CDMTCS
Research
Report
Series

An Algebraic Characterization of the Halting Probability

Gregory Chaitin
IBM T. J. Watson Research Center, USA

CDMTCS-305
April 2007

Centre for Discrete Mathematics and
Theoretical Computer Science

An Algebraic Characterization of the Halting Probability

Gregory Chaitin[*]

Abstract

Using 1947 work of Post showing that the word problem for semigroups is unsolvable, we explicitly exhibit an algebraic characterization of the bits of the halting probability Ω. Our proof closely follows a 1978 formulation of Post's work by M. Davis. The proof is self-contained and not very complicated.

1. Introduction

Algorithmic information theory [4] shows that pure mathematics is infinitely complex and contains irreducible complexity. The canonical example of such irreducible complexity is the infinite sequence of bits in the base-two expansion of the halting probability Ω. The halting probability is defined by taking the following summation

$$0 \quad < \quad \Omega \quad = \sum_{U(p) \text{ halts}} 2^{-|p|} \quad < \quad 1$$

over all the self-delimiting programs p that halt when run on a suitably defined universal Turing machine U. Here $|p|$ denotes the size in bits of the program p. The value of Ω depends on the choice of U, but its surprising properties do not.

The numerical value of Ω is *maximally unknowable* in the following precise sense. You need an N-bit theory in order to be able to determine N bits of

[*] IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598, U.S.A., *chaitin@us.ibm.com.*

Ω [7]. Nevertheless, Ω has a kind of *diophantine reality,* because there is a diophantine equation with a parameter k that has finitely or infinitely many solutions depending on whether the kth bit of Ω is respectively 0 or 1 [4]. More recently, Ord and Kieu [5] have shown that there is also a diophantine equation with a parameter k that has an even or odd number of solutions depending on whether the kth bit of Ω is respectively 0 or 1.

The purpose of this note is to discuss the fact that as well as "diophantine reality," Ω also possesses a kind of *algebraic reality,* because there is an algebraic problem with a parameter i which yields the infinite sequence of bits b_i in the binary expansion of Ω:

$$\Omega = \sum_{i=1,2,3,\ldots} b_i \times 2^{-i}.$$

First of all, note that one can calculate better and better lower bounds on Ω, for example, by using the simple LISP function given in [7, pp. 65–69]. This works because Ω is the limit of Ω_n defined as follows:

$$\Omega_n = \sum_{|p| \leq n \text{ and } U(p) \text{ halts in } \leq n \text{ steps}} 2^{-|p|}.$$

As n tends to infinity, Ω_n tends to Ω, and from some point on each bit of Ω_n will remain correct, since Ω is irrational.[1] In other words, as n tends to infinity, the values of individual bits of Ω_n will fluctuate but eventually settle down to the correct values.

Our construction closely follows the presentation in Davis [1] of the idea in Post [2]. Davis [1] explains Post [2] so well, that it seems foolish to use a different formulation here. Sections 2 through 4 are taken word from word from Davis [1], except for changes to adapt Davis [1] to our present needs.[2]

2. The Turing – Post programming language

We work with a finite alphabet of α possible tape symbols

$$0 \quad 1 \quad a \quad b \quad c \quad \ldots \quad \square$$

[1] I.e., this limiting process cannot give us .3659999... instead of .3660000... because then Ω would be a rational number and would therefore not be irreducibly complex.

[2] Davis only allows his Turing–Post programs to use the two-symbol alphabet 0, 1. However, here we use a bigger alphabet, as was originally done by Post.

Here □ stands for a blank square on the tape. Any computation can be thought of as being carried out by an automatic scanning device, working with strings of these α symbols written on a linear tape, which executes instructions of the form:

- Write the symbol 0,
- Write the symbol 1, etc.
- Move scanner one square to the right,
- Move scanner one square to the left,
- Observe the symbol currently scanned and choose the next step accordingly,
- Stop.

The procedure which our calculator is carrying out then takes the form of a numbered list of instructions of these kinds. As in modern computing practice, it is convenient to think of these kinds of instructions as constituting a special *programming language.* A list of such instructions written in this language is then called a *program.*

We are now ready to introduce the Turing–Post programming language. In this language there are $2\alpha + 3$ kinds of instructions:

WRITE 0
WRITE 1 etc.
MOVE SCANNER RIGHT
MOVE SCANNER LEFT
GO TO INSTRUCTION # i IF 0 IS SCANNED
GO TO INSTRUCTION # i IF 1 IS SCANNED etc.
STOP

A Turing–Post program is then a list of instructions, each of which is of one of these $2\alpha + 3$ kinds. Of course in an actual program the letter i in each **GO TO** instruction must be replaced by a definite (positive whole) number.

In order that a particular Turing–Post program begin to calculate, it must have some "input" data. That is, the program must begin scanning at a specific square of a tape already containing a sequence of symbols. The symbol □ functions as a "blank"; although the entire tape is infinite, there are never more than a finite number of non-□ symbols that appear on it in the course of a computation. (A reader who is disturbed by the notion of an infinite tape can replace it for our purposes by a finite tape to which blank squares—that is, squares filled with □'s—are attached to the left or the right whenever necessary.)

3. What is a word problem?

We now explain what a word problem is.

In formulating a word problem one begins with a (finite) collection, called an *alphabet*, of symbols, called *letters*. Any string of letters is called a *word* on the alphabet. A word problem is specified by simply writing down a (finite) list of equations between words. Figure 1 exhibits a word problem specified by a list of 3 equations on the alphabet a, b, c. From the given equations many other equations may be derived by making substitutions in any word of equivalent expressions found in the list of equations. In the example of Figure 1, we derive the equation $bac = abcc$ by replacing the part ba by abc as permitted by the first given equation.

We have explained how to *specify* the data for a word problem, but we have not yet stated what the problem is. It is simply the problem of determining for two arbitrary given words on the given alphabet, whether one can be transformed into the other by a sequence of substitutions that are legitimate using the given equations.

4. Converting the question of whether a Turing – Post program halts into a word problem

Consider a Turing–Post program P which we assume consists of n instructions. We now use an alphabet consisting of the $\alpha + n + 2$ symbols:

$$0 \;\; 1 \;\; a \;\; b \;\; c \;\; \ldots \;\; \Box \;\; h \;\; q_1 \;\; q_2 \;\; \ldots \;\; q_n \;\; q_{n+1}.$$

The fact that the ith step of P is about to be carried out and that there is some given tape configuration is coded by a certain word (sometimes called a Post word) in this alphabet. This Post word is constructed by writing down the string of symbols constituting the current nonblank part of the tape, placing an h to its left and right (as punctuation marks) and inserting

Given an alphabet of three symbols a, b, c, and three equations

$$\begin{aligned} ba &= abc \\ bc &= cba \\ ac &= ca \end{aligned}$$

we can obtain other equations by substitution:

$$[ba]c = abcc$$

Or

$$\begin{aligned} b[ac] = [bc]a = c[ba]a = cabca &= [ca]bca = acbca = \ldots \\ &= cab[ca] = cabac = \ldots \\ &= ca[bc]a = cacbaa = \ldots \end{aligned}$$

(The expressions in brackets are the symbols about to be replaced.) In this context can be raised questions such as: "Can we deduce from the three equations listed above that $bacabca = acbca$?" The word problem defined by the three equations is the general question: to determine of an arbitrary given equation between two words, whether or not it can be deduced from the three given equations.

Figure 1. A Word Problem

the symbol q_i (remember that it is the ith instruction which is about to be executed) immediately to the left of the symbol being scanned. For example, with a tape configuration

$$\begin{array}{c}11011\\ \quad\uparrow\end{array}$$

and instruction number 4 about to be executed, the corresponding Post word would be

$$h110q_411h.$$

This correspondence between tape configurations and words makes it possible to translate the steps of a program into equations between words. For example, suppose that the fourth instruction of a certain program is

WRITE 0.

We translate this instruction into the α equations

$$q_40 \;=\; q_50, \qquad q_41 \;=\; q_50, \qquad \text{etc.}$$

which, for example, yield the equation between Post words

$$h110q_411h \;=\; h110q_501h$$

corresponding to the next step of the computation. Suppose next that the fifth instruction is

MOVE SCANNER RIGHT.

It requires $\alpha(\alpha+1)$ equations to fully translate this instruction, of which two typical ones are

$$q_501 \;=\; 0q_61, \qquad q_51h \;=\; 1q_6\Box h.$$

In a similar manner each of the instructions of a program can be translated into a list of equations. In particular when the ith instruction is **STOP**, the corresponding equation will be:

$$q_i \;=\; q_{n+1}.$$

So the presence of the symbol q_{n+1} in a Post word serves as a signal that the computation has halted. Finally, the 2α equations

$$\begin{aligned} q_{n+1}0 \;&=\; q_{n+1}, \qquad & q_{n+1}1 \;&=\; q_{n+1},\\ 0q_{n+1} \;&=\; q_{n+1}, \qquad & 1q_{n+1} \;&=\; q_{n+1},\end{aligned}$$

etc. serve to transform any Post word containing q_{n+1} into the word $hq_{n+1}h$. Putting all of the pieces together we see how to obtain a word problem which "translates" any given Turing–Post program.

Now let a Turing–Post program P begin scanning the leftmost symbol of the string v; the corresponding Post word is hq_1vh. Then if P will eventually halt, the equation

$$hq_1vh \;=\; hq_{n+1}h$$

will be derivable from the corresponding equations as we could show by following the computation step by step. If on the other hand P will never halt, we would like to prove that this same equation will not be derivable. The problem is that every time we use one of the equations which translates an instruction, we are either carrying the computation forward, or—in case we substitute from right to left—undoing a step already taken.

5. Running computations backwards and forwards simultaneously

So the computation, when it is expressed in terms of Post words, can run both forwards and backwards! Does this ruin things? Post realized that it cannot.[3] What we actually get is a tree with all the computations that eventually halt. The final word $hq_{n+1}h$ is the root of this tree. As time goes forward, computational trajectories can merge or join, but they can never split in two. So even if the computation runs backward for a while, and even splits off from the correct trajectory, when it starts forward again it will have to retrace its steps, so this detour does not affect the final result.

Post's argument is different; it's a proof by induction.[4] He considers the last backwards step in the derivation. The step right after that goes forward (since we were looking at the last backwards step), and must undo that backwards step. Hence we can delete these two steps which mutually annihilate each other and then apply Post's argument again to the resulting 2-step-shorter derivation.

[3] Unfortunately, Davis [1] does not explain why.

[4] See Post's Lemma II in Davis [3, pp. 297–298.]

6. Converting individual bits of the halting probability Ω into word problems

Recall the definition of Ω_j given in Section 1. Ω_j approximates Ω by looking at the finite set of all programs p up to j bits in size, and running each of them for j steps. Each program p that halts that is discovered in this way contributes $1/2^{|p|}$ to Ω_j, i.e., contributes 1 over 2 raised to size in bits of p. These approximations will get better and better. More precisely, as j tends to infinity, the values of individual bits of Ω_j will fluctuate but eventually settle down to the correct values.

Now let a^j be a string of j letters a, and let b^k be a string of k letters b. Consider a Turing–Post program which when started scanning the leftmost symbol of the word $a^j b^k$ on its tape, eventually halts if the kth bit of Ω_j is a 1, and never halts if the kth bit of Ω_j is a 0. This Turing–Post program computes the kth bit of the jth approximation to Ω, and then halts at once or loops forever depending on whether this bit is a 1 or a 0. Convert this Turing–Post program to a set of equations using Post's method as explained above. Then

$$hq_1 a^j b^k h \;=\; hq_{n+1} h$$

will be derivable from this set of equations iff the kth bit of Ω_j is a 1. Hence, fixing k and letting j vary, the set of all words of the form $hq_1 a^j b^k h$ which are equal to the word $hq_{n+1}h$ will be infinite if the kth bit of Ω is a 1, and it will be finite if the kth bit of Ω is a 0. This is our algebraic characterization of the bits of the halting probability.[5]

By the way, it is well worth it to read Davis [1], reprinted in Calude [8], in its entirety, not just the parts we have excerpted here. For more on the word problem, see Chapter 12 of Rotman [9]. For the philosophical significance of Ω, see [10].

[5] Using a different construction due to Ord and Kieu [5] (explained in Chaitin [6, pp. 135–139]) we can instead construct a set of equations with the following property. Fix k and let j vary. The set of all words of the form $hq_1 a^j b^k h$ which are equal to the word $hq_{n+1}h$ will always be finite, and furthermore the cardinality of this set will be odd if the kth bit of Ω is a 1, and its cardinality will be even if the kth bit of Ω is a 0.

References

[1] M. Davis, “What is a computation?,” in L. A. Steen, *Mathematics Today: Twelve Informal Essays,* Springer-Verlag, New York, 1978, pp. 241–267. Reprinted in Calude [8].

[2] E. Post, “Recursive unsolvability of a problem of Thue,” *Journal of Symbolic Logic,* Vol. 12 (1947), pp. 1–11. Reprinted in Davis [3, pp. 293–303].

[3] M. Davis, *The Undecidable,* Dover, 2004.

[4] G. Chaitin, *Algorithmic Information Theory,* Cambridge University Press, 1987.

[5] T. Ord, T. D. Kieu, “On the existence of a new family of diophantine equations for Ω,” *Fundamenta Informaticae,* Vol. 56 (2003), pp. 273–284.

[6] G. Chaitin, *Meta Math!,* Pantheon, 2005.

[7] G. Chaitin, *The Limits of Mathematics,* Springer-Verlag, 1998.

[8] C. Calude, *Randomness & Complexity, from Leibniz to Chaitin,* World Scientific, to appear.

[9] J. J. Rotman, *An Introduction to the Theory of Groups,* Fourth edition, Springer-Verlag, 1995, Corrected second printing, 1999.

[10] G. Chaitin, *Thinking about Gödel & Turing: Essays on Complexity, 1970–2007,* in preparation.