A Highly Random Number

Verónica Becher[1], Sergio Daicz[1], and Gregory Chaitin[⋆,2]

[1] Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
{vbecher,sdaicz}@dc.uba.ar
[2] IBM Thomas J. Watson Research Center
chaitin@us.ibm.com

Abstract. In his celebrated 1936 paper Turing defined a machine to be circular iff it performs an infinite computation outputting only finitely many symbols. We define α as the probability that an arbitrary machine be circular and we prove that α is a random number that goes beyond Ω, the probability that a universal self delimiting machine halts. The algorithmic complexity of α is strictly greater than that of Ω, but similar to the algorithmic complexity of Ω', the halting probability of an oracle machine. What makes α interesting is that it is an example of a highly random number definable without considering oracles.

1 Introduction

Although almost all (in the sense of measure theory) real numbers are random, an effective example of one such number was by no means granted. Chaitin's definition of randomness lead to a natural example [4]. Based on his theory of program size, Chaitin formalized the notion of lack of structure and unpredictability of a random sequence: the prefixes of a random sequence can not be algorithmically compressed. Any structure or regularity in a sequence could be considered for compressing it, that is, for codifying certain amount of bits of the sequence in a program having substantially less bits. Thus, in Chaitin's theory a sequence is defined to be random if its prefixes are as long as the computer programs needed to generate them. Chaitin conceived a sequence with this property: the binary expansion of Ω. Ω relates to the simple task Turing found that no computer could do: solve the halting problem. Ω is the probability that a universal self delimiting machine will eventually halt. Ω is not computable because its first n bits would enable us to answer Turing's halting problem for all programs up to n bits in size, and this is impossible.

As there are countably many universal machines we can speak of the class of the Ω-numbers, the real numbers that are their halting probabilities. Recently, the complementary work of Calude [2] and Slaman [9] proved a beautiful characterization result: the class of Ω-numbers coincides with the class of *computably*

⋆ Honorary co-author.

enumerable random reals. A real number is computably enumerable if it can be approximated from below by a non decreasing computable sequence of rationals that converges to the number; that is, if it can be algorithmically approximated in a monotone non decreasing way. For any given self delimiting machine, its halting probability is computably enumerable, since by dovetailing all possible programs we can obtain a monotone non decreasing approximation.

In this note we present a natural example of a random number that goes beyond the class of Ω-numbers. The idea goes back to Turing's celebrated paper "On computable numbers, with an application to the Entscheidungsproblem" [11], where he describes the computable numbers as the real numbers whose decimal expansion is calculable by finite means. According to Turing's definition, a number is computable if its decimal expansion can be written down by a *circle free* machine. This is a machine that performs an unending computation in the course of which it writes down infinitely many symbols. Otherwise, if it writes down only finitely many, Turing defines the machine to be *circular*. Circular machines reach a configuration from which there is no possible move, or go on moving, but do not print any more symbols.

We will prove that the probability that a universal self delimiting machine outputs finitely many symbols is a random number. We call this number α.

We will show that the degree of randomness of α exceeds that of Ω, as it corresponds to the halting probability of an oracle machine. Although a universal computer could never decide in advance whether any given program will eventually halt, Turing conceived an oracle machine, which could nevertheless know one way or the other. Thus, an oracle machine could know Ω. Such a machine would have its own halting probability, Ω', which would be more random than Ω. The hierarchy goes on and on: If there is an oracle that knows Ω, it is easy to conceive a second-order oracle that knows Ω'. This machine, in turn, has its own halting probability, Ω'', which is known only by a third-order oracle. In fact, there exists an infinite sequence of increasingly more random Ωs. There is even an infinitely high-order oracle which knows all other Ωs. Ω_ω is even more random, more uncomputable than $\Omega, \Omega', \Omega'', \ldots$. Thus, $\Omega', \Omega'', \ldots$, are examples of numbers more random than Ω and they are not computably enumerable; however, they are all defined by considering oracles.

We show that α, defined without considering oracles, is essentially Ω', the halting probability of a first-order oracle. Although α and Ω' are defined in terms of recursively equivalent sets, a proof of randomness for α can not be obtained directly from the randomnes of Ω', since –as we will show later– having random probability is not a recursiveley invariant property of sets.

In sum, α is a natural example of a random not computably enumerable number. With it we dip into a topic that deserves more study, the algorithmic complexity of infinite computations. This topic was first studied by Chaitin in [5], but there was no further work in this direction. The only exception was Solovay's paper [10], which discusses the relationship between program size and algorithmic probability for infinite computations. In the last chapter of his recent book [6], Chaitin proposes continuing this line of research to fully develop a

theory of program size for infinite computations. This paper is a first step in that direction. In a subsequent paper we will show that the probability that a universal self delimiting machine outputs a co-finite set is essentially Ω'', even more random than α. That result and the main result of this work were announced in [6].

2 Preliminaries

We will 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 with λ. Σ^* is the set of all strings on the Σ alphabet. For $a \in \Sigma^*$, $|a|$ denotes the length of a. We will write $a \preceq b$ if a is a prefix of b. We define $string_i$ as the i-th string in the length and lexicographic order over Σ^*.

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 $A \subseteq \{0,1\}^*, A\Sigma^\omega$ denotes the open subset of Σ^ω whose elements have an initial segment in A. For example, for a particular string $s \in \Sigma^*$, $s\Sigma^\omega$ denotes 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. Rationals of the form $k2^{-i}$, for natural i, k, have two corresponding sequences, one ending with infinitely many 1s, the other with infinitely many 0s. Since they form a set of measure 0, this fact will not affect the considerations over probabilities that we will make on this work. We will refer to elements of $\mathbb{R}$ and elements of Σ^ω indistinctly. We will use $\mathtt{x}, \mathtt{y}$ to represent real numbers or infinite sequences and we will write $\mathtt{x}_i$ to denote the prefix of $\mathtt{x}$ of length i.

With Σ^∞ we will denote the set $\Sigma^* \cup \Sigma^\omega$, that is, the set of all strings and all sequences over Σ. We assume a partial order over Σ^∞ that extends the prefix partial order of Σ^*, so we also denote it with $\preceq$. For $u, v \in \Sigma^\infty$, $u \preceq v$ if one of the following three situations occur:

1. $u = v$.
2. $u, v \in \Sigma^*$ and $u \preceq v$.
3. $u \in \Sigma^*$, $v \in \Sigma^\omega$ and $v_{|u|} = u$.

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, $A \subseteq \Sigma^*$ is prefix free iff for $\forall a, b \in \Sigma^*$ if $a \in A$ and $b \neq \lambda$ then $ab \notin A$. For example, the set $\{\lambda\}$ is prefix free and so is $\{a^n b : n \geq 1\}$. We will also work with the dual property: 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 $ab \in B$. A trivial example is Σ^*.

Observation 1. 1. For every suffix closed set B there exists a unique prefix free set $A \subset B$ such that $A = \{a : \forall c \in \Sigma^*, ac \in B\}$.
2. For every prefix free set A there exists a unique suffix closed set B such that $B = \{ac : a \in A \text{ and } c \in \Sigma^*\}$.

Prefix free sets satisfy Kraft's inequality [8]:

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

This property allows us to conveniently express the measure of a set of all sequences extending a prefix free set or a suffix closed set.

Observation 2. 1. Let $A \subseteq \Sigma^*$ be prefix free, then $\mu(A\Sigma^\omega) = \sum_{a \in A} 2^{-|a|}$.
2. Let $B \subseteq \Sigma^*$ be suffix closed and let A be its prefix free counterpart, $B = \{ac : a \in A \text{ and } c \in \Sigma^*\}$. Then, $\mu(B\Sigma^\omega) = \mu(A\Sigma^\omega) = \sum_{a \in A} 2^{-|a|}$.

3 Self Delimiting Machines

In order to talk about probabilities over programs, Chaitin required that all program symbols be uniformly distributed. He dropped the blank used in Turing's formalization and considered programs made of just 0s and 1s without endmarkers. Consequently, the machine must realize when to finish reading the program tape, it should have a self delimiting reading behaviour. He formalized this concept requiring that when a program p halts, no extension of p halts. In other words, the set of programs that halt has to be prefix free. The abstract definition of a self delimiting machine is a partial recursive function $f : \Sigma^* \to \Sigma^*$ such that if $f(a) \downarrow$ then $f(b) \uparrow$ for all b that are proper extensions of a. And he used this formalization to develop his algorithmic complexity theory for finite computations.

In [5] Chaitin extends his work for infinite computations and this is the model we follow. However, we introduce a slight modification that will allow us to properly deal with the simulation of programs. We require that every extension of a valid program be a valid program outputting the same result. That is, the set of valid programs has to be suffix closed and every extension of a valid program has to give the same result. This condition is dual to prefix freeness and we will see that it also successfully conforms Chaitin's self delimiting requirement

The self delimiting machine that interests us is the following version of a Turing machine: a pregiven finite table that determines the computation, a program tape, a work tape and output tape. The program tape contains just 0s and 1s and can only be read by the machine, while the output tape can only be written with 0s and 1s. Both tapes are infinite to the right and their heads only move in that direction. The work tape can be read, written and erased; is infinite in both directions and its head moves in both directions. A computation starts with the heads of the program tape and output tape in their respective leftmost cells and the work tape being all blank.

We are interested in infinite computations, programs that do not halt. That is, computations that do not reach a final state, because for every reached combination of a symbol in the program tape, a symbol in the work tape and a state label there is always an entry in the machine table. For such an unending computation there are two possibilities. Either it produces an infinite number

of symbols in the output tape –a sequence–, or it produces just a finite number of symbols in the output tape and then cycles forever. Since programs are finite objects we are only interested in unending computations in the course of which only a finite number of symbols of the program tape are read. Therefore, computations that attempt to read an infinite number of symbols from the program tape are not valid. A program $p \in \Sigma^*$ is *valid* if and only if when starting an infinite computation having a sequence belonging to $p\Sigma^\omega$ in the program tape, the head of the program tape eventually reaches some bit of p and never moves forward. The set of valid programs for our machine is suffix closed, that is, if p is a valid program so is every extension of p: if the head of the program tape stops at some bit of p it will never reach any bit ahead. The stopping reading point is the same for every string extending p. We can understand now why this machine is self delimiting. Since there are no blanks in the program tape, nor any other external way of delimitation a program must contain in itself the information to know where it ends.

Now we give an abstract definition for infinite computations. Intuitively, an infinite computation is the result of executing a program (finitely many instructions) for an unlimited amount of time. We start by defining a self delimiting partial recursive function f of two arguments, a program $p \in \Sigma^*$ and the number of steps involved in the computation.

Definition 3. Let $f : \Sigma^* \times \mathbb{N} \to \Sigma^*$ be a partial recursive function such that

(monotone) If $f(p,n)\downarrow$ then $\forall m \leq n\ f(p,m) \preceq f(p,n)$.

(self delimiting) If $f(p,n)\downarrow$ then $\forall a, f(pa,n) = f(p,n)$.

(recursive domain) There is a total recursive function $\mathrm{def}_f : \Sigma^* \times \mathbb{N} \to \Sigma$ such that $\mathrm{def}_f(p,n) = 1$ iff $f(p,n)\downarrow$.

The result of computing a program p for infinitely many steps is the limit of running p for n steps, for n going to infinity. We define f^∞ as a partial function from strings to either strings or sequences, $f^\infty(p)$ is the supreme of the set of strings $f(p,n)$, for all n.

Definition 4. Let $f^\infty : \Sigma^* \to \Sigma^\infty$, $f^\infty(p) = sup\{f(p,n) :\ n \geq 1\}$.

$f^\infty(p)\downarrow$ exactly when $\forall n\ f(p,n)\downarrow$. Notice that if $\forall n f(p,n)\downarrow$ then, by the monotonicity condition of f, $\{f(p,n) :\ n \geq 1\}$ is totally ordered under $\preceq$, so the supreme of this set is defined. The result of the supreme operation is in Σ^∞. Thus, if $\{f(p,n) :\ n \geq 1\}$ is finite, then $f^\infty(p)$ will be the $\preceq$-maximal string belonging to that set. Otherwise, $f^\infty(p)$ will be a sequence. f^∞ inherits the self delimiting property of f:

Proposition 5. *If* $f^\infty(p)\downarrow$ *then* $\forall a \in \Sigma^*, f^\infty(pa) = f^\infty(p)$.

For instance, if we define a program p such that $f(p,2n) = (01)^n$ and $f(p,2n+1) = (01)^n 0, \forall n \geq 0$ we obtain $f^\infty(p) = 010101\ldots$, Turing's first example in [11].

Machines that are capable of simulating any other machine are universal. First of all we will choose a self delimiting universal machine $U : \Sigma^* \to \Sigma^*$ (a

machine for finite computations, a partial recursive function), assuming a given effective enumeration of all self delimiting machines such that M_i is the i-th machine in such an enumeration. U reads its program tape until it finds the first 1. If it read i 0s, it starts simulating the execution of M_i, taking the rest of the program tape as a program for M_i. So we get $U(0^i1p) = M_i(p)$. We set $K = dom(U)$.

We also choose a universal machine for infinite computations, $U^\infty : \Sigma^* \to \Sigma^\infty$, such that $U^\infty(0^i1p) = M_i^\infty(p)$, and denote $K^\infty = dom(U^\infty)$. We can split K^∞ into two disjoint sets, the set of programs that output finitely many symbols and those that output infinitely many. In Turing's terminology, circular and circle free programs.

$$K^\infty = K_{circular} \cup K_{circlefree}$$

We want to prove that the probability that a universal self delimiting machine outputs finitely many symbols, is a random number. That is, we want to prove that $\mu(K_{circular}\Sigma^\omega)$ is random. The rest of this paper deals with how to prove it. The usual technique for proving the randomnes of Ω uses the fact that it is computable enumerable [4]. We can not directly apply this technique because the set $K_{circular}$ is not recursively enumerable. We need a machine with an oracle for the halting problem to enumerate $K_{circular}$.

Oracle machines provide relative computability. A function that can be computed by a machine with an oracle for A is said A-computable, and a set that can be enumerated by a machine with and oracle for A is said to be A-recursively enumerable.

We fix U', a universal self delimiting machine with an oracle for the halting problem in U, $U' : \Sigma^* \to \Sigma^*$, such that $dom(U')$ is prefix free and $U'(0^i1p) = M_i'(p)$, and set $K' = dom(U')$.

Proposition 6. *$K_{circular}$ is K-recursively enumerable.*

Proof. We will denote by u the partial recursive function used to define U^∞. The following algorithm for U' enumerates $K_{circular}$.

$K_{circular} := \emptyset$
dovetail among every $s \in \Sigma^*$
  t:=1
  while ($s \notin K_{circular}$) do
    if oracle(U(n:=t;while(def$_u(s,n)$ and
          $u(s,n) = u(s,n+1)$) do; n:=n+1; end do)↑) then
      $K_{circular} := K_{circular} \cup \{s\}$
    else
      t:=t+1
  end do
end dovetail

□

Proposition 7. *Every computably enumerable real is K-computable.*

Proof. Assume r is computably enumerable. Then there is a computable sequence of rationals (a_n) that converges to r. We will denote by $dth(q, d)$ the d-th symbol of the decimal expansion of q. The following program for U' prints the decimal expansion of r, digit by digit.

```
d:=1
t:=1
do forever
  if oracle(U(n:=t;while(dth(a_n, d) = dth(a_{n+1}, d)) do; n:=n+1; end do)↑) then
    print dth(a_t, d)
    d:=d+1
  else
    t:=t+1
end do
```

□

4 Algorithmic Complexity

The algorithmic complexity of a string in a given machine is the minimal length of a program for that machine that produces it as a result. Let $f : \Sigma^* \to \Sigma^*$ be self delimiting partial recursive. Chaitin defines the algorithmic complexity of a string s in a machine f as:

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

For self delimiting oracle machines, $f' : \Sigma^* \to \Sigma^*$,

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

And we define program size complexity for infinite computations, identically. Let $f^\infty : \Sigma^* \to \Sigma^\infty$ be self delimiting.

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

In [3] Chaitin introduces a notion of universality different from the classical notion. A self delimiting partial recursive function U is Chaitin universal iff for any self delimiting partial recursive function f, there is a constant c such that for all s, $H_U(s) \leq H_f(s) + c$. For any pair of universal 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 the algorithmic complexity is more or less independent of the universal machine being used. Since universal computers are asymptotically optimal, the algorithmic complexity in a universal machine counts as an absolute measure of complexity.

The universal machine U we fixed is Chaitin universal, since for a given enumeration of all tables defining a computer such that f_i is the i-th in the

enumeration, $U(0^i1p) = f_i(p)$, $i \in \mathbb{N}$, $p \in \Sigma^*$. Then $\forall f_i$, $H_U(s) \leq H_{f_i}(s) + i + 1$. For the same reason, U^∞ and U' that we fixed in the previous section are also Chaitin universal. From now on we drop the subindexes U, U^∞, and U', and write H, H^∞, and H'.

4.1 Randomness

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

Chaitin [4] defines a sequence $\mathtt{x} \in \Sigma^\omega$ to be random iff there is a constant c such that for all n, $H(\mathtt{x}_n) > n - c$. And he defines the probability that a self delimiting universal machine U halts

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

Since $U : \Sigma^* \to \Sigma^*$, with $dom(U)$ prefix free, $\mu(K\,\Sigma^\omega) = \Omega$. Chaitin proves that Ω is computably enumerable and random.

Let us remark that the property of a prefix free set of strings of having a random probability is not recursively invariant.

Proposition 8. *For $A = \{0^{i-1}1 : string_i \in K\}$, $r = \sum_{s\in A} 2^{-|s|}$ is not random.*

Proof. By definition, the i-th bit of r is 1 iff $U(string_i)\downarrow$. Then, the first 2^n bits of r are determined by the halting behavior of all programs of length less than n. There will be m of them, $0 \leq m \leq 2^n$, halt in U.

For any n, the first 2^n bits of r can be dramatically compressed: there is an algorithm which given 2^n and m, by dovetailing all programs of length less than n finds the m that halt and determines the first 2^n bits of r. Then, $H(r_{2^n}) \leq log\ 2^n + log\ m + c \leq 2\ log\ 2^n + c = 2n + c$, for some constant c. Hence, r is not random. □

K and A are recursively equivalent. However, Ω is random and r is not.

4.2 Comparison between H, H^∞ and H'

We can now ask how this different complexity measures relate. First we show that for any string $H' \leq H^\infty \leq H$ within a constant term. Next we show that the inequalities obtained are strict, i.e.: $H - H^\infty$ and $H' - H^\infty$ can not be bounded by a constant.

Proposition 9. *1.* $\exists c \forall s \in \Sigma^*\ \ H'(s) \leq H(s) + c$.
2. $\exists c \forall s \in \Sigma^*\ \ H'(s) \leq H^\infty(s) + c$.
3. $\exists c \forall s \in \Sigma^*\ \ H^\infty(s) \leq H(s) + c$.

Proof. 1. There is an oracle machine M_i' that does not use its oracle and behaves exactly as U. Then for every $p \in \Sigma^*$, $U(p) = U'(0^i 1 p)$. Thus, if $H(s)$ is the length of the shortest program that produces s in U, there is a program of length $H(s) + i + 1$ which produces s in U'. So we can take $c = i + 1$.

2. Any program p for U^∞ that outputs finitely many symbols can be simulated on U' by increasing number of steps. At each step, the simulation polls the oracle to determine whether p would output more symbols or not. The simulation ends when there is no more output left. The following program performs that task.

```
t:=1
while( oracle(U(n:=t;while(u(p, n) = u(p, n + 1))do; n:=n+1; end do ))↓) do
   t:=t+1
end do
print(u(p,t))
```

There is an i that instructs U' to execute this program on input $0^i 1 p$. So, if p is the minimal program for s in U^∞, there is a program for U' of length $|p| + i + 1$ that outputs s.

3. There is an i that instructs U^∞ to perform exactly the same actions as U and cycle forever when U would halt. So, if p is the minimal program for s in U, there is a program for U^∞ of length $|p| + i + 1$ that outputs s.

□

Proposition 10. *1.* $\forall c \exists s \; H(s) - H^\infty(s) > c$
2. $\forall c \exists s \; H^\infty(s) - H'(s) > c$

Proof. 1. We will show a family of strings in which the diference between H and H^∞ can be made arbitrarily large. Consider the following program for U^∞ that receives a minimal program for n and outputs a string $s(n)$:

```
compute n
approx:=0 (the approximation of Ω_n)
previous:=0
dovetail among every program
   each time a program p halts
      previous:=approx
      approx:=approx+ 2^-|p|
      if (first n bits of previous)≠(first n bits of approx) then
         print the first n bits of approx
end dovetail
```

This program outputs a concatenation of strings of n bits obtained as successive approximations to Ω_n from below until the first n bits of Ω have been obtained: $b_{11} \ldots b_{1n} b_{21} \ldots b_{2n} \ldots b_{m1} \ldots b_{mn} \Omega_n$.
$s(n)$ is the result of an infinite computation by this program, so we have:

$$H^\infty(s(n)) \leq H(n) + c_1 \tag{1}$$

But given n and $s(n)$ one can easily compute Ω_n, and we know that Ω is random, so there must exist a c_2 such that for every n:

$$H(s(n)) + H(n) > n - c_2 \tag{2}$$

Joining (1) and (2) we get:

$$H(s(n)) - H^\infty(s(n)) > n - 2H(n) - c_3$$

The term on the right side can be made arbitrarily large as n increases.

2. Now we have to show that the difference between H^∞ and H' is unbounded. If u is the partial recursive function used to define U^∞, we can consider a family of recursive functions $f_n : \Sigma^* \to \Sigma^*$ such that:

$$f_n(p) = u(p, \min\{k : \ |u(p,k)| = n\})$$

With this definition, if $U^\infty(p) = \Omega_n$, then $f_n(p) = \Omega_n$. As f_n is a computable function that depends on n and its argument, we have:

$$H(\Omega_n) \leq H^\infty(\Omega_n) + H(n) + c_1 \tag{3}$$

Ω is a random number. So, from (3) we get:

$$H^\infty(\Omega_n) > n - H(n) - c_2 \tag{4}$$

On the other side, Ω is K-computable by proposition 7, so:

$$H'(\Omega_n) \leq H(n) + c_3 \tag{5}$$

Joining (4) and (5) we get:

$$H^\infty(\Omega_n) - H'(\Omega_n) > n - 2H(n) - c_4$$

And again, the term on the right side can be made arbitrarily large as n increases.

□

5 The Probability of Circular Programs

We define the probability that a self delimiting universal machine for infinite computations produces only a finite amount of output.

$$\alpha = \sum_{U^\infty(p) \text{ is finite}} 2^{-|p|} = \mu(K_{circular}\Sigma^\omega).$$

We will show that α is incompressible even if we count with an oracle for the halting problem in U. To prove it we will first establish a correspondence between the programs that halt in U' and a subset of the circular programs in U^∞. This

correspondence will allow us to show that if we have the first n bits of α we can solve the halting problem for all programs for U' of length less than or equal to n minus a constant. Then using Chaitin's original argument for his proof of the randomness of Ω we will prove that α is random.

To prove the correspondence result, we will take from [5] a technique called *simulation in the limit* that is as follows: a computation in U' is simulated by increasing number of steps. In step t the oracle for the halting problem is simulated answering that $U(q)$ halts iff it does so within time t. Using this technique, we will be able prove that there is a prefix % such that $U^\infty(\%pa)$ is finite for almost every $a \in \Sigma^*$ if and only if $U'(p)$ halts. Therefore, the probability that $U^\infty(\%p)$ is finite is exactly equal to Ω'. We have to allow possible extensions a of p because the simulation in the limit may initially read extra bits. This happens because the domain of U' is not recursively enumerable, and the algorithm for the simulation runs on U^∞, so it has no way to determine where the program p ends until it has the correct oracle answers. In the meantime, it may have read extra bits from the program tape, and the head of the program tape can not move backwards. But the actual value of the extra bits is irrelevant because once the algorithm reaches the correct oracle answers it will know where p ends. For example, suppose p is the following simple program for U'.

```
if oracle(U(factorial 100)↓) then halt
else do forever read next bit of the program tape
```

Our simulation will need to compute factorial 100. The simulated oracle will give the wrong answer for many steps t, until t is large enough to complete the task. The simulation will read as many extra bits from the program tape as needed steps to complete the simulation of factorial 100. In general, the extra bits may determine different execution paths at intermediate steps of the simulation. But for each t the simulation of $U'(p)$ is restarted. So, although they are unavoidable, in the end the extra bits are ignored. Any sequence of 0s and 1s after p on the program tape, will lead to the correct simulation of $U'(p)$. This motivates the following

Definition 11. A set $A \subseteq \Sigma^*$ is unavoidable iff A is prefix free and every sequence of Σ^ω has a prefix in A.

For example, $\{0, 10, 110, 1110, 1111\}$ is unavoidable, and $\{0\}$ is not.

Proposition 12. *For every unavoidable set A, $\mu(A\Sigma^\omega) = \sum_{a\in A} 2^{-|a|} = 1$.*

Hence, if A is unavoidable, s is any string and $B = \{sa : a \in A\}$, then $\sum_{b\in B} 2^{-|b|}$ $= \sum_{sa\in B} 2^{-|sa|} = \sum_{sa\in B} 2^{-|s|}2^{-|a|} = 2^{-|s|} \sum_{a\in A} 2^{-|a|} = 2^{-|s|}$.

Lemma 13. *$\exists\%\forall p\exists A$ unavoidable such that $\forall a \in A, \%pa$ is circular iff $U'(p)$ halts.*

Proof. % contains instructions for U^∞ to perform the simulation in the limit of the program for U' that comes afterwards in the program tape. The following

algorithm performs that task:

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

1. 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 at most t steps.
2. 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.
3. `t:=t+1`

```
end do
```

Let us see that this % has the desired property. Suppose $U'(p)\downarrow$. Then p halts in finitely many steps. Then it can perform only finitely many oracle questions. Let us call Q the set of programs for U that are consulted to the oracle. Every $q \in Q$ such that $U(q)\downarrow$ halts in some finite time. Let us call m the maximum time required to halt by the programs of Q. For values of t less than m, the simulation of some oracle questions will be wrong, but for every value of $t \geq m$, they will be correct. If $U'(p)$ halts after 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. Thus, for every $a \in \Sigma^*$ of length greater than m and n, $U^\infty(\%pa)$ will be finite. The set of all strings of length $\max(m,n)$ is an unavoidable set.

Suppose now that $U^\infty(\%pa)$ is finite. Then, during the simulation the head of the output tape prints finitely many positive integers, and then it prints no more. Say the last number printed is n. We have to show that $U'(p)$ halts.

Suppose not. Then 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 performed, over a finite number of 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 perform 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. Thus, it must be $U'(p)\downarrow$. □

Now we can prove that α is incompressible even if we count with an oracle for the halting problem. In this sense, α is highly random.

Theorem 14. $\exists c \forall n,\ H'(\alpha_n) > n - c$.

Proof. Consider the following algorithm for U' that receives as input a minimal program for U' that computes α_n:

1. Compute α_n.
2. As we proved in Proposition 6, $K_{circular}$ is K-recursively enumerable. Let g be a K-recursive function whose range is $K_{circular}$. Enumerate enough programs $g(1), g(2), \ldots$ until we have a prefix free set $X = \{g(i_1), \ldots, g(i_m)\}$, such that $w_m = \sum_{x \in X} 2^{-|x|} > \alpha_n$.
3. Enumerate the set K' to obtain a set $Y \subset K'$ such that for every $\%p$ in X there exists a $y \in Y$ such that $y \preceq p$.
4. Output z the first string that does not belong to $\{U'(p) : p \in Y\}$ and halt.

We have to show that $H'(z) > n - |\%|$. There must be some minimal program p for U' such that $U'(p) = z$, but we know that $p \notin Y$. By Lemma 13, there exists an unavoidable set A such that for all $a \in A$, $\%pa$ is circular. Since $p \notin Y$, then for all $a \in A$, $\%pa \notin X$. We shall now consider the contribution of these circular programs to α. Since $\sum_{a \in A} 2^{-|\%pa|} = 2^{-|\%p|}$, the unavoidable set A can be ignored. By our construction,

$$\alpha_n < w_m$$

Let us add $2^{-|\%p|}$ to both sides of the inequality,

$$\alpha_n + 2^{-|\%p|} < w_m + 2^{-|\%p|}$$

Since there are infinitely many circular programs we have that $w_m + 2^{-|\%p|} < \alpha$, then

$$\alpha_n + 2^{-|\%p|} < w_m + 2^{-|\%p|} < \alpha$$

Finally, using that $\alpha \leq \alpha_n + 2^{-n}$, we obtain

$$\alpha_n + 2^{-|\%p|} < w_m + 2^{-|\%p|} < \alpha \leq \alpha_n + 2^{-n}$$

Thus, $\alpha_n + 2^{-|\%p|} < \alpha_n + 2^{-n}$, which means that

$$|\%p| > n$$

We conclude that $n - |\%| < H'(z)$.

Since we obtained z as the output of the algorithm above, there is a constant q such that $H'(z) \leq H'(\alpha_n) + q$. Thus, $H'(\alpha_n) > n - q - |\%|$. Taking $c = q + |\%|$, we obtain the desired result. □

Proposition 15. $\exists c \forall n \; H'(\Omega'_n) \leq H'(\alpha_n) + c$

Proof. Given a minimal program for U' for the first n bits of α_n we can compute the first $n - |\%|$ bits of Ω' with the following algorithm for U'.

As in the proof of Theorem 14, enumerate enough programs of $K_{circular}$ until we have a prefix free set $X = \{g(i_1), \ldots, g(i_m)\}$ such that $\sum_{x \in X} 2^{-|x|} > \alpha_n$. Applying Lemma 13 we obtain $\Omega'_{n-|\%|} = \sum_{\%p \in X} 2^{-|p|}$. Thus, $H'(\Omega_n) \leq H'(\alpha_n) + |\%| + constant$. □

We conclude with the following:

Corollary 16. *α is random and not computably enumerable.*

Proof. From Theorem 14 and Proposition 9, α is random. Also from Theorem 14, α can not be K-computable, therefore by Proposition 7, it can not be computably enumerable. □

We have chosen the formulation of α as the probability that a self delimiting universal machine outputs finitely many symbols. However, there are a number of alternative though equivalent formulations of α. One is to define it as the probability that a self delimiting universal machine (for finite and infinite computations) reads a finite number of bits of the program tape, as we did it in [7]. It is also possible to define α is as the probability that a universal self delimiting machine enumerates a finite set. And yet another equivalent formulation is that it computes a partial recursive function with a finite graph.

Acknowledgements. We thank Cristian Calude for his valuable comments in an earlier stage of this work. Serge Grigorieff and Max Dickmann provided us with useful comments that helped us improve the presentation of this work.

The first author is supported by grant 11-05382 from the Agencia de Promoción Científica y Tecnológica and a postdoctoral fellowship from the CONICET.

References

1. Cristian. S. Calude. *Information and Randomness. An Algorithmic Perspective.* Springer-Verlag, Berlin, 1994.
2. Cristian S. Calude, Peter H. Hertlind, Bakhadyr Khoussainov, and Yongee Wang. Recursively enumerable reals and Chaitin Ω numbers. *Theoretical Computer Science.* In press.
3. G. J. Chaitin. Information-theoretic limitations of formal systems. *J. ACM*, 21:403–424, 1974.
4. G. J. Chaitin. A theory of program size formally identical to information theory. *J. ACM*, 22:329–340, 1975.
5. G. J. Chaitin. Algorithmic entropy of sets. *Computers & Mathematics with Applications*, 2:233–245, 1976.
6. G. J. Chaitin. *Exploring Randomness.* Springer-Verlag, London, 2001.
7. S. Daicz. Una nueva versión de la probabilidad de detención. Tesis de licenciatura, Facultad de Cs. Exactas y Naturales, Universidad de Buenos Aires, 2000.
8. 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.
9. T. Slaman. Randomness and recursive enumerability. *SIAM J. on Computing.* to appear.
10. 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*, pages 283–307. North-Holland Publishing Company, 1977.
11. A. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society, 2nd series*, 42:230–265, 1936.