Fundamenta Informaticae 51(4) (2002) 325–338
IOS Press

Another Example of Higher Order Randomness

Verónica Becher [C]
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Pabellón I, Ciudad Universitaria, (1428) Buenos Aires, Argentina
vbecher@dc.uba.ar

Gregory Chaitin
IBM Thomas J. Watson Research Center
P O Box 218, Yorktown Heights, NY 10598, USA
chaitin@us.ibm.com

Abstract. We consider the notion of algorithmic randomness relative to an oracle. We prove that the probability β that a program for infinite computations (a program that never halts) outputs a cofinite set is random in the *second* jump of the halting problem. Indeed, we prove that β is exactly as random as the halting probability of a universal machine equipped with an oracle for the second jump of the halting problem, in spite of the fact that β is defined without considering oracles.

Keywords: randomness, program size complexity, infinite computations.

1. Introduction

The theory of program size defines randomness on the basis of computability: an infinite sequence is random if and only if its initial segments have essentially the same size as the shortest (minimal length) computer programs needed to generate them [5].

The primary notion of effective computability is given by Turing machines. But when these machines are equipped with an oracle for a subset A of natural numbers, i.e., an external procedure that answers

[C]Corresponding author

questions of the form "is n in A", they define the notion of relative computability (or Turing reducibility). A set A is computable from B (or recursive in B) if there is a Turing machine which, when equipped with an oracle for B, computes the characteristic function of A. The relation of relative computability induces an equivalence relation on the subsets of natural numbers (Turing equivalence classes or Turing degrees), and a partial order on the equivalence classes [11].

The notion of relative computability allows for a definition of relative randomness: a sequence is random in A if its initial segments can have essentially the same size as the shortest programs needed to generate them in a computer equipped with and oracle A.

If a set A is computable from B, then randomness in B implies randomness in A. Thus, the Turing equivalence relation on oracles induces an equivalence relation on sets of sequences according to their property of relative randomness, and a partial order on these equivalence classes.

Although the definition of randomness is given for infinite sequences (over a finite alphabet), the definition immediately extends to real numbers. A specific example of a random real is Ω [5], the probability that a self-delimiting universal machine eventually halts. Ω is plainly random, or random in the empty set. The halting probability Ω' of a self-delimiting universal machine equipped with an oracle for the halting problem is random in the first jump of the halting problem. Similarly, Ω'' , Ω''' ,..., the halting probabilities of self-delimiting universal machines equipped with oracles for the respective jumps of the halting problem, are random in the respective jumps.

We are interested in examples of higher order randomness that are not defined in terms of oracles. In [1] we proved that the probability that a program for infinite computations outputs finitely many symbols is random in the *first* jump of the halting problem. In this note we go one step further: we show that the probability β that a program for infinite computations outputs a cofinite set is random in the *second* jump of the halting problem.

We show that β is exactly as random as Ω''. However, although β and Ω'' are Turing equivalent, the proof of randomness of β is not immediate from the randomness of Ω''. This is because randomness is not a recursively invariant property: if two real numbers are computable from Turing equivalent sets, one may be random but the other may not [3] (see Prop. 2.2).

The present work proves results announced by the second author in [8], and continues early work on the program size complexity of infinite computations [6, 14].

2. Preliminaries

We denote $\mathbb{N}$ the set of natural numbers and we work with the binary alphabet $\Sigma = \{0, 1\}$. As usual, we refer to a finite sequence of elements of Σ as a string, and we denote the empty string by λ. Σ^* is the set of all strings on the Σ alphabet.

For $a \in \Sigma^*$, $|a|$ denotes the length of a. We write $a \preceq b$ if a is a prefix of b, and we write $a \prec b$ if a is a proper prefix of b (that is, $|a| < |b|$). We assume the recursive bijection $string(i)$ as the i-th string in the length and lexicographic order over Σ^*.

The abstract definition of a Turing machine is a partial recursive function $f : \Sigma^* \to \Sigma^*$; if the machine is a equipped with an oracle A, the function is recursive in A. The domain of f is considered a family of programs and the value of $f(p)$ —if there is any— is the output of a halting computation on the Turing machine on input p. As usual, we write $f(p)\downarrow$ when the function is defined, and $f(p)\uparrow$ otherwise. To deal with program inputs we consider a recursive bijection $\langle . , . \rangle : \Sigma^* \to \Sigma^*$, and we use

the convention $f(ps_1s_2 \dots s_n) = f(\langle p, \langle s_1, \dots \langle s_{n-1}, s_n\rangle \dots\rangle$. A function that can be computed by a machine equipped with an oracle A is said recursive in A, and a set that can be enumerated by a machine with an oracle A is said to be recursively enumerable in A.

In this work we are only concerned with oracles that are defined by applications of the *jump* operation. Assume a given effective enumeration of all finite lists of Turing machine instructions relative to an arbitrary oracle (notice that the enumeration does not depend on the oracle). For any set A, the jump of A is $A' = \{x : \varphi_x^A(x)\downarrow\}$, where φ_x^A is the partial function recursive in A determined by the $x+1$-th list of instructions relative to the set A. We concentrate exclusively on the first and second jumps of the empty set. The first is $\emptyset' = \{x : \varphi_x^{\emptyset}(x)\downarrow\}$, which is recursively enumerable. The second jump is $\emptyset''$ $= \{x : \varphi_x^{\emptyset'}(x)\downarrow\}$, which is recursively enumerable in $\emptyset'$.

The Arithmetical Hierarchy provides a classification of arithmetic relations according to their syntactical representation in First Order Logic. Σ_n^0 is the class of relations definable by a formula in prenex form with recursive matrix and n quantifier alternations in the prefix, the outer quantifier being existential. Π_n^0 is defined similarly, with the outer quantifier being universal. Δ_n^0 is $(\Sigma_n^0 \cap \Pi_n^0)$, i.e., the class of relations definable in both the n-quantifier forms. The Arithmetical Hierarchy admits also a purely recursion-theoretical definition: A relation is Δ_{n+1}^0 if and only if it is recursive in a Σ_n^0 or Π_n^0 relation. A relation is Σ_{n+1}^0 if and only if it is recursively enumerable in a Σ_n^0 or Π_n^0 relation. A relation is Π_{n+1}^0 if and only if its complement is recursively enumerable in a Σ_n^0 or Π_n^0 relation. Thus, Δ_1^0 is the class of recursive relations and Σ_1^0 is the class of recursively enumerable relations.

The Arithmetical Hierarchy relativizes to a given oracle A, defining the classes $\Sigma_n^{0,A}$, $\Pi_n^{0,A}$ and $\Delta_n^{0,A}$, by simply substituting "recursive" with "recursive in A". The Arithmetical Hierarchy and the hierarchy generated by the iterated application of the jump operator are related: $\forall n \geq 0$ $\emptyset^{(n)}$ is Σ_n^0-complete and $A \in \Delta_{n+1}^0$ iff A is recursive in $\emptyset^{(n)}$.

Let's also recall Shoenfield's Limit Lemma ([12] p.56)

Proposition 2.1. (Shoenfield's Limit Lemma)
A set A is in Δ_2^0 of the Arithmetical Hierarchy if and only if its characteristic function is the limit of a recursive function g, i.e., $c_A(x) = lim_{t\to\infty} g(x,t)$.

$\emptyset'$ is in Σ_1^0, thus also in Δ_2^0; hence its characteristic function is obtainable as the limit of a recursive function. In general, $\emptyset^{(n+1)}$ is in Σ_n^0, hence in $\Delta_2^{0,\emptyset^{(n)}}$; so, its characteristic function is obtainable as the limit of a function recursive in $\emptyset^{(n)}$.

Σ^ω is the set of all infinite binary sequences. For $X \subseteq \Sigma^\omega$ the set theoretic measure of X is denoted by $\mu(X)$ and represents the probability that any arbitrary sequence belongs to X. For $B \subseteq \Sigma^*, B\Sigma^\omega$ denotes the open subset of Σ^ω whose elements have an initial segment in B. For example, for a particular string $s \in \Sigma^*$, $s\Sigma^\omega$ is the set of all sequences starting with s, and $\mu(s\Sigma^\omega) = 2^{-|s|}$. Infinite binary sequences can be identified with real numbers in $[0,1]$, when the sequence is taken as the binary expansion of a real number. Hence, every real in $[0,1]$ has a corresponding sequence in Σ^ω. Rational numbers of the form $k2^{-i}$, for natural numbers i,k, have two corresponding sequences, one ending with infinitely many 1's, the other with infinitely many 0's. Since they form a set of measure 0, this fact does not affect the considerations over probabilities that we make in this work. We refer to real numbers and elements of Σ^ω indistinctly. We use $\mathtt{x}$ to denote a real number or an infinite sequence and we write $\mathtt{x}_i$ to denote the prefix of $\mathtt{x}$ of length i.

A set of strings is *prefix free* if and only if no proper extension of an element of the set belongs to the

set. That is, $B \subseteq \Sigma^*$ is prefix free iff $\forall a, b \in \Sigma^*$ if $a \in B$ and $b \neq \lambda$ then $ab \notin B$. For example, the set $\{\lambda\}$ is prefix free and so is $\{0^n 1 : n \geq 0\}$.

Prefix free sets satisfy Kraft's inequality [10]:

$$\text{If } B \subseteq \Sigma^* \text{ is prefix free, then } 0 \leq \sum_{a \in B} 2^{-|a|} \leq 1.$$

This property allows us to conveniently express the measure of a set of all sequences extending a prefix free set: If $B \subseteq \Sigma^*$ is prefix free then $\mu(B\Sigma^\omega) = \sum_{a \in B} 2^{-|a|}$.

A prefix free set $B \subset \Sigma^*$ is maximal when $\forall a \notin B$, $B \cup \{a\}$ is not prefix free. If B is finite and maximal prefix free then every sequence $\mathtt{x} \in \Sigma^\omega$ has an initial segment in B, and $\sum_{a \in B} 2^{-|a|} = 1$.

A set of strings is *suffix closed* if and only if every extension of an element of the set also belongs to the set. That is, $B \subseteq \Sigma^*$ is suffix closed iff $\forall a, b \in \Sigma^*$ if $a \in B$ then $ab \in B$.

Let us remark that for every suffix closed set B there exists a unique prefix free set $A \subset B$ such that $A\Sigma^* = B$. This prefix free set A is the set of minimal elements of B with respect to the prefix ordering $\preceq$. It holds that $\mu(B\Sigma^\omega) = \mu(A\Sigma^\omega) = \sum_{a \in A} 2^{-|a|}$.

Finally we recall two definitions.

A real $\mathtt{x}$ is *computable* [15] iff there is a total recursive function $f : I\!N \to \Sigma$ such that $f(n)$ is the n-th symbol in the binary expansion of the fractional part of $\mathtt{x}$.

A real $\mathtt{x}$ is *left computably enumerable* [13] (resp. right computably enumerable) iff its left (resp. right) Dedekind cut is recursively enumerable. I.e., iff there exists a recursive increasing (decreasing) sequence of rationals whose limit is $\mathtt{x}$.

Both definitions relativize to any oracle A.

2.1. Randomness: incompressibility in self-delimiting machines

For defining random sequences as those whose initial segments are algorithmically incompressible it is required to consider *self-delimiting* Turing machines (see [7, 2, 9] for an explanatory exposition). The space of programs has to be interpretable as a probabilistic space with all program symbols uniformly distributed. This rules out having a blank marking the end of the program. Since there are no blanks on the input tape, nor any other external way of delimitation, a program must contain in itself the information to know where it ends, so the machine can realize when to finish reading the input tape; this is what self-delimiting means.

Self-delimiting Turing machines are assumed to have: a pre-given finite table that determines the computation, an input (or program) tape, a work tape and an output tape. The input tape contains a first dummy cell (to allow for no input) and then just 0's and 1's. The input tape can only be read by the machine, while the output tape can only be written with 0's and 1's. Both tapes are infinite to the right and their heads *only* move in that direction. The work tape can be read, written and erased; it is infinite in both directions and its head moves in both directions.

Assume an effective enumeration of machines. This enumeration is possible because each machine is determined by its table of instructions, which is finite.

Self-delimiting machines can be equipped with an oracle A, adding to the previous architecture an oracle tape, infinite to the right, that can only be read by the machine. The i-th square contains 1 if $string(i) \in A$, and 0 otherwise. Assume also an effective enumeration of machines equipped with an arbitrary oracle (the enumeration is independent of the oracle).

The abstract definition of a self-delimiting Turing machine (with an oracle A) is a function $f : \Sigma^* \to \Sigma^*$ partial recursive (in A), whose domain is prefix free [5]: if $f(a)\downarrow$ then $f(b)\uparrow$ for all b that are proper extensions of a.

Machines that are capable of simulating any other machine are universal. We choose a self-delimiting universal machine $U : \Sigma^* \to \Sigma^*$ (with no oracle) such that $U(0^i 1p) = M_i(p)$, where M_i is the i-th machine in the effective enumeration. U reads its input tape until it finds the first 1. If it read i 0's, it starts simulating the execution of M_i, taking the rest of the input tape as an input for M_i. We also fix self-delimiting universal machines $U' : \Sigma^* \to \Sigma^*$ and $U'' : \Sigma^* \to \Sigma^*$, containing oracles for the first and second jump of the halting problem respectively, such that $U'(0^i 1p) = M'_i(p)$ and $U''(0^i 1p) = M''_i(p)$, where i denotes is the index in the given enumeration.

The *program size complexity* [5] in a given self-delimiting machine $f : \Sigma^* \to \Sigma^*$ is a function $H_f : \Sigma^* \to I\!N$ which maps a string s to the length of the shortest programs that output s.

$$H_f(s) = \begin{cases} min\{|p| : f(p) = s\} & \text{if } s \text{ is in the range of } f. \\ \infty & otherwise \end{cases}$$

Since the subscript f can be any machine, even one equipped with an oracle, this is a definition of program size complexity for both, effective and relative computability. The function H_f is not recursive. When f is a universal machine, H_f is total and $\exists c\, \forall s \;\; H_f(s) \leq |s| + H_f(|s|) + c \leq |s| + 2 \;\; log\, |s| + c$, where log stands for base 2 logarithm. This upper bound is obtained considering a program that contains explicitly the actual string s plus a codification of the string length.

A machine f is *asymptotically optimal* if and only if for any machine g, there is a constant c such that for all s, $H_f(s) \leq H_g(s) + c$. For any pair of asymptotically optimal machines U_1 and U_2 there is a constant c such that for every string s, $|H_{U_1}(s) - H_{U_2}(s)| \leq c$. This is known as the invariance theorem and implies that program size complexity is independent of the asymptotically optimal machine. Thus, program size complexity on asymptotically optimal machines counts as an absolute measure of complexity, up to an additive constant. The universal self-delimiting machine $U(0^i 1p) = M_i(p), i \in I\!N$, is asymptotically optimal because $\forall i \forall s \in \Sigma^*$, $H_U(s) \leq H_{M_i}(s) + i + 1$. For the same reason, U' and U'' that we fixed are also asymptotically optimal.

Randomness of a sequence is defined in terms of the program size complexity of its initial segments.

Definition 2.1. ([5] Random in A)
Let U^A be a self-delimiting universal machine with oracle A.
A sequence $\mathtt{x} \in \Sigma^\omega$ is random in A iff $\exists c\, \forall n \quad H_{U^A}(\mathtt{x}_n) > n - c$.

A real number in $[0, 1]$ is random if its corresponding binary sequence is random. The definition is given for the alphabet $\Sigma = \{0, 1\}$, but it can be shown to be invariant under any alphabet [2]. That is, the property of being random is inherent to the number and it is independent of the system in which it is represented.

A significant class of random reals was defined by Chaitin [5]:

$$\Omega = \sum_{U(p)\downarrow} 2^{-|p|}$$

for a self-delimiting universal machine U. Since the domain of U is prefix free, $\Omega = \mu(dom(U)\Sigma^\omega)$ is the probability that a self-delimiting universal machine U halts. Ω is left computably enumerable and random [5, 4].

We remark that the property of being random is not recursively invariant in the following sense: given two Turing equivalent real numbers, one may be random but the other may not. A general result about this fact was given by Calude and Nies [3].

Proposition 2.2. For prefix sets $X \subseteq \Sigma^*$ let $r(X) = \sum_{p \in X} 2^{-|p|}$. There are recursively isomorphic sets $X, Y \subset \Sigma^*$ such that $r(X)$ is random but $r(Y)$ is not random.

Proof:
Let X be the $dom(U)$ and $Y = \{0^{i-1}1 : string(i) \in dom(U)\}$. By definition, the i-th bit of $r(Y)$ is 1 iff $U(string(i))\downarrow$. Then, the first 2^n bits of $r(Y)$ are determined by the halting behavior of all programs of length less than n. There will be m of them, $0 \leq m < 2^n$ that halt in U.

For any n, the first 2^n bits of $r(Y)$ can be dramatically compressed: there is an algorithm which given 2^n and m, dovetails all programs of length less than n and finds the m ones that halt and determines the first 2^n bits of $r(Y)$. Then, $H_U(r(Y)_{2^n}) \leq \log(2^n) + \log(m) + c \leq 2\log(2^n) + c = 2n + c$, for some constant c. Hence, $r(Y)$ is not random, while $dom(U)$ and Y are recursively isomorphic and $r(dom(U)) = \Omega$ is random. □

As expected, the help of oracles can contribute to have programs that are shorter in length.

Proposition 2.3. For any string $H_{U''} \leq H_{U'} \leq H_U$ within constant terms.

Proof:
We prove $H_{U'} \leq H_U$. The other is exactly alike. There is a machine M_i' equipped with an oracle for the halting problem which ignores its oracle and behaves exactly as U. Then for every $p \in \Sigma^*$, $U(p) = U'(0^i 1p)$. If $H_U(s)$ is the length of the minimal size program that produces s in U, there is a program of length $H_U(s) + i + 1$ which produces s in U'. Take the constant term equal to $i + 1$. □

The inequalities of Proposition 2.3 can be strict, as exemplified by sufficiently large initial segments of Ω. On the one hand Ω is random in the empty set, so $\exists c\ \forall n\ H_U(\Omega_n) > n - c$. On the other hand, Ω is computable from the domain of U, that is Ω is recursive in $\emptyset'$. In general, the initial segments of computable sequences have very low program size complexity: If $\mathtt{x}$ is computable from A (or recursive in A), then $\exists c\ \forall i\ \ H_{U^A}(\mathtt{x}_i) \leq H_{U^A}(i) + c$, for a self-delimiting universal machine U^A with oracle A. Thus, $\exists d\ \forall n\ H_{U'}(\Omega_n) \leq H_{U'}(n) + d \leq 2\ log\ n + d$. For every n sufficiently greater than $c + d$, $H_{U'}(\Omega_n) < H_U(\Omega_n)$.

3. Random in the second jump of the halting problem

3.1. Machines for infinite computations

Turing's original definition of the "automatic machine" in his fundamental paper in 1936 [15], is indeed a machine that performs unending or infinite computations. Turing defines a number to be computable if its decimal expansion can be written down by an automatic machine.

Infinite computations do not reach a final state because for every reached combination of a symbol in the input tape, a symbol in the work tape and a state label there is always an entry in the machine table.

However, with respect to the output, an infinite computation may either produce just finite or infinitely many symbols.

An infinite computation is the limit of executing a program for an unlimited number of steps. In order to give an abstract definition we first define a partial recursive function f of two arguments, an input $p \in \Sigma^*$ and a number of steps t. We consider f as a machine that on input p computes t steps and halts. If the number of steps t are insufficient for the machine to read all of p, we require $f(p,t)$ to be undefined. We want $f(p,t)\downarrow$ only if the head of the input tape reaches the last symbol of p in at most t steps. When defined, $f(p,t)$ is the output of the computation, which we interpret as a finite set of natural numbers. n is in the output set iff 0 is written by the machine on the output tape followed by exactly n 1s, followed by another 0.

We write $\mathcal{P}(\mathbb{N})$ for the set of all subsets of $\mathbb{N}$, and $\mathcal{P}_{fin}(\mathbb{N})$ for the the set of all finite subsets of $\mathbb{N}$.

Definition 3.1. (Self-delimiting step machine) Let $f : \Sigma^* \times \mathbb{N} \to \mathcal{P}_{fin}(\mathbb{N})$ be a partial recursive function such that

initialized:	$f(\lambda, 0) \downarrow$
continuously defined:	If $f(p,n)\downarrow$ then $\forall s \prec p\ \exists m < n$ such that $f(s,m)\downarrow$.
	If $f(p,n)\downarrow$ then $\forall m < n\ \exists s \preceq p$ such that $f(s,m)\downarrow$.
self-delimiting:	If $f(p,n)\downarrow$ then $\forall a \neq \lambda$, $f(pa,n)\uparrow$.
recursive domain:	There is a total recursive function $\mathit{def}_f : \Sigma^* \times \mathbb{N} \to \{0,1\}$
	such that $\mathit{def}_f(p,n) = 1$ iff $f(p,n)\downarrow$.
monotone:	If $m < n$, $s \preceq p$, $f(s,m)\downarrow$ and $f(p,n)\downarrow$ then $f(s,m) \subseteq f(p,n)$.

The function f induces a definition for infinite computations. In principle programs are finite objects, just strings. However, we will also face unending computations which are determined by possibly infinitely many bits of the input tape. We define f^∞ for this general case, as a function from sequences to sets of natural numbers:

Definition 3.2. (Machine for infinite computation)
Let $\mathtt{x}_{i_t}$ be the unique prefix of $\mathtt{x} \in \Sigma^\omega$ such that $f(\mathtt{x}_{i_t}, t)\downarrow$ if such prefix exists, and let $f^\infty : \Sigma^\omega \to \mathcal{P}(\mathbb{N})$ be

$$f^\infty(\mathtt{x}) = \begin{cases} \bigcup_{t\geq 0} f(\mathtt{x}_{i_t}, t) & \text{If } \forall t\ \exists i_t\ \ f(\mathtt{x}_{i_t}, t)\downarrow. \\ \uparrow & \text{otherwise.} \end{cases}$$

The result of computing $\mathtt{x}$ for infinitely many steps is the limit of running the initial segments of $\mathtt{x}$ for t steps, for t going to infinity. If defined, $f^\infty(\mathtt{x})$ may be either a finite or an infinite set.

We fix universal machines $U^\infty, U'^\infty : \Sigma^\omega \to \mathcal{P}(\mathbb{N})$ such that U'^∞ is equipped with an oracle for the halting problem in U. As usual, assuming the given enumerations of all tables of instructions, which are finite, $U^\infty(0^i 1\mathtt{x}) = M_i^\infty(\mathtt{x})$ and $U'^\infty(0^i 1\mathtt{x}) = M_i'^\infty(\mathtt{x})$ for the enumerations M_1^∞, M_2^∞, ... and $M_1'^\infty$, $M_2'^\infty$, ... Let u and u' the self-delimiting step machines that respectively induce U^∞ and U'^∞.

3.2. The probability of programs for cofinite sets

We define β as the probability that an arbitrary string $p \in \Sigma^*$ determines a cofinite output:

$$P_{cofinite} = \{p \in \Sigma^* : \exists m \, \forall n > m \, \forall \mathtt{x} \in \Sigma^\omega \;\; n \in U^\infty(p\mathtt{x})\}$$
$$\beta = \mu(P_{cofinite}\Sigma^\omega)$$

We will show that the real number β is random in the second jump of the halting problem. To prove it we use two correspondence results:

(a) between all the programs that halt in U'' and a subset of the programs that output a finite set in U'^∞. And,

(b) between all the programs that output a finite set in U'^∞ and a subset of programs that output a cofinite set in U^∞.

Motivated by different purposes, the left to right side of (a) and (b) were proved by Chaitin in [6] (as Theorems 14 and 16). We prove here the full correspondences using *simulation in the limit* technique, also used in [6], that tells how to perform a simulation of a computation relative to an oracle, in a machine for infinite computations that lacks the oracle. The problem is how to answer the questions to the lacking oracle. The technique requires that the oracle be recursively enumerable in the machine performing the simulation. The simulated program is run in increasing number of steps, using a fake oracle: at step t a question to the oracle is answered "no" unless the question is found to be true less than t steps. As the number of steps t goes to infinity any finite set of questions will eventually be answered correctly by the fake oracle. The existence of this simulation algorithm is guaranteed by Shoenfield's Limit Lemma (Prop. 2.1), which ensures that, in the limit, one jump of the halting problem can be obtained. For (b) we have to simulate $\emptyset'$, which is obtainable in the limit of a recursive function. And for (a) we have to simulate $\emptyset''$, which the limit of a function recursive in $\emptyset'$.

A critical feature of the simulation in the limit of self-delimiting machines is the so called *harmless overshoot* [6]. Consider a simulation of a program p with oracle instructions. Although in the limit the fake oracle realizes its mistakes and provides the correct answers, the simulation may already have read beyond the program p. This happens because the domain of the machine being simulated is not recursively enumerable so the simulation may not know where the program p ends until it has the correct oracle answers. In the meantime, the machine performing the simulation may have read extra symbols from the input tape, while its head can not move backwards. However, the actual value of the extra bits is irrelevant because once the simulation reaches the correct oracle answers it will know where p actually ends. A simulation of a program that halts will have at most finitely many symbols of harmless overshoot.

Lemma 3.1. (adapted from [1])
$\exists \% \in \Sigma^*$ with the following property:

1. $\forall p \in \Sigma^*$ [if $U'(p)$ halts then $\forall \mathtt{x} \in \Sigma^\omega \;\; U^\infty(\%p\mathtt{x})$ is finite].

2. $\forall p \in \Sigma^*$ [if $U'(p)$ halts then $\exists m \, \forall \mathtt{x} \in \Sigma^\omega \, \forall t \, \exists i \leq m \;\; u(\%p\mathtt{x}_i, t)\downarrow$].
(finite harmless overshoot)

3. $\forall p \in \Sigma^* \forall \mathtt{x} \in \Sigma^\omega$ [if $U^\infty(\%p\mathtt{x})$ is finite then $\exists s \in \Sigma^*$ such that $s \prec p\mathtt{x}$ and $U'(s)$ halts].

Proof:
$\%$ contains instructions for U^∞ to perform the simulation in the limit of the program for U' that comes afterwards in the input tape. The following algorithm does that task:

t:=1
maximum:=0 (number of simulated instructions of U' when it halts)
`do forever`

Simulate U' for at most t instructions. For each question to the oracle of whether $U(q)$ halts, simulate $U(q)$ and take as an answer whether it halts in less than t steps.

If U' did not halt, then print t on the output tape. Else, let c be the actual number of instructions of U' that have been simulated (the simulation for $U(q)$ is not charged). If c exceeds the maximum number of simulated instructions for all previous values of t, then update the maximum to c and print c on the output tape. Otherwise nothing is printed on the output tape.

t:=t+1

`end do`

Let us verify that $\%$ has the desired property. Assume $\mathtt{x}$ be any sequence after p on the input tape.

To see that 1 and 2 hold, suppose $U'(p)$ halts. Then p halts in finitely many steps and it asks only finitely many oracle questions. Let Q be the set of programs for U that are consulted to the oracle. Every $q \in Q$ such that $U(q)\downarrow$, does so in some finite number of steps. Let M be the maximum number of steps required to halt by the programs of Q that indeed halt. For values of t less than M, the simulation of some oracle questions will be wrong, but for every value of $t > M$, they will be correct. If $U'(p)$ halts in N steps, then when t exceeds the maximum between M and N, the simulation at step t will find out that p halts. For larger values of t the maximum number of executed steps will stabilize in N and there will be no more symbols printed on the output tape.

Since the simulation finds out that $U'(p)$ halts in at most $\max(M+1, N)$ steps, the amount of bits read from the input tape after $\%$ will never exceed $\max(M+1, N)$. Hence, $\forall t\ \exists i \leq max(M+1, N)$ such that $u(\%(p\mathtt{x})_i, t)\downarrow$ (finite harmless overshoot). Consequently,
$\forall n > max(M+1, N)\ \forall t\ \exists i \leq max(M+1, N)$ such that $u(\%(p\mathtt{x})_i, t)\downarrow$ and $n \notin u(\%(p\mathtt{x})_i, t)$.
Thus, $U^\infty(\%p\mathtt{x})$ is finite.

To prove 3, suppose $\exists m\ \forall n > m\ \ n \notin U^\infty(\%p\mathtt{x})$. Then, during the simulation only finitely many numbers are printed. Say the last number printed is N. We have to show that for some string $s \prec p\mathtt{x}$, $U'(s)$ halts. Suppose not. Then, in particular, the execution of U' does not halt in less than $N+1$ steps. In such $N+1$ steps only a finite number of oracle questions can be asked, over a finitely many programs. Following the same reasoning as before, there is a maximum number of steps M that lead to the correct answer for all the programs $q \in Q$ that halt in U. Therefore, in the maximum step between M and $N+1$, the simulation should find out exactly what U' does in the first $N+1$ instructions, which we assumed does not halt. Then, the number $N+1$ is printed in the output tape, contradicting that the maximum number printed was N. Hence, for some $s \prec p\mathtt{x}$, $|s| \leq N$, it must be $U'(s)$ halts. □

The relativization of this lemma considering U'' and its simulation on U'^{∞}, proves the correspondence (a).

Now we turn to the correspondence (b), between all programs for U'^{∞} that output a finite set and a subset of programs for U^{∞} that output a cofinite set. Again we have to consider the harmless overshoot, which in this case might be infinite. For example, let the following instructions be part of a program for U'^{∞}:

```
do forever
   if (O(U(factorial i)↑)) then read i bits from the input tape
   ...
   i:= i+1
end do
```

When these instructions are run on U'^{∞}, no symbols from the input tape are read because the oracle `O` knows that U(`factorial i`) halts. Instead, the simulation running on U^{∞}, in the limit, will read infinitely many symbols, because infinitely many times the fake oracle will be wrong.

Lemma 3.2. $\exists \# \in \Sigma^*$ with the following property:

$$\forall \mathtt{x} \in \Sigma^{\omega}\ [U'^{\infty}(\mathtt{x}) \text{ is finite} \quad \text{iff} \quad U^{\infty}(\#\mathtt{x}) \text{ is cofinite}\,].$$

Proof:
$\#$ contains instructions for U^{∞} to perform the simulation in the limit of the program for U'^{∞} that comes afterwards in the input tape. The following algorithm does that task.

The algorithm maintains a set HOLES of pairs $(y,\ t)$, meaning that since step t the element y is believed to belong to the output of $U'^{\infty}(\mathtt{x})$. If at some subsequent step t' the element y is no longer believed in the output of $U'^{\infty}(\mathtt{x})$ (this happens when the fake oracle changes at step t' some of the responses it gave at previous steps), the algorithm removes the pair $(y,\ t)$ from HOLES.

We write π_1(HOLES) for the set $\{y : \exists n\ (y, n) \in \text{HOLES}\}$.

$t := 1$
`do forever`

> Y_t := the output set obtained by simulating the computation of t steps of U'^{∞} using the fake oracle (for each question to the oracle of whether $U(q)$ halts, simulate $U(q)$ and take as an answer whether it halts in less than t steps)
>
> For each $y \in Y_t$ such that $y \notin \pi_1(\text{HOLES})$
> (y is a new element encountered at this step t)
>
> > $HOLES := HOLES \cup \{(y, t)\}$
> > $t := t + 1$ (skip as many values of t as the number of new elements)
>
> For each $(z, t_z) \in$ HOLES such that $z \notin Y_t$
> (z was mistakenly believed to be in the output of U'^{∞}).

$HOLES := HOLES \setminus \{(z, t_z)\}$
print t_z (fix the hole witnessed by z)

print t

$t := t + 1$

`end do`

Let us verify that # has the desired property. Let x be any sequence after # on the input tape.

$\Rightarrow$. Suppose $U'^{\infty}(\mathtt{x})$ is a finite set. Then there is a step t_0, since which U'^{∞} outputs no new elements (only repeated elements or no elements at all). In such t_0 steps only finitely many oracle questions Q can be performed. Every $q \in Q$ such that $U(q)$ halts, it does so in some finite number of steps. Let m_Q be the maximum number of steps required to halt by the halting programs in Q. For values of t less than $m_Q + 1$, the simulation of some oracle questions will be wrong, but for every value of $t > m_Q$, they will be correct. Then, for $m = \max(t_0, m_Q + 1)$ the simulation will have found out all the elements in $U'^{\infty}(\mathtt{x})$ and the output of the simulation will contain at least as many holes as the number of elements in $U'^{\infty}(\mathtt{x})$. Namely, $\forall t \geq m\ Y_t \supseteq U'^{\infty}(\mathtt{x})$. And for each $y \notin U'^{\infty}(\mathtt{x})$, for infinitely many values of t, $y \notin Y_t$. Hence, any missing number greater than m, sooner or later will be printed (i.e, all incorrect holes sooner or later will be fixed). Therefore, $U'^{\infty}(\mathtt{x})$ is cofinite.

$\Leftarrow$. Each element $y \in U'^{\infty}(\mathtt{x})$ is obtained after finitely computation steps of U'^{∞}, say t_y, which involve just finitely many oracle questions Q_y. The simulation on U^{∞} using the fake oracle finds the right oracle answers for Q_y in m_y steps. Then, each element $y \in U'^{\infty}$ is found by the simulation no later than step $max(t_y, m_y)$, and for every step $t \geq max(t_y, m_y)$ $y \in Y_t$. Therefore, a pair (y, T_y) —for some value $T_y \leq max(t_y, m_y)$— will remain forever in HOLES, and the value T_y will remain forever missing in the output. Hence, if $U'^{\infty}(\mathtt{x})$ is infinite, $U^{\infty}(\#\mathtt{x})$ is coinfinite. □

Concatenating Lemmas 3.1 and 3.2 we obtain:

Proposition 3.1. $\exists \#\% \in \Sigma^* \ \forall p \in \Sigma^*$ such that

1. if $U''(p)$ halts then $\#\%p \in P_{cofinite}$.
2. if $\#\%p \in P_{cofinite}$ then either $\exists s \in \Sigma^*\ s \preceq p$ and $U''(s)$ halts, or
   $\exists S \subset \Sigma^*$ finite and maximal prefix free such that $\forall s \in S\ U''(ps)$ halts.

Proof:
1. Suppose $U''(p)$ halts. By relativizing Lemma 3.1, we obtain $\forall \mathtt{x} \in \Sigma^\omega\ U'^{\infty}(\%p\mathtt{x})$ is finite and $\exists m\ \forall \mathtt{x}\ \forall t\ \exists i \leq m\ u'((\%p\mathtt{x})_i, t)\downarrow$ (finite harmless overshoot). Since no computation on input $\%p\mathtt{x}$ reads beyond m symbols of the input tape, there are at most 2^m different output sets; that is, $\{U'^{\infty}(\%p\mathtt{x}) : \mathtt{x} \in \Sigma^\omega\}$ is finite and contains at most 2^m different finite sets. For each $\mathtt{x} \in \Sigma^\omega$ let t_x be the minimum step number at which all the different elements have already been found: $\forall t \geq t_x\ \exists i\ u'(\%p\mathtt{x}_i, t) = U'^{\infty}(\%p\mathtt{x})$. It follows that $\{t_x : \mathtt{x} \in \Sigma^\omega\}$ is also finite.

By Lemma 3.2, if $U'^{\infty}(\%p\mathtt{x})$ is finite, $U^{\infty}(\#\%p\mathtt{x})$ is cofinite. According to the algorithm associated with # in the proof of Lemma 3.2, for each x the simulation on U^{∞} discovers the last different element of $U'^{\infty}(\%p\mathtt{x})$ by step $max(t_x, m_x)$, where m_x stands for the number of steps required by the fake oracle

to get the correct answers for all the oracle questions asked in the t_x steps. So, $U^\infty(\#\%p\mathtt{x})$ prints every value greater than $max(t_x, m_x)$. Since $\{max(t_x, m_x) : \mathtt{x} \in \Sigma^\omega\}$ is again finite, it has a maximum element $M = max \bigcup \{max(t_x, m_x) : \mathtt{x} \in \Sigma^\omega\}$. Consequently, $\forall n > M\ \forall \mathtt{x} \in \Sigma^\omega\ \ n \in U^\infty(\#\%p\mathtt{x})$.

2. Assume $\%\#p \in P_{cofinite}$, and let M be such that $\forall n > M\ \forall \mathtt{x} \in \Sigma^\omega\ \ n \in U^\infty(\#\%p\mathtt{x})$. By the proof of Lemma 3.2, we know that $U^\infty(\#\%p\mathtt{x})$ performs the simulation of $U'^\infty(\%p\mathtt{x})$. The missing values in the output of U^∞ are the step numbers at which new elements have been found. Thus, all definitive holes in the output of U^∞ have been produced at some step $\leq M$. Consequently, all the elements of $U'^\infty(\%p\mathtt{x})$ have been found by the simulation on U^∞ by step M, after having read at most M symbols after $\#$ from the input tape. So, $\forall \mathtt{x}\ U'^\infty(\%p\mathtt{x})$ is finite.

Now by Lemma 3.1, for each $\mathtt{x} \in \Sigma^\omega\ \exists s$ such that $s \prec p\mathtt{x}$ and $U''(s)$ halts. A simple application of König's lemma yields that there exists a finite prefix free set $S \subset \Sigma^*$ such that $\forall \mathtt{x} \in \Sigma^\omega\ \ \exists s \in S$, $s \prec p\mathtt{x}$ and $U''(s)$ halts. □

To prove the main result of the paper we use that $P_{cofinite}$ is recursively enumerable in the second jump of the halting problem:

Proposition 3.2. $P_{cofinite}$ is Σ^0_3 in the Arithmetical Hierarchy.

Proof:
Let $u : \Sigma^* \times I\!N \to \mathcal{P}(I\!N)$ be the partial recursive function that induces U^∞ and def_u the total recursive function that decides the domain of u.

$$\begin{aligned} p \in P_{cofinite} \quad &\Leftrightarrow \quad \exists m\, \forall n > m\, \forall \mathtt{x} \in \Sigma^\omega\, \forall t\, \exists i\ \ [def_u((p\mathtt{x})_i, t) = 1 \wedge\ n \in u((p\mathtt{x})_i, t)] \\ &\Leftrightarrow \quad \exists m\, \forall n > m\, \forall t\, \forall s \in \Sigma^*\, \exists s_t \in \Sigma^* \\ &\qquad\quad [(s_t \preceq ps\ \vee\ ps \preceq s_t) \wedge def_u(s_t, t) = 1 \wedge\ n \in u(s_t, t)]. \end{aligned}$$

Since this is a $\exists\, \forall\, \exists$ formula, the set $P_{cofinite}$ is Σ^0_3. □

Clearly $P_{cofinite}$ is suffix closed, since for each $p \in P_{cofinite}$, $\forall a \in \Sigma^*$, $pa \in P_{cofinite}$. The prefix free counterpart of $P_{cofinite}$ is the set of minimal elements in the prefix order $\preceq$:

$$PF_{cofinite} = \{p \in P_{cofinite} : \forall q \prec p,\ q \notin P_{cofinite}\}.$$

We now show that

$$\beta = \mu(P_{cofinite}\Sigma^\omega) = \sum_{p \in PF_{cofinite}} 2^{-|p|}$$

is random in the second jump of the halting problem.

Theorem 3.1. β is random in $\emptyset''$: $\exists c\, \forall n\ \ H_{U''}(\beta_n) > n - c$.

Proof:
Consider the following algorithm for U'' that receives as input a minimal size program for U'' that outputs β_n:

1. Compute β_n.

2. By Proposition 3.2 there is function g recursive in $\emptyset''$ whose range is $P_{cofinite}$. Enumerate enough programs $g(1), g(2), \ldots$ until we have a prefix free set $X = \{g(1), \ldots, g(m)\}$, such that $\beta_n < \sum_{x \in X} 2^{-|x|}$.

3. Since X is prefix free, it contains only minimal elements in the prefix ordering $\preceq$. By Prop. 3.1, for each $\#\%x \in X$ there exists $S \subset \Sigma^*$ finite and maximal prefix free such that $\forall s \in S,\ U''(xs)$ halts. Let $Y = \{p \in \Sigma^* : \exists x\ [\#\%x \in X\ \wedge x \preceq p\ \wedge U^{''}(p)\downarrow]\}$.

4. Output z such that $z \notin \{U^{''}(p) : p \in Y\}$.

5. Halt.

We give bounds for $H_{U''}(z)$. On the one hand, by the universality of $U^{''}$, there exists a minimal size program p such that $U^{''}(p) = z$. But $\#\%p \notin X$ (and no prefix nor extension can be in X), so, by Proposition 3.1, $\#\%p \in P_{cofinite}$. Hence, $\#\%p$ contributes to β with $2^{-|\#\%p|}$.

$$\beta_n + 2^{-|\#\%p|} < \sum_{x \in X} 2^{-|x|} + 2^{-|\#\%p|} < \beta$$

Using that $\beta \leq \beta_n + 2^{-n}$ we obtain

$$\beta_n + 2^{-|\#\%p|} < \sum_{x \in X} 2^{-|x|} + 2^{-|\#\%p|} < \beta \leq \beta_n + 2^{-n}$$

Therefore, $2^{-|\#\%p|} < 2^{-n}$. Thus, $n < |\#\%p|$ and $n - |\#\%| < |p| = H_{U''}(z)$.

On the other hand, z is the output of the algorithm above, so there is a constant q such that $H_{U''}(z) \leq H_{U''}(\beta_n) + q$. Hence, $n - |\#\%| < H_{U''}(z) \leq H_{U''}(\beta_n) + q$. So, $H_{U''}(\beta_n) > n - q - |\#\%|$. Putting $c = q + |\#\%|$ the theorem is proved. □

Proposition 3.3. β is exactly as random as Ω'': $\exists c\, \forall n\ \ |H_{U''}(\Omega''_n) - H_{U''}(\beta_n)| \leq c$.

Proof:
As in the proof of Theorem 3.1, assume given a minimal size program for $U^{''}$ for β_n. Compute β_n and enumerate enough programs of $P_{cofinite}$ until we have a prefix free set $X = \{g(1), \ldots, g(m)\}$ such that $\beta_n < \sum_{x \in X} 2^{-|x|}$.

Since X is prefix free, it contains only minimal elements in the prefix ordering $\preceq$. By Prop 3.1, for each $\#\%x \in X$ there exists $S \subset \Sigma^*$ finite and maximal prefix free such that $\forall s \in S,\ U''(xs)$ halts; since S is finite maximal prefix free, it holds that $\sum_{s \in S} 2^{-|xs|} = 2^{-|x|}$. We conclude that $\Omega''_{n-|\#\%|}$= $\sum_{\#\%x \in X} 2^{-|x|}$. Hence, $H_{U''}(\Omega''_n) \leq H_{U''}(\beta_n) + |\#\%| + constant$.

For the other inequality, assume given a minimal size program for $U^{''}$ for Ω''_n. Compute Ω''_n and enumerate $dom(U^{''})$ until $X = \{g(1), \ldots g(m)\}$ such that $\Omega''_n < \sum_{x \in X} 2^{-|x|}$. By Proposition 3.2, $P_{cofinite}$ is recursively enumerable in $U^{''}$, which implies that there is an algorithm for $U^{''}$ that does the following:

" if $p \in P_{cofinite}$ then halt else loop forever. "

Therefore, $\exists @ \in \Sigma^*$ such that $p \in P_{cofinite}$ iff $U^{''}(@p)$ halts. Then, $\beta_{n-|@|} = \sum_{@p \in X} 2^{-|p|}$. Thus, $H_{U''}(\beta_n) \leq H_{U''}(\Omega_n) + |@| + constant$. □

We conclude that β is computably enumerable in $\emptyset''$ and random in $\emptyset''$, hence, also random in $\emptyset'$, and $\emptyset$. β and Ω'' are equally random, while β is more random than Ω', and more random than the probability α that a program for infinite computations outputs finitely many symbols. This is because both, Ω' and α, are computable from $\emptyset''$, hence not random in $\emptyset''$. And of course, β is not random in $\emptyset'''$ because it is computable from $\emptyset'''$.

The definitions and techniques used in this paper seem to be appropriate to generalize the results for significant classes of real numbers, computably enumerable and random in any jump of the halting problem.

Acknowledgements: We thank Serge Grigorieff and Denis Hirschfeldt for their valuable comments.

References

[1] V. Becher, S. Daicz, and G. Chaitin. A highly random number. In C. S. Calude, M. J. Dineen, and S. Sburlan, editors, *Combinatorics, Computability and Logic: Proceedings of the Third Discrete Mathematics and Theoretical Computer Science Conference (DMTCS'01)*, Springer-Verlag London, pp. 55–68, 2001.

[2] C. Calude. *Information and Randomness. An Algorithmic Perspective*. Springer-Verlag, Berlin, 1994.

[3] C. S. Calude, A. Nies. Chaitin Ω numbers and strong reducibilities, *J. UCS* 3 , pp. 1161–1166, 1997.

[4] C. Calude. A characterization of c.e. random reals. *Theoretical Computer Science* 271(1-2), pp. 3-14, 2002.

[5] G. J. Chaitin. A theory of program size formally identical to information theory. *J. ACM*, 22 , pp. 329–340, 1975.

[6] G. J. Chaitin. Algorithmic entropy of sets. *Computers & Mathematics with Applications*, 2 , pp. 233–245, 1976.

[7] G. J. Chaitin. *Algorithmic Information Theory*. Cambridge University Pres, Cambridge, 1987.

[8] G. J. Chaitin. *Exploring Randomness.* Springer-Verlag, London, 2001.

[9] M. Ferbus-Zanda and S. Grigorieff. Is randomness "native" to Computer Science?. Logic in Computer Science Column. *Bulletin of EATCS*, vol 74. , pp. 78–118, 2001.

[10] L. G. Kraft. A device for quantizing, grouping and coding amplitude modulated pulses. Master's thesis, Dept. of Electrical Engineering, M.I.T., Cambridge, Massachusets, 1949.

[11] P.G. Odifreddi. *Classical Recursion Theory*. North Holland, Amsterdam, Vol. 1, 1989.

[12] J.R. M. Shoenfield. *Recursion theory*, Lecture Notes in Logic, vol. 1, 1993, reprinted 2001, A K Peters, Ltd.

[13] R. Soare. Recursion theory and Dedekind cuts. *Trans. Amer. Math. Soc.*, vol. 140, 271–294, 1969.

[14] R. M. Solovay. On random r.e. sets. In A. I. Arruda, N. C. A. da Costa, and R. Chuaqui, editors, *Non-Classical Logics, Model Theory and Computability*, pp. 283–307. North-Holland Publishing Company, 1977.

[15] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 2nd series*, 42, pp. 230–265, 1936.