(\sigma_i, t_0 + i - 1) = 0$ because Σ is prefix-free). This ensures that $a(\Sigma, t_0 + n) = |\Sigma|2^{-r}$. The second step is to wait until a stage t_1, when the opponent responds by setting $K_{m,t_1}(\sigma) \leq r$ for all $\sigma \in \Sigma$. If this never happens, then for some $\sigma \in \Sigma$, $K_m(\sigma) > r = -\log(2^{-r}) = -\log(\lim_{t\to\infty} a(\sigma, t)) = -\log(a(\sigma))$, so outcome (i) occurs.

If at some stage t_1, $K_{m,t_1}(\sigma) \leq r$ for all $\sigma \in \Sigma$, then for all $i \in \mathbb{N}$, $1 \leq i \leq n$ our opponent must have enumerated some $\langle \tau_i, \sigma_i' \rangle$ into U_{t_1} where $s_i = |\tau_i| \leq r$ and $\sigma_i' \succeq \sigma_i$. In this case we will show that outcome (ii) occurs. First (a) holds as $a(\Sigma, t_1) = a(\Sigma, t_0 + n)$.

For any i, let τ_r be any extension of τ_i of length r. By definition, $\tau_r \in F_r(\sigma_i, t_1)$, so $\tau_r \in \bigcup_{\sigma\in\Sigma} F_r(\sigma, t_1) = F_r(\Sigma, t_1)$. If $\tau_r \in I_r(\sigma_i, t_0)$, then for some τ' comparable with τ_r and some ρ incomparable with σ_i, $\langle \tau', \rho \rangle \in U_{t_0} \subseteq U_{t_1}$ but this would contradict the definition of a monotone machine as it implies both $\tau' \approx \tau_i$ and $\rho \mid \sigma_i'$. So $\tau_r \notin I_r(\sigma_i, t_0)$ and thus $\tau_r \notin \bigcap_{\sigma_i\in\Sigma} I_r(\sigma_i, t_0) = I_r(\Sigma, t_0)$. Hence $G_r^r(\Sigma; t_0, t_1)$, which by definition is $[F_r(\Sigma, t_1)] \setminus [I_r(\sigma, t_0)]$, contains all infinite extensions of τ_r and hence all infinite extensions of τ_i. So $[\tau_i] \subseteq G_r^r(\Sigma; t_0, t_1)$.

Now if $i \neq j$, $[\tau_i] \cap [\tau_j] = \emptyset$ because if they are comparable, then σ_i and σ_j must be comparable but Σ is a prefix-free set. Thus $\mu(G_r^r(\Sigma; t_0, t_1)) \geq \sum_{i=1}^n \mu([\tau_i]) = \sum_{i=1}^n 2^{-s_i} \geq \sum_{i=1}^n 2^{-r} = |\Sigma|2^{-r} = a(\Sigma, t_1) = (1 + \frac{0}{2})a(\Sigma, t_1)$. Hence (b) holds.

Let $\hat{\Sigma} \subseteq \Sigma$. Let $I = \{1, \ldots, n\}$ and $J = \{i \in I : \sigma_i \in \hat{\Sigma}\}$. By the same logic,

$$\begin{aligned}
\mu(G_r^r(\hat{\Sigma}; t_0, t_1)) &\geq \sum_{j\in J} 2^{-r} \\
&= |I|2^{-r} - (|I \setminus J|)2^{-r} \\
&= a(\Sigma, t_1) - a(\Sigma \setminus \hat{\Sigma}, t_1) \\
&\geq (1 + \frac{0}{2})(a(\Sigma, t_1) - 2a(\Sigma \setminus \hat{\Sigma}, t_1)).
\end{aligned}$$

So (c) holds and outcome (ii) occurs. □

4.2. **Using strategies sequentially.** We are going to work up to developing an $(r, f + 1, \Sigma)$-strategy, using (r, f, Σ)-strategies. Before we present the main algorithm that does this, there is some more preparatory work to do. Ideally, when we implement a strategy, we want to achieve outcome (i). However, if this does not happen, then at some stage the strategy will terminate and we can start a new strategy. We want to ensure that if the second strategy also achieves outcome (ii), then the gain of running the strategies sequentially is at least the sum

of the minimum gain expected from the strategies individually. The definition of the (r, f, Σ)-strategy is designed to allow this as Proposition 3 will show. Before proving this proposition, we need the following lemma.

Lemma 4. *If $\sigma_0 \mid \sigma_1$, then $F_p(\sigma_0, t) \subseteq I_p(\sigma_1, t)$.*

Proof. If $\tau \in F_p(\sigma_0, t)$, then there exists $\langle \tau', \sigma_0' \rangle \in U_t$ such that $\tau' \approx \tau$ and $\sigma_0' \succeq \sigma_0$. Now this implies that $\sigma_0' \mid \sigma_1$ and so $\tau \in I_p(\sigma_1, t)$. □

Proposition 3. *If $r_0 \leq r_1 \leq \ldots \leq r_n$ and $t_0 \leq t_1 \leq \ldots \leq t_n$ are sequences of $\mathbb{N}$, and $\Sigma_0, \Sigma_1, \ldots, \Sigma_{n-1}$ are pairwise disjoint prefix-free subsets of $2^{<\omega}$, and $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \ldots \cup \Sigma_{n-1}$ is also prefix-free, then:*

$$\mu(G^{r_n}_{r_0}(\Sigma; t_0, t_n)) \geq \sum_{i=0}^{n-1} \mu(G^{r_{n-i}}_{r_{n-1-i}}(\Sigma_i; t_i, t_{i+1})).$$

Proof. First if $i \in \mathbb{N}, 0 \leq i < n$, then $\Sigma \preceq \Sigma_i$, so by chaining together Lemma 3: $[F_{r_{n-i-1}}(\Sigma_i, t_{i+1})] \subseteq [F_{r_0}(\Sigma_i, t_{i+1})] \subseteq [F_{r_0}(\Sigma, t_{i+1})] \subseteq [F_{r_0}(\Sigma, t_n)]$ and $[I_{r_n}(\Sigma, t_0)] \subseteq [I_{r_n}(\Sigma, t_i)] \subseteq [I_{r_n}(\Sigma_i, t_i)] \subseteq [I_{r_{n-i}}(\Sigma_i, t_i)]$.

Hence $[F_{r_0}(\Sigma, t_n)] \setminus [I_{r_n}(\Sigma, t_0)] \supseteq [F_{r_{n-i-1}}(\Sigma_i, t_{i+1})] \setminus [I_{r_{n-i}}(\Sigma_i, t_i)]$, which by definition means that $G^{r_n}_{r_0}(\Sigma; t_0, t_n) \supseteq G^{r_{n-i}}_{r_{n-i-1}}(\Sigma_i; t_i, t_{i+1})$.

Now assume $0 \leq i < j < n$. We have that

$$[F_{r_{n-i-1}}(\Sigma_i, t_i)] \subseteq [F_{r_{n-j}}(\Sigma_i, t_j)] \subseteq [I_{r_{n-j}}(\Sigma_j, t_j)].$$

The first inclusion is by Lemma 3. The second inclusion follows because if $\tau \in F_{r_{n-j}}(\Sigma_i, t_j)$, then $\tau \in F_{r_{n-j}}(\sigma_i, t_j)$ for some $\sigma_i \in \Sigma_i$. But for all $\sigma_j \in \Sigma_j$, σ_j is incomparable with σ_i. So $\tau \in I_{r_{n-j}}(\sigma_j, t_j)$ by Lemma 4. Thus $\tau \in \bigcap_{\sigma_j \in \Sigma_j} I_{r_{n-j}}(\sigma_j, t_j) = I_{r_{n-j}}(\Sigma_j, t_j)$. So we can conclude that

$$([F_{r_{n-i-1}}(\Sigma_i, t_{i+1})] \setminus [I_{r_{n-i}}(\Sigma_i, t_i)]) \cap ([F_{r_{n-j-1}}(\Sigma_j, t_{j+1})] \setminus [I_{r_{n-j}}(\Sigma_j, t_j)]) = \emptyset,$$

which again by definition means that

$$G^{r_{n-i}}_{r_{n-i-1}}(\Sigma_i; t_i, t_{i+1}) \cap G^{r_{n-j}}_{r_{n-j-1}}(\Sigma_j; t_j, t_{j+1}) = \emptyset.$$

□

Buried in the above proof is why we define $I_r(\Sigma, t)$ to be the intersection of $I_r(\sigma, t)$ for all $\sigma \in \Sigma$. The reason is this. Take Σ_0 and Σ_1 to be disjoint sets whose union is prefix-free. If $\tau \in F_r(\Sigma_0, t)$, then τ is fragmented by some string in Σ_0. As all strings in Σ_0 are incomparable with all strings in Σ_1, so $\tau \in I_r(\Sigma_1, t)$. This is why the above proposition works as it avoids double counting any gains.

Example 3. To illustrate why this proposition is useful, we will show how we can use it to gradually increase the measure on a string without losing any gain made along the way. Consider the following implementation of three strategies in sequence and assume that they all have outcome (ii). Take some $r_0 \in \mathbb{N}$. Fix an $f \in \mathbb{N}$ and assume we have an (r, f, Σ) strategy for any appropriate r and Σ. Let r_1 be the r' computable from (r_0, f). Similarly let r_2 be the r' computable from (r_1, f) and r_3 be the r' computable from (r_2, f). Take any string σ and let $\Sigma_2 = \{\sigma 00\}$, $\Sigma_1 = \{\sigma 01\}\{0,1\}^{r_1 - r_0}$, $\Sigma_0 = \{\sigma 1\}\{0,1\}^{r_2 - r_0}$ and $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$.

Starting at stage t_0, first we implement an (r_2, f, Σ_0)-strategy. When this finishes at stage t_1, we implement an (r_1, f, Σ_1)-strategy. Finally when it finishes at stage t_2, we implement an (r_0, f, Σ_2)-strategy. Let t_3 be the stage that this final strategy finishes.

Now from Proposition 3,

$$\begin{aligned}\mu(G^{r_3}_{r_0}(\Sigma;t_0,t_3)) &\geq \sum_{i=0}^{2}\mu(G^{r_{3-i}}_{r_{3-1-i}}(\Sigma_i;t_i,t_{i+1}))\\ &\geq \sum_{i=0}^{2}\left(1+\frac{f}{2}\right)a(\Sigma_i,t_{i+1})\\ &= \left(1+\frac{f}{2}\right)a(\Sigma,t_3).\end{aligned}$$

The last step follows provided we do not increase the measure on any other string comparable with a string in Σ except as required by the strategies. Now $a(\sigma) \geq a(\Sigma) \geq |\Sigma_0|2^{-r_2} + |\Sigma_1|2^{-r_1} + |\Sigma_2|2^{-r_0} = 3 \cdot 2^{-r_0}$. This approach has allowed us to use strategies to increase the measure on σ in three increments of 2^{-r_0} without losing any of the gain made by the three strategies individually. This ability to increase the measure on a string in small increments will be exploited in the main proof.

4.3. **Implementing more difficult strategies.** We have seen that we can use strategies sequentially without losing any gain. However, we need to do better than this. We need to be able to combine strategies in such a way as to increase the gain we make. To do this, we will make use of reservations. Our goal is to implement an $(r, f+1, \Sigma)$-strategy using a finite sequence of (r_i, f, Σ_i)-strategies.

We will improve Algorithm 1. The basic idea remains the same. When we assign a measure to a string, e.g. if at stage t_0 we set $a(\sigma, t_0) = 2^{-g}$, then our opponent has to respond by allocating a string of length at most g to describe σ. Say in response to our setting $a(\sigma, t_0) = 2^{-g}$, our opponent enumerates $\langle \tau, \sigma \rangle$ into U_{t_1}, where $|\tau| = g$. Take $p < g$ and let τ_p be the first p bits of τ. If $\tau_p \notin R_p(\sigma, t)$, then $\tau_p \in I_p(\sigma, t)$, so our opponent can never enumerate $\langle \tau_p, \sigma \rangle$ into U. So if we set $a(\sigma, t_1) = 2^{-p}$, our opponent must find some new string υ of length at most p to describe σ. The original τ description of σ is effectively a waste of resources.

On the other hand, while $\tau_p \in R_p(\sigma, t)$, we also have a gain because our opponent can only use τ_p to describe a string comparable with σ. If we only increase the measure on strings incomparable with σ, then our opponent cannot use τ_p in response. If we only use increments of measure of size at least 2^{-p}, then our opponent cannot make use of any extension of τ_p (it would be too long to be useful). Note that this is why we need an r' in the definition of an (r, f, Σ)-strategy. The r' is the smallest amount of measure this strategy needs. Being able to compute r' means we know how much space strategies need in terms of units of measure. With this knowledge, we can ensure that the smallest measure needed by a strategy will not interfere with any reservations established by previous strategies.

We noted earlier that part of the problem with Algorithm 1 was that the amount of measure that it assigned to a string ranged between 2^{-g} and 2^{-r}. This range was too great to allow the algorithm to be used recursively. Hence our algorithm has two objectives: create a certain amount of gain, and ensure that a certain amount of measure is allocated. These two objectives will be achieved by splitting the algorithm into two phases. In the first, the advantage phase, the objective will be to force our opponent to reserve a number of strings of length p. We do not know how much measure we will need to use before our opponent makes this many

reservations. If we do not use enough measure, then we proceed with the second phase, the spend phase. The spend phase is where we make sure that we have placed enough measure on each string so that the strategies can be used recursively. The use of these two phases is another new feature of this proof.

The spend phase must occur after the advantage phase because only then will we know how much more measure we need to allocate. Accordingly, the spend phase will use larger units of measure than the advantage phase. From r we will compute a p, q and r' such that $r < p < q < r'$. The advantage phase will use units of measure between q and r' while the spend phase will use units of measure between r and p (see Figure 1). This gives us a bit of a problem. During the advantage phase we can only require reservations of length at most p. If p is much larger than r, then we will need to make many reservations in order to achieve anything. To get around this problem, we define $\Upsilon = \Sigma\{0\}\{0,1\}^{p-r}$ (where Σ is the set of input strings for the algorithm). Note that the purpose of the $\{0\}$ in the definition of Υ is just to separate the strings used by the advantage phase from those used by the spend phase. We will run the algorithm on the strings in Υ until enough of these have a reservation of length p. An important new idea in this proof is to run each pass of the algorithm *simultaneously* on all the strings in Υ that do not have a reservation.

As in Algorithm 1, we will not increase the measure on the elements of Υ directly, but rather we will make a series of small increments of measure on some extensions of them. We will let $\Xi = \Upsilon\{0,1\}^e$ (e depends on f and will be defined shortly). The idea is this. If $\upsilon \in \Upsilon$ and $\xi_0, \ldots, \xi_{2^e-1}$ are the elements of Ξ that extend υ, then we will sequentially set the measure of each ξ_i to 2^{-p-e} until a reservation of size p occurs for υ. If we do this for all such ξ_i, then the measure on υ will be 2^{-p} and the opponent must allocate a string of length p to describe υ. However, it is not quite that simple. We want to increase the measure on ξ_i by running some strategy on it. Every time we run a strategy we need to use larger units of measure. So instead what we will do is take many extensions of ξ_0. Then we can run our first strategy on all these extensions, spending a little bit of measure on each. Then we will take slightly fewer extensions of ξ_1, and run a strategy that spends a little more measure on each of these extensions, and so on. This is just like what we did in Example 3. This gives us sets $\Psi_0, \Psi_1, \ldots$, where Ψ_i is the ith set of extensions of elements of Ξ that we want to increase measure on. The sets used by the advantage phase are shown in Figure 2. In this figure arrows represent extensions of strings.

The algorithm would ideally achieve a reservation of length p for every element of Υ but there is a problem. Once a reservation is achieved for some $\upsilon \in \Upsilon$, we do not want to increase the measure on that υ. However, reservations are not monotonic; they can appear at one stage and then disappear at the next. Consider $\upsilon_1, \upsilon_2 \in \Upsilon$. Say we increase the measure on both υ_1, υ_2 by 2^{-p-e} and then the opponent responds by giving a p reservation to υ_1, but not υ_2. Then we could increase the measure on υ_2 and the opponent might respond by giving a p reservation to υ_2 but taking away the p reservation from υ_1. By adopting this sort of tactic, the opponent could force us to make many separate increases in measure in order to achieve reservations for all elements of Υ. Every time we increase the measure on some subset of Υ we need a new Ψ_i. The more Ψ_i's we need, the deeper we need to go into the binary tree. To improve the lower bound, we want to limit the number of Ψ_i's we need. Each time we increase the measure, we do

so simultaneously on all the elements of some subset of Υ. We want to make sure that this subset includes at least one-quarter of the elements of Υ. To do this, we will terminate the advantage phase of the algorithm when we have a reservation for three-quarters of the elements of Υ.

The spend phase is similar but simpler. In this phase we will identify those elements $\sigma \in \Sigma$ that do not have enough measure. We will increase the measure on σ in increments of 2^{-r-3} until we have exceeded 2^{-r}. Again like Example 3, we will not increase the measure directly on σ but rather on some set of extensions of it. These extensions will form the sets $\Phi_0, \ldots, \Phi_7$.

4.4. **The S function.** The length of extensions that we take to form the sets Ψ_i and Φ_i is governed by the space that a strategy needs to run, in terms of quantities of measure. The function $S(f)$ is used to determine this space. We will show that there exist (r, f, Σ)-strategies such that $r' = r + S(f)$. From our base strategy we can set $S(0) = 0$ because in this case $r = r'$.

In the process of determining $S(f+1)$ we will also compute all the other variables we need for the main algorithm. These are illustrated in Figure 1. To determine $S(f+1)$, first we compute an increasing sequence $r_0 \leq r_1 \leq \ldots \leq r_8$ by $r_0 = r + 3$ and $r_{i+1} = r_i + S(f)$. This gives us room for the spend phase. Let $p = r_8$. Let $e = \lceil \log(20 + 10f) \rceil$. e is chosen so that $\frac{5}{2} 2^{-e} \left(1 + \frac{f}{2}\right) \leq \frac{1}{8}$.

Let $q = p + e$. Now compute an increasing sequence $r_0^* \leq r_1^* \leq \ldots \leq r_n^*$, where $r_0^* = q$, $n = 2^{e+2}$ and $r_{i+1}^* = r_i^* + S(f)$. This gives us room for the advantage phase. Finally let $r' = r_n^*$, and $S(f+1) = r' - r$. This is well defined as $S(f+1)$ does not depend on the initial choice of r.

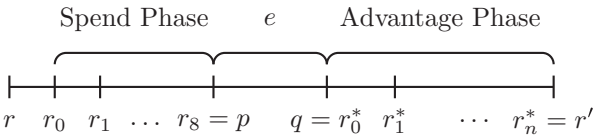

FIGURE 1. Algorithm parameters

4.5. **An f+1 strategy algorithm.** We are finally ready to present the main algorithm. We are given $r, f \in \mathbb{N}$ and a prefix-free set of strings Σ such that $|\Sigma| \frac{5}{4} 2^{-r} \leq 1$. We let t_0 be the stage that the algorithm starts and we require that $a(\Sigma, t_0) = 0$. In order to implement an $(r, f+1, \Sigma)$-strategy first we determine all the values $n, p, q, e, r_i, r_i^*, r'$ as shown in Figure 1. In the algorithm, Υ' represents those elements of Υ whose measure we will increase in the current pass through the advantage phase of the algorithm. Now with all the values that we need computed, the algorithm that we will use to implement an $(r, f+1, \Sigma)$-strategy is as follows:

Advantage Phase. Let t be the current stage. Let $\Upsilon = \Sigma\{0\}\{0,1\}^{p-r}$. Let $\Xi = \Upsilon\{0,1\}^e$. Let $\Upsilon' = \{\upsilon \in \Upsilon : R_p(\upsilon, t) = \emptyset\}$. If $|\Upsilon'| < \frac{1}{4}|\Upsilon|$, then move to the spend phase.

For each $\upsilon \in \Upsilon'$, choose a $\xi_\upsilon \in \Xi$ such that $\xi_\upsilon \succeq \upsilon$ and $a(\xi_\upsilon, t) = 0$. Let $\Xi' = \{\xi_\upsilon : \upsilon \in \Upsilon'\}$. Choose the least $i \in \{0, \ldots, n-1\}$ that has not been used previously. Let $\Psi_i = \Xi'\{0,1\}^{r^*_{n-i-1} - q}$ and implement an (r^*_{n-i-1}, f, Ψ_i)-strategy.

If this strategy finishes at stage t', then wait until stage t'' such that $K_{m,t''}(\upsilon) \leq p$ for all $\upsilon \in \Upsilon$ such that $a(\upsilon, t') \geq 2^{-p}$. Repeat the advantage phase.

Spend Phase. Let t be the current stage. Let $\Sigma' = \{\sigma \in \Sigma : a(\sigma, t) < 2^{-r}\}$. If Σ' is the empty set, then terminate the algorithm. Otherwise choose the least $i \in \{0, \ldots, 7\}$ such that i has not been used previously in the spend phase. Let $\Phi_i = \Sigma'\{1\}\{0^i 1\}\{0,1\}^{r_{7-i}-r-3}$ and run a (r_{7-i}, f, Φ_i)-strategy. Repeat the spend phase.

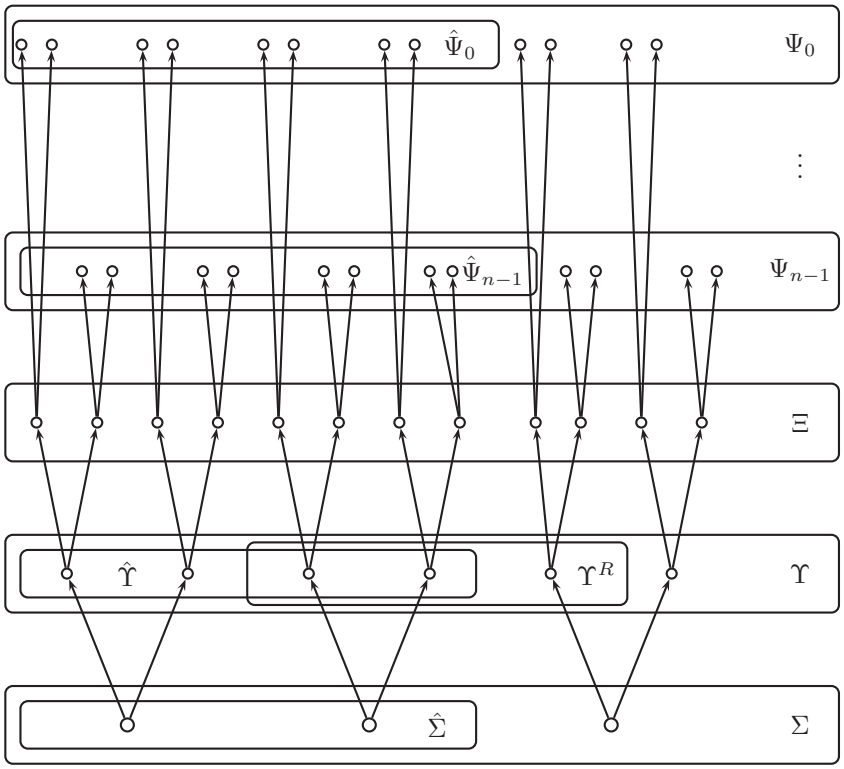

FIGURE 2. Algorithm strings

5. Verification of algorithm

The algorithm starts at stage t_0. We will define the following parameters assuming that the algorithm does terminate at some stage t_1. Let n_a be the number of times the advantage phase is run. Let n_s be the number of times the spend phase is run. For all $0 \leq i < n_a$, let t_i^* be the stage when the ith strategy in the advantage phase begins (starting with $i = 0$). Let t_{mid} be the stage when the algorithm completes the advantage phase and let $t_{n_a}^* = t_{mid}$. For all $0 \leq i < n_s$, let t_i' be the stage when the ith strategy in the spend phase begins (starting with $i = 0$). Let $t'_{n_s} = t_1$. Let $\Phi = \Phi_0 \cup \ldots \cup \Phi_{n_s-1}$ and $\Psi = \Psi_0 \cup \ldots \cup \Psi_{n_a-1}$. Let $\hat{\Sigma} \subseteq \Sigma$.

The first step in the verification is to prove that the algorithm runs without error. To prove this, we need to show that each time we 'choose' something in

the algorithm, there is something valid to choose. First we note that the opponent cannot add a description of σ of length $\leq p$ without making a reservation of length p.

Lemma 5. *If $t, p \in \mathbb{N}$, and $\sigma \in 2^{<\omega}$, with $K_{m,t}(\sigma) \leq p$, then $R_p(\sigma, t) \neq \emptyset$.*

Proof. If $K_{m,t}(\sigma) \leq p$, then there exists $\langle \tau, \sigma' \rangle \in U_t$ such that $s = |\tau| \leq p$ and $\sigma' \succeq \sigma$. Consider $\tau' = \tau 0^{p-s}$. As $|\tau'| = p$, $\tau \preceq \tau'$ and $\sigma' \succeq \sigma$, so $\tau' \in F_p(\sigma, t)$. If $\tau' \in I_p(\sigma, t)$, then there would exist $\tau'' \approx \tau'$ and $\rho \mid \sigma$ such that $\langle \tau'', \rho \rangle \in U_t$. However, then $\tau'' \approx \tau$ and $\rho \mid \sigma'$, so this would contradict the definition of a monotone machine. Thus $\tau' \notin I_p(\sigma, t)$ and so $\tau' \in R_p(\sigma, t)$. □

Proposition 4. *The advantage phase of the algorithm never runs out of choices for ξ_υ or i, and the spend phase never runs out of choices for i.*

Proof. First we will show that there is always a choice for ξ_υ. Take any $\upsilon \in \Upsilon$. Take any $j \in \mathbb{N}$ such that $0 \leq j < n_a$. Assume at stage t_j^* for any $\xi \in \Xi$ such that $\xi \succeq \upsilon$, $a(\xi, t_j^*) \neq 0$. If so, then given any $\xi \in \Xi$ such that $\xi \succeq \upsilon$, there is some $i < j$ such that an (r^*_{n-i-1}, f, Ψ_i)-strategy has been implemented with $\{\xi\}\{0,1\}^{r^*_{n-i-1}-q} \subseteq \Psi_i$. There are $2^{r^*_{n-i-1}-q}$ extensions of ξ in Ψ_i, so

$$a(\xi, t_j^*) \geq 2^{r^*_{n-i-1}-q} 2^{-r^*_{n-i-1}} = 2^{-q}.$$

Hence $a(\upsilon, t_j^*) \geq 2^e 2^{-q} = 2^{-p}$, because there are 2^e extensions of υ in Ξ. This means that at the end of the iteration of the previous advantage phase, the algorithm will wait until a stage t when the opponent uses a string of length at most p to describe υ before running the advantage phase again. As this implies that $R_p(\upsilon, t_j^*) \neq \emptyset$ by Lemma 5, it follows that no choice of ξ_υ will be needed.

The advantage phase stops if $|\Upsilon'| < \frac{1}{4}|\Upsilon|$. So each time the phase is run, at least $\frac{1}{4}|\Upsilon|$ elements of Ξ are selected. This can only happen $4 \cdot 2^e = 2^{e+2}$ times (otherwise the algorithm would run out of choices for some ξ_υ), so there is always a choice for i.

The spend phase will terminate if $a(\sigma, t_i') \geq 2^{-r}$ for all $\sigma \in \Sigma$. During each iteration through the phase, $a(\sigma, t_{i+1}') \geq a(\sigma, t_i') + 2^{-r-3}$ (if it is not greater than 2^{-r} already), so after a maximum of 8 steps, $a(\sigma, t) \geq 2^{-r}$ for all $\sigma \in \Sigma$ and so the algorithm will not run out of choices for i. □

The next step in the verification is to show that the required bounds on the measure that the algorithm uses are met.

Proposition 5. *For all $\sigma \in \Sigma$, if the algorithm terminates at stage t_1, then $2^{-r} \leq a(\sigma, t_1) < \frac{5}{4}2^{-r}$.*

Proof. As $a(\sigma, t_0) = 0$, any measure placed on σ by stage t_{mid} must be due to an increase on measure on some $\psi \in \Psi$, where $\psi \succeq \sigma$. Hence for any $\sigma \in \Sigma$, $a(\sigma, t_{mid}) = \sum_{\psi \in \Psi, \psi \succeq \sigma} a(\psi, t_{mid})$. If $\psi \in \Psi_i$, then we increase the measure on ψ by $< \frac{5}{4}2^{-r^*_{n-i-1}}$. Now if $\psi \succeq \sigma$, then $|\psi| = |\sigma| + 1 + p - r + e + r^*_{n-i-1} - q = |\sigma| + 1 - r + r^*_{n-i-1}$, so $r^*_{n-i-1} = |\psi| - |\sigma| - 1 + r$. Thus $a(\psi, t_{mid}) < \frac{5}{4}2^{-(|\psi| - |\sigma| - 1 + r)}$.

All such ψ form a prefix-free set above $\sigma 0$, so

$$\begin{aligned} a(\sigma, t_{mid}) = a(\sigma 0, t_{mid}) &= \sum_{\psi \in \Psi, \psi \succeq \sigma 0} a(\psi, t_{mid}) \\ &< \sum_{\psi \in \Psi, \psi \succeq \sigma 0} \frac{5}{4} 2^{-|\psi| + |\sigma 0| - r} \\ &\leq \frac{5}{4} 2^{-r}. \end{aligned}$$

The spend phase will only run if $a(\sigma) < 2^{-r}$. Each iteration of the spend phase increases the measure on σ by less than $\frac{5}{4}2^{-r-3} < 2^{-r-2}$, so at the end of the spend phase, $2^{-r} \leq a(\sigma, t_1) < 2^{-r} + 2^{-r-2} = \frac{5}{4}2^{-r}$. □

Note that as $a(\Sigma, t_1) < |\Sigma|\frac{5}{4}2^{-r} \leq 1$, we do not run out of measure. There are two possible outcomes to an $(r, f+1, \Sigma)$-strategy. These are related to whether or not the algorithm terminates.

Proposition 6. *If the algorithm does not terminate, then outcome (i) is achieved.*

Proof. The number of times any phase is repeated is bounded, so if the algorithm does not terminate, then this must be caused by waiting for a response from the opponent or while waiting for another strategy to finish. Using the basic strategy as an inductive base case, this can only be caused if for some $\sigma \in \Sigma$, there is a $\sigma' \succeq \sigma$ with $K_m(\sigma') > -\log(a(\sigma'))$. Hence (i) (a) holds. Now (i) (b) holds by Proposition 5 because if the algorithm does not exceed the measure upper bound when it terminates, then it will not exceed the upper bound if it takes no action from some point onwards. □

Now for the difficult stage of the verification. Let us examine what happens if the algorithm does terminate. We are going to look at the gain made during the advantage phase and the spend phase separately. This is because we know from Proposition 3 that the overall gain is at least the sum of the gain which occurs during these phases. That is:

$$\mu(G_r^{r'}(\hat{\Sigma}; t_0, t_1)) \geq \mu(G_p^{r'}(\hat{\Sigma}\{0\}; t_0, t_{mid})) + \mu(G_r^p(\hat{\Sigma}\{1\}; t_{mid}, t_1)).$$

We know what we can expect from the spend phase of the algorithm because this is just a series of (r, f, Φ_i)-strategies. However, in the advantage phase we have to show that the overall gain is increased by the fact that some reservations of length p are made; i.e. for some $\upsilon \in \Upsilon$, $R_p(\upsilon, t_{mid}) \neq \emptyset$.

First let us look at the spend phase. For all $i \in \mathbb{N}$, $0 \leq i < n_s$, let $\hat{\Phi}_i = \{\phi \in \Phi_i : \exists \sigma \in \hat{\Sigma}, \phi \succeq \sigma\}$. Note that $\hat{\Sigma}\{1\} \preceq \hat{\Phi}_i$. Let $\hat{\Phi} = \hat{\Phi}_0 \cup \ldots \cup \hat{\Phi}_{n_s - 1}$.

Proposition 7. *If the algorithm terminates, then*

$$\mu(G_r^p(\hat{\Sigma}\{1\}; t_{mid}, t_1)) \geq \left(1 + \frac{f}{2}\right)\left(a(\Sigma\{1\}, t_1) - 2a(\Sigma\{1\} \setminus \hat{\Sigma}\{1\}, t_1)\right).$$

Proof. From Proposition 3 and the inductive hypothesis of the existence of (r, f, Σ)-strategies we know that

$$\begin{aligned}\mu(G_r^p(\hat{\Sigma}\{1\}; t_{mid}, t_1)) &\geq \sum_{i=0}^{n_s-1} \mu(G_{r_{7-i}}^{r_{8-i}}(\hat{\Phi}_i; t_i', t_{i+1}')) \\ &\geq \sum_{i=0}^{n_s-1} (1+\frac{f}{2})(a(\Phi_i, t_{i+1}') - 2a(\Phi_i \setminus \hat{\Phi}_i, t_{i+1}')) \\ &= (1+\frac{f}{2})(a(\Sigma\{1\}, t_1) - 2a(\Sigma\{1\} \setminus \hat{\Sigma}\{1\}, t_1)).\end{aligned}$$

The second inequality is a consequence of condition (c) of outcome (ii). The final equality is due to the fact that we only increase the measure on extensions of $\Sigma\{1\}$ during the spend phase, and that the sets $\Phi_0, \Phi_1, \ldots, \Phi_{n_s-1}$ are incomparable. □

Before we examine the advantage phase, we need some more definitions. We need to define those subsets of Υ that have p reservations, or extend elements of $\hat{\Sigma}$, or both.

(i) Let $\Upsilon^R = \{\upsilon \in \Upsilon : R_p(\upsilon, t_{mid}) \neq \emptyset\}$.
(ii) Let $\hat{\Upsilon} = \{\upsilon \in \Upsilon : \exists \sigma \in \hat{\Sigma}, \upsilon \succeq \sigma\}$.
(iii) Let $\hat{\Upsilon}^R = \Upsilon^R \cap \hat{\Upsilon}$.

For all $\upsilon \in \hat{\Upsilon}^R$, we choose a specific τ_υ in $R_p(\upsilon, t_{mid})$.

The gain made during the advantage phase can be broken down into two parts. First, for every $\upsilon \in \hat{\Upsilon}^R$, $[\tau_\upsilon] \subseteq G_p^{r'}(\hat{\Sigma}\{0\}; t_0, t_{mid})$. This is true because $\tau_\upsilon \in R_p(\upsilon, t_{mid}) = F_p(\upsilon, t_{mid}) \setminus I_p(\upsilon, t_{mid})$. As $\upsilon \succeq \sigma 0$ for some $\sigma \in \hat{\Sigma}$, so $\tau_\upsilon \in F_p(\sigma 0, t_{mid}) \subseteq F_p(\hat{\Sigma}\{0\}, t_{mid})$. Additionally, $\tau_\upsilon \notin I_p(\sigma 0, t_{mid})$, so $\tau_\upsilon \notin I_p(\hat{\Sigma}\{0\}, t_{mid}) \supseteq I_p(\hat{\Sigma}\{0\}, t_0)$. Now as $[I_p(\hat{\Sigma}\{0\}, t_0)] \supseteq [I_{r'}(\hat{\Sigma}\{0\}, t_0)]$, so $[\tau_\upsilon] \cap [I_{r'}(\hat{\Sigma}\{0\}, t_0)] = \emptyset$. Thus:

$$[\tau_\upsilon] \subseteq [F_p(\hat{\Sigma}\{0\}, t_{mid})] \setminus [I_{r'}(\hat{\Sigma}\{0\}, t_0)] = G_p^{r'}(\hat{\Sigma}\{0\}; t_0, t_{mid}).$$

Secondly, we can count the gains made by the recursive use of the strategies in the advantage phase as:

$$\bigcup_{i=0}^{n_a-1} G_{r_{n-i-1}^*}^{r_{n-i}^*}(\hat{\Psi}_i; t_i^*, t_{i+1}^*) \subseteq G_p^{r'}(\hat{\Sigma}\{0\}; t_0, t_{mid}).$$

We would like to combine these to say:

$$\mu(G_p^{r'}(\hat{\Sigma}\{0\}; t_0, t_{mid})) \geq \mu(\bigcup_{\upsilon \in \hat{\Upsilon}^R} [\tau_\upsilon]) + \mu(\bigcup_{i=0}^{n_a-1} G_{r_{n-i-1}^*}^{r_{n-i}^*}(\hat{\Psi}_i; t_i^*, t_{i+1}^*)).$$

However we cannot do this as given υ and i, $[\tau_\upsilon]$ and $G_{r_{n-i-1}^*}^{r_{n-i}^*}(\hat{\Psi}_i; t_i^*, t_{i+1}^*)$ are not necessarily disjoint. Our goal is to find some subsets $\hat{\Psi}_i^R \subseteq \hat{\Psi}_i$, such that for all $\upsilon \in \hat{\Upsilon}^R$, $0 \leq i < n_a$ to have that $[\tau_\upsilon]$ and $G_{r_{n-i-1}^*}^{r_{n-i}^*}(\hat{\Psi}_i \setminus \hat{\Psi}_i^R, t_i^*, t_{i+1}^*)$ are disjoint. To this end we define $\hat{\Psi}_i^R = \{\psi \in \hat{\Psi}_i : \exists \upsilon, \tau_\upsilon \in F_p(\psi, t_{i+1}^*)\}$.

Lemma 6. *For all $\upsilon \in \hat{\Upsilon}^R$, $0 \leq i < n_a$ we have that $[\tau_\upsilon]$ and $G_{r_{n-i-1}^*}^{r_{n-i}^*}(\hat{\Psi}_i \setminus \hat{\Psi}_i^R, t_i^*, t_{i+1}^*)$ are disjoint.*

Proof. Consider if $\psi \in \hat{\Psi}_i \setminus \hat{\Psi}_i^R$. Then by definition it must be that $\tau_\upsilon \notin F_p(\psi, t^*_{i+1})$ and so $[\tau_\upsilon] \cap [F_{r^*_{n-i-1}}(\psi, t^*_{i+1})] = \emptyset$. This implies that $[\tau_\upsilon] \cap G^{r^*_{n-i}}_{r^*_{n-i-1}}(\hat{\Psi}_i \setminus \hat{\Psi}_i^R, t^*_i, t^*_{i_1}) = \emptyset$. □

Our objective now is to figure out how much we lose by removing these sets $\hat{\Psi}_i^R$. To do this we want to move everything down to the level of the Υ strings. We define $\hat{\Upsilon}_i^R = \{\upsilon \in \hat{\Upsilon}^R : \tau_\upsilon \in F_p(\hat{\Psi}_i^R, t^*_{i+1})\}$.

Lemma 7. *If $\tau_\upsilon \in F_p(\psi, t^*_{i+1})$, then $\upsilon \preceq \psi$.*

Proof. If $\upsilon \not\preceq \psi$, then $\upsilon \mid \psi$, so $\tau_\upsilon \in I_p(\upsilon, t^*_{i+1})$ (Lemma 4), which means that $\tau_\upsilon \notin R_p(\upsilon, t_{mid})$. □

Lemma 8. $|\hat{\Psi}_i^R| \leq 2^{r^*_{n-i-1}-q} |\hat{\Upsilon}_i^R|$.

Proof. If $\psi \in \hat{\Psi}_i^R$, then for some υ, $\tau_\upsilon \in F_p(\psi, t^*_{i+1})$. This implies that $\upsilon \preceq \psi$ and further $\upsilon \in \hat{\Upsilon}_i^R$. The result follows as for any $\upsilon \in \Upsilon$ there are at most $2^{r^*_{n-i-1}-q}$ extensions of τ_υ in Ψ_i. □

Lemma 9. *If $i \neq j$, then $\hat{\Upsilon}_i^R \cap \hat{\Upsilon}_j^R = \emptyset$.*

Proof. Assume that $i < j \leq n_a$ and $\upsilon \in \hat{\Upsilon}_i^R$. We have that $\tau_\upsilon \in F_p(\hat{\Psi}_i^R, t^*_{i+1})$. This means that for some $\psi \in \hat{\Psi}_i^R$, $\tau_\upsilon \in F_p(\psi, t^*_{i+1})$, which implies that $\upsilon \preceq \psi$ (Lemma 7), so $\tau_\upsilon \in F_p(\upsilon, t^*_{i+1}) \subseteq F_p(\upsilon, t^*_j)$.

Now as $\tau_\upsilon \in R_p(\upsilon, t_{mid})$ it must be that $\tau_\upsilon \notin I_p(\upsilon, t_{mid})$, which implies that $\tau_\upsilon \notin I_p(\upsilon, t^*_j)$. This gives us that $\tau_\upsilon \in R_p(\upsilon, t^*_j)$. Now during the advantage phase, our algorithm only chooses those elements of Υ that do not currently have a reservation of length p. As υ does have a reservation of length p at stage t^*_j, υ is not picked, so there is no extension of υ in $\Psi_j \supseteq \hat{\Psi}_j^R$. Hence by Lemma 7 we have that $\tau_\upsilon \notin F_p(\hat{\Psi}_j^R, t^*_{i+1})$. This means that $\upsilon \notin \hat{\Upsilon}_j^R$. □

The final lemma that we need is a lower bound on the size of $|\hat{\Upsilon}^R|$.

Lemma 10. $|\hat{\Upsilon}^R| \geq (\frac{3}{4}|\Sigma| - |\Sigma \setminus \hat{\Sigma}|)2^{p-r}$.

Proof. Note that $|\Upsilon^R| \geq \frac{3}{4}|\Upsilon| = \frac{3}{4}|\Sigma|2^{p-r}$ as this is the condition for the algorithm to move to the spend phase. Additionally, $|\Upsilon \setminus \hat{\Upsilon}| = |\Sigma \setminus \hat{\Sigma}|2^{p-r}$. Hence:

$$\begin{aligned}
|\hat{\Upsilon}^R| &= |\Upsilon^R| + |\hat{\Upsilon}| - |\Upsilon^R \cup \hat{\Upsilon}| \\
&\geq |\Upsilon^R| + |\hat{\Upsilon}| - |\Upsilon| \\
&= |\Upsilon^R| - |\Upsilon \setminus \hat{\Upsilon}| \\
&\geq \frac{3}{4}|\Sigma|2^{p-r} - |\Sigma \setminus \hat{\Sigma}|2^{p-r} \\
&= (\frac{3}{4}|\Sigma| - |\Sigma \setminus \hat{\Sigma}|)2^{p-r}.
\end{aligned}$$

□

Proposition 8. *If the algorithm terminates and if $\hat{\Sigma} \subseteq \Sigma$, then*

$$\mu(G_p^{r'}(\hat{\Sigma}\{0\}; t_0, t_{mid})) \geq \left(1 + \frac{f}{2}\right)\Big(a(\Sigma\{0\}, t_1) - 2a(\Sigma\{0\} \setminus \hat{\Sigma}\{0\}, t_1)\Big) + \frac{5}{8}|\Sigma|2^{-r} - |\Sigma \setminus \hat{\Sigma}|2^{-r}.$$

Proof. Lemma 6 shows that the sets $[\tau_\upsilon]$ and $G^{r^*_{n-1+1}}_{r^*_{n-i}}(\hat{\Psi}_i \setminus \hat{\Psi}_i^R; t_i, t_{i+1})$ are disjoint. So:

$$\mu(G_p^{r'}(\hat{\Sigma}\{0\}; t_0, t_{mid})) \geq \sum_{\upsilon \in \hat{\Upsilon}^R} \mu([\tau_\upsilon]) + \sum_{i=0}^{n_a-1} \mu(G^{r^*_{n-i}}_{r^*_{n-i-1}}(\hat{\Psi}_i \setminus \hat{\Psi}_i^R; t_i^*, t_{i+1}^*)).$$

This is the point at which we use part (c) of outcome (ii). We know that the amount of gain that occurs on $\hat{\Psi}_i^R$ is bounded. So if we subtract it from $\hat{\Psi}_i$, we can determine the maximum amount we can lose. Note that as $\hat{\Psi}_i \subseteq \Psi_i$, it follows that $\Psi_i \setminus (\hat{\Psi}_i \setminus \hat{\Psi}_i^R)$ is the disjoint union of $\Psi_i \setminus \hat{\Psi}_i$ and $\hat{\Psi}_i^R$.

With this identity we can determine that

$$\begin{aligned}
&\mu(G^{r^*_{n-i}}_{r^*_{n-i-1}}(\hat{\Psi}_i \setminus \hat{\Psi}_i^R; t_i^*, t_{i+1}^*)) \\
&\quad \geq (1+\frac{f}{2})(a(\Psi_i, t_{i+1}^*) - 2a(\Psi_i \setminus (\hat{\Psi}_i \setminus \hat{\Psi}_i^R), t_{i+1}^*)) \\
&\quad = (1+\frac{f}{2})(a(\Psi_i, t_{i+1}^*) - 2a(\Psi_i \setminus \hat{\Psi}_i, t_{i+1}^*) - 2a(\hat{\Psi}_i^R, t_{i+1}^*)).
\end{aligned}$$

The first inequality comes from part (c) of outcome (ii) applied to our (r, f, Ψ_i)-strategies. Hence we can combine these results along with the facts that $a(\Psi_i, t_{i+1}^*) = a(\Psi_i, t_1)$ and $\sum_{\upsilon \in \hat{\Upsilon}^R} \mu([\tau_\upsilon]) = |\hat{\Upsilon}^R| 2^{-p}$ to get that

$$\begin{aligned}
\mu(G_r^p(\hat{\Sigma}\{0\}; t_0, t_{mid})) \geq & |\hat{\Upsilon}^R| 2^{-p} + \sum_{i=0}^{n_a-1} (1+\frac{f}{2}) a(\Psi_i, t_1) \\
& - \sum_{i=0}^{n_a-1} (1+\frac{f}{2}) 2a(\Psi_i \setminus \hat{\Psi}_i, t_1) \\
& - \sum_{i=0}^{n_a-1} (1+\frac{f}{2}) 2a(\hat{\Psi}_i^R, t_1). \qquad (5.1)
\end{aligned}$$

These terms can be simplified. First,

$$\begin{aligned}
&\sum_{i=0}^{n_a-1} (1+\frac{f}{2}) a(\Psi_i, t_1) - \sum_{i=0}^{n_a-1} (1+\frac{f}{2}) 2a(\Psi_i \setminus \hat{\Psi}_i, t_1) \\
&\qquad = (1+\frac{f}{2})(a(\Sigma\{0\}, t_1) - 2a(\Sigma\{0\} \setminus \hat{\Sigma}\{0\}, t_1)). \qquad (5.2)
\end{aligned}$$

Further $a(\hat{\Psi}_i^R, t_1) < \frac{5}{4} 2^{r^*_{n-i-1}} |\hat{\Psi}_i^R|$ by condition (b) of outcome (ii). Hence:

$$\begin{aligned}
\sum_{i=0}^{n_a-1} (1+\frac{f}{2}) 2a(\hat{\Psi}_i^R, t_1) &\leq \sum_{i=0}^{n_a-1} (1+\frac{f}{2}) 2 \cdot \frac{5}{4} \cdot 2^{r^*_{n-i-1}} |\hat{\Psi}_i^R| \\
&\leq \frac{5}{2}(1+\frac{f}{2}) \sum_{i=0}^{n_a-1} |\hat{\Upsilon}_i^R| 2^{-q} \\
&\leq \frac{5}{2}(1+\frac{f}{2}) |\hat{\Upsilon}^R| 2^{-q}. \qquad (5.3)
\end{aligned}$$

The second and third inequalities are consequences of Lemmas 8 and 9 respectively.

Combining (5.1), (5.2) and (5.3) gives that

$$\begin{aligned}\mu(G_p^{r'}(\hat{\Sigma}\{0\};t_0,t_{mid})) &\geq (1+\frac{f}{2})(a(\Sigma\{0\},t_1)-2a(\Sigma\{0\}\setminus\hat{\Sigma}\{0\},t_1))\\ &\quad + |\hat{\Upsilon^R}|2^{-p}-\frac{5}{2}(1+\frac{f}{2})|\hat{\Upsilon}^R|2^{-q}. \qquad (5.4)\end{aligned}$$

Now because $|\hat{\Upsilon}^R| \geq (\frac{3}{4}|\Sigma|-|\Sigma\setminus\hat{\Sigma}|)2^{p-r}$ (Lemma 10), $q=p+e$, and by choice of e, $\frac{5}{2}2^{-e}(1+\frac{f}{2}) \leq \frac{1}{8}$, it follows that

$$\begin{aligned}|\hat{\Upsilon^R}|2^{-p}-\frac{5}{2}(1+\frac{f}{2})|\hat{\Upsilon}^R|2^{-q} &= |\hat{\Upsilon^R}|2^{-p}(1-\frac{5}{2}2^{-e}(1+\frac{f}{2}))\\ &\geq |\hat{\Upsilon^R}|2^{-p}\frac{7}{8}\\ &\geq (\frac{3}{4}|\Sigma|-|\Sigma\setminus\hat{\Sigma}|)2^{p-r}2^{-p}\frac{7}{8}\\ &\geq \frac{5}{8}|\Sigma|2^{-r}-|\Sigma\setminus\hat{\Sigma}|2^{-r}. \qquad (5.5)\end{aligned}$$

The proposition follows from (5.4) and (5.5). □

Proposition 9. *If the algorithm terminates, then*

$$\mu(G_r^{r'}(\hat{\Sigma};t_0,t_1)) \geq (1+\frac{f+1}{2})(a(\Sigma,t_1)-2a(\Sigma\setminus\hat{\Sigma},t_1)).$$

Proof. Proposition 3 tells us that

$$\mu(G_r^{r'}(\hat{\Sigma};t_0,t_1)) \geq \mu(G_p^{r'}(\hat{\Sigma}\{0\};t_0,t_{mid})) + \mu(G_r^p(\hat{\Sigma}\{1\};t_{mid},t_1)). \qquad (5.6)$$

All we need to do now is to combine and simplify Propositions 7 and 8. We have the following terms to deal with: $(1+\frac{f}{2})a(\Sigma\{1\},t_1)$, $-(1+\frac{f}{2})2a(\Sigma\{1\}\setminus\hat{\Sigma}\{1\},t_1)$, $(1+\frac{f}{2})a(\Sigma\{0\},t_1)$, $-(1+\frac{f}{2})2a(\Sigma\{0\}\setminus\hat{\Sigma}\{0\},t_1)$, $\frac{5}{8}|\Sigma|2^{-r}$ and $-|\Sigma\setminus\hat{\Sigma}|2^{-r}$.

First, grouping the positive terms gives that

$$\begin{aligned}&(1+\frac{f}{2})a(\Sigma\{1\},t_1)+(1+\frac{1}{2})a(\Sigma\{0\},t_{mid})+\frac{5}{8}|\Sigma|2^{-r}\\ &\quad=(1+\frac{f}{2})a(\Sigma,t_1)+\frac{1}{2}\frac{5}{4}|\Sigma|2^{-r}\\ &\quad>(1+\frac{f}{2})a(\Sigma,t_1)+\frac{1}{2}a(\Sigma,t_1)\\ &\quad=(1+\frac{f+1}{2})a(\Sigma,t_1). \qquad (5.7)\end{aligned}$$

Secondly, the negative terms give that

$$\begin{aligned}&(1+\frac{f}{2})2a(\Sigma\{1\}\setminus\hat{\Sigma}\{1\})+(1+\frac{f}{2})2a(\Sigma\{0\}\setminus\hat{\Sigma}\{0\})+|\Sigma\setminus\hat{\Sigma}|2^{-r}\\ &\quad=(1+\frac{f}{2})2a(\Sigma\setminus\hat{\Sigma})+|\Sigma\setminus\hat{\Sigma}|2^{-r}\\ &\quad\leq(1+\frac{f}{2})2a(\Sigma\setminus\hat{\Sigma})+a(\Sigma\setminus\hat{\Sigma})\\ &\quad=(1+\frac{f+1}{2})2a(\Sigma\setminus\hat{\Sigma}). \qquad (5.8)\end{aligned}$$

Both inequalities in the above arguments follow from Proposition 5. The lemma follows by combining (5.6), (5.7) and (5.8) along with Propositions 7 and 8. □

Proposition 10. *If the algorithm terminates, then outcome (ii) occurs.*

Proof. By Proposition 5, condition (ii) (a) holds. By Proposition 9, condition (ii) (c) holds. This implies condition (ii) (b) by taking $\hat{\Sigma}$ to be Σ. □

Hence we have established the existence of an $(r, f+1, \Sigma)$-strategy. By induction we can assert the existence of an $(r, f+1, \Sigma)$-strategy for any $f, r \in \mathbb{N}$ such that $|\Sigma|\frac{5}{4}2^{-r} < 1$.

Proposition 11. *If $r \geq 1$ and $f2^{-r-1} \geq 1$, then for any $\sigma \in 2^{<\omega}$ such that $a(\sigma, t_0) = 0$, an $(r, f, \{\sigma\})$-strategy achieves outcome (i).*

Proof. Because $|\{\sigma\}|\frac{5}{4}2^{-r} < 1$ we have a sufficient measure to implement such a strategy. However, if the algorithm terminates at some stage t_1, then

$$\mu(G_r^{r'}(\{\sigma\}; t_0, t_1)) \geq (1 + \frac{f}{2})a(\sigma, t_1) \geq (1 + \frac{f}{2})2^{-r} = 2^{-r} + f2^{-r-1} > 1.$$

This is impossible, so outcome (i) must occur. □

5.1. **Construction of the semimeasure a.** We can run a countable number of winning strategies in our construction of a. To do so, at stage s we find the least prime p that divides s. If p is the ith prime, then we run one step of the ith strategy.

Proposition 12. *There is a c.e. semimeasure a such that for all i, $a(0^i1) \leq 2^{-2i-1}$ and there exists a σ_i such that $K_m(0^i1\sigma_i) > -\log(a(0^i\sigma_i))$.*

Proof. The semimeasure a can be constructed by for all i running a $(2i+2, 2^{2i+3}, \{0^i1\})$-strategy. Then for all i, $a(0^i1) < \frac{5}{4}2^{-2i-2} < 2^{-2i-1}$. Also by Proposition 11 because $2^{2i+3}2^{-2i-2-1} = 1$, the strategy achieves outcome (i) and hence there is some σ_i such that $K_m(0^i1\sigma_i) > -\log(a(0^i1\sigma_i))$. □

Now Theorem 1 holds by Proposition 1 and Proposition 12.

6. A lower bound on the difference

To determine a lower bound on the difference between K_m and KM, we need to determine the maximum string length used by an (r, f, Σ)-strategy. We will approach this by determining an upper bound for the number of bits appended to a string in Σ by the strategy. If $f = 0$, then the strategy does not use any extensions of strings in Σ, so the number of extra bits appended is 0. If $f > 0$, an upper bound is the number of bits needed to extend a string in Σ to a string in Ψ_0. This requires $1 + r^*_{n-1} - r = 1 + r' - r - S(f-1) = 1 + S(f) - S(f-1)$ bits (the extra 1 corresponds to the use of a 0 or 1 to distinguish between the advantage and spend phases).

Now an upper bound on the maximum string length used can be obtained by taking the maximum length l of any string in Σ and adding the upper bound for the number of bits added by an (r, i, Σ)-strategy for all i, $1 \leq i \leq f$. Hence we get that the maximum possible string length is

$$l + \sum_{i=1}^{f}(1 + S(i) - S(i-1)) = l + f + S(f) - S(0) = l + f + S(f).$$

Lemma 11. *If n is sufficiently large, then $S(n+1) \leq 2^{2n \log n}$.*

Proof. Unraveling the definition of $S(n)$ gives that $S(0) = 0$, and $S(n+1) = (8 + 2^{e+2})S(n) + e + 3$, where $e = \lceil \log(20 + 10n) \rceil$. So for all $n \in \mathbb{N}$,

$$S(n+1) \leq (2^3(20 + 10n))S(n) + \log(20 + 10n) + 4.$$

So there is some N such that for all $n \geq N$, $S(n+1) < 81nS(n)$. Hence for such an n,

$$S(n+1) < S(N) \cdot 81^{n-N} \frac{n!}{(N-1)!} < (81S(N))^n n^n.$$

Thus the lemma holds if $n \geq 81S(N)$ as in such cases $S(n+1) < n^{2n} = 2^{2n \log n}$. □

We will now prove our main theorem.

Proof of Theorem 7. To prove this, we need to generalize Proposition 1 a little. For any $d \in \mathbb{N}$, the series $\sum_{i \in \mathbb{N}} 2^{-\frac{i}{d}}$ converges by the ratio test, so we can take $k \in \mathbb{N}$ to be such that 2^k is larger than the sum of this series. Now we can construct a semimeasure a such that for all $i \in \mathbb{N}$, $a(0^i 1) \leq 2^{-(1+\frac{1}{d})i-k}$. We construct a by for all $j \in \mathbb{N}$, running a $(j(d+1)+k+1, 2^{j(d+1)+k+2}, \{0^{jd}1\})$-strategy. We do not exceed the bound on the measure for a because

$$\begin{aligned} a(0^{jd}1) &< \frac{5}{4} 2^{-j(d+1)-k-1} \\ &< 2^{-j(d+1)-k} \\ &= 2^{-(1+\frac{1}{d})jd-k}. \end{aligned}$$

Because $f 2^{-r-1} = 2^{j(d+1)+k+2} 2^{-j(d+1)-k-2} = 1$, the strategy achieves outcome (i). Thus for all $j \in \mathbb{N}$, there is some $\sigma_j \succeq 0^{jd}1$, such that $K_m(\sigma_j) > -\log a(\sigma_j)$. We can apply the same scaling to a to construct a semimeasure m; i.e. for all $\sigma \in 2^{<\omega}$, $m(0^i\sigma) = 2^i a(0^i\sigma)$. Again:

$$\begin{aligned} m(\lambda) &= \sum_{i \in \mathbb{N}} m(0^i 1) \\ &= \sum_{i \in \mathbb{N}} 2^i a(0^i 1) \\ &\leq \sum_{i \in \mathbb{N}} 2^{-\frac{i}{d}-k} \\ &= 2^{-k} \sum_{i \in \mathbb{N}} 2^{-\frac{i}{d}} \leq 1. \end{aligned}$$

We now get that for all $j \in \mathbb{N}$, there is some $\sigma_j \succeq 0^{jd}1$ such that $K_m(\sigma_j) > -\log a(\sigma_j) = -\log(2^{-jd} m(\sigma_j)) = -\log m(\sigma_j) + jd$. By the discussion at the start of this section, $|\sigma_j| \leq |0^{jd}1| + S(n) + n$, where in this case $n = 2^{j(d+1)+k+2}$. If j is large enough, then by Lemma 11, the maximum string used by this strategy is less than

$$jd + 1 + 2^{2(2^{j(d+1)+k+2}(j(d+1)+k+2))}.$$

Again if j is large enough this term is less than $2^{2^{j(d+2)}}$. Hence for infinitely many j, there exists a string σ_j such that $|\sigma_j| < 2^{2^{j(d+2)}}$ or $\log\log|\sigma_j| < j(d+2)$. As M_U

majorizes all c.e. continuous semimeasures, there must be some constant b_d such that $K_m(\sigma_j) - KM(\sigma_j) > jd - b_d$. So:

$$\begin{aligned} K_m(\sigma_j) - KM(\sigma_j) &> jd\frac{j(d+2)}{j(d+2)} - b_d \\ &> \frac{d}{(d+2)} \log\log|\sigma_j| - b_d. \end{aligned}$$

Hence for any $c < 1$, there is a d such that $\frac{d}{(d+2)} > c + \epsilon$ for some $\epsilon \in \mathbb{R}^{>0}$. Now for sufficiently large j it must be that $\epsilon \log\log|\sigma_j| > b_d$. Hence

$$\begin{aligned} K_m(\sigma_j) - KM(\sigma_j) &> (c+\epsilon)\log\log|\sigma_j| - b_d \\ &> c\log\log|\sigma_j|. \end{aligned}$$

□

While it has been significantly reduced, there is still a gap between the best known upper bound on the difference between KM and K_m, and this new lower bound. Hence there is further work that could be done lowering the upper bound, or potentially improving on the approach presented here.

Acknowledgments

The author would like to thank his Ph.D. supervisors, Rod Downey and Noam Greenberg, for helpful discussions. The author also acknowledges the support of the New Zealand Tertiary Education Commission. Finally, the author would like to thank Peter Gács for his helpful comments on an early draft and in particular his observation that constant c in the main theorem could be improved to 1.

References

1. William C. Calhoun, *Degrees of monotone complexity*, J. Symbolic Logic **71** (2006), 1327–1341. MR2275862 (2008i:68047)
2. G.J. Chaitin, *Incompleteness theorems for random reals*, Adv. Appl. Math. **8** (1987), no. 2, 119–146. MR886921 (88h:68038)
3. R.G. Downey and D.R. Hirschfeldt, *Algorithmic randomness and complexity*, Springer-Verlag, to appear.
4. Péter Gács, *On the relation between descriptional complexity and algorithmic probability*, Theor. Comput. Sci. **22** (1983), 71–93, Short version: Proc 22nd IEEE FOCS (1981) 296-303. MR693050 (84h:60010)
5. L.A. Levin, *On the concept of a random sequence*, Soviet Math. Dokl. **212** (1973), no. 5, 1413–1416. MR0366096 (51:2346)
6. L.A. Levin and A.K. Zvonkin, *The complexity of finite objects and the development of the concepts of information and randomness of means of the theory of algorithms*, Russian Math. Surveys **25** (1970), no. 6. MR0307889 (46:7004)
7. Ming Li and Paul Vitányi, *An introduction to Kolmogorov complexity and its applications (2nd ed.)*, Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 1997. MR1438307 (97k:68086)
8. André Nies, *Computability and randomness*, Oxford University Press, 2009. MR2548883
9. C.P. Schnorr, *Process complexity and effective random test*, Journal of Computer and System Sciences **7** (1973). MR0325366 (48:3713)
10. ———, *A survey of the theory of random sequences*, pp. 193–210, in: Basic Problems in Methodology and Linguistics, D. Reidel, Dordrecht, Holland, 1977. MR0517133 (58:24431)
11. V.A. Uspensky and A. Shen, *Relations between varieties of Kolmogorov complexities*, Math. Systems Theory **29** (1996), 271–292. MR1374498 (97c:68074)

School of Mathematics, Statistics and Operations Research, Victoria University of Wellington, Wellington 6140, New Zealand

E-mail address: adam.day@msor.vuw.ac.nz