Every computably enumerable random real is provably computably enumerable random

Cristian S. Calude and Nicholas J. Hay, *Department of Computer Science, University of Auckland, Private Bag 92019, Auckland, New Zealand E-mail: {cristian,nickhay}@cs.auckland.ac.nz*

Abstract

We prove that every computably enumerable (c.e.) random real is provable in Peano Arithmetic (PA) to be c.e. random. A major step in the proof is to show that the theorem stating that "a real is c.e. and random iff it is the halting probability of a universal prefix-free Turing machine" can be proven in PA. Our proof, which is simpler than the standard one, can also be used for the original theorem.

Our positive result can be contrasted with the case of computable functions, where not every computable function is provably computable in PA, or even more interestingly, with the fact that almost all random finite strings are not provably random in PA.

We also prove two negative results: a) there exists a universal machine whose universality cannot be proved in PA, b) there exists a universal machine U such that, based on U, PA cannot prove the randomness of its halting probability.

The paper also includes a sharper form of the Kraft-Chaitin Theorem, as well as a formal proof of this theorem written with the proof assistant Isabelle.

1 Introduction

A real in the unit interval is *computably enumerable* (*c.e.*) if it is the limit of a computable, increasing sequence of rationals. We identify a real with its infinite binary expansion. In contrast with the case of a computable real, whose bits are given by a computable function, during the process of approximation of a c.e. real one may never know how close one is to the limit. A real is (algorithmic) random if its binary expansion is an algorithmic random (infinite) sequence [4, 7, 8, 11, 20].

A prefix-free machine is a Turing machine, shortly, *machine*, from strings to strings whose domain is a prefix-free set. A machine is universal if it can simulate every machine. Chaitin [7] introduced the halting probability Ω_U of a universal machine U, Chaitin's Omega number

$$\Omega_U = \sum_{U(x) \text{ is defined}} 2^{-|x|},$$

and proved that Ω_U *is c.e. and random.* As shown by Calude, Hertling, Khoussainov, Wang [6] and Kučera, Slaman [16], (see also [3]) there are no other c.e. random reals:

Theorem 1 *The set of c.e. random reals coincides with the set of halting probabilities of all universal machines.*

C.e. random reals have been intensively studied in recent years, with many results summarised in [4, 11].

Theorem 2 (Chaitin [7]) *Assume that* ZFC *(Zermelo-Fraenkel set theory with choice) is arithmetically sound (that is, any theorem of arithmetic proved by* ZFC *is true). Then, for every universal machine U,* ZFC *can determine the value of only finitely many bits of* Ω_U*, and one can calculate a bound on the number of bits of* Ω_U *which* ZFC *can determine.*

The real Ω_U depends on U, and so by tuning this choice one gets:

Theorem 3 (Solovay [19]) *We can chose a universal machine U so that* ZFC *(if arithmetically sound) cannot determine any bit of* Ω_U*.*

This result was generalised as follows:

Theorem 4 (Calude [2]) *Assume that* ZFC *is arithmetically sound. Let* $i \geq 1$ *and consider the c.e. random real* $\alpha = 0.1^{i-1}0\alpha_{i+1}\cdots$ *Then, we can effectively construct a universal machine U (depending upon* ZFC *and* α*) such that* PA *(Peano Arithmetic) proves the universality of U,* ZFC *can determine at most* $i-1$ *initial bits of* Ω_U *and* $\alpha = \Omega_U$*.*

The proof of Theorem 4 in [2] starts by fixing a universal machine V such that the universality of V is provable in PA and $\Omega_V = \alpha$. Solovay [18] observed that "it is by no means evident that there is a universal prefix-free machine whose universality is provable in PA and whose halting probability is α".

> Let $\alpha \in (0, 1)$ be c.e. and random. Is there any *representation* of α for which PA can *prove* that α c.e. and random?

We give an affirmative answer to this question. A major step in the proof is to show that Theorem 1 can be proved in PA. Our proof, which is simpler than the standard one, can be used also for the original theorem.

The paper also includes a sharper form of the Kraft-Chaitin Theorem, as well as a formal proof of this theorem written with the proof assistant Isabelle.

In what follows proofs will be written in Solovay's style [19]. All necessary steps are presented in sufficient detail to leave the remaining formalisation routine. The formalisation of the Kraft-Chaitin Theorem is presented with full details, and then a sketch of the formal proof in Isabelle is discussed.

The paper is organised as follows. Sections 2 and 3 present all facts on formal provability and Algorithmic Information Theory needed for this paper. The Kraft-Chaitin Theorem is presented in Section 4. Section 5 presents three ways to prove randomness, using Martin-Löf tests, prefix-free complexity, and Solovay representation formula. In Section 6 we revisit

Chaitin's Theorem on the randomness of the halting probability of a universal machine. In Section 7 we prove that a real $\alpha \in (0,1)$ is provably Chaitin-random iff it is provable that $\alpha = \Omega_U$ for some provably universal machine U (see Theorem 18). In Section 8 we prove our main theorem: every c.e. random real is provably random (Theorem 22). In Section 9 we construct a universal machine U based on which PA cannot prove the randomness of its halting probability. Section 10 presents a formal proof of the Kraft-Chaitin Theorem written with Isabelle. The final Section 11 includes a few general remarks.

2 Provability

By $\mathcal{L}_A$ we denote the first-order language of arithmetic whose non-logical symbols consist of the constant symbols 0 and 1, the binary relation symbol $<$ and two binary function symbols $+$ (addition) and $\cdot$ (multiplication). Peano Arithmetic (see [15], shortly, PA) is the first-order theory given by a set of 15 axioms defining discretely ordered rings, together with induction axioms for each formula $\varphi(x, y_1, \ldots, y_n)$ in $\mathcal{L}_A$:

$$\forall \overline{y}(\varphi(0, \overline{y}) \wedge \forall x(\varphi(x, \overline{y}) \rightarrow \varphi(x+1, \overline{y})) \rightarrow \forall x(\varphi(x, \overline{y})).$$

The structure $\mathbf{N}$ whose domain is the set of naturals $\mathbb{N} = \{0, 1, 2, \ldots\}$, where the symbols in $\mathcal{L}_A$ have the obvious interpretation, satisfies the axioms of PA; this is the standard model for PA. There are non-standard models of PA that are not isomorphic to $\mathbf{N}$. If M is a structure for $\mathcal{L}_A$ and $\varphi(\overline{x})$ is an $\mathcal{L}_A$-formula with free-variables $\overline{x} = (x_1, \ldots, x_n)$ and $\overline{a} = (a_1, \ldots, a_n) \in M$, then we write $M \vDash \varphi(\overline{a})$ to mean that "φ is true in M when each variable x_i is interpreted by a_i". We blur the distinction between n and the closed term of $\mathcal{L}_A$, $(\cdots(((1+1)+1)+1)+\cdots 1)$, ($n$ times).

A formula $\theta(\overline{x})$ of $\mathcal{L}_A$ is Δ_0 if all its quantifiers are bounded. A formula $\psi(\overline{x})$ of $\mathcal{L}_A$ is Σ_1 if it is of the form $\psi(\overline{x}) = \exists y \theta(\overline{x}, y)$ with $\theta(\overline{x}, y) \in \Delta_0$; $\psi(\overline{x})$ of $\mathcal{L}_A$ is Π_1 if it is of the form $\psi(\overline{x}) = \forall y \theta(\overline{x}, y)$ with $\theta(\overline{x}, y) \in \Delta_0$.

By PA $\vdash \theta$ we mean "there is a proof in PA for θ". It is useful to know that PA proves the *least number principle*: PA $\vdash \forall y(\exists x \varphi(x, y) \rightarrow \exists z(\varphi(z, y) \wedge \forall w < z \neg \varphi(w, y)))$, for each formula $\varphi(x, y)$ of $\mathcal{L}_A$.

An important link between computability and provability is given by the following results.

Theorem 5 *A partial function from $\mathbb{N}$ to $\mathbb{N}$ is partial computable iff its graph is equivalent to a Σ_1 $\mathcal{L}_A$-formula.*

Corollary 6 *A set $A \subset \mathbb{N}^k$ is computably enumerable (c.e.) if there is a Σ_1 $\mathcal{L}_A$-formula $\varphi(\overline{x})$ such that for all $\overline{x} \in \mathbb{N}^k$, $\overline{x} \in A$ iff $\mathbf{N} \vDash \varphi(\overline{x})$.*

A total function $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is represented in PA if there is an $\mathcal{L}_A$-formula $\theta(\overline{x})$ such that for all $\overline{n} \in \mathbb{N}^k$:

1. PA $\vdash \exists! y \theta(\overline{n}, y)$, and
2. if $k = f(\overline{n})$ then PA $\vdash \theta(\overline{n}, k)$.

(Here ∃! means "there exists a unique".) One can show that every total computable function is represented by a Σ_1-formula of PA [15].

A function $f: \mathbb{N} \to \mathbb{N}$ is *provably computable* [12, 15] if there exists a Σ_1-formula of PA $\varphi(x, y)$ such that:

1. $\{(n, m) \mid m, n, \in \mathbb{N}, f(n) = m\} = \{(n, m) \mid \mathbf{N} \vDash \varphi(n, m)\}$,
2. $\text{PA} \vdash \forall x \exists! y \varphi(x, y)$.

In view of Corollary 6, any provably computable function has a c.e. graph, so it is total and computable. These functions can be viewed as computable functions whose totality is proved by PA.

Theorem 7 ([15]) *Every primitive recursive function is provably computable, but there exist computable functions which are not provably computable in* PA.

If f is computable but not provably computable in PA, then the statement "f is total" is true but unprovable in PA. In contrast with the case of computable functions, *c.e. sets are provably enumerable* [12] (because every non-empty c.e. set can be enumerated by a primitive recursive function, [1], p. 138).

In what follows all computations will be implemented by primitive recursive functions. Hence, we will work with a special type of Σ_1 formulae. By abuse of language we say that a formula of PA is Σ_1^0 if it has the form $\exists x P(x)$, for some primitive recursive predicate $P(x)$. A formula of PA is Π_2^0 if it has the form $\forall x \exists y P(x, y)$, for some primitive recursive predicate $P(x, y)$.

Our metatheory is ZFC. We fix a (relative) interpretation of PA in ZFC. Each formula of $\mathcal{L}_A$ has a translation into a formula of ZFC determined by the interpretation of PA in ZFC. By abuse of language we shall use the phrase "sentence of arithmetic" to mean a formula with no free variables of ZFC that is the translation of some formula of PA. We assume that ZFC is 1–consistent, that is, if it proves a Σ_1^0 sentence then that sentence is true (in the standard model of PA).

Theorem 8 (Solovay [19]) *Every Π_2^0 sentence proved by* ZFC *is true.*

As a consequence, it follows that *if U is a machine which* PA *can prove universal and* ZFC *can prove the sentence "the i-th digit of Ω_U is k", then the sentence is true.* Whenever we talk about the provability of a sentence of arithmetic we mean that PA proves its corresponding translation formula.

If there is a proof in PA for statement A we say that A provable in PA. We say that A is provably P (where P is a property) if the statement "A has P" is provable in PA.

3   Algorithmic Information Theory: Some Definitions and Results

All reals are in the unit interval. A c.e. real α is represented by an increasing computable sequence of rationals converging to α. We blur the distinction between the real α and the

infinite base-two expansion of α, i.e. the infinite sequence $\alpha_1\alpha_2\cdots\alpha_n\cdots$ $(\alpha_n \in \{0,1\})$ such that $\alpha = 0.\alpha_1\alpha_2\cdots\alpha_n\cdots$ By $\alpha(n)$ we denote the string of length n, $\alpha_1\alpha_2\cdots\alpha_n$.

The set of (bit) strings is denoted by Σ^*; ε denotes the empty string. If s is a string then $|s|$ denotes the length of s. We import the theory of computability from natural numbers to strings by fixing the canonical bijection between Σ^* and $\mathbb{N}$ induced by the linear order $s < t$ if $|s| < |t|$ or $|s| = |t|$ and s lexicographically precedes t.

A machine U is *universal* if for every machine V there is a constant c (depending upon U and V) such that for all strings s, t, if $V(s) = t$, then $U(s') = t$ for some string s' of length $|s'| \leq |s| + c$. The domain of U is the set $\{x \in \Sigma^* \mid U(x) \text{ is defined}\}$. The Omega number $\Omega_U = \sum_{x \in \mathrm{dom}\, U} 2^{-|x|}$ is halting probability of U. The *prefix-free complexity* of the string $x \in \Sigma^*$ (relatively to the machine C) is $H_C(x) = \min\{|y| \mid y \in \Sigma^*,\ C(y) = x\}$ $(\min \emptyset = \infty)$. If U is a universal machine, then for every machine we can effectively construct a constant c (depending on U and C) such that for all x, $H_U(x) \leq H_C(x) + c$.

A real α is *Chaitin-random* if there exists a universal machine U and constant c such that for all $n \geq 1$, $H_U(\alpha(n)) \geq n - c$.

A c.e. open set is a c.e. union of intervals with rationals endpoints $[a, b)$ and μ is Lebesgue measure. If S is a prefix-free set, then $\mu(S)$ denotes the Lebesgue measure of the cylinder denoted by S, i.e. all reals whose infinite binary expansions have a prefix in S. To the string x we associate the interval $[0.x, 0.x + 2^{-|x|})$ of measure $2^{-|x|}$. A Martin-Löf test (shortly, ML test) A is a uniformly c.e. sequence of c.e. open sets $A = (A_n)$ such that for all $n \geq 1$, $\mu(A_n) \leq 2^{-n}$. A real α is *Martin-Löf-random* (shortly, *ML-random*) if for every ML test A there exists an i such that $\alpha \notin A_i$. A classical theorem states that *a real is Chaitin-random iff it is ML-random* [4, 8].

Note that Chaitin and Martin-Löf definitions apply to any real. In the special case of c.e. reals the following Solovay representation formula stated in [18] is used: A real α is *c.e. and random* if there exists a universal machine U, an integer $c > o$ and a c.e. real $\gamma > 0$ such that $\alpha = 2^{-c} \cdot \Omega_U + \gamma$ (see Lemma 20).

4 Kraft-Chaitin Theorem Revisited

We start by showing that PA can prove the Kraft-Chaitin Theorem [4, 8].

Theorem 9 *Suppose $(n_i, y_i)_i \in \mathbb{N} \times \Sigma^*$ is a primitive recursive enumeration of "requests" which provably satisfies $\sum_i 2^{-n_i} \leq 1$. Then there exists a provably prefix-free machine M and a primitive recursive enumeration $(x_i)_i$ of $\mathrm{dom}(M)$ such that the following is provable in* PA:

1. $\mu(\mathrm{dom}(M)) = \sum_i 2^{-n_i}$,
2. $|x_i| = n_i$ for all $i \in \mathbb{N}$,
3. $M(x_i) = y_i$ for all $i \in \mathbb{N}$.

Proof. Algorithm 1 below enumerates the graph of M. Intuitively, S_i keeps track of the tree of prefixes we haven't allocated yet. To start with we have allocated nothing, so $S_0 = \{\epsilon\}$. At each step we want a string (node) of a given length (depth) n_i. The program selects the deepest leaf it can, then creates the smallest number of new leaves to create the node we need.

Algorithm 1

1: $S_0 = \{\epsilon\}$, $T_0 = \emptyset$, $r_0 = 0$, $i \leftarrow 0$.
2: **loop**
3:     Let s_i be the longest element of S of length at most n_i. If no such string exists, terminate.
4:     **if** $|s_i| = n_i$ **then**
5:         $S_{i+1} = (S_i \setminus \{s_i\})$.
6:     **else**
7:         $S_{i+1} = (S_i \setminus \{s_i\}) \cup \{s_i 1, s_i 0 1, s_i 0^2 1, \ldots, s_i 0^{n_i - |s_i| - 1} 1\}$.
8:     **end if**
9:     Define $M(s_i 0^{n_i - |s_i|}) = y_i$.
10:     $T_{i+1} = T_i \cup \{s_i 0^{n_i - |s_i|}\}$.
11:     $r_{i+1} = r_i + 2^{-n_i}$.
12:     $i \leftarrow i + 1$.
13: **end loop**

Examining Algorithm 1, it is clear that the sequence $x_i = s_i 0^{n_i - |s_i|}$ is a primitive recursive enumeration of $\text{dom}(M)$, and whenever x_i is defined we have $M(x_i) = y_i$ and $|x_i| = n_i$. It remains to show that x_i is defined for all $i \in \mathbb{N}$ (i.e. the program never terminates), that $\text{dom}(M)$ is prefix-free, and $\mu(\text{dom}(M)) = \sum_i 2^{-n_i}$.

It suffices to establish, for all i , the following invariants:

1. $S_i \cup T_i$ is prefix-free (which implies that S_i and T_i individually are prefix-free),
2. $\mu(S_i \cup T_i) = 1$,
3. $\mu(T_i) = r_i$,
4. $\sum_{j \geq i} 2^{-n_j} \leq \mu(S_i)$,
5. If $2^{-n} \leq \mu(S_i)$, then S_i contains a string of length at most n (equivalently, S_i contains strings of distinct length).

The base case is trivial. For the inductive step, first observe that line 3 of Algorithm 1 doesn't terminate since $2^{-n_i} \leq \mu(S_i)$ by invariant 5. We see that

$$S_{i+1} \cup T_{i+1} = ((S_i \cup T_i) \setminus \{s_i\}) \cup \{s_i 1, s_i 0 1, s_i 0^2 1, \ldots, s_i 0^{n_i - |s_i| - 1} 1, s_i 0^{n_i - |s_i|}\}$$

which is prefix-free establishing invariant 1. From this we can see invariant 2 holds: $\mu(S_{i+1} \cup T_{i+1}) = \mu(S_i \cup T_i) = 1$. Next observe invariant 3 holds too: $\mu(T_{i+1}) = \mu(T_i) + 2^{-n_i} = r_{i+1}$, which implies that $\mu(S_{i+1}) = \mu(S_i) - 2^{-n_i}$. From this follows invariant 4: $\sum_{j \geq i+1} 2^{-n_j} \leq \mu(S_{i+1})$. Finally, since s_i is the longest string of length at most n_i in S_i , and we add strings of distinct length between $s_i + 1$ and n_i to S_i to form S_{i+1} , we see that S_{i+1} consists of strings of distinct lengths. This establishes invariant 5. ∎

5 Randomness and Provability

In this section we discuss three forms of provability for randomness.

There are two ways to represent a c.e. real number $\alpha \in (0,1)$ in PA: 1) by giving an increasing 1-1 primitive recursive function that enumerates a c.e. prefix-free set of strings $\{s_i\}$ such that $\alpha = \sum_i 2^{-|s_i|}$, 2) by giving an increasing primitive recursive sequence $(a_i)_i$ of rationals in the unit interval whose limit is α. It is clear that given the representation 1) one can effectively get the representation 2). The converse is also true.

Lemma 10 *Let α be a c.e. real defined by the increasing primitive recursive sequence $(a_i)_i$ of rationals. Then there is a primitive recursive sequence $(n_i)_i$ of natural numbers such that PA proves $\sum_i 2^{-n_i} = \alpha$.*

Proof. Without loss of generality assume $a_1 > 0$. Define the primitive recursive sequences $(r_i)_i \in \mathbb{Q}$ and $(n_i)_i \in \mathbb{N}$ by $r_0 = 0$ and for $i \geq 1$ by

$$n_i = \lceil -\log_2(a_i - r_{i-1}) \rceil, r_i = r_{i-1} + 2^{-n_i}.$$

Since $(r_i)_i$ is strictly increasing we can establish by induction the inequality $r_{i-1} < a_i$ for all i, making the logarithm well-defined. By construction we have $\sum_i 2^{-n_i} = \lim_{i \to \infty} r_i$. Define $\beta = \lim_{i \to \infty} r_i$. Since $-\log_2(a_i - r_{i-1}) \leq n_i \leq -\log_2(a_i - r_{i-1}) + 1$ we have $(a_i + r_{i-1})/2 \leq r_i \leq a_i$. Taking the limit we see that $(\alpha + \beta)/2 \leq \beta \leq \alpha$ establishing our result. ∎

Corollary 11 *Let α be a c.e. real defined by the increasing primitive recursive sequence $(a_i)_i$ of rationals. Then there is a machine M such that PA proves that $\alpha = \mu(\mathrm{dom}(M))$.*

Proof. Use Lemma 10 and Theorem 9. ∎

In what follows a c.e. real is given by one of the above representations.

A c.e. real α is *provably Chaitin-random* if there exists a provably universal machine U and constant c such that PA proves that for all $n \geq 1$, $H_U(\alpha(n)) \geq n - c$. A c.e. real α is *provably ML-random* if for every set A which PA proves to be a ML test and PA proves that there exists an i such that $\alpha \notin A_i$.

The classical theorem which states that *a real is Chaitin-random iff it is ML-random* is provable in PA. However, for the goal of this paper only one implication is needed:

Theorem 12 *Every c.e. provably Chaitin-random real is provably ML-random.*

Proof. Take a c.e. real α, a machine U which is provably universal and a natural $c > 0$ such that PA proves that for all $n \geq 1$, $H_U(\alpha(n)) \geq n - c$.

We wish to prove that for every $A = (A_n)$ which PA proves to be a ML test there exists an i such that PA proves that $\alpha \notin A_i$. Following the proof of Proposition 6.3.4 in [4] it follows that PA proves the existence of a c.e. set $S \subset \Sigma^* \times \mathbb{N}$ such that each $S_i = \{x \in \Sigma^* \mid (x, i) \in S\}$ is prefix-free, and by taking $A_i = \{\beta \mid \beta(m) \in S_i, \text{ for some } m \geq 1\}$ we get $\mu(A_i) = \sum_{s \in S_i} 2^{-|s|}$.

Let $g \colon \mathbb{N} \to \Sigma^* \times \Sigma^*$ be a 1-1 primitive recursive enumeration of the graph of U. Denote by $\pi_i \colon \Sigma^* \times \Sigma^* \to \Sigma^*$ for $i = 1, 2$ the projection functions and $f(i) = \pi_1(g(i))$ is a 1-1 primitive recursive enumeration of $\mathrm{dom}(U)$. Note that $H_U(x)$ can be expressed in PA: $H_U(x) = \min_i\{|v| \colon U(v) = x\} = \min_i\{|\pi_1(g(i))| \mid \pi_2(g(i)) = x\}$,

We have:

$$\sum_{n\geq 2}\sum_{s\in S_{n^2}} 2^{-\lfloor |s|-n\rfloor} = \sum_{n\geq 2} 2^n \mu(S_{n^2}) \leq \sum_{n\geq 2} 2^n 2^{-n^2} \leq 1.$$

We can now use Theorem 9: There exists a provably prefix-free machine M such that: $\mu(\mathrm{dom}(M)) = \sum_{n\geq 2}\sum_{s\in S_{n^2}} 2^{-\lfloor |s|-n\rfloor}$, $\mathrm{dom}(M) = \{r_{n,s} \mid n\geq 2, s\in S_{n^2}, |r_{n,s}| = |s|-n\}$, $M(r_{n,s}) = s$. Since U is provably universal there is a constant $d\geq 1$ such that for all strings $x, H_U(x) \leq H_M(x) + d$, so in particular, PA proves that for all $n\geq 2$, if $s\in S_{n^2}$, then $H_U(s) \leq H_M(s) + d \leq |s|-n+d < |s|-n+d+1$.

We are now in a position to find a natural $n\geq 2$ such that PA proves that $\alpha\notin A_{n^2}$ showing that α is provably ML-random. Note that for $n\geq 2$, PA proves $\alpha\notin A_{n^2}$ iff for all $m\geq 1$, PA proves that $\alpha(m)\notin S_{n^2}$. For all $m\geq 1$, PA proves $\alpha(m)\in S_{n^2}$ implies $H_U(\alpha(m)) < m-n+d+1$. Hence, for $2\leq n < c+d+1$, PA proves that $H_U(\alpha(m)) \geq m-c$ implies $\alpha(m)\notin S_{n^2}$, so because α is provably Chaitin-random PA proves that $\alpha\notin A_{n^2}$. ∎

Comment   The above proof shows that that $T_m^U = \{\beta \mid H_U(\beta(n)) < n-m, \text{ for some } n\geq 1\}$, where U is a provably universal machine, is a provably ML test such that for all $n\geq 2$ and provably ML test A there exists $d > 0$ such that PA proves the inclusion $A_{n^2}\subset T_{n-d-1}^U$, i.e. $(T_m^U)_m$ is a provably universal ML test.

To be able to complete our program we need to choose a specific representation for a c.e. and random real which can be "understood" by PA and, even more importantly, PA can extract from it a proof of the randomness of the real (c.e. is obvious). First we work with Solovay representation formula discussed at the end of Section 3.

A real α is *c.e. and provably random* if there exists a representation of α in the form

$$\alpha = 2^{-c}\cdot\Omega_V + \gamma, \tag{1}$$

where V is a provably universal machine, $c\geq 0$ is an integer and $\gamma > 0$ is a provably c.e. real. Theorem 20 shows that all c.e. random reals have a representation of this form. In detail, PA receives an algorithm for a machine V, a proof that V is prefix-free and universal, an integer $c\geq 0$ and a computable increasing sequence of rational converging to a real $\gamma > 0$. The goal is to prove that PA can use this information to prove that $\alpha = 2^{-c}\cdot\Omega_V + \gamma$ is c.e. and random.

6   Chaitin's Theorem Revisited

Chaitin [7] proved that the halting probability of a universal machine is Chaitin-random. This theorem is provable in PA:

Theorem 13 *Suppose U is provably universal. Then $\Omega_U = \sum_{p\in\mathrm{dom}(U)} 2^{-|p|}$ is provably Chaitin-random.*

Proof. Let $g\colon \mathbb{N}\to\Sigma^*\times\Sigma^*$ be a 1-1 primitive recursive enumeration of the graph of U. Denote by $\pi_i\colon \Sigma^*\times\Sigma^*\to\Sigma$ for $i = 1,2$ the projection functions and $f(i) = \pi_1(g(i))$ is a 1-1

primitive recursive enumeration of $\mathrm{dom}(U)$. Recall that $H_U(x)$ can be expressed in PA. Define the machine M by $M(0^{|x|}1x) = x$. Since U is provably universal, there is a c such that for all x, $H_U(x) \leq H_M(x) + c = 2|x| + c + 1$. This shows that $H_U(x)$ is provably total and U is provably onto.

Define the primitive recursive sequence of rationals $\omega_k = \sum_i^k 2^{-|\pi_1(g(i))|}$ and notice that $(\omega_k)_k$ is provably strictly increasing; Ω_U is, by definition, the limit of this sequence.

Define $C(x) = v$ if there exist t, j such that

1. $\pi_1(g(j)) = x$ (i.e. $x \in \mathrm{dom}(U)$),
2. t is the least such that $0.\pi_2(g(j)) \leq \omega_t$ (i.e. $0.U(x) \leq \omega_t$),
3. v is the lexicographically least string such that $v \neq \pi_2(g(s))$ for all $1 \leq s \leq t$.

This defines a provably prefix-free machine. Observe that if $C(x)$ is defined and $U(x) = U(x')$ then $C(x) = C(x')$. From this we can establish that whenever $C(x)$ is defined we have $H_C(C(x)) \leq H_U(U(x))$. As U is provably universal, there exists an a such that for all y, $H_U(y) \leq H_C(y) + a$ is provable in PA.

Denote by Ω_i the ith digit of Ω_U. Since U is provable onto, for each n there exists a string x_n such that $U(x_n) = \Omega_1 \cdots \Omega_n$. Since $0.\Omega_1 \cdots \Omega_n < \Omega_U$ we know that $C(x_n)$ is defined. Let t be the least natural (found when evaluating $C(x_n)$) such that $0.U(x_n) \leq \omega_t < \Omega_U < 0.U(x_n) + 2^{-n}$. The inequality $\sum_{i \geq t+1} 2^{-\pi_1(g(i))} \leq 2^{-n}$ is easy consequence, so for all $i \geq t+1$ we have $|\pi_2(g(i))| \geq n$. Since $C(x_n)$ equals $g(i)$ for some $i \geq t+1$ by construction, we have that for all n

$$n \leq H_U(C(x_n)) \leq H_U(U(x_n)) + a = H_U(\Omega_1 \cdots \Omega_n) + a$$

is provable in PA. That is, Ω is provably Chaitin-random. ∎

From Theorem 13, we deduce that PA can prove the implication: "if U is a provably universal machine, then Ω_U is Chaitin-random." We know that every c.e. and random real is the halting probability of a universal machine, but we need more: *Can any c.e. and random real be represented as the halting probability of a* provably *universal machine?* First we have to check whether every universal machine is provably universal.

Theorem 14 *There exist a provably universal machine and a universal machine that is not provably universal.*

Proof. The set of all provably prefix-free machines is c.e., so if $(M_i)_i$ is a computably enumeration of provably prefix-free machines, then the machine U defined by $U(0^i1x) = M_i(x)$ is a provably universal machine.

Let $(f_i)_i$ be a c.e. enumeration of all primitive recursive functions $f_i : \mathbb{N} \rightarrow \Sigma^*$ and (T_i) a c.e. enumeration of all machines. Fix a universal machine U and consider the computable function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that:

$$T_{g(i)}(x) = \begin{cases} U(x), & \text{if for some } j > 0, \#\{f_i(1), f_i(2), \ldots, f_i(j)\} > |x|, \\ \infty, & \text{otherwise.} \end{cases}$$

For every i, $T_{g(i)}$ is a universal machine iff $f_i(\mathbb{N})$ is infinite (if $f_i(\mathbb{N})$ is finite, then so is $T_{g(i)}$). Since the set of all indices of primitive recursive functions with infinite range is not c.e. it follows that there is an i such that PA cannot prove that $T_{g(i)}$ is universal. ∎

Theorem 14 does not imply a negative answer for the previous question; in fact, Corollary 25 shows that the answer is affirmative. Theorem 14 produces examples of true and unprovable (in PA) statements of the form "V is universal".

7 Provably C.E. Random Reals

In this section we sharpen Theorem 1 by proving that a real is provably c.e. and Chaitin-random iff it is provable that the real is the halting probability of a provably universal machine.

According to Solovay [20] a c.e. real α *Solovay dominates* a c.e. real β (we write $\beta \leq_S \alpha$) if there are two computable, increasing sequences $(a_i)_i$ and $(b_i)_i$ of rationals and a constant c with $\lim_{n \to \infty} a_n = \alpha$, $\lim_{n \to \infty} b_n = \beta$, and $c(\alpha - a_n) \geq \beta - b_n$, for all n.

For c.e. reals α, β, PA proves $\beta \leq_S \alpha$ if there are two primitive recursive, increasing sequences $(a_i)_i$ and $(b_i)_i$ of rationals and a constant c such that PA proves $\lim_{n \to \infty} a_n = \alpha$, $\lim_{n \to \infty} b_n = \beta$, and $c(\alpha - a_n) \geq \beta - b_n$, for all n.

Theorem 15 *If α is c.e. and provably ML-random, and β is c.e., then $\beta \leq_S \alpha$ is provable in* PA.

Proof. Let $(a_i)_i$ and $(b_i)_i$ be primitive recursive sequences of rationals with limits α and β respectively. Let $a_0 = b_0 = 0$.

For each n, for $i \geq 1$ if $a_i \notin \bigcup_{j=1}^{i-1} T_n[j]$ then define $T_n[i] = [a_i, a_i + 2^{-n}(b_i - b_{s^n}))$, where $s^n = \max_{j < i}\{j : T_n[j] \neq \emptyset\}$ is the most recent non-empty stage, or $s^n = 0$ if this is the first non-empty stage. Otherwise define $T_n[i] = \emptyset$.

Let s_j^n denote the jth non-empty stage, wherever that is well-defined, and let $s_0^n = 0$. Observe that

$$T_n = \bigcup_i T_n[i] = \bigcup_{j \geq 1} [a_{s_j^n}, a_{s_j^n} + 2^{-n}(b_{s_j^n} - b_{s_{j-1}^n}))$$

and that all the sets in the above union are disjoint by construction. As a result $\mu(T_n) = \sum_{j \geq 1} 2^{-n}(b_{s_j^n} - b_{s_{j-1}^n}) \leq 2^{-n}$, so PA proves that $(T_n)_n$ is a ML-test.

Because α is provably ML-random, PA proves that there exists an m such that $\alpha \notin T_m$, so for all $j \geq 1$ we know that s_j^m is well-defined. By construction we have the inequality $a_{s_{j+1}^m} \notin [a_{s_j^m}, a_{s_j^m} + 2^{-m}(b_{s_j^m} - b_{s_{j-1}^m}))$ which implies that $b_{s_j^m} - b_{s_{j-1}^m} \leq 2^m(a_{s_{j+1}^m} - a_{s_j^m})$.

Defining $a_j' = a_{s_j^m}$ and $b_j' = b_{s_{j-1}^m}$, we have for all $j \geq 1$ that $b_{j+1}' - b_j' \leq 2^m(a_{j+1}' - a_j')$, where $(a_j')_j$ and $(b_j')_j$ are primitive recursive sequences of rationals which provably converge to α and β respectively. So PA proves $\beta \leq_S \alpha$. ∎

Corollary 16 *If α is c.e. and provably Chaitin-random and β is c.e., then $\beta \leq_S \alpha$ is provable in PA.*

Proof. See Theorems 12 and 15. ∎

Theorem 17 *Suppose V is a provably universal machine, α is c.e., and $\Omega_V \leq_S \alpha$ is provable in PA. Then there exists a provably universal machine U such that $\Omega_U = \alpha$ is provable in PA.*

Proof. Since $\Omega_V \leq_S \alpha$, there exist primitive recursive increasing sequences $(a_i)_i$ and $(b_i)_i$ of rationals, with limits α and Ω_V respectively, and a constant $c \geq 0$ such that for all n

$$b_{n+1} - b_n < 2^c (a_{n+1} - a_n). \tag{2}$$

Define $a_0 = b_0 = 0$. Form the real

$$\gamma = \alpha - 2^{-c} \cdot \Omega_V = \sum_n (a_{n+1} - a_n) - 2^{-c}(b_{n+1} - b_n).$$

By equation (2) the terms of the sum are positive, so $\gamma \in (0, 1)$ is c.e. Applying Lemma 10 to γ we get a primitive recursive sequence $(m_i)_i$ of natural numbers such that $\sum_i 2^{-m_i} = \gamma$.

Let $(v_i)_i$ be a 1-1 primitive recursive enumeration of $\mathrm{dom}(V)$, and define the sequence of requests $y_{2i} = v_i, n_{2i} = |v_i| + c, y_{2i+1} = v, n_{2i+1} = m_i$, where v is an arbitrarily fixed element in $\mathrm{dom}(V)$.

By Theorem 9 we get a provably prefix-free machine M and a primitive recursive enumeration $(x_i)_i$ of $\mathrm{dom}(M)$ such that the following three statements are provable: 1) $\mu(\mathrm{dom}(M)) = \sum_i 2^{-n_i}$, 2) $|x_i| = n_i$ for all i, 3) $M(x_i) = y_i$ for all i.

Consider the machine $U = V \circ M$. The machine U is provably universal. Indeed, $U(x_{2i}) = V(M(x_{2i})) = V(y_{2i}) = V(v_i)$ and $|x_{2i}| = n_{2i} = |v_i| + c$, by construction of M. Finally, it is provable that $\Omega_U = \sum_{p \in \mathrm{dom}(U)} 2^{-p} = \sum_i 2^{-n_{2i}} + \sum_i 2^{-n_{2i+1}} = 2^{-c} \cdot \Omega_V + \gamma = \alpha$. ∎

Using all results above we obtain:

Theorem 18 *A c.e. real α is provably Chaitin-random iff it is provable that $\alpha = \Omega_U$ for some provably universal machine U.*

Proof. Suppose α is provably c.e. and Chaitin-random. By Theorem 12, α it is provably ML-random. Take a provably universal machine V (Theorem 14). From Theorem 15 we see that $\Omega_V \leq_S \alpha$ is provable in PA. By Theorem 17 we effectively get a U which is provably universal and prefix-free such that $\alpha = \Omega_U$ is provable in PA. The converse is exactly Theorem 13. ∎

Corollary 19 *Every provably c.e. and Chaitin-random real is provably random.*

Proof. If α is provably Chaitin-random and c.e. then by Theorem 18, $\alpha = \Omega_U$ for some provably universal machine U, so α satisfies Solovay's formula (1) with $c = 1, \gamma = 1/2 \cdot \Omega_U$. ∎

8   Every Random C.E. Real Is Provably C.E. Random

This section proves its title. We start with the following result by Solovay [18]:

Lemma 20 *Let V be a universal machine. If α is c.e. and ML-random, then there exists an integer $c \geq 0$ and a c.e. real $\gamma > 0$ such that (1) is satisfied.*

Proof. Using the proof of Theorem 15, we deduce that $\Omega_V \leq_S \alpha$ (because α is c.e. and ML-random). Consequently, we can consider the primitive recursive increasing sequences $(a_i)_i$ and $(b_i)_i$ of rationals, with $a_0 = b_0 = 0$ and converging to α and Ω_V respectively, and a constant $c \geq 0$ such that for all n, $b_{n+1} - b_n < 2^c(a_{n+1} - a_n)$. The c.e. real $\gamma = \alpha - 2^{-c} \cdot \Omega_V = \sum_n (a_{n+1} - a_n) - 2^{-c}(b_{n+1} - b_n)$ is positive and $\alpha = 2^{-c} \cdot \Omega_V + \gamma$. ∎

It is not difficult to see that the converse implication in Lemma 20 is also true. In fact, a sharper result can be proved:

Theorem 21 *Let V be provably universal, $c \geq 0$ be an integer, γ a positive c.e. real. Then $\alpha = 2^{-c} \cdot \Omega_V + \gamma$ is provably Chaitin-random (ML-random).*

Proof. Let (v_n) be a primitive recursive enumeration of the domain of V and $(b_n)_n$ be a primitive recursive increasing sequence with limit γ. The sequence of rationals $\alpha_n = 2^{-c} \sum_{i=1}^n 2^{-|v_i|} + b_n$ is primitive recursive, increasing and converges to α.

Take $a_n = \sum_{i=1}^n 2^{-|v_i|}$ and observe that for all n, $a_{n+1} - a_n \leq 2^c(\alpha_{n+1} - \alpha_n)$, hence PA proves that $\Omega_V \leq_S \alpha$. Using Theorem 17 we can find a provably universal machine U such that $\Omega_U = \alpha$ is provable in PA. By Theorem 13, α is provably Chaitin-random and by Theorem 12, α is provably ML-random. ∎

We can now state our main result:

Theorem 22 *Every c.e. and random real is provably c.e. and Chaitin-random (ML-random), hence provably c.e. and random.*

Proof. Start with a provably universal machine V (Theorem 14). By Lemma 20 there exist c and γ defining the representation (1) for α: $\alpha = 2^{-c} \cdot \Omega_V + \gamma$. Since V is provably universal, Theorem 21 shows that that $2^{-c} \cdot \Omega_V + \gamma$ is provably Chaitin-random (ML-random). Therefore α is provably Chaitin-random (ML-random). Finally use Corollary 19 to deduce that α is provably random. ∎

Theorem 18 can now be stated in the form:

Theorem 23 *A real α is provably c.e. and random iff it is provable that $\alpha = \Omega_U$ for some provably universal machine U.*

Proof. Use Theorem 22 and Corollary 19. ∎

Corollary 24 *For every universal machine U there exists a provably universal machine U' such that $\Omega_U = \Omega_{U'}$.*

Proof. Since Ω_U is c.e. and random, by Theorem 22 we deduce that Ω_U is provably Chaitin-random, so by Theorem 18 we get a provably universal machine U' such that $\Omega_U = \Omega_{U'}$. ■

Corollary 25 *Every c.e. and random real can be written as the halting probability of a provably universal machine.*

Proof. Use Theorems 22 and 23. ■

9 A Negative Result

From the previous two sections we know that every c.e. random real can be written as the halting probability of a provably universal machine, so it is provable random. Does there exist a universal machine whose halting probability is not provable random? By Theorem 13 such a machine should not be provably universal (and such machines exist by Theorem 14).

We answer in the affirmative this question. To this aim we fix an effective enumeration of all c.e. reals in $(0,1)$ $(\gamma_i)_i$ (for example, by enumerating all increasing primitive recursive sequences of rationals in $(0,1)$) and define the set $\mathfrak{R}_{ce} = \{\gamma \in (0,1) \mid \gamma \text{ is c.e.}\}$. A set $A \subseteq \mathfrak{R}_{ce}$ is called c.e. if the set $\{i \in \mathbb{N} \mid \gamma_i \in A\}$ is c.e. Note that in A we enumerate all indices for all elements in A.

Lemma 26 [14] *If $A \subseteq \mathfrak{R}_{ce}$ is c.e., then for all c.e. reals $\alpha \in A$ and $\beta > \alpha$ we have $\beta \in A$.*

Proof. Let $K = \{k_i\}$ be a c.e. not computable set of natural numbers enumerated by a primitive recursive function $i \mapsto k_i$, and for each n let $(a_i^n)_i$ be a primitive recursive increasing sequence of rationals in $(0,1)$ such that $\lim_{i \to \infty} a_i^n = \gamma_n$. Let $\alpha = \lim_{i \to \infty} a_i^s, \beta = \lim_{i \to \infty} a_i^t$ and define the function

$$\Gamma(j,i) = \begin{cases} a_i^s, & \text{if } j \neq k_m, \text{ for all } m \leq i, \\ \max\{a_i^s, a_i^t\}, & \text{otherwise.} \end{cases}$$

Because $\beta > \alpha$ there exists a natural i_0 such that $a_{i_0}^s < a_{i_0}^t$. If $j \in K$, then there exists an m such that $j = k_m$, hence $\Gamma(j,i) = a_i^t$, for $i \geq \max\{i_0, m\}$, so $\lim_{i \to \infty} \Gamma(i,j) = \beta$. If $j \notin K$, then for all i, $\Gamma(j,i) = a_i^s$, so $\lim_{i \to \infty} \Gamma(i,j) = \alpha$.

Because of the uniform definition of $\Gamma(i,j)$ we can construct a computable function f such that $\lim_{i \to \infty} \Gamma(i,j) = \gamma_{f(j)}$.

Finally, let's assume by absurdity that $\beta \notin A$. The set $\{j \in \mathbb{N} \mid \gamma_{f(j)} \in A\}$ is c.e. because A is c.e., but in view of the definition of Γ, $\{j \in \mathbb{N} \mid \gamma_{f(j)} \in A\} = \{j \in \mathbb{N} \mid \gamma_{f(j)} = \alpha\} = \{j \in \mathbb{N} \mid j \notin K\}$, a non c.e. set. ■

Let $(U_i)_i$ be a c.e. enumeration of all universal machines. Consider now the sets $\mathfrak{R}_{halt} = \{\Omega_{U_i}\}$ and $\mathfrak{R}_{cerand}^{PA} = \{\gamma \in \mathfrak{R}_{ce} \mid \gamma \text{ is provably random}\}$. By enumerating proofs in PA we deduce that $\mathfrak{R}_{cerand}^{PA}$ is c.e., so $\mathfrak{R}_{cerand}^{PA} = \{\gamma_{f(i)}\}$, for some primitive recursive function f.

We have: $\{\gamma_{f(i)}\} \subseteq \{\Omega_{U_i}\} \subset \{\gamma_i\}$. Is $\mathfrak{R}_{halt}$ c.e.? The answer is negative:

Theorem 27 *There exists a universal machine U_t such that $\Omega_{U_t} \neq \gamma_{f(i)}$, for all i.*

Proof. Take a universal machine U such that $\Omega_U \geq 1/2$ and construct the c.e. real $\beta = \Omega_U(n+1)11\cdots$, where $\Omega_U(n+1) = 1^n 0$. As $\beta > \Omega_U$ and β is not random, $\beta \neq \Omega_{U_i}$, for all i, so by Lemma 26, $\{\Omega_{U_i}\}$ is not c.e., hence the theorem is proved. ∎

Comment There is no contradiction between Corollary 24 and Theorem 27: there exist a universal machine U_t and a provably universal machine U_j such that $\Omega_{U_t} = \Omega_{U_j}$ and $\Omega_{U_t} \neq \gamma_{f(i)}$, for all i: PA cannot prove the randomness of Ω_{U_t} based on U_t, but can prove the randomness of $\Omega_{U_t} = \Omega_{U_j}$ based on U_j.

10 Formal Proof of the Kraft-Chaitin Theorem

In the above we gave proofs that various statements, once suitably formalised in the language of first order logic, were derivable from the axioms of PA. In principle, but for lack of space and patience, we could have presented complete PA derivations of each statement proved. Instead, as is common practice for all but the simplest of results, we sketched constructions which leave the actual derivations implicit.

Recent advances in theorem proving computer programs, such as the proof assistant Isabelle [17], have allowed complete formal derivations of nontrivial mathematical results. In such systems, humans write a sequence of proof commands, and the computer system searches for a complete derivation, if one exists. Essentially, the human user gives a sequence of intermediate lemma with proof directions, and the computer interpolates the full derivation. (For a recent perspective on the importance of formalising mathematics see [10].)

Using Isabelle, we formalised and proved the Kraft-Chaitin Theorem (Theorem 9), a key result in our above proof. To keep our presentation self-contained, we begin by showing how to formalise and prove a simple result about strings; for a full introduction to the Isabelle system see [17]. We follow with a formalisation of the Kraft-Chaitin Theorem, then sketch its formal proof. The full proof script is available online [13].

10.1 Formalising Results in Isabelle

To illustrate Isabelle and its use, we will formalise and prove the following simple property of strings:

Lemma 28 *Given strings $x, y, z \in \Sigma^*$, if x extends y then xz extends y.*

Strings are naturally represented by the Isabelle list data-type. Here `[]` represents the empty list, and `y#ys` represents the list formed by concatenating the element `y` with the list `ys`. For example, the string 001 is represented by `0 # 0 # 1 # []` (or `[0,0,1]` for short). The following code inductively defines whether the list `A` extends `B`, denoted `extends A B`:

```
fun extends :: "'A list => 'A list => bool"
where
  "extends [] [] = True"
| "extends [] (y#ys) = False"
```

```
| "extends x [] = True"
| "extends (x#xs) (y#ys) = ((x=y) & (extends xs ys))"
```

When faced with the above definition, Isabelle automatically proves termination (in this case, by observing that the first argument always decreases in length with each recursive call).

Let us first prove that any list extends the empty list. We enter into Isabelle:

```
lemma extends1: "extends A []"
```

It responds with the propositions we need to prove:

```
goal (1 subgoal):
 1. extends A []
```

It is natural to prove this by induction on A, by entering the command `apply(induct A)`. This results in two proof obligations, one for the base case and the other for the inductive step:

```
goal (2 subgoals):
 1. extends [] []
 2. !!a A. extends A [] ==> extends (a # A) []
```

The first proposition is one of the cases in our definition of `extend`. In the second `!!` denotes universal quantification and this similarly follows from one of our definition cases. We tell Isabelle to simplify these expressions with the command `apply(simp_all)`. Isabelle manages to simplify all these expressions down to `True`, using rewrite rules for simplifying conjunctions, variable identity, and expanding the definition of `extends`. As a result we get:

```
goal:
No subgoals!
```

Having completed the proof, we compactly store it in the following format:

```
lemma extends1: "extends A []"
  apply(induct A) apply(simp_all)
done
```

We can now attempt our original goal:

```
lemma extends2: "extends (A@B) A"

goal (1 subgoal):
 1. extends (A @ B) A
```

The concatenation of lists `A` and `B` is denoted `A @ B`. We again induct with the command `apply(induct A)`, then simplify with the command `apply(simp_all)`

```
goal (2 subgoals):
 1. extends ([] @ B) []
 2. !!a A. extends (A @ B) A ==> extends ((a # A) @ B) (a # A)

goal (1 subgoal):
 1. extends B []
```

Since we proved this before, we use the command `apply(simp only: extends1)` to reuse our previous result, completing the proof. In sum:

```
lemma extends2: "extends (A@B) A"
  apply(induct A) apply(simp_all) apply(simp only: extends1)
done
```

10.2 Formalising the Kraft-Chaitin Theorem

The proof of the Kraft-Chaitin Theorem is algorithmic: it describes a particular algorithm (Algorithm 1 of Theorem 9) for selecting strings of the required lengths, and proves that the algorithm is correct. In what follows we will implement this algorithm in Isabelle and will prove its correctness.

The following Isabelle code implements Algorithm 1. We give the definition of each function, then explain what it does.

```
 fun extend :: "nat list => nat => nat list list"
  where
  "extend l 0 = [l]"
| "extend l (Suc n) = (hd (extend l n) @ [0]) # (hd (extend l n) @ [1])
                                             # tl (extend l n)"
```

For `l` a binary list representing a binary string, and `n` a natural number, `extend l n` computes the list

$$[l0^{n-|l|}, l0^{n-|l|-1}1, \ldots, l01, l1].$$

For example, in Isabelle the expression `extend [0,0,1] 5` evaluates to

$$[[0,0,1,0,0],\ [0,0,1,0,1],\ [0,0,1,1]]$$

This corresponds to the set $\{00100, 00101, 0011\}$ of binary strings.

The set of unallocated prefixes S_i and the set of allocated strings T_i are represented by lists of strings. The free prefixes are ordered by decreasing length, the allocated strings by the order of allocation.

Consider one iteration of the main loop. Let `A` be the list of previously allocated strings, `F` the list of free prefixes, and `n` the length of the string we want to allocate at this step. (These are denoted T_i, S_i, and n_i in the original algorithm.) `kcstep A F n` returns the updated pair of allocated strings and free prefixes (S_{i+1} and T_{i+1}).

```
 consts kcstep ::
"nat list list => nat list list => nat
                                  => (nat list list * nat list list)"
primrec
  "kcstep A [] n      = (A, [])"   (* fail case *)
  "kcstep A (f # F) n = (if length f <= n
                then ((hd (extend f (n - length f))) # A,
                      (tl (extend f (n - length f))) @ F)
                else (fst (kcstep A F n), f # snd (kcstep A F n)))"
```

`kcstep` searches through the list `F` of free prefixes for the longest string of length at most n. One it finds it, it calls `extend`, which returns a list of extended prefixes. It takes the first string in the list, guaranteed to have exactly length `n`, and adds it to the allocated strings list. The rest of the strings are placed on the free prefixes list.

For example, `kcstep [] [[]] 2` evaluates to

$$([[0,0]], [[0,1], [1]])$$

which corresponds to the list 00 of allocated strings and the set {1,01} of free prefixes.

```
consts kcloop :: "nat list => (nat list list * nat list list)
                            => (nat list list * nat list list)"
primrec
  "kcloop [] X = X"
  "kcloop (l#ls) X = (kcstep (fst (kcloop ls X)) (snd (kcloop ls X)) l)"
```

For a list of lengths `l` and a pair `(A,F)` of allocated strings and free prefixes, `kcloop l (A,F)` runs `kcstep` to allocate strings for every length in `l`. For example, `kcloop [3,4,2] ([], [[]])` allocates a string of length 2, then one of length 4, then one of length 3, starting from the initial state where no strings are yet allocated (`[]`) and the empty string is our free prefix (`[[]]`).

For example, `kcloop [3,2] ([], [[]])` evaluates to

$$([[0,1,0], [0,0]], [[0,1,1], [1]])$$

which corresponds to the list 00, 010 of allocated strings (note that we reverse the list), and the set {1,011} of free prefixes.

```
fun kc :: "nat list => nat list list"
where
    "kc ls = (fst (kcloop ls ([],[[]])))"
```

For a list of lengths `l`, `kc l` returns the list of strings allocated by running `kcloop` on the list starting from the initial state where no strings have been allocated. For example, `kc [4,3,2]` evaluates to

$$[[0,1,1,0], [0,1,0], [0,0]]$$

which corresponds to the sequence $00, 010, 0110$ of allocated strings.

This implements Kraft-Chaitin's algorithm, for we will prove that:

1. If our list of lengths obeys Kraft's inequality, $\sum_i 2^{-n_i} \leq 1$, then `kc ls` is a list of strings, and the ith element of `kc ls` has length equal to the ith element of `ls`.
2. `kc ls` is always a prefix-free list (no two distinct elements of the list are prefixes of each other).
3. If we add new lengths to the start of `ls`, then this adds new strings to the end of `kc ls` without changing the old ones. That is, once a string of a given length is allocated it is not changed.

To prove the above we need to define what a prefix-free list is, a function to evaluate Kraft's inequality, a function which checks whether the lengths of one list match the lengths in another, and a tool to check whether one list extends another.

```
fun prefixes :: "nat list => nat list => bool"
where
  "prefixes [] x = True"
| "prefixes x [] = True"
| "prefixes (x#xs) (y#ys) = ((x=y) & (prefixes xs ys))"

consts incomparable :: "nat list => nat list list => bool"
primrec
  "incomparable x [] = True"
  "incomparable x (y # ys) = (~(prefixes x y) & (incomparable x ys))"

consts prefixfree :: "nat list list => bool"
primrec
  "prefixfree [] = True"
  "prefixfree (x # xs) = ((incomparable x xs) & (prefixfree xs))"
```

If `x` is a prefix of `y`, or vice versa, then `prefixes x y`. For example `prefixes [0,0,1] [0,0]` is true. `incomparable x A` holds if `x` is not a prefix of any string in `A`, for instance `incomparable [0,0] [[1,0], [1,1,1]]` holds. `prefixfree L` holds if the list `L` is prefix-free, for instance `prefixfree [[0,0], [1,0], [1,1,1]]` holds.

```
consts expn2 :: "nat => rat"
primrec
  "expn2 0 = 1"
  "expn2 (Suc n) = (1/2) * expn2 n"
consts meas_nat :: "nat list => rat"
primrec
```

```
"meas_nat [] = 0"
"meas_nat (f#F) = (expn2 f + meas_nat F)"
```

We define `expn2 n` equal to 2^{-n}. `meas_nat F` computes the "measure" of a sequence of natural numbers `F`, for example `meas_nat [4,3,2]` equals 7/16.

```
fun lengthsmatch :: "nat list list => nat list => bool"
where
  "lengthsmatch [] [] = True"
| "lengthsmatch [] (l#ls) = False"
| "lengthsmatch (x#xs) [] = False"
| "lengthsmatch (x#xs) (l#ls) = ((length x = l) & (lengthsmatch xs ls))"
```

The expression `lengthsmatch X Y` holds if the lengths of each string in `X` matches the corresponding number in `Y`. For example, we have `lengthsmatch [[0,0], [1,0], [1,1,1]] [2,2,3]` is True.

```
fun extends :: "'A list => 'A list => bool"
where
  "extends [] [] = True"
| "extends [] (y#ys) = False"
| "extends x [] = True"
| "extends (x#xs) (y#ys) = ((x=y) & (extends xs ys))"
```

Finally, `extends A B` holds if the list `A` extends the list `B`, so `extends [0,1] [0]` holds.

With the above definitions we can state the three results which establish correctness:

```
theorem kc_correct1: "meas_nat ls <= 1 ==> lengthsmatch (kc ls) ls"

theorem kc_correct2: "prefixfree (kc ls)"

theorem kc_extend:   "extends (rev (kc (L2 @ L1))) (rev (kc L1))"
```

The first says that if `ls` is a list of natural numbers n_i which satisfies Kraft's inequality $\sum_i 2^{-n_i} \leq 1$, then the strings `kc ls` allocated by running Algorithm 1 on this list have exactly the lengths `ls` we asked for.

The second says the strings allocated are prefix-free.

The last says that when Algorithm 1 allocates additional strings it does not change strings it has previously allocated. To see this, note that when we run `kc L` the algorithm allocates strings starting from the end of the list `L`. This means, the first element of `kc L` is the last string allocated. `kc (L2 @ L1)` is the list of strings allocated if we allocate strings with lengths in `L1` then strings with lengths in `L2`.

Together, establishing these would show that the `kc` algorithm constructively establishes the Kraft-Chaitin Theorem.

10.3 Proof Outline

All the above merely formalised the algorithm and stated the theorem we wish Isabelle to prove. This gets the order mixed slightly, since formalising this theorem unearthed a mistake in the algorithm, so the process was mutual. In some sense this formalisation of the theorem is the major creative work, the rest is just technical detail. As one might guess, however, most of the work is in these details. To prove the above theorems we must guide Isabelle to them by establishing numerous intermediate lemmas, and telling Isabelle which proof techniques to use to establish each. Often we just advise Isabelle to induct on a variable then simplify, but sometimes we must give more detailed guidance.

The Isabelle proof follows the proof given for Theorem 9: we establish that the inner loop preserves some invariants, and use these invariants to establish correctness.

Recall the algorithm has two variables: the list of allocated strings and the list of free strings. Each pass through the loop will (potentially) add one new allocated string, and modify the free strings. We then show that these two lists combined remain prefix-free, their joint measure never decreases, and that there are never two free strings of the same length.

For reasons of space we give only the definitions required to state the above intermediate results and show how they are formalised in Isabelle. The proof in its entirety is available online [13].

```
fun strictlysorted :: "nat list list => bool"
where
  "strictlysorted [] = True"
| "strictlysorted [x] = True"
| "strictlysorted (x1 # x2 # xs) = ((length x1 > length x2)
                                    & (strictlysorted (x2 # xs)))"
```

`strictlysorted L` holds if the strings in `L` are ordered by (strictly) decreasing length. In particular, this means there can be no two strings of the same length in `L`.

```
fun inv1 :: "nat list list * nat list list => bool"
where
  "inv1 X = strictlysorted (snd X)"

fun inv2 :: "nat list list * nat list list => bool"
where
  "inv2 X = prefixfree ((fst X) @ (snd X))"
```

The first invariant is that the list of free strings is strictly sorted. This is needed to show both that there is at most one string of any given length and to show that the algorithm will always select the longest string it is able to.

```
fun inv :: "nat list list * nat list list => bool"
```

```
where
  "inv X = ((inv1 X) & (inv2 X))"
```

```
theorem kcstep_inv: "inv (A,F) ==> inv (kcstep A F n)"
```

This says simply that if the invariants held of the variables before running through the loop once, then they hold afterwards.

```
consts meas :: "nat list list => rat"        (* The measure of a prefix free set *)
primrec
  "meas [] = 0"
  "meas (x # xs) = expn2 (length x) + meas xs"
```

This defines the measure of a list of strings: the usual $\sum_{x \in X} 2^{-|x|}$.

```
lemma kcstep_meas: "meas ((fst (kcstep A F n)) @ (snd (kcstep A F n)))
                    = meas (A@F)"
```

This says that measure is preserved at each step of the loop. This measure will be 1 for all the intermediate states of the kc algorithm, but we need this more general result for the inductive proofs to work.

A number of further intermediate results are required both to establish the above invariants and to apply them to the main theorems. Below are three of the most important, which one may recall from the proof of Theorem 9 (in total, there are 102 theorems and lemmas proved).

```
theorem meas_alloc: "[| expn2 n <= meas F; strictlysorted F |]
                    ==> length (last F) <= n"
```

```
lemma kcstep_correct1: "[|inv (A,F); expn2 n <= meas F|]
          ==> (tl (fst (kcstep A F n)) = A)
            & (length (hd (fst (kcstep A F n))) = n)"
```

```
lemma kcstep_correct2: "[|inv (A,F); expn2 n <= meas F|]
          ==> meas (fst (kcstep A F n)) = meas A + expn2 n"
```

Theorem meas_alloc formalises the result that if 2^{-n} is smaller than the measure of a set F, and that set has no two strings of the same length, then there is a string of length at most n in F. Lemma kcstep_correct1 says that if the invariants are satisfied by the current variables A and F, and the measure of F is at least 2^{-n}, then kcstep succeeds. This means that we allocate one new string of length exactly n, leaving the old strings untouched. Lemma kcstep_correct2 expresses an implied result: if the algorithm succeeds, then the measure of the list of allocated strings increases by exactly 2^{-n} (Isabelle will often not notice conclusions that seem obvious to the prover; they must be spelt out).

11 Final Remarks

If PA receives an algorithm for a machine V, a proof that V is universal and prefix-free, an integer $c \geq 0$, and a computable increasing sequence of rationals converging to a real $\gamma > 0$, then PA can prove that $\alpha = 2^{-c} \cdot \Omega_V + \gamma$ is c.e. and random. Similarly, if PA receives an algorithm for a machine U, a proof that U is universal and prefix-free, then it can prove that Ω_U is c.e. and random. This implies that every c.e. random real is provably c.e. and random—as stated in Solovay's email [18].

We have offered two representations for c.e. and random reals from which PA can prove that the real is c.e. and random. In the first we fix a provably universal prefix-free machine V and we vary the integer $c \geq 0$ and the c.e. real $\gamma > 0$ to get via the formula $2^{-c} \cdot \Omega_V + \gamma$ all c.e. and random reals. In the second we vary all provably universal prefix-free machines U to get via Ω_U all c.e. and random reals.

A key result was to show that the theorem that "a real is c.e. and random iff it is the halting probability of a universal machine" [4, 6, 16] can be proved in PA. Our proof, which is simpler than the standard one, can be used also for the original theorem.

We proved two negative results showing the importance of the representation for provability of randomness: a) there exists a universal machine whose universality cannot be proved in PA, b) there exists a universal machine U such that, based on U, PA cannot prove the randomness of Ω_U.

Chaitin [9] explicitly computed a constant c such that if N is larger than the size in bits of the program for enumerating the theorems of PA plus c, then PA cannot prove that a specific string x has complexity greater than N, $H_U(x) > N$. Consequently, PA cannot prove randomness of almost all random (finite) strings. Our positive result shows an interesting difference between the finite and the infinite cases of (algorithmic) randomness.

Does our positive result contradict Chaitin and Solovay's negative results discussed in the Introduction? The answer is negative because the digits of the binary expansion of a random c.e. real are not computable.

Our positive result would not be satisfactory without demonstrating our proofs with an automatic theorem prover. We have chosen Isabelle [17] to obtain an automatic proof of our version of the Kraft-Chaitin Theorem, one of the key results of this paper. The paper contains a description of the formalisation (for Isabelle) of the Kraft-Chaitin Theorem and the description of the main steps of the automatic proof; the full proof is available online [13].

Finally we speculate about the role of the automatic prover. How can an automatic theorem prover help understanding/proving a mathematical statement?[1] There are at least three possibilities. a) Use the prover to verify the theorem by discovering a proof, call it "Solovay mode" (because this corresponds to the result reported in this paper: Bob Solovay communicated to one of us the statement to be proved and we found a proof). It is worth observing that the Kraft-Chaitin Theorem has two "roles": one, as an algorithm, to be executed, the other, as a mathematical statement, to be proved. Previous formalisation efforts focused

---

[1]The reader may note that we don't question the fact that an automatic theorem prover *helps* understanding mathematics, [5].

only on the first part[2]; our present work was directed towards the second. One could imagine that mathematical journals might use such systems in the process of refereeing [5, 12]. b) The second possibility is to use the prover to verify a human-made proof—a full Isabelle proof for all results in this paper is under construction. c) The third possibility is to use the prover as some kind of "assistant" in an interactive process of discovery/proving. During the work to automate the proof of the Kraft-Chaitin Theorem a mistake in our human-made argument was unearthed and corrected. We also used the experience with Isabelle to test the adequacy of the representation of a c.e. random real in meeting the goal: to obtain the PA proof of randomness.

Acknowledgment

References

[1] W. S. Brainerd, L. H. Landweber. *Theory of Computation*, Wiley, New York, 1974.

[2] C. S. Calude. Chaitin Ω numbers, Solovay machines and incompleteness, *Theoret. Comput. Sci.* 284 (2002), 269–277.

[3] C. S. Calude. A characterization of c.e. random reals, *Theoret. Comput. Sci.* 271 (2002), 3–14.

[4] C. S. Calude. *Information and Randomness. An Algorithmic Perspective*, 2nd Edition, Revised and Extended, Springer Verlag, Berlin, 2002.

[5] C. S. Calude, E. Calude, S. Marcus. Proving and Programming, in C. S. Calude (ed.). *Randomness & Complexity, from Leibniz to Chaitin*, World Scientific, Singapore, 2007, 310–321.

[6] C. S. Calude, P. Hertling, B. Khoussainov, and Y. Wang. Recursively enumerable reals and Chaitin Ω numbers, in: M. Morvan, C. Meinel, D. Krob (eds.), *Proceedings of the 15th Symposium on Theoretical Aspects of Computer Science (Paris)*, Springer–Verlag, Berlin, 1998, 596–606. Full paper in *Theoret. Comput. Sci.* 255 (2001), 125–149.

[7] G. J. Chaitin. A theory of program size formally identical to information theory, *J. Assoc. Comput. Mach.* 22 (1975), 329–340.

[8] G. J. Chaitin. *Algorithmic Information Theory*, Cambridge University Press, Cambridge, 1987 (3rd printing 1990).

[9] G. J. Chaitin. *The Limits of Mathematics*, Springer, Singapore, 1998.

[10] T. C. Hales. Formal proof. *Notices of the AMS* 11 (2008), 1370–1380.

[11] R. Downey, D. Hirschfeldt. *Algorithmic Randomness and Complexity*, Springer, Heidelberg, to appear.

[12] P. C. Fischer. Theory of provable recursive functions, *Trans. Amer. Math. Soc.* 117 (1965), 494–520.

---

[2]The use of Lisp and Mathematica in Algorithmic Information Theory were pioneered by Chaitin—see [9].

[13] N. J. Hay. Formal proof of the Kraft-Chaitin theorem in Isabelle. Available online at
    http://www.cs.auckland.ac.nz/˜nickjhay/KraftChaitin.thy.

[14] M. Hoyrup, C. Rojas. Personal communication to C. Calude, 11 September 2008.

[15] R. Kaye. *Models of Peano Arithmetic*, Oxford Press, Oxford, 1991.

[16] A. Kučera, T. A. Slaman. Randomness and recursive enumerability, *SIAM J. Comput.*,
    31, 1 (2001), 199-211.

[17] T. Nipkow, L. C. Paulson, M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-
    Order Logic*, Springer, LLNCS 2283, 2002.

[18] R. M. Solovay. Personal communication to C. Calude, 23 March 2007.

[19] R. M. Solovay. A version of Ω for which ZFC can not predict a single bit, in C.S.
    Calude, G. Păun (eds.). *Finite Versus Infinite. Contributions to an Eternal Dilemma*,
    Springer-Verlag, London, 2000, 323–334.

[20] R. M. Solovay. *Draft of a paper (or series of papers) on Chaitin's work ...done for the
    most part during the period of Sept.–Dec. 1974*, unpublished manuscript, IBM Thomas
    J. Watson Research Center, Yorktown Heights, New York, May 1975, 215 pp.