Computing a Glimpse of Randomness

Cristian S. Calude, Michael J. Dinneen, and Chi-Kou Shu

CONTENTS

A Chaitin Omega number is the halting probability of a universal Chaitin (self-delimiting Turing) machine. Every Omega number is both *computably enumerable* (the limit of a computable, increasing, converging sequence of rationals) and *random* (its binary expansion is an algorithmic random sequence). In particular, every Omega number is strongly noncomputable. The aim of this paper is to describe a procedure, that combines Java programming and mathematical proofs, to compute the *exact values of the first 64 bits of a Chaitin Omega:*

0000001000000100000110001000011010001111110010111011101000010000.

Full description of programs and proofs will be given elsewhere.

1. INTRODUCTION

Any attempt to compute the uncomputable or to decide the undecidable is without doubt challenging, but hardly a new endeavor (see, for example, [Marxen and Buntrock 90], [Stewart 91], [Casti 97]). This paper describes a hybrid procedure (which combines Java programming and mathematical proofs) for computing the *exact values of the first 64 bits of a concrete Chaitin Omega number*, Ω_U, the halting probability of the universal Chaitin (self-delimiting Turing) machine U, see [Chaitin 90a]. Note that any Omega number is not only noncomputable, but random, making the computing task even more demanding.

Computing lower bounds for Ω_U is not difficult: we just generate more and more halting programs. Are the bits produced by such a procedure exact? *Hardly.* If the first bit of the approximation happens to be 1, then yes, it is exact. However, if the provisional bit given by an approximation is 0, then, due to possible overflows, nothing prevents the first bit of Ω_U from being either 0 or 1. This situation extends to other bits as well. Only an initial run of 1s may give exact values for some bits of Ω_U.

The paper is structured as follows. Section 2 introduces the basic notation. Computably enumerable (c.e.) reals, random reals, and c.e. random reals are presented

2000 AMS Subject Classification: Primary 68Q30; Secondary 68Q17

Keywords: Chaitin Omega number, halting problem, algorithmic randomness

in Section 3. Various theoretical difficulties preventing the exact computation of any bits of an Omega number are discussed in Section 4. The register machine model of Chaitin [Chaitin 90a] is discussed in Section 5. In Section 6 we summarize our computational results concerning the halting programs of up to 84 bits long for U. They give a lower bound for Ω_U which is proved to provide the exact values of the first 64 digits of Ω_U in Section 7.

Chaitin [Chaitin 00b] has pointed out that the self-delimiting Turing machine constructed in the preliminary version of this paper [Calude et al. 00] is universal in the sense of Turing (i.e., it is capable to simulate any self-delimiting Turing machine), but it is not universal in the sense of algorithmic information theory because the "price" of simulation is not bounded by an additive constant; hence, the halting probability is not an Omega number (but a computably enumerable real with some properties close to randomness). The construction presented in this paper is a self-delimiting Turing machine. Full details will appear in [Shu 03].

2. NOTATION

We will use notation that is standard in algorithmic information theory and assume familiarity with Turing machine computations, computable and computably enumerable (c.e.) sets (see, for example, [Bridges 94], [Odifreddi 99], [Soare 87], [Weihrauch 87]), and elementary algorithmic information theory (see, for example, [Calude 94]).

By $\mathbf{N}, \mathbf{Q}$, we denote the set of nonnegative integers (natural numbers) and rationals, respectively. If S is a finite set, then $\#S$ denotes the number of elements of S. Let $\Sigma = \{0, 1\}$ denote the binary alphabet. Let Σ^* be the set of (finite) binary strings, and Σ^ω the set of infinite binary sequences. The length of a string x is denoted by $|x|$. A subset A of Σ^* is *prefix-free* if whenever s and t are in A and s is a prefix of t, then $s = t$.

For a sequence $\mathbf{x} = x_0 x_1 \cdots x_n \cdots \in \Sigma^\omega$ and an nonnegative integer $n \geq 1$, $\mathbf{x}(n)$ denotes the initial segment of length n of $\mathbf{x}$ and x_i denotes the ith digit of $\mathbf{x}$, i.e. $\mathbf{x}(n) = x_0 x_1 \cdots x_{n-1} \in \Sigma^*$. Due to Kraft's inequality, for every prefix-free set $A \subset \Sigma^*$, $\Omega_A = \sum_{s \in A} 2^{-|s|}$ lies in the interval $[0, 1]$. In fact Ω_A is a probability: Pick, at random using the Lebesgue measure on $[0, 1]$, a real α in the unit interval and note that the probability that some initial prefix of the binary expansion of α lies in the prefix-free set A is exactly Ω_A.

Following Solovay [Solovay 75, Solovay 00], we say that C is a *(Chaitin)* (self-delimiting Turing) *machine*, shortly, a *machine*, if C is a Turing machine processing binary strings such that its program set (domain) $PROG_C = \{x \in \Sigma^* \mid C(x) \text{ halts}\}$ is a prefix-free set of strings. Clearly, $PROG_C$ is c.e.; conversely, every prefix-free c.e. set of strings is the domain of some machine. The *program-size complexity* of the string $x \in \Sigma^*$ (relatively to C) is $H_C(x) = \min\{|y| \mid y \in \Sigma^*,\ C(y) = x\}$, where $\min \emptyset = \infty$. A major result of algorithmic information theory is the following invariance relation: We can effectively construct a machine U (called *universal*) such that for every machine C, there is a constant $c > 0$ (depending upon U and C) such that for every $x, y \in \Sigma^*$ with $C(x) = y$, there exists a string $x' \in \Sigma^*$ with $U(x') = y$ (U simulates C) and $|x'| \leq |x| + c$ (the overhead for simulation is no larger than an additive constant). In complexity-theoretic terms, $H_U(x) \leq H_C(x) + c$. Note that $PROG_U$ is c.e., but not computable.

If C is a machine, then $\Omega_C = \Omega_{PROG_C}$ represents its halting probability. When $C = U$ is a universal machine, then its halting probability Ω_U is called a *Chaitin* Ω *number*, shortly, Ω *number*.

3. COMPUTABLY ENUMERABLE AND RANDOM REALS

Reals will be written in binary, so we start by looking at random binary sequences. Two complexity-theoretic definitions can be used to define random sequences (see [Chaitin 75, Chaitin 00a]): an infinite sequence $\mathbf{x}$ is *Chaitin random* if there is a constant c such that $H(\mathbf{x}(n)) > n - c$, for every integer $n > 0$, or, equivalently, $\lim_{n\to\infty} H(\mathbf{x}(n)) - n = \infty$. Other *equivalent* definitions include the Martin-Löf [Martin-Löf 66, Martin-Löf 66] definition using statistical tests (*Martin-Löf random sequences*), the Solovay [Solovay 75] measure-theoretic definition (*Solovay random sequences*), and the Hertling and Weihrauch [Hertling and Weihrauch 98] topological approach to define randomness (*Hertling–Weihrauch random sequences*). Independent proofs of the equivalence between the Martin-Löf and Chaitin definitions have been obtained by Schnorr and Solovay, see [Chaitin 90a, Chaitin 01]. In what follows, we will simply call "random" a sequence satisfying one of the above equivalent conditions. Their equivalence motivates the following "randomness hypothesis"([Calude 00]): *A sequence is "algorithmically random" if it satisfies one of the above equivalent conditions.* Of course, randomness implies strong noncomputability (see, for example, [Calude 94]), but the converse is false.

A real α is random if its binary expansion $\mathbf{x}$ *(i.e. $\alpha = 0.\mathbf{x}$) is random.* The choice of the binary base does not play any role, see [Calude and Jürgensen 94], [Hertling and Weihrauch 98], [Staiger 91]: *randomness is a property of reals not of names of reals.*

Following Soare [Soare 69], a real α is called c.e. if there is a computable, increasing sequence of rationals which converges (*not necessarily computably*) to α. We will start with several characterizations of c.e. reals (see [Calude et al. 01]). If $0.\mathbf{y}$ is the binary expansion of a real α with infinitely many ones, then $\alpha = \sum_{n \in X_\alpha} 2^{-n-1}$, where $X_\alpha = \{i \mid y_i = 1\}$.

Theorem 3.1. *Let α be a real in $(0, 1]$. The following conditions are equivalent:*

(i) There is a computable, nondecreasing sequence of rationals which converges to α.

(ii) The set $\{p \in \mathbf{Q} \mid p < \alpha\}$ of rationals less than α is c.e..

(iii) There is an infinite prefix-free c.e. set $A \subseteq \Sigma^*$ with $\alpha = \Omega_A$.

(iv) There is an infinite prefix-free computable set $A \subseteq \Sigma^*$ with $\alpha = \Omega_A$.

(v) There is a total computable function $f : \mathbf{N}^2 \to \{0, 1\}$ such that

(a) If for some k, n we have $f(k, n) = 1$ and $f(k, n + 1) = 0$, then there is an $l < k$ with $f(l, n) = 0$ and $f(l, n + 1) = 1$.

(b) We have: $k \in X_\alpha \iff \lim_{n\to\infty} f(k, n) = 1$.

We note that following Theorem 3.1, (v), given a computable approximation of a c.e. real α via a total computable function f, $k \in X_\alpha \iff \lim_{n\to\infty} f(k, n) = 1$; the values of $f(k, n)$ may oscillate from 0 to 1 and back; we will not be sure that they stabilized until 2^k changes have occurred (of course, there need not be so many changes, but in this case, there is no guarantee of the exactness of the value of the kth bit).

Chaitin [Chaitin 75] proved the following important result:

Theorem 3.2. *If U is a universal machine, then Ω_U is c.e. and random.*

The converse of Theorem 3.2 is also true: It has been proved by Kučera and Slaman [Kučera and Slaman 01] based on work reported in [Calude et al. 01] (see also [Calude and Chaitin 99], [Calude 02a], [Downey 02]):

Theorem 3.3. *Let $\alpha \in (0, 1)$. The following conditions are equivalent:*

(i) The real α is c.e. and random.

(ii) For some universal machine U, $\alpha = \Omega_U$.

4. THE FIRST BITS OF AN OMEGA NUMBER

We start by noting that

Theorem 4.1. *Given the first n bits of Ω_U, one can decide whether $U(x)$ halts or not on an arbitrary string x of length at most n.*

The first 10,000 bits of Ω_U include a tremendous amount of mathematical knowledge. In Bennett's words [Bennett and Gardner 79]:

> [Ω] embodies an enormous amount of wisdom in a very small space . . . inasmuch as its first few thousands digits, which could be written on a small piece of paper, contain the answers to more mathematical questions than could be written down in the entire universe.
>
> Throughout history mystics and philosophers have sought a compact key to universal wisdom, a finite formula or text which, when known and understood, would provide the answer to every question. The use of the Bible, the Koran and the I Ching for divination and the tradition of the secret books of Hermes Trismegistus, and the medieval Jewish Cabala exemplify this belief or hope. Such sources of universal wisdom are traditionally protected from casual use by being hard to find, hard to understand when found, and dangerous to use, tending to answer more questions and deeper ones than the searcher wishes to ask. The esoteric book is, like God, simple yet undescribable. It is omniscient, and transforms all who know it . . . Omega is in many senses a cabalistic number. It can be known of, but not known, through human reason. To know it in detail, one would have to accept its uncomputable digit sequence on faith, like words of a sacred text.

It is worth noting that even if we get, by some kind of miracle, the first 10,000 digits of Ω_U, the task of solving the problems whose answers are embodied in these bits is computable, but unrealistically difficult: The time it takes to find all halting programs of length less than n from $0.\Omega_0\Omega_2 \ldots \Omega_{n-1}$ grows faster than any computable function of n.

Computing some initial bits of an Omega number is even more difficult. According to Theorem 3.3, c.e. random reals can be coded by universal machines through their halting probabilities. How "good" or "bad" are these names? In [Chaitin 75] (see also [Chaitin 97, Chaitin 99]), Chaitin proved the following:

Theorem 4.2. *Assume that ZFC[1] is arithmetically sound.*[2] *Then, for every universal machine U, ZFC can determine the value of only finitely many bits of Ω_U.*

In fact, one can give a bound on the number of bits of Ω_U which ZFC can determine; this bound can be explicitly formulated, but it *is not computable*. For example, in [Chaitin 97] Chaitin described, in a dialect of Lisp, a universal machine U and a theory T, and proved that U can determine the value of at most $H(T)+15,328$ bits of Ω_U; $H(T)$ is the program-size complexity of the theory T, an *uncomputable* number.

Fix a universal machine U and consider all statements of the form

$$\text{“The } n^{th} \text{ binary digit of the expansion of } \Omega_U \text{ is } k\text{”}, \tag{4–1}$$

for all $n \geq 0, k = 0, 1$. How many theorems of the form (4–1) can ZFC prove? More precisely, is there a bound on the set of nonnegative integers n such that ZFC proves a theorem of the form (4–1)? From Theorem 4.2, we deduce that ZFC can prove only finitely many (true) statements of the form (4–1). This is Chaitin information-theoretic version of Gödel's incompleteness (see [Chaitin 97, Chaitin 99]):

Theorem 4.3. *If ZFC is arithmetically sound and U is a universal machine, then almost all true statements of the form (4–1) are unprovable in ZFC.*

Again, a bound can be explicitly found, but not effectively computed. Of course, for every c.e. random real α, we can construct a universal machine U such that $\alpha = \Omega_U$ and ZFC is able to determine finitely (but as many as we want) bits of Ω_U.

A machine U for which Peano Arithmetic can prove its universality and ZFC cannot determine more than the initial block of 1 bits of the binary expansion of its halting probability, Ω_U, will be called *Solovay machine*.[3]

[1] Zermelo set theory with choice.
[2] That is, any theorem of arithmetic proved by ZFC is *true*.
[3] Clearly, U depends on ZFC.

To make things worse Calude [Calude 02b] proved the following result:

Theorem 4.4. *Assume that ZFC is arithmetically sound. Then, every c.e. random real is the halting probability of a Solovay machine.*

For example, if $\alpha \in (3/4, 7/8)$ is c.e. and random, then in the worst case, ZFC can determine its first two bits (11), but no more. For $\alpha \in (0, 1/2)$, we obtained Solovay's Theorem [Solovay 00]:

Theorem 4.5. *Assume that ZFC is arithmetically sound. Then, every c.e. random real $\alpha \in (0, 1/2)$ is the halting probability of a Solovay machine which cannot determine any single bit of α. No c.e. random real $\alpha \in (1/2, 1)$ has the above property.*

The conclusion is that the worst fears discussed in the first section proved to materialize: In general, only the initial run of 1s (if any) can be exactly computed.

5. REGISTER MACHINE PROGRAMS

We start with the register machine model used by Chaitin [Chaitin 90a]. Recall that any register machine has a finite number of registers, each of which may contain an arbitrarily large nonnegative integer. The list of instructions is given below in two forms: our compact form and its corresponding Chaitin [Chaitin 90a] version. The main difference between Chaitin's implementation and ours is in the encoding: we use 7 bit codes instead of 8 bit codes.

L: ? L1 **(L: GOTO L1)**

This is an unconditional branch to L1. L1 is a label of some instruction in the program of the register machine.

L: ∧ R L1 **(L: JUMP R L1)**

Set the register R to be the label of the next instruction and go to the instruction with label L1.

L: @ R **(L: GOBACK R)**

Go to the instruction with a label which is in R. This instruction will be used in conjunction with the jump instruction to return from a subroutine. The instruction is illegal (i.e., run-time error occurs) if R has not been explicitly set to a valid label of an instruction in the program.

L: = R1 R2 L1 **(L: EQ R1 R2 L1)**

This is a conditional branch. The last 7 bits of register R1 are compared with the last 7 bits of register R2. If they are equal, then the execution continues at the instruction with label L1. If they are not equal, then execution continues with the next instruction in sequential order. R2 may be replaced by a constant which can be represented by a 7-bit ASCII code, i.e., a constant from 0 to 127.

L: # R1 R2 L1 **(L: NEQ R1 R2 L1)**

This is a conditional branch. The last 7 bits of register R1 are compared with the last 7 bits of register R2. If they are not equal, then the execution continues at the instruction with label L1. If they are equal, then execution continues with the next instruction in sequential order. R2 may be replaced by a constant which can be represented by a 7-bit ASCII code, i.e., a constant from 0 to 127.

L:) R **(L: RIGHT R)**

Shift register R right 7 bits, i.e., the last character in R is deleted.

L: (R1 R2 **(L: LEFT R1 R2)**

Shift register R1 left 7 bits, add to it the rightmost 7 bits of register R2, and then shift register R2 right 7 bits. The register R2 may be replaced by a constant from 0 to 127.

L: & R1 R2 **(L: SET R1 R2)**

The contents of register R1 are replaced by the contents of register R2. R2 may be replaced by a constant from 0 to 127.

L: ! R **(L: READ R)**

One bit is read into the register R, so the numerical value of R becomes either 0 or 1. Any attempt to read past the last data-bit results in a run-time error.

L: / **(L: DUMP)**

All register names and their contents, as bit strings, are written out. This instruction is also used for debugging.

L: % **(L: HALT)**

Halts the execution. This is the last instruction for each register machine program.

A *register machine program* consists of a finite list of labeled instructions from the above list, with the restriction that the `HALT` instruction appears only once, as the last instruction of the list. The data (a binary string) follows immediately the `HALT` instruction. The use of undefined variables is a run-time error. A program not reading the whole data, or attempting to read past the last data-bit, results in a run-time error. Because of the position of the `HALT` instruction and the specific way data is read, register machine programs are Chaitin machines.

To be more precise, we present a context-free grammar $G = (N, \Sigma, P, S)$ in Backus-Naur form which generates the register machine programs.

(1) N is the finite set of nonterminal variables:

$$\begin{aligned} N &= \{S\} \cup INST \cup TOKEN \\ INST &= \{\langle \mathrm{RMS_{Ins}}\rangle, \langle ?_{\mathrm{Ins}}\rangle, \langle \wedge_{\mathrm{Ins}}\rangle, \langle @_{\mathrm{Ins}}\rangle, \langle =_{\mathrm{Ins}}\rangle, \langle \#_{\mathrm{Ins}}\rangle, \\ &\quad \langle)_{\mathrm{Ins}}\rangle, \langle (_{\mathrm{Ins}}\rangle, \langle \&_{\mathrm{Ins}}\rangle, \langle !_{\mathrm{Ins}}\rangle, \langle /_{\mathrm{Ins}}\rangle, \langle \%_{\mathrm{Ins}}\rangle\} \\ TOKEN &= \{\langle \textsc{Data}\rangle, \langle \textsc{Label}\rangle, \langle \textsc{Register}\rangle, \langle \textsc{Constant}\rangle, \\ &\quad \langle \textsc{Special}\rangle, \langle \textsc{Space}\rangle, \langle \textsc{Alpha}\rangle, \langle \mathrm{LS}\rangle\} \end{aligned}$$

(2) Σ, the alphabet of the register machine programs, is a finite set of terminals, disjoint from N:

$$\begin{aligned} \Sigma &= \langle \textsc{Alpha}\rangle \cup \langle \textsc{Special}\rangle \cup \langle \textsc{Space}\rangle \cup \langle \textsc{Digit}\rangle \\ \langle \textsc{Alpha}\rangle &= \{a, b, c, \ldots, z\} \\ \langle \textsc{Special}\rangle &= \{:, /, ?, \wedge, @, =, \#,), (, \&, !, \%\} \\ \langle \textsc{Space}\rangle &= \{\text{`space'}, \text{`tab'}\} \\ \langle \textsc{Digit}\rangle &= \{0, 1, \ldots, 9\} \\ \langle \textsc{Constant}\rangle &= \{d \mid 0 \le d \le 127\} \end{aligned}$$

(3) P (a subset of $N \times (N \cup \Sigma)^*$) is the finite set of rules (productions):

$$\begin{aligned} S &\to \langle \mathrm{RMS_{Ins}}\rangle^* \langle \%_{\mathrm{Ins}}\rangle \langle \textsc{Data}\rangle \\ \langle \textsc{Data}\rangle &\to (0|1)^* \\ \langle \textsc{Label}\rangle &\to 0 \mid (1|2|\ldots|9)(0|1|2|\ldots|9)^* \\ \langle \mathrm{LS}\rangle &\to\ : \langle \textsc{Space}\rangle^* \\ \langle \textsc{Register}\rangle &\to \langle \textsc{Alpha}\rangle(\langle \textsc{Alpha}\rangle \cup (0|1|2|\ldots|9))^* \\ \langle \mathrm{RMS_{Ins}}\rangle &\to \langle ?_{\mathrm{Ins}}\rangle \mid \langle \wedge_{\mathrm{Ins}}\rangle \mid \langle @_{\mathrm{Ins}}\rangle \mid \langle =_{\mathrm{Ins}}\rangle \mid \langle \#_{\mathrm{Ins}}\rangle \mid \\ &\quad \langle)_{\mathrm{Ins}}\rangle \mid \langle (_{\mathrm{Ins}}\rangle \mid \langle \&_{\mathrm{Ins}}\rangle \mid \langle !_{\mathrm{Ins}}\rangle \mid \langle /_{\mathrm{Ins}}\rangle \end{aligned}$$

`(L: HALT)`

$$\langle \%_{\mathrm{Ins}}\rangle \to \langle \textsc{Label}\rangle\langle \mathrm{LS}\rangle\%$$

`(L: GOTO L1)`

$$\langle ?_{\mathrm{Ins}}\rangle \to \langle \textsc{Label}\rangle\langle \mathrm{LS}\rangle ? \langle \textsc{Space}\rangle^* \langle \textsc{Label}\rangle$$

`(L: JUMP R L1)`

$$\langle \wedge_{\mathrm{Ins}}\rangle \to \langle \textsc{Label}\rangle\langle \mathrm{LS}\rangle \wedge \langle \textsc{Space}\rangle^* \langle \textsc{Register}\rangle \langle \textsc{Space}\rangle^+ \langle \textsc{Label}\rangle$$

`(L: GOBACK R)`

$$\langle @_{\mathrm{Ins}}\rangle \to \langle \textsc{Label}\rangle\langle \mathrm{LS}\rangle @ \langle \textsc{Space}\rangle^* \langle \textsc{Register}\rangle$$

`(L: EQ R 0/127 L1 or L: EQ R R2 L1)`

$$\begin{aligned} \langle =_{\mathrm{Ins}}\rangle \to\ & \langle \textsc{Label}\rangle\langle \mathrm{LS}\rangle = \langle \textsc{Space}\rangle^* \langle \textsc{Register}\rangle \langle \textsc{Space}\rangle^+ \\ & \langle \textsc{Constant}\rangle \langle \textsc{Space}\rangle^+ \langle \textsc{Label}\rangle \mid \langle \textsc{Label}\rangle\langle \mathrm{LS}\rangle = \\ & \langle \textsc{Space}\rangle^* \langle \textsc{Register}\rangle \langle \textsc{Space}\rangle^+ \langle \textsc{Register}\rangle \\ & \langle \textsc{Space}\rangle^+ \langle \textsc{Label}\rangle \end{aligned}$$

(L: NEQ R 0/127 L1 or L: NEQ R R2 L1)

$$\langle \#_{\text{Ins}} \rangle \rightarrow \langle \text{Label} \rangle \langle \text{LS} \rangle \# \langle \text{Space} \rangle^* \langle \text{Register} \rangle \langle \text{Space} \rangle^+ \langle \text{Constant} \rangle \langle \text{Space} \rangle^+ \langle \text{Label} \rangle \mid \langle \text{Label} \rangle \langle \text{LS} \rangle \# \langle \text{Space} \rangle^* \langle \text{Register} \rangle \langle \text{Space} \rangle^+ \langle \text{Register} \rangle \langle \text{Space} \rangle^+ \langle \text{Label} \rangle$$

(L: RIGHT R)

$$\langle)_{\text{Ins}} \rangle \rightarrow \langle \text{Label} \rangle \langle \text{LS} \rangle) \langle \text{Space} \rangle^* \langle \text{Register} \rangle$$

(L: LEFT R L1)

$$\langle (_{\text{Ins}} \rangle \rightarrow \langle \text{Label} \rangle \langle \text{LS} \rangle (\langle \text{Space} \rangle^* \langle \text{Register} \rangle \langle \text{Space} \rangle^+ \langle \text{Constant} \rangle \mid \langle \text{Label} \rangle \langle \text{LS} \rangle (\langle \text{Space} \rangle^* \langle \text{Register} \rangle \langle \text{Space} \rangle^+ \langle \text{Register} \rangle$$

(L: SET R 0/127 or L: SET R R2)

$$\langle \&_{\text{Ins}} \rangle \rightarrow \langle \text{Label} \rangle \langle \text{LS} \rangle \& \langle \text{Space} \rangle^* \langle \text{Register} \rangle \langle \text{Space} \rangle^+ \langle \text{Constant} \rangle \mid \langle \text{Label} \rangle \langle \text{LS} \rangle \& \langle \text{Space} \rangle^* \langle \text{Register} \rangle \langle \text{Space} \rangle^+ \langle \text{Register} \rangle$$

(L: READ R)

$$\langle !_{\text{Ins}} \rangle \rightarrow \langle \text{Label} \rangle \langle \text{LS} \rangle ! \langle \text{Space} \rangle^* \langle \text{Register} \rangle$$

(L: DUMP)

$$\langle /_{\text{Ins}} \rangle \rightarrow \langle \text{Label} \rangle \langle \text{LS} \rangle /$$

(4) $S \in N$ is the start symbol for the set of register machine programs.

It is important to observe that the above construction is *universal* in the sense of algorithmic information theory (see the discussion at the end of Section 1). Register machine programs are self-delimiting because the HALT instruction is at the end of any valid program. Note that the data, which immediately follows the HALT instruction, is read bit by bit with no endmarker: This type of construction was first programmed in Lisp by Chaitin [Chaitin 90a, Chaitin 00a].

To minimize the number of programs of a given length that need to be simulated, we have used "canonical programs" instead of general register machines programs. A *canonical program* is a register machine program in which (1) labels appear in increasing numerical order starting with 0; (2) new register names appear in increasing lexicographical order starting from 'a'; (3) there are no leading or trailing spaces; (4) operands are separated by a single space; (5) there is no space after labels or operators; and (6) instructions are separated by a single space. Note that *for every register machine program, there is a unique canonical program which is equivalent to it*, that is, both programs have the same domain and produce the same output on a given input. If x is a program and y is its canonical program, then $|y| \leq |x|$.

Here is an example of a canonical program:

```
0:!a 1:^b 4 2:!c 3:?11 4:=a 0 8 5:&c 110
6:(c 101 7:@b 8:&c 101 9:(c 113 10:@b 11:%10
```

To facilitate the understanding of the code, we rewrite the instructions with additional comments and spaces:

```
0:! a        // read the first data bit into register a
1:^ b 4      // jump to a subroutine at line 4
2:! c        // on return from the subroutine call c is
             //written out
3:? 11       // go to the halting instruction
4:= a 0 8    // the right most 7 bits are compared with
             // 127; if they are equal, then go to label 8
5:& c 'n'    // else, continue here and
6:( c 'e'    // store the character string 'ne' in register
7:@ b        // c; go back to the instruction with label 2
             // stored in register b
8:& c 'e'    // store the character string 'eq' in
             // register c
9:( c 'q'
10:@ b
11:%         // the halting instruction
10           // the input data
```

For optimization reasons, our particular implementation designates the first maximal sequence of SET/LET instructions as (static) register preloading instructions. We "compress" these canonical programs by (1) deleting all labels, spaces and the colon symbol with the first nonstatic instruction having an implicit label 0, (2) separating multiple operands by a single comma symbol, and (3) replacing constants with their ASCII numerical values. The compressed format of the above program is

```
!a^b,4!c?11=a,0,8&c,110(,c,101@b&,c,101(,c,113@b%10
```

Note that compressed programs are canonical programs because during the process of "compression," everything remains the same except for the elimination of space. Compressed programs use an alphabet with 49 symbols (including the halting character). The length is calculated as the sum of the program length and the data length (7 times the number of characters). For example, the length of the above program is $7 \times (49 + 2) = 357$.

For the remainder of this paper, we will be focusing on compressed programs.

6. SOLVING THE HALTING PROBLEM FOR PROGRAMS UP TO 84 BITS

A Java version interpreter for register machine compressed programs has been implemented; it imitates Chaitin's universal machine in [Chaitin 90a]. This interpreter has been used to test the Halting Problem for all register machine programs of at most 84 bits long. The results have been obtained according to the following procedure:

Program plus data length	Number of halting programs	Program plus data length	Number of halting programs
7	1	49	1012
14	1	56	4382
21	3	63	19164
28	8	70	99785
35	50	77	515279
42	311	84	2559837

TABLE 1. Distribution of halting programs.

1. Start by generating all programs of 7 bits and test which of them stops. All strings of length 7 which can be extended to programs are considered prefixes for possible halting programs of length 14 or longer; they will simply be called *prefixes*. In general, all strings of length n which can be extended to programs are *prefixes* for possible halting programs of length $n+7$ or longer. *Compressed prefixes* are prefixes of compressed (canonical) programs.

2. Testing the Halting Problem for programs of length $n \in \{7, 14, 21, \ldots, 84\}$ was done by running all candidates (that is, programs of length n which are extensions of prefixes of length $n-7$) for up to 100 instructions, and proving that any generated program which does not halt after running 100 instructions never halts. For example, (uncompressed) programs that match the regular expression `"0:\^ a 5.* 5:\? 0"` never halt on any input.

For example, the programs `"!a!b!a!b/%10101010"` and `"!a?0%10101010"` produce run-time errors; the first program "under reads" the data and the second one "over reads" the data. The program `"!a?1!b%1010"` loops.

One would naturally want to know the shortest program that halts with more than 100 steps. If this program is larger than 84 bits, then all of our looping programs never halt. The trivial program with a sequence of 100 dump instructions runs for 101 steps, but can we do better? The answer is yes. The following family of programs $\{P_1, P_2, \ldots\}$ recursively counts to 2^i, but has linear growth in size. The programs P_1 through P_4 are given below:[4]

```
/&a,0=a,1,5&a,1?2%
/&a,0&b,0=b,1,6&b,1?3=a,1,9&a,1?2%
/&a,0&b,0&c,0=c,1,7&c,1?4=b,1,10&b,1?3=a,1,13&a,1?2%
/&a,0&b,0&c,0&d,0=d,1,8&d,1?5=c,1,11&c,1?4=b,1,14&b,1
    ?3=a,1,17&a,1?2%
```

[4]In all cases the data length is zero.

In order to create the program P_{i+1} from P_i only 4 instructions are added, while updating "goto" labels.

The running time $t(i)$ (excluding the halt instruction) of program P_i is found by the following recurrence: $t(1) = 6$, $t(i) = 2 \cdot t(i-1) + 4$. Thus, since $t(4) = 86$ and $t(5) = 156$, P_5 is the smallest program in this family to exceed 100 steps. The size of P_5 is 86 bytes (602 bits), which is smaller than the trivial dump program of 707 bits. What is the smallest program that halts after 100 steps is an open question. A hybrid program, given below, created by combining P_2 and the trivial dump programs is the smallest known.

```
&a,0/&b,0/////////////////////=b,1,26&b,1?2=a,1,29
    &a,1?0%
```

This program of 57 bytes (399 bits) runs for 102 steps. Note that the problem of finding the smallest program with the above property is undecidable (see [Chaitin 99]).

The distribution of halting compressed programs of up to 84 bits for U, the universal machine processing compressed programs, is presented in Table 1. All binary strings representing programs have the length divisible by 7.

7. THE FIRST 64 BITS OF Ω_U

Computing all halting programs of up to 84 bits for U seems to give the exact values of the first 84 bits of Ω_U. However, this is false! To understand the point, let's first ask ourselves whether the converse implication in Theorem 4.1 is true? The answer is *negative*. Globally, if we can compute all bits of Ω_U, then we can decide the Halting Problem for every program for U and conversely. However, if we can solve for U the Halting Problem for all programs up to N bits long, we might not still get any exact value for any bit of Ω_U (less all values for the first N bits). Reason: A large set of very long halting programs can contribute to the values of more significant bits of the expansion of Ω_U.

$$
\begin{aligned}
\Omega_U^{7} &= 0.0000001\\
\Omega_U^{14} &= 0.00000010000001\\
\Omega_U^{21} &= 0.000000100000010000011\\
\Omega_U^{28} &= 0.0000001000000100000110001000\\
\Omega_U^{35} &= 0.00000010000001000001100010000110010\\
\Omega_U^{42} &= 0.000000100000010000011000100001101000110111\\
\Omega_U^{49} &= 0.0000001000000100000110001000011010001111101110100\\
\Omega_U^{56} &= 0.00000010000001000001100010000110100011111100101100011110\\
\Omega_U^{63} &= 0.000000100000010000011000100001101000111111001011101100111011100\\
\Omega_U^{70} &= 0.0000001000000100000110001000011010001111110010111011100111001111001001\\
\Omega_U^{77} &= 0.00000010000001000001100010000110100011111100101110111010000011100000101001111\\
\Omega_U^{84} &= 0.000000100000010000011000100001101000111111001011101110100001000000111101101101101101
\end{aligned}
$$

TABLE 2. Successive approximations for Ω_U.

So, to be able to compute the exact values of the first N bits of Ω_U, we need to be able to *prove* that longer programs do not affect the first N bits of Ω_U. And, fortunately, this is the case for our computation. Due to our specific procedure for solving the Halting Problem discussed in Section 6, any compressed halting program of length n has a compressed prefix of length $n-7$. This gives an upper bound for the number of possible compressed halting programs of length n.

Let Ω_U^n be the approximation of Ω_U given by the summation of all halting programs of up to n bits in length. Compressed prefixes are partitioned into two cases—ones with a `HALT` (%) instruction and ones without. Hence, halting programs may have one of the following two forms: either "$x\,y$ `HALT` u," where x is a prefix of length k not containing `HALT`, y is a sequence of instructions of length $n-k$ not containing `HALT`, and u is the data of length $m \geq 0$, or "$x\,u$," where x is a prefix of length k containing one occurrence of `HALT` followed by data (possibly empty) and u is the data of length $m \geq 1$. In both cases, the prefix x has been extended by at least one character. Accordingly, the "tail" contribution to the value of

$$\Omega_U = \sum_{n=0}^{\infty} \sum_{\{|w|=n,\, U(w)\,\mathrm{halts}\}} 2^{-|w|}$$

is bounded from above by the sum of the following two convergent series (which reduce to two independent sums of geometric progressions):

$$\sum_{m=0}^{\infty}\sum_{n=k}^{\infty} \underbrace{\#\{x \mid \text{prefix } x \text{ not containing } \mathrm{HALT}, |x| = k\}}_{x} \cdot \underbrace{48^{n-k}}_{y} \cdot \underbrace{1}_{\mathrm{HALT}} \cdot \underbrace{2^m}_{u} \cdot 128^{-(n+m+1)},$$

and

$$\sum_{m=0}^{\infty} \underbrace{\#\{x \mid \text{prefix } x \text{ containing } \mathrm{HALT}, |x| = k\}}_{x} \cdot \underbrace{2^m}_{u} \cdot 128^{-(m+k)}.$$

The number 48 comes from the fact that the alphabet has 49 characters and the last instruction before the data is `HALT` (%).

There are 402906842 prefixes not containing `HALT` and 1748380 prefixes containing `HALT`. Hence, the "tail" contribution of all programs of length 91 or greater is bounded by:

$$
\begin{aligned}
&\sum_{m=0}^{\infty}\sum_{n=13}^{\infty} 402906842 \cdot 48^{n-13} \cdot 2^m \cdot 128^{-(n+m+1)}\\
&\quad + \sum_{m=0}^{\infty} 1748380 \cdot 2^m \cdot 128^{-(m+13)}\\
&= 402906842 \cdot \frac{64}{128 \cdot 48^{13}} \cdot \sum_{n=13}^{\infty} \left(\frac{48}{128}\right)^n\\
&\quad + 1748380 \cdot \frac{1}{63 \cdot 128^{13}} < 2^{-68}, \qquad (7\text{–}1)
\end{aligned}
$$

that is, the first 68 bits of Ω_U^{84} "may be" correct by our method. Actually, we do not have 68 correct bits, but *only* 64 because adding a 1 to the 68th bit may cause an overflow up to the 65th bit. From (7–1) it follows that no other overflows may occur.

The list in Table 2 presents the main results of the computation:
The exact bits are underlined in the 84 approximation:

$$
\begin{aligned}
\Omega_U^{84} = 0.&\underline{0000001000000100000110001000011010001111}\\
&\underline{110010111011101000010000}01111011011011011101
\end{aligned}
$$

In summary, the first 64 exact bits of Ω_U are:

0000001000000100000110001000011010001111110010111011101000010000

8. CONCLUSIONS

The computation described in this paper is the first attempt to compute some initial exact bits of a random real. The method, which combines programming with mathematical proofs, can be improved in many respects. However, due to the impossibility of testing that long looping programs never actually halt (the undecidability of the Halting Problem), the method is essentially non-scalable.

As we have already mentioned, solving the Halting Problem for programs of up to n bits might not be enough to compute exactly the first n bits of the halting probability. In our case, we have solved the Halting Problem for programs of at most 84 bits, but we have obtained only 64 exact initial bits of the halting probability.

Finally, there is no contradiction between Theorem 4.5 and the main result of this paper. Ω's are halting probabilities of Chaitin universal machines, and each Ω is the halting probability of an infinite number of such machines. Among them, there are those (called Solovay machines in [Calude 02b]) which are in a sense "bad," as ZFC cannot determine more than the initial run of 1s of their halting probabilities. But the same Ω can be defined as the halting probability of a Chaitin universal machine which is not a Solovay machine, so ZFC, if supplied with that different machine, may be able to compute more (but always, as Chaitin proved, only finitely many) digits of the same Ω. Such a machine has been used for the Ω discussed in this paper.

All programs used for the computation as well as all intermediate and final data files (3 giga-bytes in gzip format) can be found at ftp://ftp.cs.auckland.ac.nz/pub/CDMTCS/Omega/

ACKNOWLEDGMENTS

We thank Greg Chaitin for pointing out an error in our previous attempt to compute the first bits of an Omega number [Calude and Dineen 00], and for continuous advice and encouragement.

REFERENCES

[Bennett and Gardner 79] C. H. Bennett, M. Gardner. "The random number omega bids fair to hold the mysteries of the universe." *Scientific American* **241** (1979), 20–34.

[Bridges 94] D. S. Bridges. *Computability—A Mathematical Sketchbook*, Springer Verlag, Berlin, 1994.

[Calude 94] C. S. Calude. *Information and Randomness. An Algorithmic Perspective*. Springer-Verlag, Berlin, 1994.

[Calude 00] C. S. Calude. "A glimpse into algorithmic information theory." in *Logic, Language and Computation*, P. Blackburn, N. Braisby, L. Cavedon, A. Shimojima (eds.), Volume 3, CSLI Series, pp. 67–83, Cambridge University Press, Cambridge, 2000.

[Calude 02a] C. S. Calude. "A characterization of c.e. random reals." *Theoret. Comput. Sci.*, **217** (2002), 3–14.

[Calude 02b] C. S. Calude. "Chaitin Ω numbers, Solovay machines and incompleteness." *Theoret. Comput. Sci.* **284** (2002), 269–277.

[Calude and Chaitin 99] C. S. Calude and G. J. Chaitin. "Randomness everywhere." *Nature*, **400**:22 (July 1999), 319–320.

[Calude et al. 00] C. S. Calude, M. J. Dinneen, and C. Shu. "Computing 80 Initial Bits of A Chaitin Omega Number: Preliminary Version." *CDMTCS Research Report* **146** (2000)

[Calude et al. 01] C. S. Calude, P. Hertling, B. Khoussainov, and Y. Wang. "Recursively enumerable reals and Chaitin Ω numbers." in *Proceedings of the 15th Symposium on Theoretical Aspects of Computer Science (Paris)*, M. Morvan, C. Meinel, D. Krob (eds.), pp. 596–606, Springer–Verlag, Berlin, 1998. Full paper in *Theoret. Comput. Sci.* **255** (2001), 125–149.

[Calude and Jürgensen 94] C. Calude and H. Jürgensen. "Randomness as an invariant for number representations." in *Results and Trends in Theoretical Computer Science*, H. Maurer, J. Karhumäki, G. Rozenberg (eds.), pp. 44–66, Springer-Verlag, Berlin, 1994.

[Casti 97] J. L. Casti. "Computing the uncomputable." *The New Scientist*, **154/2082** (17 May 1997), 34.

[Chaitin 75] G. J. Chaitin. "A theory of program size formally identical to information theory." *J. Assoc. Comput. Mach.* **22** (1975), 329–340. (Reprinted in: [Chaitin 90b], 113–128)

[Chaitin 90a] G. J. Chaitin. *Algorithmic Information Theory*, Cambridge University Press, Cambridge, 1987. (Third printing 1990)

[Chaitin 90b] G. J. Chaitin. *Information, Randomness and Incompleteness, Papers on Algorithmic Information Theory*, World Scientific, Singapore, 1987. (2nd ed., 1990)

[Chaitin 97] G. J. Chaitin. *The Limits of Mathematics*. Springer-Verlag, Singapore, 1997.

[Chaitin 99] G. J. Chaitin. *The Unknowable*, Springer-Verlag, Singapore, 1999.

[Chaitin 00a] G. J. Chaitin. *Exploring Randomness*, Springer-Verlag, London, 2000.

[Chaitin 00b] G. J. Chaitin. Personal communication to C. S. Calude, November 2000.

[Chaitin 01] G. J. Chaitin. Personal communication to C. S. Calude, December 2001.

[Downey 02] R. G. "Downey. Some Computability-Theoretical Aspects of Reals and Randomness." *CDMTCS Research Report* **173** (2002).

[Hertling and Weihrauch 98] P. Hertling and K. Weihrauch. "Randomness spaces." in *Automata, Languages and Programming, Proceedings of the 25th International Colloquium, ICALP'98* (Aalborg, Denmark), K. G. Larsen, S. Skyum, and G. Winskel (eds.), pp. 796–807, Springer-Verlag, Berlin, 1998.

[Kučera and Slaman 01] A. Kučera and T. A. Slaman. "Randomness and recursive enumerability." *SIAM J. Comput.*, **31**:1 (2001), 199–211.

[Martin-Löf 66] P. Martin-Löf. *Algorithms and Random Sequences*, Erlangen University, Nürnberg, Erlangen, 1966.

[Martin-Löf 66] P. Martin-Löf. "The definition of random sequences." *Inform. and Control* **9** (1966), 602–619.

[Marxen and Buntrock 90] H. Marxen and J. Buntrock. "Attacking the busy beaver 5." *Bull EATCS* **40** (1990), 247–251.

[Odifreddi 99] P. Odifreddi. *Classical Recursion Theory*, North-Holland, Amsterdam, Vol.1, 1989, Vol. 2, 1999.

[Shu 03] C. Shu. *Computing Exact Approximations of a Chaitin Omega Number*, Ph.D. Thesis, University of Auckland, New Zealand, 2003.

[Soare 69] R. I. Soare. "Recursion theory and Dedekind cuts." *Trans. Amer. Math. Soc.* **140** (1969), 271–294.

[Soare 87] R. I. Soare. *Recursively Enumerable Sets and Degrees*, Springer-Verlag, Berlin, 1987.

[Solovay 75] 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.

[Solovay 00] R. M. Solovay. "A version of Ω for which *ZFC* can not predict a single bit." in *Finite Versus Infinite. Contributions to an Eternal Dilemma*, C.S. Calude, G. Păun (eds.), pp. 323–334, Springer-Verlag, London, 2000.

[Staiger 91] L. Staiger. "The Kolmogorov complexity of real numbers." in *Proc. Fundamentals of Computation Theory*, Lecture Notes in Comput. Sci. No. 1684, G. Ciobanu and Gh. Păun (eds.), pp. 536–546, Springer–Verlag, Berlin, 1999.

[Stewart 91] I. Stewart. "Deciding the undecidable." *Nature* **352** (1991), 664–665.

[Weihrauch 87] K. Weihrauch. *Computability*, Springer-Verlag, Berlin, 1987.

Cristian S. Calude, Department of Computer Science, University of Auckland, Private Bag 92019, Auckland, New Zealand (cristian@cs.auckland.ac.nz)

Michael J. Dinneen, Department of Computer Science, University of Auckland, Private Bag 92019, Auckland, New Zealand (mjd@cs.auckland.ac.nz)

Chi-Kou Shu, Department of Computer Science, University of Auckland, Private Bag 92019, Auckland, New Zealand (cshu004@cs.auckland.ac.nz)

Received January 31, 2002; accepted February 11, 2002.