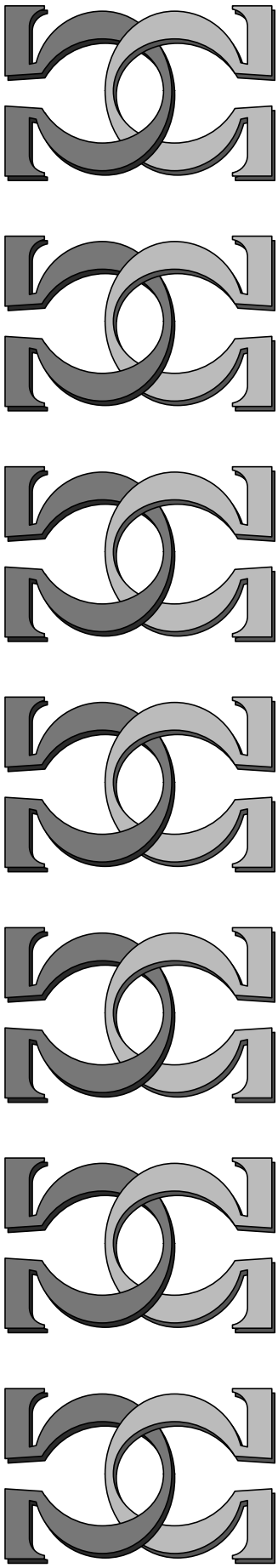

CDMTCS
Research
Report
Series

Computing 80 Initial Bits of A Chaitin Omega Number: Preliminary Version

C. S. Calude, M. J. Dinneen and C.-K. Shu
Department of Computer Science, Univeristy of Auckland, Auckland, New Zealand

CDMTCS-146
November 2000

Centre for Discrete Mathematics and Theoretical Computer Science

Computing 80 Initial Bits of A Chaitin Omega Number: Preliminary Version

Cristian S. Calude, Michael J. Dinneen and Chi-Kou Shu
Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
E-mails: {cristian,mjd,cshu004}@cs.auckland.ac.nz

Abstract

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 non-computable. The aim of this paper is to describe a procedure, which combines Java and Perl programming and mathematical proofs, for computing the *exact values of the first 80 bits of a Chaitin Omega:* 0.00000000000000000000100000010000001000000100000010000011100100111000101 0001010000. 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 new (see, for example, Marxen and Buntrock [20], Stewart [28]). This paper describes a hybrid procedure leading to the computation of the exact values of the first 80 bits of a concrete Omega number, Ω_U, the halting probability of the universal Chaitin (self-delimiting Turing) machine U constructed by Chaitin in [12]. Note that any Omega number is not only uncomputable, 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 sure, 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 to be either 0 or 1. This situation extends to other bits as well. Only an initial run of 1's may give exact values for some bits of Ω_U.

The aim of this paper is to describe informally a procedure, which combines Java and Perl programming and mathematical proofs, for computing the *exact values of the first 80 bits of a concrete Chaitin Omega number.* 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 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 [12] is discussed in section 5. In section 6 we summarize our computational results concerning the halting programs of up to 98 bits long for U. They give a lower bound for Ω_U which is proved to provide the exact values of the first 80 digits of Ω_U in section 7.

2 Notation

We will use notation that is standard in algorithmic information theory; we will assume familiarity with Turing machine computations, computable and computably enumerable (c.e.) sets (see, for example, Bridges [2], Odifreddi [21], Soare [24], Weihrauch [29]) and elementary algorithmic information theory (see, for example, Calude [4]).

By $\mathbf{N}, \mathbf{Q}$ we denote the set of nonnegative integers (natural numbers) and rationals, respectively. 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 [25, 26] 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 [11, 16]): an infinite sequence $\mathbf{x}$ is *Chaitin–Schnorr random* if there is a constant c such

that $H(\mathbf{x}(n)) > n - c$, for every integer $n > 0$, and, an infinite sequence $\mathbf{x}$ is *Chaitin random* if $\lim_{n\to\infty} H(\mathbf{x}(n)) - n = \infty$. Other *equivalent* definitions include Martin-Löf [19, 18] definition using statistical tests (*Martin-Löf random sequences*), Solovay [25] measure-theoretic definition (*Solovay random sequences*) and Hertling and Weihrauch [17] topological approach to define randomness (*Hertling–Weihrauch random sequences*). 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 [5]): *A sequence is "algorithmically random" if it satisfies one of the above equivalent conditions.* Of course, randomness implies strong non-computability (cf., for example, Calude [4]), 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, cf. Calude and Jürgensen [10], Hertling and Weihrauch [17], Staiger [27]: *randomness is a property of reals not of names of reals.*

Following Soare [23], 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 (cf. Calude, Hertling, Khoussainov and Wang [9]). 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 1 *Let α be a real in $(0, 1]$. The following conditions are equivalent:*

1. *There is a computable, nondecreasing sequence of rationals which converges to α.*
2. *The set $\{p \in \mathbf{Q} \mid p < \alpha\}$ of rationals less than α is c.e.*
3. *There is an infinite prefix-free c.e. set $A \subseteq \Sigma^*$ with $\alpha = \Omega_A$.*
4. *There is an infinite prefix-free computable set $A \subseteq \Sigma^*$ with $\alpha = \Omega_A$.*
5. *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 1, 5), 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 won't 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 [11] proved the following important result:

Theorem 2 *If U is a universal machine, then Ω_U is c.e. and random.*

The converse of Theorem 2 is also true: it has been proved by Slaman [22] based on work reported in Calude, Hertling, Khoussainov and Wang [9] (see also Calude and Chaitin [8] and Calude[6]):

Theorem 3 *Let $\alpha \in (0, 1)$. The following conditions are equivalent:*

1. *The real α is c.e. and random.*
2. *For some universal machine U, $\alpha = \Omega_U$.*

4 The First Bits of An Omega Number

We start by noting that

Theorem 4 *Given the first n bits of Ω_U one can decide whether $U(x)$ halts or not on an arbitrary program 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 [1]:

> *[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, c.e. random reals can be coded by universal machines through their halting probabilities. How "good" or "bad" are these names? In [11] (see also [14, 15]), Chaitin proved the following:

Theorem 5 *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 [14] Chaitin described, in a dialect of Lisp, a universal machine U and a theory T, and

[1] Zermelo set theory with choice.

[2] That is, any theorem of arithmetic proved by ZFC is *true*.

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

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

for all $n \geq 0, k = 0, 1$. How many theorems of the form (1) can ZFC prove? More precisely, is there a bound on the set of non-negative integers n such that ZFC proves a theorem of the form (1)? From Theorem 5 we deduce that ZFC can prove only finitely many (true) statements of the form (1). This is Chaitin information-theoretic version of Gödel's incompleteness (see [14, 15]):

Theorem 6 *If ZFC is arithmetically sound and U is a universal machine, then almost all true statements of the form (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 PA[3] 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.*[4] To make things worse Calude [7] proved the following result:

Theorem 7 *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 [26]:

Theorem 8 *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 1's (if any) can be exactly computed.

5 Register Machine Programs

We are going to define the register machine model used by Chaitin [12]. Recall that any register machine has a finite number of registers, each of which may contain an arbitrarily large non-negative integer. The list of instructions is given below in two forms: our compact form and its corresponding Chaitin [12] version. The only difference between Chaitin's implementation and ours is in encoding: we use 7 bits codes instead of 8 bits codes used by Chaitin.

L: ? L2 **(L: GOTO L2)**

[3] PA means Peano Arithmetic.
[4] Clearly, U depends on ZFC.

This is an unconditional branch to L2. L2 is a label of some register machine instruction in the program.

L: ∧ R L2 **(L: JUMP R L2)**

Set the register R to the label of the next sentence and go to the instruction with label L2.

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 program is invalid if R doesn't contain the specified label of an instruction in the program.

L: = R1 R2 L2 **(L: EQ R1 R2 L2)**

This is a conditional branch. The rightmost 7 bits of register R1 are compared with the rightmost 7 bits of register R2. If there are equal, then the execution continues at the instruction with label L2. If they are not equal, then execution continues with the next instruction in sequential order. R2 may be replaced by a constant.

L: # R1 R2 L2 **(L: NEQ R1 R2 L2)**

This is a conditional branch. The rightmost 7 bits of register R1 are compared with the rightmost 7 bits of register R2. If they are not equal, then the execution continues at the instruction with label L2. If they are equal, then execution continues with the next instruction in sequential order. R2 may be replaced by a constant.

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

Shift register R right 7 bits, i.e., the first 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, i.e. the first character in register R2 has been removed and added at the beginning of the character string in register R1. The register R2 may be replaced by a constant.

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

The contents of register R1 is replaced by the contents of register R2. R2 may be replaced by a constant.

L: ! R **(L: OUT R)**

The character string in register R is written out. This instruction is usually used for debugging.

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

Each register's name and the character string that it contains are written out. This instruction is also used for debugging.

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

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

The alphabet of the register machine programs consists of standard ASCII 7 bit codes for

- the above 11 special specification symbols, ?, ∧ , @, =, #,), (, &, !, /, % for instructions,
- the special characters : (line terminating character) and ‘ (quotation mark),
- all lower case letters, a, b, . . . , z,
- the digits 0, 1, . . . , 9,
- the upper case letters X and C (for constants),
- the space character,

which totals 52 out of 128 possible ASCII codes.

A *register machine program* consists of finite list of labelled instructions from the above list, with the restriction that the halt instruction appears only once, as the last instruction of the list. Because of the priviledged position of the halt instruction, register machine programs are Chaitin machines.

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 order starting with 0, (2) new register names appear in increasing lexicographical order starting from a, (3) there are no leading or trailing spaces. 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 string of 7 bit characters for a program and y is its canonical program, then $|y| \leq |x|$.

Here is an example of a canonical program (the additional comments do not form a part of the program, they are just facilitating understanding).

```
0: & a 41    // assign 41 to 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 127 8 // the right most 7 bits are compared with 127; if they
             // are equal, then go to label 8
5: & c C'n'  // else, continue here and
```

```
6: ( c C'e'  // store the character string 'ne' in register c
7: @ b       // go back to the instruction with label 2 stored in
             // register b
8: & c C'e'  // store the character string 'eq' in register c
9: ( c C'q'
10: @ b
11: %        // the halting instruction
```

6 Solving the Halting Problem for Programs Up to 98 Bits

A Java version interpreter for register machine programs has implemented Chaitin universal machine in [12]. This interpreter has been used to test the Halting Problem for all register machine programs of at most 98 bits long. The results have been obtained according to the following procedure:

1. Start by generating all programs of 7 bits and test which of them stops. All strings of length 7 wich can be extended to programs are considered prefixes for possible halting programs of length 14 or longer; they will be called simply *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. *Canonical prefixes* are prefixes of canonical programs.

2. Testing the Halting Problem for programs of length $n \in \{7, 14, 21, \ldots, 98\}$ 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, filtering out time-limit exceeding programs, and proving that any generated program which doesn't halt after running 100 instructions never halts.

The statistics of halting canonical programs of up to 98 bits for U (all binary strings representing programs have the length divisible by 7) is presented below.

Program length	**Number of halting programs**	**Program length**	**Number of halting programs**
7	0	56	7
14	0	63	19
21	1	70	66
28	1	77	382
35	1	84	1506
42	1	91	4686
49	2	98	12265

7 The First 80 Bits of Ω_U

Computing all halting programs of up to 98 bits for U seems to give the exact values of the first 98 bits of Ω_U. False! To understand the point let's first ask ourselves whether the converse implication in Theorem 4 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: Longer halting programs can contribute to the value of a "very early" bit of the expansion of 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 "too much" the first N bits of Ω_U. And, fortunately, this is the case for our computation. Due to the procedure of solving the Halting Problem discussed in section 6, any halting program of length n has as prefix of length $n-7$. This gives an upper bound for the number of possible halting programs of length n. For example, there are 668,424 prefixes of length 91, so the number of halting programs of length 98 is less than $668,424 \times 51 = 34,089,777$. Similarly, the number of prefixes length 105 is less than $2,003,645 \times 51 = 102,186,048$. The number 51 comes from the fact that the alphabet has 52 characters and a halting program has a unique halting instruction, the last one. 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. Accordingly, the "tail" contribution to the value of

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

is bounded from above by the series

$$\sum_{n=k}^{\infty} \#\{x \mid x \text{ prefix}, |x| = k\} \cdot 51^{n-k} \cdot 128^{-n}.$$

For example, the 'tail" contribution of all programs of length at most 91 is bounded by

$$\sum_{n=14}^{\infty} 668,424 \cdot 51^{n-14} \cdot 128^{-n} < 2^{-77},$$

that is, the first 77 bits of Ω_U^{91} proven correct by our method. For the case of length 98 we have

$$\sum_{n=15}^{\infty} 2,003,645 \cdot 51^{n-15} \cdot 128^{-n} < 2^{-83}, \tag{2}$$

so the first 80 bits of Ω_U^{98} "may be" correct. Actually we don't have 83 correct bits because the 82th and 83th bits are 1 which means that may be overflowed to 100. From (2) it follows that no other overflows may occur.

The following list presents the main results of the computation:

$\Omega_U^7 = 0.0000000$
$\Omega_U^{14} = 0.00000000000000$

$\Omega_U^{21} = 0.000000000000000000001$
$\Omega_U^{28} = 0.00000000000000000000010000001$
$\Omega_U^{35} = 0.0000000000000000000010000001000000010000001$
$\Omega_U^{42} = 0.000000000000000000001000000100000010000001000000100000111$
$\Omega_U^{49} = 0.000000000000000000001000000100000010000001000000100000100000111$
$\Omega_U^{56} = 0.000000000000000000001000000100000010000001000000100000100000111$
$\Omega_U^{63} = 0.000000000000000000001000000100000010000001000000100000100000111001001$
$\Omega_U^{70} = 0.000000000000000000001000000100000010000001000000100000111001001110000$
10
$\Omega_U^{77} = 0.000000000000000000001000000100000010000001000000100000111001001110000$
1001111110
$\Omega_U^{84} = 0.000000000000000000001000000100000010000001000000100000111001001110000$
10100010011100010
$\Omega_U^{91} = 0.000000000000000000001000000100000010000001000000100000111001001110000$
$10100010100000110100111 0$
$\Omega_U^{98} = 0.000000000000000000001000000100000010000001000000100000111001001110000$
$10100010100000111010110111010 01$

The exact bits are underlined in the case of the 91 and 98 approximations:

$\Omega_U^{91} = 0.\underline{000000000000000000001000000100000010000001000000100000111001001110000}$
$\underline{10100010100}0000110100111 0$

$\Omega_U^{98} = 0.\underline{000000000000000000001000000100000010000001000000100000111001001110000}$
$\underline{1010001010000}0111010110111010 01$

In conclusion, the first 80 exact bits of Ω_U are:

0.00000000000000000000100000010000001000000100000100000111001001110001010001010000

References

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

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

[3] C. Calude. *Theories of Computational Complexity*, North-Holland, Amsterdam, 1988.

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

[5] C. S. Calude. A glimpse into algorithmic information theory, in P. Blackburn, N. Braisby, L. Cavedon, A. Shimojima (eds.). *Logic, Language and Computation*, Volume 3, CSLI Series, Cambridge University Press, Cambridge, 2000, 65–81.

[6] C. S. Calude. A characterization of c.e. random reals, *Theoret. Comput. Sci.*, to appear.

[7] C. S. Calude. Chaitin Ω numbers, Solovay machines and incompleteness, *Theoret. Comput. Sci.*, to appear.

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

[9] 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 to appear in *Theoret. Comput. Sci.*

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

[11] G. J. Chaitin. A theory of program size formally identical to information theory, *J. Assoc. Comput. Mach.* 22 (1975), 329–340. (Reprinted in: [13], 113–128)

[12] G. J. Chaitin. *Algorithmic Information Theory*, Cambridge University Press, Cambridge, 1987. (third printing 1990)

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

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

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

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

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

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

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

[20] H. Marxen, J. Buntrock. Attaching the busy beaver 5, *Bull EATCS* 40 (1990), 247–251.

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

[22] T. A. Slaman. Randomness and recursive enumerability, *SIAM J. Comput.*, to appear.

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

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

[25] 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.

[26] 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.

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

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

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