Chaitin's Constant: An Uncomputable Number

Brendan Saxberg

Contents

1 Introduction

In short, Chaitin's constant is the probability that a random program of fixed finite length will terminate. This constant is deeply embedded in the realm of algorithmic information theory and has ties to the halting problem, Godel's incompleteness theorems, and statistics. The aim of this paper is to introduce the formalism surrounding the number and discuss strategies for enumerating parts of the number.

2 Definitions

Before we begin to delve into the terminology of information theory, some terms must be defined.

2.1 Terms

1. **Turing Machine**: A theoretical machine that takes in binary information stored on cells in an infinite strip of tape. This machine can read and write data by moving back and forth on the strip. It also has a state register that stores the state of the Turing machine and and action table that the Turing machine checks with to map instructions to actions. There are multiple precise definitions of Turing machines, but our discussion will not require any intimate detail with these differences. For the purposes of dealing with the Chaitin number in generality, we simply require that the machine be able to perform all these actions consistently.

2. **Register Machine**: A machine resembling a Turing machine, except it can store arbitrarily large integers on unique registers. Register machine programs are rules which the register machine follows to termination. It is very important to note that register machines are *not* Turing machines.

3. **Canonical program**: a register program for which

   (a) labels appear in increasing numerical order starting with 0

   (b) new register names appear in increasing lexicographical order

   (c) there are no leading or trailing spaces

   (d) operands are separated by a single space

   (e) labels are deleted as they implicitly appear in increasing order starting with 0

   (f) spaces are deleted and multiple operands are separate by a comma

   Canonical programs are constructed this way because they have the unique property that any register program can be represented by an equivalent canonical program. The specifics on syntax vary, but this standard was introduced by Chaitin himself when first using register machines. The

exact procedures listed above are not vital to our discussion. It is very important to note that the above enables us to greatly reduce the complexities of dealing with random programs, as will be discussed below in further detail.

4. **Computable**: A computable number is one for which there exists a finite terminating algorithm that can calculate the number to within arbitrary precision.

5. **Prefix free universal computable functions**: This corresponds to a programming language that has several nice properties which are required for proofs in halting problems. Prefix free means that our code has distinguishable commands. A sufficient condition satisfying this is having our theoretical strip of bits be read by the Turing machine one at a time. This is normally the default operation of theoretical Turing machines, but it is important to keep track of our requirements. The modifier Universal means that we can express any computable function in our language. A simple example would be prefacing a numerical value with the command to add 1 to that number.

2.2 Properties of Ω_U

1. **Algorithmically Random**: The shortest program to output the first n bits of an algorithmically random number must be of size at least n-O(1), or n minus a constant

2. **Normal number**: the digits of the decimal representation of the number are equidistributed (as if the digits were decided by coin tosses)

3. **arithmetical number**: a number that can be defined by a formula of first-order Peano arithmetic

3 The Halting Problem

The halting problem is a classic thought experiment in computability theory. To begin, we suppose for the sake of contradiction that the Halting problem is decidable. That is, there exists an algorithm to determine if a program will halt given a specific input. Let us call this program Halt(P,I) where a program P takes an input I, and

$$Halt(P, I) = \begin{cases} 1 & \text{if P(I) terminates} \\ 0 & \text{otherwise} \end{cases}$$

Let us create another program ξ, such that

$$\xi(x) = \begin{cases} terminate & \text{if Halt(x,x)=0} \\ loop & \text{if Halt(x,x)=1} \end{cases}$$

This may seem convoluted, but the crux of this argument comes from passing ξ to itself. If we do so, then

$$\xi(\xi) = \begin{cases} terminate & \text{if Halt}(\xi,\xi)=0 \\ loop & \text{if Halt}(\xi,\xi)=1 \end{cases}$$

However, Halt(ξ,ξ)=0 when $\xi(\xi)$ fails to terminate, and Halt(ξ,ξ)=1 when$\xi(\xi)$ terminates. Thus a contradiction is produced in either event of $\xi(\xi)$. So we know that Halt(P,I) cannot exist. So, there is no Turing machine program that can tell us whether another program will ever terminate or not.

4 Chaitin's Constant Ω

4.1 Definition

In the following definition, p is a program expressed in binary form so that the number of different programs of a fixed length. For a given universal Turing machine U and a program p of length $|p|$, the Chaitin constant is defined to be

$$\Omega_U = \sum_{p \ halts} 2^{-|p|}.$$

An intuitive way of organizing these programs is by length. Then the sum takes the form

$$\Omega_U = \sum_{n=0}^{\infty} \sum_{|p|=n} 2^{-n}$$

where U(p) halts.

It may seem initially as though Ω can diverge. For instance, a program language and Turing machine created in such a way that the Turing machine halts on both one bit programs (0 and 1) and halts for another (say 11), then

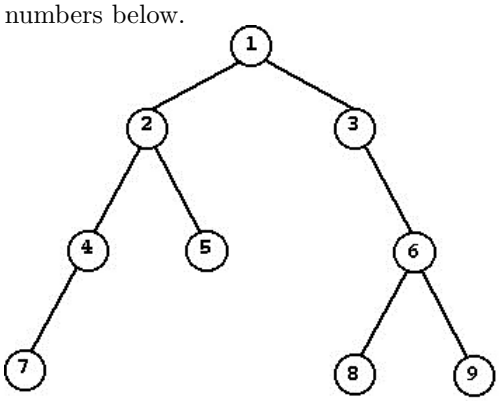
$$\Omega_U = \sum_{n=0}^{\infty} \sum_{|p|=n} 2^{-n} \geq \frac{1}{2} * 2 + \frac{1}{4} > 1.$$

This presents a problem. If we wish to interpret Ω as a probability, then it must be a real number between 0 and 1. The solution comes from a prerequisite on the languages allowed for Chaitin's constant. Kraft's inequality tell us that any binary tree satisfies

$$\sum_{l \in leaves} 2^{-depth(l)} \leq 1$$

where l is the specific leaf in the binary tree, and depth(l) is the depth of the specified leaf inside the tree. If our language is uniquely decodable then our language can be interpreted in binary tree form. This is because our programs p can be interpreted as prefix codes. Thus, we can use Kraft's inequality, implying that $\Omega \in [0, 1]$. For example, the following is a binary tree with 9 elements.

7,5,8, and 9 are leaves on this tree because they do not branch into any other numbers below.



To equate our language to a binary tree, imagine we have a program I to analyze. The canonical representation of this program will be another program I' such that $|I'| < |I|$. Additionally, the canonical program I' will be a leaf on the binary tree representation of programs. It is this "collapse" of any program to it's canonical program that allows us to apply Kraft's inequality and know that Ω is between 0 and 1.

4.2 Strength, and Difficulty, of Ω

Ω captures a significant amount of information about computer programs. Suppose we were to have the first N digits of Chaitin's number. Then, for any arbitrary prefix free universal computable function $U(x)$ and a string x of length no more than N, we can determine whether or not $U(x)$ halts.[1] Thus, given Ω_U to some precision, the Halting problem is solved for programs with string arguments of length dependent on that precision. This is an incredible result! In section three it was established that the halting problem was undecidable, yet with Ω_U at hand we are able to make headway into deciding algorithmic termination. However, this leads us to the question: What exactly is the value of Ω_U, given a particular machine and language?

Attempting to calculate Chaitin's constant leads to some disturbing realities surrounding the number. Theorem 4.2 [1] states

Assume that ZFC is arithmetically sound. Then, for every universal machine U, ZFC can determine the value of only finitely many bits of Ω_U.

Since we have proved that the Halting problem is undecidable in section three, this is no surprise. If we were able to obtain all of the bits of Ω_U for an arbitrary universal machine U, then we would have solved the halting problem.

It is important to recognize what we have done here. The definition for Ω_U is clear and defined in predicate logic, yet the number itself is not computable.

There is no Turing algorithm that can generate the number to arbitrary precision.

4.3 Computing the uncomputable

The main result of "Computing a Glimpse of Randomness" is the use of register machine programs and register machines to approximate Ω_U for Turing machines. In the paper [1] register machines with a program generator are used to solve the halting problem for programs up to 84 bits in their language. The reason this is significant is that the register machine and language developed admit a specific Ω which is *also* the Ω for an infinite number of Chaitin universal machines, some of which are Turing machines. Thus, solving the problem for this register machine allows us to say we have solved the problem for some set of Turing machines as well.

The method used for enumerating Ω in the cited paper is to test all programs up to and including 84 bits to see whether or not they terminate. This may seem like a blatant contradiction to the previous part of the paper, but the language compression that is performed by the register program requirements (and the specific instructions listed in [1] p. 364-366) force programs that run longer than 100 executions to never terminate. Since these register programs are canonical, "all" programs can be represented uniquely by these, so the property holds to all of the duplicates and larger programs covered by the canonical programs. Thus, computing these small-sized programs will get us started towards writing down a part of Ω. If you so wish, the link to some of the computational software is inside [1] and can be used by you to compute the same value.

The number retrieved after this process is, in binary form,

Ω=0.000000100000010000011000100001101000111111001011101110100001000001111011011011011101

which is 84 digits long. However, how do we know what we calculated actually resembles the true Chaitin number for our specific register machine? As we enumerate these programs it is possible to have a large string of programs that end up affecting digits above their program length by rounding. So, we need to prove that programs beyond 84 bits in length cannot contribute enough to the termination probability to affect our Ω, or failing that, prove that there is some number N for which the bits of Ω before bit N remain unchanged. The latter turns out to be true in the case of this paper.

The language used guarantees that the tail contribution to our calculation of Ω due to the programs of length 91 or longer are the sum of:

$$\sum_{m=0}^{\infty}\sum_{n=k}^{\infty}\#\{x|\text{prefix x not containing HALT}, |x|=k\}$$

and

$$\sum_{m=0}^{\infty} \#\{x|\text{prefix x containing HALT}, |x| = k\}.$$

Instructions in the language are 7 bits long, so our canonical program list up to length 84 includes programs up to length 90, since the programs greater than 84 but less than 90 represent additional data, and not instructions to the program (see [1] pp. 367.) In the language used in the paper, and for programs of length less than or equal to 84, there are 402906842 prefixes not containing the HALT command, and 1748380 prefixes containing the HALT command. Thus, after some work involving the multiplicity of canonical programs with HALT and without[1], the sums reduce to

$$402906842 * \frac{64}{128 * 48^{13}} * \sum_{n=13}^{\infty} (\frac{48}{128})^n + 1748380 * \frac{1}{63 * 128^{13}} < 2^{-68}.$$

With this inequality we know now that the tail contribution from the rest of the programs can be no larger than an additional 1 in the 68th bit in our Chaitin number Ω. Unfortunately, we have a run of 1's from the 65th bit to the 68th bit, so adding 1 to that last bit can cause a rounding up in the 65th bit. Thus, we only know Ω to the 64th bit with certainty. But disparage not, for we have successfully calculated Ω to the 64th digit! The first 64 digits in binary form are as follows:

$\Omega = 0.0000010000001000001100010000110100011111100101110111010000010000$

5 Reflection

In case you are curious the decimal equivalent of our enumerated Ω is around 0.0157499939956247687. Thus, randomly constructed programs in the language used in [1] have a probability of terminating at about 1.575%. Our numerical work here is not exactly enlightening, but the end result is quite bizarre. Feeding ZFC a Turing machine can result in almost no information about Ω at all. Solovay machines (a subset of Turing machines) can only have the initial run of 1's in its binary expansion determined for their Chaitin number. This sequence doesn't even have a single digit if Ω¡1/2 for that machine. Yet feeding ZFC a register machine yields significantly more information about its Chaitin number. Thus the strength of ZFC in enumerating Ω is dependent on the type of machine we give it. Although, no matter what machine we choose, we are always restricted to knowing a finite number of digits of Ω.

6 Improvements

Admittedly our success is short-lived. The problem with computing Ω this way is obvious. We never set out to find the Chaitin number of a specific Turing

machine that we had. Instead, we enumerated the Chaitin number of a register machine, for which the Turing machines that have the same Chaitin number are not known. All we know is that there exist Turing machines that have the same value. The first step in improving the results of this paper would be to associate a sub-class of Turing machines which have the Chaitin number of the register machines we are examining. Alternatively we could attempt to construct yet another type of machine with a Chaitin number that was enumerable in a similar fashion but whose equivalent Turing machines were more explicitly defined.

References

[1] Cristian S. Calude, Michael J. Dinneen, and Chi-Kou Shu, *Computing a Glimpse of Randomness*, 2000.