

AN ALGEBRAIC EQUATION FOR THE HALTING PROBABILITY

In R. Herken, *The Universal Turing Machine*, Oxford University Press, 1988, pp. 279–283

Gregory J. Chaitin

Abstract

We outline our construction of a single equation involving only addition, multiplication, and exponentiation of non-negative integer constants and variables with the following remarkable property. One of the variables is considered to be a parameter. Take the parameter to be $0, 1, 2, \dots$ obtaining an infinite series of equations from the original one. Consider the question of whether each of the derived equations has finitely or infinitely many non-negative integer solutions. The original equation is constructed in such a manner that the answers to these ques-

tions about the derived equations are independent mathematical facts that cannot be compressed into any finite set of axioms. To produce this equation, we start with a universal Turing machine in the form of the LISP universal function EVAL written as a register machine program about 300 lines long. Then we “compile” this register machine program into a universal exponential Diophantine equation. The resulting equation is about 200 pages long and has about 17,000 variables. Finally, we substitute for the program variable in the universal Diophantine equation the Gödel number of a LISP program for Ω , the halting probability of a universal Turing machine if n -bit programs have measure 2^{-n} . Full details appear in a book.¹

More than half a century has passed since the famous papers of Gödel (1931) and Turing (1936) that shed so much light on the foundations of mathematics, and that simultaneously promulgated mathematical formalisms for specifying algorithms, in one case via primitive recursive function definitions, and in the other case via Turing machines. The development of computer hardware and software technology during this period has been phenomenal, and as a result we now know much better how to do the high-level functional programming of Gödel, and how to do the low-level machine language programming found in Turing’s paper. And we can actually run our programs on machines and debug them, which Gödel and Turing could not do.

I believe that the best way to actually program a universal Turing machine is John McCarthy’s universal function EVAL. In 1960 McCarthy proposed LISP as a new mathematical foundation for the theory of computation (McCarthy 1960). But by a quirk of fate LISP has largely been ignored by theoreticians and has instead become the standard programming language for work on artificial intelligence. I believe that pure LISP is in precisely the same role in computational mathematics that set theory is in theoretical mathematics, in that it

¹This article is the introduction of the book G. J. Chaitin, *Algorithmic Information Theory*, copyright © 1987 by Cambridge University Press, and is reprinted by permission.

provides a beautifully elegant and extremely powerful formalism which enables concepts such as that of numbers and functions to be defined from a handful of more primitive notions.

Simultaneously there have been profound theoretical advances. Gödel and Turing's fundamental undecidable proposition, the question of whether an algorithm ever halts, is equivalent to the question of whether it ever produces any output. In another paper (Chaitin 1987a) I have shown that much more devastating undecidable propositions arise if one asks whether an algorithm produces an infinite amount of output or not.

Gödel expended much effort to express his undecidable proposition as an arithmetical fact. Here too there has been considerable progress. In my opinion the most beautiful proof is the recent one of Jones and Matijasevič (1984), based on three simple ideas:

1. the observation that $11^0 = 1$, $11^1 = 11$, $11^2 = 121$, $11^3 = 1331$, $11^4 = 14641$ reproduces Pascal's triangle, makes it possible to express binomial coefficients as the digits of powers of 11 written in high enough bases;
2. an appreciation of E. Lucas's hundred-year-old remarkable theorem that the binomial coefficient $\binom{n}{k}$ is odd if and only if each bit in the base-two numeral for k implies the corresponding bit in the base-two numeral for n ;
3. the idea of using register machines rather than Turing machines, and of encoding computational histories via variables which are vectors giving the contents of a register as a function of time.

Their work gives a simple straight-forward proof, using almost no number theory, that there is an exponential Diophantine equation with one parameter p which has a solution if and only if the p^{th} computer program (i.e., the program with Gödel number p) ever halts. Similarly, one can use their method to arithmetize my undecidable proposition. The result is an exponential Diophantine equation with the parameter n and the property that it has infinitely many solutions if and only if the n^{th} bit of Ω is a 1. Here Ω is the halting probability of a universal Turing machine if an n -bit program has measure 2^{-n} (Chaitin 1986a, 1986b).

Ω is an algorithmically random real number in the sense that the first N bits of the base-two expansion of Ω cannot be compressed into a program shorter than N bits, from which it follows that the successive bits of Ω cannot be distinguished from the result of independent tosses of a fair coin. It can also be shown that an N -bit program cannot calculate the positions and values of more than N scattered bits of Ω , not just the first N bits (Chaitin 1987a). This implies that there are exponential Diophantine equations with one parameter n which have the property that no formal axiomatic theory can enable one to settle whether the number of solutions of the equation is finite or infinite for more than a finite number of values of the parameter n .

What is gained by asking if there are infinitely many solutions rather than whether or not a solution exists? The question of whether or not an exponential Diophantine equation has a solution is in general undecidable, but the answers to such questions are not independent. Indeed, if one considers such an equation with one parameter k , and asks whether or not there is a solution for $k = 0, 1, 2, \dots, N - 1$, the N answers to these N questions really only constitute $\log_2 N$ bits of information. The reason for this is that we can in principle determine which equations have a solution if we know how many of them are solvable, for the set of solutions and of solvable equations is r.e. On the other hand, if we ask whether the number of solutions is finite or infinite, then the answers can be independent, if the equation is constructed properly.

In view of the philosophical impact of exhibiting an algebraic equation with the property that the number of solutions jumps from finite to infinite at random as a parameter is varied, I have taken the trouble of explicitly carrying out the construction outlined by Jones and Matijasevič. That is to say, I have encoded the halting probability Ω into an exponential Diophantine equation. To be able to actually do this, one has to start with a program for calculating Ω , and the only language I can think of in which actually writing such a program would not be an excruciating task is pure LISP. It is in fact necessary to go beyond the ideas of McCarthy in three fundamental ways:

1. First of all, we simplify LISP by only allowing atoms to be one character long. (This is similar to McCarthy's "linear LISP.")

2. Secondly, EVAL must not lose control by going into an infinite loop. In other words, we need a safe EVAL that can execute garbage for a limited amount of time, and always results in an error message or a valid value of an expression. This is similar to the notion in modern operating systems that the supervisor should be able to give a user task a time slice of CPU, and that the supervisor should not abort if the user task has an abnormal error termination.
3. Lastly, in order to program such a safe time-limited EVAL, it greatly simplifies matters if we stipulate “permissive” LISP semantics with the property that the only way a syntactically valid LISP expression can fail to have a value is if it loops forever. Thus, for example, the head (CAR) and tail (CDR) of an atom is defined to be the atom itself, and the value of an unbound variable is the variable.

Proceeding in this spirit, we have defined a class of abstract computers which, as in Jones and Matijasevič’s treatment, are register machines. However, our machine’s finite set of registers each contain a LISP *S*-expression in the form of a character string with balanced left and right parentheses to delimit the list structure. And we use a small set of machine instructions, instructions for testing, moving, erasing, and setting one character at a time. In order to be able to use subroutines more effectively, we have also added an instruction for jumping to a subroutine after putting into a register the return address, and an indirect branch instruction for returning to the address contained in a register. The complete register machine program for a safe time-limited LISP universal function (interpreter) EVAL is about 300 instructions long. To test this LISP interpreter written for an abstract machine, we have written in 370 machine language a register machine simulator. We have also rewritten this LISP interpreter directly in 370 machine language, representing LISP *S*-expressions by binary trees of pointers rather than as character strings, in the standard manner used in practical LISP implementations. We have then run a large suite of tests through the very slow interpreter on the simulated register machine, and also through the extremely fast 370 machine language interpreter,

in order to make sure that identical results are produced by both implementations of the LISP interpreter.

Our version of pure LISP also has the property that in it we can write a short program to calculate Ω in the limit from below. The program for calculating Ω is only a few pages long, and by running it (on the 370 directly, not on the register machine!), we have obtained a lower bound of $^{127}/_{128}$ -ths for the particular definition of Ω we have chosen, which depends on our choice of a self-delimiting universal computer.

The final step was to write a compiler that compiles a register machine program into an exponential Diophantine equation. This compiler consists of about 700 lines of code in a very nice and easy to use programming language invented by Mike Cowlshaw called REXX (Cowlshaw 1985). REXX is a pattern-matching string processing language which is implemented by means of a very efficient interpreter. It takes the compiler only a few minutes to convert the 300-line LISP interpreter into a 200-page 17,000-variable universal exponential Diophantine equation. The resulting equation is a little large, but the ideas used to produce it are simple and few, and the equation results from the straight-forward application of these ideas.

I have published the details of this adventure (but not the full equation!) as a book (Chaitin 1987b). My hope is that this book will convince mathematicians that randomness not only occurs in non-linear dynamics and quantum mechanics, but that it even happens in rather elementary branches of number theory.

References

Chaitin, G.J.

- 1986a Randomness and Gödel's theorem. *Mondes en Développement* No. 54–55 (1986) 125–128.
- 1986b *Information-theoretic computational complexity and Gödel's theorem and information*. In: *New Directions in the Philosophy of Mathematics*, ed. T. Tymoczko. Boston: Birkhäuser (1986).

1987a Incompleteness theorems for random reals. *Adv. Appl. Math.* **8** (1987) 119–146.

1987b *Algorithmic Information Theory*. Cambridge, England: Cambridge University Press (1987).

Cowlishaw, M.F.

1985 *The REXX Language*. Englewood Cliffs, NJ: Prentice-Hall (1985).

Gödel, K.

1931 On formally undecidable propositions of *Principia mathematica* and related systems I. In: *Kurt Gödel: Collected Works, Volume I: Publications 1929–1936*, ed. S. Feferman. New York: Oxford University Press (1986).

Jones, J.P., and Y.V. Matijasevič

1984 Register machine proof of the theorem on exponential Diophantine representation of enumerable sets. *J. Symb. Log.* **49** (1984) 818–829.

McCarthy, J.

1960 Recursive functions of symbolic expressions and their computation by machine, Part I. *ACM Comm.* **3** (1960) 184–195.

Turing, A.M.

1936 On computable numbers, with an application to the Entscheidungsproblem. *P. Lond. Math. Soc. (2)* **42** (1936) 230–265; with a correction, *Ibid. (2)* **43** (1936-7) 544–546; reprinted in: *The Undecidable*, ed. M. Davis. Hewlett, NY: Raven Press (1965).