INFORMATION-THEORETIC INCOMPLETENESS

G. J. Chaitin
IBM, P O Box 704
Yorktown Heights, NY 10598
chaitin@watson.ibm.com

Published 1992 by
World Scientific, Singapore

Acknowledgments—The author and the publisher are grateful to the following for permission to reprint the papers included in this volume:

- *La Recherche*;
- IPC Magazines Ltd. (*New Scientist*);
- Elsevier Science Publishing Co., Inc.
  (*Applied Mathematics and Computation*);
- Kluwer Academic Publishers B.V. and Marc E. Carvallo
  (*Nature, Cognition, and System, Vol. 3*);
- Springer-Verlag New York Inc. (*The Mathematical Intelligencer*).

Thanks are due to Abraham Peled, Michael Blasgen and Martin Hopkins (Computer Science Department, Watson Research Center) for supporting this research. Also to Axel Leijonhufvud and Vela Velupillai, whose invitation to the author to give a series of lectures at the UCLA Center for Computable Economics was an important stimulus.

To my parents Sara and Norman,
and to Maureen and Carmen for being so sweet!

Foreword

My book *Algorithmic Information Theory* is like hiking up a mountain by the fastest route, not stopping until one reaches the top. Here we shall instead take a look at interesting sights along the way and explore some alternate routes, in roughly the order that I discovered them.

In this book I shall survey eight different theories of program size complexity based on eight different programming models. And I'll discuss the incompleteness results that one gets in each of these eight theories.

I decided to tell this story in the form of a **mathematical** autobiography.

GREGORY CHAITIN

Contents

A LIFE IN MATH

"Guts and imagination!"

—The ambitious young director EASTON in
NORMAN CHAITIN's 1962 film *The Small Hours*

"The open secret of real success is to throw your whole personality at a problem."

—G. POLYA, *How to Solve It*

"To appreciate the living spirit rather than the dry bones of mathematics, it is necessary to inspect the work of a master at first hand. Textbooks and treatises are an unavoidable evil... The very crudities of the first attack on a significant problem by a master are more illuminating than all the pretty elegance of the standard texts which has been won at the cost of perhaps centuries of finicky polishing."

—ERIC TEMPLE BELL,
Mathematics: Queen & Servant of Science

Beginnings

In which we consider the plight of a bright child growing up in Manhattan and attempting to learn everything, including general relativity and Gödel's incompleteness theorem, on his own.

It was fun being a child in Manhattan in the late 1950s and early 1960s. I was lucky to go to a series of good public schools, and to take advantage of many special programs for bright children, and many accelerated school programs. The Bronx High School of Science was an exciting place, with an excellent school library, and lots of

very bright students. There were new almost-college-level experimental math, physics, biology and chemistry courses that had been provoked by the Russian success in orbiting the artificial satellite Sputnik before America could put a satellite in orbit. While at the Bronx High School of Science I got into a special program for bright high school students run by Columbia University, called the Science Honors Program, where I was able to play with computers. This was great fun, and not at all usual at that time, because computers were still a novelty.

We lived a block away from Central Park, near many branches of the New York City Public Library full of interesting books, and also at walking distance from the Museum of Modern Art (conveniently located across the street from a branch of the Public Library). I spent much of my time in the park, at the library, and in the Museum of Modern Art, where I saw many interesting old and foreign films. (I've never lost my taste for such films; the early INGMAR BERGMAN films and ERIC ROHMER's films are among my favorites as an adult.)

I would hike across Central Park, looking for interesting rocks, particularly crystals and fossils, which I would then compare with the collection of minerals in the Museum of Natural History across the park from our apartment. (As an adult I've continued hiking. First in Argentina, then in New York's Hudson Valley and Maine's Mt. Desert Island, and lately in Switzerland.)

My parents were involved with the United Nations and with the theater and film making. Recently I was flying back to New York on a Swissair flight after lecturing in GÖDEL's classroom in Vienna, and a documentary about MARILYN MONROE was shown on the plane. All of a sudden, there on the screen for a moment were my father and several others with MARILYN MONROE!

All this gave me the feeling that anything was possible, that the sky was the limit. As the ambitious young director EASTON says in my father's 1962 film *The Small Hours,* all it takes is "Guts and imagination!"

Two big steps in my childhood were when I was given permission to take out books from the adult section of the New York Public Library, even though I was much too young. Also when in high school as a member of the Columbia University Science Honors Program I was given the run of the stacks of the Columbia University mathematics

library, and could have first-hand contact with the collected works of masters like NEILS HENRIK ABEL and LÉONARD EULER. The library at the Bronx High School of Science was also good, and later I had permission to enter the stacks of the City College library. I read a lot; I was a sponge!

Scientific American with MARTIN GARDNER's *Mathematical Games* and the *Amateur Astronomer* and *Amateur Scientist* departments played a big role in my childhood. I read every issue, and tried out many of the mathematical games and amateur scientist experiments. One of my amateur scientist projects was building a VAN DE GRAAFF electrostatic generator, which I did when I was eleven years old.

My first loves were physics and astronomy. I wanted very badly to learn EINSTEIN's theories and build a telescope on my own. (I didn't quite manage it then, but as an adult I did.) One problem was that to read the physics books one had to understand mathematics. And it was hard to get equipment for physics experiments and astronomy. So I started studying math and experimenting with computers. I spent a lot of time in high school programming EDWARD F. MOORE's "Gedanken-Experiments on Sequential Machines,"[1] which led to my first publication, written while I was in high school.[2]

In high school I was also interested in game theory, information theory and in GÖDEL's incompleteness theorem. These subjects were still relatively new and exciting then, and there were not many books about them or about computers either, which were also a novelty at that time. I first had the idea of defining randomness via algorithmic incompressibility as part of the answer for an essay question on the entrance exam to get into the Science Honors Program! But I forgot the idea for a few years.

The summer between the Bronx High School of Science and the City College of the City University of New York, I thought about the simplicity and speed of programs for computing infinite sets of natural

[1] EDWARD F. MOORE, "Gedanken-Experiments on Sequential Machines," in CLAUDE E. SHANNON and JOHN MCCARTHY, *Automata Studies,* Annals of Mathematics Studies, No. 34, Princeton: Princeton University Press (1956), pp. 129–153.

[2] GREGORY J. CHAITIN, "An Improvement on a Theorem of E. F. Moore," *IEEE Transactions on Electronic Computers* EC-14 (1965), pp. 466–467.

numbers.[3] In this connection I came up with my first incompleteness theorem.[4] GÖDEL's original proof I had found to be infinitely fascinating, but incomprehensible. Once I formulated and proved an incompleteness theorem of my own, I began to understand GÖDEL's. (To really understand something, I believe that one must discover it oneself, not learn it from anyone else!)

During my first year at City College, while reading VON NEUMANN and MORGENSTERN's *Theory of Games and Economic Behavior* where they invoke random ("mixed") strategies for zero-sum two-person games without a saddle point ("Justification of the Procedure as Applied to an Individual Play"),[5] I recalled my idea for a definition of randomness, and decided to try to develop it mathematically using the notion of a TURING machine. I did this the next summer, the summer of 1965, between my first and second years at City College, while seeing interesting films at the Museum of Modern Art and thinking in Central Park. When college started in the fall, I was excused from attending classes to finish writing up my results![6,7,8,9] These papers contain two theories of program size based on counting the number of states in TURING machines, and one theory of program size based on counting bits in

[3] This eventually became: GREGORY J. CHAITIN, "On the Simplicity and Speed of Programs for Computing Infinite Sets of Natural Numbers," *Journal of the ACM* 16 (1969), pp. 407–422.

[4] Maximizing over all provably total recursive functions, one obtains a total recursive function that grows more quickly than any function that is provably total. See the discussion at the end of Section 2 of: GREGORY J. CHAITIN, "Gödel's Theorem and Information," *International Journal of Theoretical Physics* 22 (1982), pp. 941–954.

[5] JOHN VON NEUMANN and OSKAR MORGENSTERN, *Theory of Games and Economic Behavior,* Princeton: Princeton University Press (1953), pp. 146–148.

[6] GREGORY J. CHAITIN, "On the Length of Programs for Computing Finite Binary Sequences by Bounded-Transfer Turing Machines," Abstract 66T–26, *AMS Notices* 13 (1966), p. 133.

[7] GREGORY J. CHAITIN, "On the Length of Programs for Computing Finite Binary Sequences by Bounded-Transfer Turing Machines II," Abstract 631–6, *AMS Notices* 13 (1966), pp. 228–229.

[8] GREGORY J. CHAITIN, "On the Length of Programs for Computing Finite Binary Sequences," *Journal of the ACM* 13 (1966), pp. 547–569.

[9] GREGORY J. CHAITIN, "On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations," *Journal of the ACM* 16 (1969), pp. 145–159. Publication was inexplicably delayed.

programs for more abstract binary computers; altogether three different theories of program size! (I did all this when I was eighteen.) Then my parents and I moved to Buenos Aires, a sophisticated European city of cafes, cinemas, restaurants, beer halls, and rowing clubs. There my adult life began and I almost immediately joined IBM working as a programmer, continuing my research as a hobby.

Adult Activities

In August 1968 I presented a paper summarizing my ideas at the Pan-American Symposium of Applied Mathematics in Buenos Aires. This was published in 1970.[10] In 1970 I also published some thoughts on possible connections between my work and JOHN VON NEUMANN's ideas on self-reproducing automata[11] in the newsletter of the ACM's Special Interest Committee on Automata and Computability Theory (SICACT).[12] (SICACT was later renamed SIGACT, the Special Interest Group on Automata and Computability Theory).

I discovered my first information-theoretic incompleteness theorem in 1970 at the age of twenty-two while I was visiting a Rio de Janiero university and enjoying the tropical beaches and the excitement of Carnival.[13,14] (I should have discovered this incompleteness theorem in 1965, because this theorem is an immediate consequence of the proof that program-size complexity is uncomputable that I give in my 1966 *Journal of the ACM* paper. The reason that I didn't discover this theorem in 1965 is that I was so interested in randomness that I temporarily forgot about incompleteness! The moment that I thought about randomness **and** incompleteness, I realized that one cannot prove that

[10] GREGORY J. CHAITIN, "On the Difficulty of Computations," *IEEE Transactions on Information Theory* IT–16 (1970), pp. 5–9.

[11] JOHN VON NEUMANN, *Theory of Self-Reproducing Automata,* Urbana: University of Illinois Press (1966). Edited and completed by ARTHUR W. BURKS.

[12] GREGORY J. CHAITIN, "To a Mathematical Definition of 'Life'," *ACM SICACT News* 4 (January 1970), pp. 12–18.

[13] GREGORY J. CHAITIN, "Computational Complexity and Gödel's Incompleteness Theorem," Abstract 70T–E35, *AMS Notices* 17 (1970), p. 672.

[14] GREGORY J. CHAITIN, "Computational Complexity and Gödel's Incompleteness Theorem," *ACM SIGACT News* 9 (April 1971), pp. 11–12.

a string is random because of very strong information-theoretic constraints on the power of mathematical reasoning.)

And in Rio I obtained a copy of the original MIT LISP 1.5 manual.[15] When I returned to Buenos Aires I learned LISP by writing a LISP interpreter. This was the first of many LISP interpreters that I was to write, and the beginning of a long love affair with LISP. (I was having so much fun playing with LISP on computers that I did not realize that my theoretical ideas on program-size complexity apply beautifully to LISP.[16])

The early 1970s were a time of intense activity.[17,18,19,20,21,22] I gave a course on LISP and two courses on "Computability and Metamathematics" at Ciencias Exactas, the School of Exact Sciences of the University of Buenos Aires.

In the early 1970s I continued developing my information-theoretic approach to incompleteness. In October 1971, JACOB T. SCHWARTZ presented a paper on this for me at the Courant Institute Computational Complexity Symposium in New York. In June 1973, TERRENCE L. FINE presented a paper on this for me at the IEEE International Symposium on Information Theory in Ashkelon, Israel.[23] In 1974 these

---

[15] JOHN MCCARTHY, PAUL W. ABRAHAMS, DANIEL J. EDWARDS, TIMOTHY P. HART, and MICHAEL I. LEVIN, *LISP 1.5 Programmer's Manual,* Cambridge: MIT Press (1985).

[16] See pages 37–105 in this volume.

[17] GREGORY J. CHAITIN, Information-Theoretic Aspects of the Turing Degrees, Abstract 72T–E77, *AMS Notices* 19 (1972), pp. A–601, A–602.

[18] GREGORY J. CHAITIN, "Information-Theoretic Aspects of Post's Construction of a Simple Set," Abstract 72T–E85, *AMS Notices* 19 (1972), p. A–712.

[19] GREGORY J. CHAITIN, "On the Difficulty of Generating all Binary Strings of Complexity less than N," Abstract 72T–E101, *AMS Notices* 19 (1972), p. A–764.

[20] GREGORY J. CHAITIN, "On the Greatest Natural Number of Definitional or Information Complexity $\leq N$," *Recursive Function Theory: Newsletter* 4 (January 1973), pp. 11–13.

[21] GREGORY J. CHAITIN, "A Necessary and Sufficient Condition for an Infinite Binary String to be Recursive," *Recursive Function Theory: Newsletter* 4 (January 1973), p. 13.

[22] GREGORY J. CHAITIN, "There are Few Minimal Descriptions," *Recursive Function Theory: Newsletter* 4 (January 1973), p. 14.

[23] GREGORY J. CHAITIN, "Information-Theoretic Computational Complexity," *Abstracts of Papers, 1973 IEEE International Symposium on Information Theory,*

two papers were published as an invited paper in the *IEEE Transactions on Information Theory*[24] and a much longer paper in the *Journal of the ACM*.[25]

While visiting the IBM THOMAS J. WATSON Research Center in New York in 1974, I realized that it is very important to use abstract binary computers to measure program-size complexity that have **self-delimiting** programs.[26] (Indeed, the Turing machines that I had studied had self-delimiting programs. Dropping this when going to abstract binary computers had been a mistake.) The definition of **relative** complexity also needed a correction. This new approach made a world of difference: It was during this visit that I devised the halting probability Ω and showed that Ω is random. At this point it became appropriate to refer to this subject as "algorithmic information theory;" the definitions were now right. I was invited to speak about this at the opening plenary session of the 1974 IEEE International Symposium on Information Theory.[27]

In 1975 my first *Scientific American* article appeared.[28] I visited the IBM THOMAS J. WATSON Research Center again, wrote a few

June 25–29, 1973, King Saul Hotel, Ashkelon, Israel, IEEE Catalog No. 73 CHO 753–4 IT, p. F1–1.

[24]GREGORY J. CHAITIN, "Information-Theoretic Computational Complexity," *IEEE Transactions on Information Theory* IT–20 (1974), pp. 10–15. Reprinted in: THOMAS TYMOCZKO, *New Directions in the Philosophy of Mathematics,* Boston: Birkhäuser (1986), pp. 287–299.

[25]GREGORY J. CHAITIN, "Information-Theoretic Limitations of Formal Systems," *Journal of the ACM* 21 (1974), pp. 403–424.

[26]GREGORY J. CHAITIN, "A Theory of Program Size Formally Identical to Information Theory," *Journal of the ACM* 22 (1975), pp. 329–340.

[27]GREGORY J. CHAITIN, "A Theory of Program Size Formally Identical to Information Theory," *Abstracts of Papers, 1974 IEEE International Symposium on Information Theory,* October 28–31, 1974, University of Notre Dame, Notre Dame, Indiana, USA, IEEE Catalog No. 74 CHO 883–9 IT, p. 2.

[28]GREGORY J. CHAITIN, "Randomness and Mathematical Proof," *Scientific American* 232 (May 1975), pp. 47–52. Also published in the French, Italian and Japanese editions of *Scientific American.*

papers,[29,30,31] and then became a permanent member of the staff at the WATSON Research Center. For a decade I was heavily involved in pioneering work on RISC technology. In spite of this, I managed to publish a few survey papers,[32,33,34] an improved version of my 1970 SICACT paper on biology,[35] and a note with JACOB T. SCHWARTZ.[36]

In the mid 1980s, after working on a computer physics course,[37] I received an invitation to write the first book in the new Cambridge University Press series *Cambridge Tracts in Theoretical Computer Science.* This invitation was a tremendous stimulus. It encouraged me to re-think incompleteness using self-delimiting complexity (my 1974 definitions). While working on the book I discovered that there is randomness in arithmetic.[38,39] I proved this by transforming Ω into

[29] GREGORY J. CHAITIN, "Information-Theoretic Characterizations of Recursive Infinite Strings," *Theoretical Computer Science* 2 (1976), pp. 45–48.

[30] GREGORY J. CHAITIN, "Algorithmic Entropy of Sets," *Computers & Mathematics with Applications* 2 (1976), pp. 233–245.

[31] GREGORY J. CHAITIN, "Program Size, Oracles, and the Jump Operation," *Osaka Journal of Mathematics* 14 (1977), pp. 139–149.

[32] GREGORY J. CHAITIN, "Algorithmic Information Theory," *IBM Journal of Research and Development* 21 (1977), pp. 350–359, 496.

[33] GREGORY J. CHAITIN, "Algorithmic Information Theory," in SAMUEL KOTZ, NORMAN L. JOHNSON and CAMPBELL B. READ, *Encyclopedia of Statistical Sciences,* Vol. 1, New York: Wiley (1982), pp. 38–41.

[34] GREGORY J. CHAITIN, "Gödel's Theorem and Information," *International Journal of Theoretical Physics* 22 (1982), pp. 941–954. Reprinted in: THOMAS TYMOCZKO, *New Directions in the Philosophy of Mathematics,* Boston: Birkhäuser (1986), pp. 300–311.

[35] GREGORY J. CHAITIN, "Toward a Mathematical Definition of 'Life'," in RAPHAEL D. LEVINE and MYRON TRIBUS, *The Maximum Entropy Formalism,* Cambridge: MIT Press (1979), pp. 477–498.

[36] GREGORY J. CHAITIN and JACOB T. SCHWARTZ, "A Note on Monte Carlo Primality Tests and Algorithmic Information Theory," *Communications on Pure and Applied Mathematics* 31 (1978), pp. 521–527.

[37] GREGORY J. CHAITIN, "An APL2 Gallery of Mathematical Physics—A Course Outline," in IBM JAPAN, *Proceedings Japan 85 APL Symposium,* form N:GE18–9948–0 (1985), pp. 1–56.

[38] GREGORY J. CHAITIN, "Random Reals and Exponential Diophantine Equations," Research Report RC–11788, Yorktown Heights: IBM THOMAS J. WATSON Research Center (March 1986), 6 pp.

[39] GREGORY J. CHAITIN, "Incompleteness Theorems for Random Reals," *Advances in Applied Mathematics* 8 (1987), pp. 119–146.

a two-hundred page algebraic equation involving only whole numbers. The book[40] is full of new results, and has a foreword by JACOB T. SCHWARTZ. A collection of my papers[41] was published at the same time.[42]

My discovery of randomness in arithmetic attracted a great deal of attention, starting with an article by IAN STEWART in *Nature.*[43] I had the pleasure of writing about it not only in *Scientific American,*[44] but also in the English magazine *New Scientist*[45] and the French magazine *La Recherche.*[46] I also had the pleasure of being invited to speak on this work in the very room of the Mathematical Institute of the University of Vienna where GÖDEL used to teach.[47] The room where I lectured is called the GÖDEL classroom and has a stone plaque on the wall announcing that "HIER WIRKTE KURT GÖDEL VON 1932–1938". I also gave a less technical talk on the background and the broader

[40] GREGORY J. CHAITIN, *Algorithmic Information Theory,* Cambridge Tracts in Theoretical Computer Science, No. 1, Cambridge: Cambridge University Press (1987). Reprinted with corrections in 1988 and 1990.

[41] GREGORY J. CHAITIN, *Information, Randomness & Incompleteness—Papers on Algorithmic Information Theory,* Singapore: World Scientific (1987). A second edition was published in 1990.

[42] Incidentally, my father published a book when he was twenty. I was forty when these two books appeared.

[43] IAN STEWART, "The Ultimate in Undecidability," *Nature* 232 (10 March 1988), pp. 115–116.

[44] GREGORY J. CHAITIN, "Randomness in Arithmetic," *Scientific American* 259 (July 1988), pp. 80–85. Also published in the French, German, Italian, Japanese and Spanish editions of *Scientific American.*

[45] GREGORY J. CHAITIN, "A Random Walk in Arithmetic," *New Scientist* 125 (24 March 1990), pp. 44–46. Reprinted in: NINA HALL, *The New Scientist Guide to Chaos,* Harmondsworth: Penguin (1991), pp. 196–202. Also reprinted in this volume, p. 137.

[46] GREGORY J. CHAITIN, "Le Hasard des Nombres," *La Recherche* 22 (May 1991), pp. 610–615. Reprinted in this volume, p. 171. Also published in the Spanish edition of *La Recherche.*

[47] GREGORY J. CHAITIN, "Randomness in Arithmetic," in MARC E. CARVALLO, *Nature, Cognition and System,* Vol. 3, Dordrecht: Kluwer (1993), in press. Reprinted in this volume, p. 161.

significance of my work.[48,49]

How is the two-hundred page equation for Ω constructed? I start with an interpreter for a toy version of LISP that I originally used in the LISP course that I gave at Ciencias Exactas (the School of Exact Sciences of the University of Buenos Aires) in the early 1970s. I then take this LISP interpreter and transform it into an equation using ideas of JONES and MATIJASEVIČ.[50] Finally, I plug a LISP program for computing approximations to Ω into the equation that I produced from the LISP interpreter. The software for doing all of this was originally in assembly language. It has been rewritten twice: in C, and in the very high level language SETL2. (This software is available on request.)

I had the pleasure of speaking at two SOLVAY conferences in Brussels. One took place in October 1985, the other in September 1989. Both conferences were organized by PROF ILYA PRIGOGINE and sponsored by the SOLVAY Institute and the HONDA Foundation. Both conferences were quite stimulating. My first talk is summarized in a paper.[51] My second talk was filmed; a transcript is included in the second collection of my papers[52] (my *New Scientist* article is a condensed version of this talk; my *La Recherche* article is an expanded version).

I am grateful to the KING and QUEEN of Belgium, MR HONDA, MR SOLVAY and PROF PRIGOGINE for their hospitality. Two memorable moments: One, a dinner at MR SOLVAY's city residence. Among the guests, the KING and QUEEN of Belgium, the CROWN PRINCE of Japan, MR HONDA and PROF PRIGOGINE. The other, a party hosted

[48] GREGORY J. CHAITIN, "Zahlen und Zufall," in HANS-CHRISTIAN REICHEL and ENRIQUE PRAT DE LA RIBA, *Naturwissenschaft und Weltbild,* Vienna: Verlag Hölder–Pichler–Tempsky (1992), pp. 30–44.

[49] GREGORY J. CHAITIN, "Number and Randomness," in MARC E. CARVALLO, *Nature, Cognition and System,* Vol. 3, Dordrecht: Kluwer (1993), in press. Reprinted in this volume, p. 145.

[50] JAMES P. JONES and YURI V. MATIJASEVIČ, "Register Machine Proof of the Theorem on Exponential Diophantine Representation of Enumerable Sets," *Journal of Symbolic Logic* 49 (1984), pp. 818–829.

[51] GREGORY J. CHAITIN, "Randomness and Gödel's Theorem," *Mondes en Développement* 54–55 (1986), pp. 125–128.

[52] GREGORY J. CHAITIN, "Undecidability & Randomness in Pure Mathematics," in GREGORY J. CHAITIN, *Information, Randomness & Incompleteness—Papers on Algorithmic Information Theory,* Singapore: World Scientific (1990), pp. 307–313.

by MR HONDA, with a string quartet playing MOZART and dressed in period costumes. During a break, one of the performers standing in his wig and knee-breeches, eating sushi with chopsticks!

Recently I have given much thought to program size in LISP,[53,54,55,56] which works out nicely.

My latest paper presents surprising reformulations of my most basic incompleteness theorems using self-delimiting **enumeration** complexity.[57]

This book is the story of my mathematical adventures, and this story is told roughly in the order that I made the discoveries. The first part surveys eight different theories of program-size complexity: one based on TURING machines, three based on LISP, and four using abstract binary computers (see the index of complexity measures on p. 12). In each case the emphasis is on incompleteness theorems. The second part discusses the significance of information-theoretic incompleteness.

> "The essential in the being of a man of my type lies precisely in *what* he thinks and *how* he thinks, not in what he does or suffers."
>
> "This exposition has fulfilled its purpose if it shows the reader how the efforts of a life hang together and why they have led to expectations of a definite form."
>
> —ALBERT EINSTEIN, *Autobiographical Notes*

[53] GREGORY J. CHAITIN, "LISP Program-Size Complexity," *Applied Mathematics and Computation* 49 (1992), pp. 79–93. Reprinted in this volume, p. 37.

[54] GREGORY J. CHAITIN, "LISP Program-Size Complexity II," *Applied Mathematics and Computation* 52 (1992), pp. 103–126. Reprinted in this volume, p. 55.

[55] GREGORY J. CHAITIN, "LISP Program-Size Complexity III," *Applied Mathematics and Computation* 52 (1992), pp. 127–139. Reprinted in this volume, p. 83.

[56] GREGORY J. CHAITIN, "LISP Program-Size Complexity IV," *Applied Mathematics and Computation* 52 (1992), pp. 141–147. Reprinted in this volume, p. 97.

[57] GREGORY J. CHAITIN, "Information-Theoretic Incompleteness," *Applied Mathematics and Computation* 52 (1992), pp. 83–101. Reprinted in this volume, p. 107.

Index of Complexity Measures		
Notation	Name ..... Page	Unit
	Turing Machines Complexity Measure	
H_{tm}	Turing Machine Complexity 17	States
	LISP Complexity Measures	
H_{lisp}	(Real) LISP Complexity 37	Characters
H_{pf}	Parenthesis-Free LISP Complexity 83	Characters
H_{cs}	Character-String LISP Complexity 97	Characters
	Abstract Binary Computer Complexity Measures	
H_{b}	Blank-Endmarker Complexity 25	Bits
H_{be}	Blank-Endmarker Enumeration Complexity 30	Bits
H	Self-Delimiting Complexity 108	Bits
H_{e}	Self-Delimiting Enumeration Complexity . 113	Bits

Part I

Technical Survey

"The mathematician's patterns, like the painter's or poet's, must be *beautiful;* the ideas, like the colours or the words, must fit together in a harmonious way. Beauty is the first test: there is no permanent place in the world for ugly mathematics."

"A mathematician, like a painter or poet, is a maker of patterns. If his patterns are more permanent than theirs, it is because they are made with *ideas.*"

—G. H. HARDY, *A Mathematician's Apology*[1]

"Two proofs are better than one."

"Look out for simple intuitive ideas: Can you see it at a glance?"

"Can you use the result, or the method, for some other problem?"

"Quite often, when an idea that could be helpful presents itself, we do not appreciate it, for it is so inconspicuous. The expert has, perhaps, no more ideas than the inexperienced, but appreciates more what he has and uses it better."

"The future mathematician...should solve problems, choose the problems which are in his line, meditate upon their solution, and invent new problems. By this means, and by all other means, he should endeavor to make his first important discovery: he should discover his likes and his dislikes, his taste, his own line."

—G. POLYA, *How to Solve It*

[1]See also: ROBERT KANIGEL, *The Man Who Knew Infinity: A Life of the Genius Ramanujan,* New York: Scribners (1991).

"If we marvel at the patience and the courage of the pioneers, we must also marvel at their persistent blindness in missing the easier ways through the wilderness and over the mountains. What human perversity made them turn east to perish in the desert, when by going west they could have marched straight through to ease and plenty?"

—Eric Temple Bell,
Mathematics: Queen & Servant of Science

"'A mathematician's reputation rests on the number of bad proofs he has given.' (Pioneer work is clumsy.)"

—A. S. Besicovitch quoted in
J. E. Littlewood, *A Mathematician's Miscellany*

"On revient toujours à ses premières amours."
[One always returns to one's first loves.]

—Mark Kac, *Enigmas of Chance*

"He who seeks for methods without having a definite problem in mind seeks for the most part in vain."

—David Hilbert,
International Congress of Mathematicians,
Paris, 1900

"A theory is the more impressive the greater the simplicity of its premises is, the more different kinds of things it relates, and the more extended is its area of applicability."

—Albert Einstein, *Autobiographical Notes*

TURING MACHINES

G. J. Chaitin

Abstract

We review the Turing machine version of a fundamental information-theoretic incompleteness theorem, which asserts that it is difficult to prove that specific strings s have high Turing machine state complexity $H_{\rm tm}(s)$. We also construct a Turing machine "halting probability" $\Omega_{\rm tm}$ with the property that the initial segments of its base-two expansion asymptotically have the maximum possible Turing machine complexity $H_{\rm tm}$.

1. Turing Machine Complexity $H_{\rm tm}$

Following [1], consider Turing machines with a single one-way infinite tape (infinite to the right), a single read-write scanner, and a tape-symbol alphabet consisting of blank, 0 and 1. Such an n-state 3-tape-symbol Turing machine is defined by a $3 \times n$ table. This table gives the action to be performed and the next state as a function of the current state and the symbol currently being scanned on the tape. There are six possible actions: write blank (erase tape), write 0, write 1, shift tape left, shift tape right, or halt.

This chapter of the survey has not been published elsewhere.

Define the Turing machine complexity $H_{\rm tm}(s)$ of a bit string s to be the number of states in the smallest n-state 3-tape-symbol Turing machine that starts with the scanner at the beginning of a blank tape and halts with s written at the beginning of the tape, with the rest of the tape blank, and with the scanner on the first blank square of the tape.[1]

In [1] it is shown that the maximum possible Turing machine complexity $H_{\rm tm}(s)$ of an n-bit string s is asymptotic to $n/2\log_2 n$. Furthermore, most n-bit strings s have Turing machine complexity close to $n/2\log_2 n$. Equivalently, most $2n\log_2 n$ bit strings have Turing machine complexity close to n. Moreover, it is proposed in [1] that such strings are "random"; for example, it is shown that in the limit of large n such s have exactly the same relative frequency of 0's and 1's.

The sequel [2] considers random infinite bit strings and laboriously constructs an example, i.e., a specific infinite string whose initial n-bit segments have Turing machine complexity close to the maximum possible, which is $n/2\log_2 n$. As we show in Section 2, there is a much better way to do this: via a halting probability.

2. The "Halting Probability" $\Omega_{\rm tm}$

There are

$$n\,\frac{(6n)^{3n}}{n!}$$

n-state 3-tape-symbol Turing machines. The factor n on the left is to specify the initial state. The exponent $3n$ is because there are that many entries in the table defining the Turing machine. The base $6n$ is because each of the $3n$ table entries specifies one of six operations and one of n states. The denominator $n!$ is because any permutation of the state numbers gives the same behavior.

$\log n! \sim n\log n$. Thus

$$\log_2(\text{the total number of } n\text{-state Turing machines}) \sim 2n\log_2 n.$$

[1] As will be discussed below, this definition illustrates our "linkage convention."

Define

$$\Omega_{\rm tm} = \sum_{n=1}^{\infty} \left[\frac{\#\text{ of } n\text{-state Turing machines that halt}}{2^{\lceil \log_2(\text{total \# of } n\text{-state Turing machines})\rceil}} \right] 2^{-2\lceil \log_2 n\rceil - 1}.$$

The factor of $2^{-2\lceil \log_2 n\rceil - 1}$ is in order to insure convergence; it is easy to see that $\sum_{n=1}^{\infty} 2^{-2\lceil \log_2 n\rceil - 1} < 1$. The denominator of (total # of n-state Turing machines) is increased to the next highest power of two, so that only straight-forward binary arithmetic is involved in calculating $\Omega_{\rm tm}$.

If we knew the first $2n\log_2 n + o(n\log n)$ bits of $\Omega_{\rm tm}$, this would enable us to solve the halting problem for all $\le n$ state Turing machines.

Thus the first $\sim 2n\log_2 n$ bits of $\Omega_{\rm tm}$ tell us the Turing machine complexity of each string with $\le 2n\log_2 n$ bits. Hence the Turing machine complexity of the first $2n\log_2 n$ bits of $\Omega_{\rm tm}$ is asymptotic to the maximum possible for a $2n\log_2 n$ bit string, which is $\sim n$. Thus $\Omega_{\rm tm}$ is a Turing-machine-random bit string and therefore a normal real number in BOREL's sense.

The construction of a complex infinite string presented in this section is much better than our original approach in [2, Section 7]. The description in this section is very concise. For more details, see the analogous discussion for LISP in [3, Sections 4, 7 and 9].

3. Proving Lower Bounds on $H_{\rm tm}$

We will measure the complexity of a formal system by the number of states needed for a proof-checker.

We need a Turing machine linkage convention and should only measure the size of Turing machine programs that obey this linkage convention. A good convention is that nothing to the left of the position of the scanner upon entry to a subroutine is ever read or altered by the subroutine. This makes it possible for the calling routine to save information on the tape. One problem with Turing machines is that a subroutine can be called from only one place, because the return address must be "wired" into the subroutine as a fixed state number to return to after finishing.

Theorem: *An n-state formal system cannot prove that a specific bit string has Turing machine complexity $> n + O(\log n)$.*

The "$O(\log n)$" here is because we can run a formal system's proof-checker to get theorems, but the proof-checker does not tell us how many states it has, i.e., what its Turing machine complexity is.[2]

Proof (Original Version): Suppose we are given a n-state Turing machine for checking purported proofs in a formal system. The output is either a message that the proof is incorrect, or else the theorem established by the proof. We assume that the proof-checker is coded in such a way that it can be invoked as a subroutine, i.e., it obeys the linkage conventions we discussed above. We add $\lfloor \log_2 k \rfloor + c$ states to tell us an arbitrary natural number k that we will determine later, and c' more states that keep calling the proof-checker until we find a proof that a specific bit string s has Turing machine state complexity $> k$. Then we output s and halt. This gives us a Turing machine with

$$n + \lfloor \log_2 k \rfloor + c + c'$$

states that produces a string s with Turing machine complexity

$$> k.$$

That is, there is a Turing machine with

$$\leq n + \log_2 k + c + c'$$

states that produces a string s with Turing machine complexity

$$> k.$$

Let us take

$$k = n + \log_2 n + \Delta + c + c',$$

which must be an integer. Thus there is a Turing machine with

$$\leq n + \log_2(n + \log_2 n + \Delta + c + c') + c + c'$$

[2] One could (but we won't) add to our Turing machines a way to write on the tape the current state number. This could then be used to determine the size of a program by subtracting the initial state from the final state. In fact, real computers can get a return address for a subroutine into a register, so that the subroutine knows where to branch after it is finished.

states that produces a string s with Turing machine complexity

$$> n + \log_2 n + \Delta + c + c'.$$

This implies that

$$n + \log_2 n + \Delta + c + c' \ < \ n + \log_2(n + \log_2 n + \Delta + c + c') + c + c'.$$

I.e.,

$$\log_2 n + \Delta < \log_2(n + \log_2 n + \Delta + c + c'),$$

or

$$\Delta < \log_2 \left(1 + \frac{\log_2 n + \Delta + c + c'}{n}\right).$$

We may assume that n is greater than or equal to 1. From this it is easy to see that $(\log_2 n)/n$ is always less than or equal to 1. This implies that

$$\Delta < \log_2 \left(1 + \frac{\log_2 n + \Delta + c + c'}{n}\right) \leq \log_2(1 + 1 + \Delta + c + c').$$

Hence

$$\Delta < \log_2(1 + 1 + \Delta + c + c').$$

Clearly, this last inequality can hold for at most finitely many values of Δ. More precisely, it implies that $\Delta < c''$, where c'' does not depend on the complexity n of the proof-checker. Thus an n-state formal system can establish that a specific bit string has complexity

$$> k = n + \log_2 n + \Delta + c + c'$$

only if $\Delta < c''$. I.e., an n-state formal system cannot establish that a specific bit string has complexity

$$> n + \log_2 n + c'' + c + c'.$$

Q.E.D.

These inequalities are obvious if viewed properly. We are talking about the size of the base-two numeral for n. How much can this grow if we add it to its own size? For all sufficiently large n, this size, which

is $\lfloor \log_2 n \rfloor + 1$, is much less than n. Therefore the number of bits to express this size is much less than the number of bits to express n. (More precisely, the number of bits in the base-two numeral for the number of bits in n is in the limit a vanishing fraction of the number of bits in the base-two numeral for n.) So when the size of the base-two numeral for n is added to n, the base-two numeral for the result will usually be the same size as the base-two numeral for n. At most one bit is added to its size in the unlikely event that overflow occurs (i.e., a carry out of the left-most non-zero bit position).

History: This was my very first information-theoretic incompleteness theorem [4, 5]. The only difference is that here I spell out all the inequalities.

4. Proving Lower Bounds II

Here is an arrangement of the proof that avoids the messy inequalities.

Proof (Improved Version): Suppose we are given an n-state Turing machine for checking purported proofs in a formal system. The output is either a message that the proof is incorrect, or else the theorem established by the proof. We assume that the proof-checker is coded in such a way that it can be invoked as a subroutine. We add $\lfloor \log_2 k \rfloor + c$ states to tell us an arbitrary natural number k that we will determine later, and c' more states that keep calling the proof-checker until we find a proof that a specific bit string s has Turing machine complexity $> k + \lfloor \log_2 k \rfloor + c$. Then we output s and halt.

This gives us a Turing machine with

$$n + \lfloor \log_2 k \rfloor + c + c'$$

states that produces a string s with Turing machine complexity

$$> k + \lfloor \log_2 k \rfloor + c.$$

Taking $k = n + c'$, we have a Turing machine with

$$n + \lfloor \log_2 (n + c') \rfloor + c + c'$$

states that produces a string s with Turing machine complexity

$$> n + c' + \lfloor \log_2 (n + c') \rfloor + c,$$

which is impossible. Thus an n-state formal system cannot establish that a specific bit string has complexity

$$> n + c' + \lfloor \log_2(n + c') \rfloor + c = n + O(\log n).$$

Q.E.D.

It is much easier to formulate this information-theoretic incompleteness theorem in LISP [3, Section 3]. The LISP result is also much sharper.

References

[1] G. J. CHAITIN, "On the length of programs for computing finite binary sequences," *Journal of the ACM* **13** (1966), 547–569.

[2] G. J. CHAITIN, "On the length of programs for computing finite binary sequences: statistical considerations," *Journal of the ACM* **16** (1969), 145–159.

[3] G. J. CHAITIN, "LISP program-size complexity II," *Applied Mathematics and Computation,* in press.

[4] G. J. CHAITIN, "Computational complexity and Gödel's incompleteness theorem," Abstract 70T–E35, *AMS Notices* **17** (1970), 672.

[5] G. J. CHAITIN, "Computational complexity and Gödel's incompleteness theorem," *ACM SIGACT News* **9** (April 1971), 11–12.

BLANK-ENDMARKER PROGRAMS

G. J. Chaitin

Abstract

We review four different versions of the information-theoretic incompleteness theorem asserting that it is difficult to prove that specific strings s have high blank-endmarker complexity $H_{\rm b}(s)$. We also construct a blank-endmarker "halting probability" $\Omega_{\rm b}$ with the property that the initial segments of its base-two expansion asymptotically have the maximum possible blank-endmarker complexity $H_{\rm b}$.

1. Blank-Endmarker Complexity $H_{\rm b}$

Following [1, Appendix], we define the abstract setting.

A computer C is a partial recursive function that maps a program p that is a bit string into an output $C(p)$ that is also an individual bit string. The complexity $H_C(x)$ of the bit string x based on the computer C is the size in bits $|p|$ of the smallest program p for computing x with

This chapter of the survey has not been published elsewhere.

C:

$$H_C(x) = \min_{C(p)=x} |p|.$$

Define a universal computer U as follows:

$$U(p1\overbrace{000\cdots000}^{i\ 0\text{'s}}) = C_i(p).$$

Here C_i is the computer with Gödel number i, i.e., the ith computer. Hence

$$H_U(x) \leq H_{C_i}(x) + (i+1)$$

for all strings x. The general definition of a universal computer U is that it is one that has the property that for each computer C there is a constant sim_C (the cost in bits of **sim**ulating C) such that

$$H_U(x) \leq H_C(x) + \mathrm{sim}_C$$

for all x. The universal computer U we defined above satisfies this definition with $\mathrm{sim}_{C_i} = i+1$. We pick this particular universal computer U as our standard one and define the blank-endmarker complexity $H_{\mathrm{b}}(x)$ to be $H_U(x)$:

$$H_{\mathrm{b}}(x) = H_U(x).$$

2. Discussion

The first thing to do with a new complexity measure is to determine the maximum complexity that an n-bit string can have and whether most n-bit strings have close to this maximum complexity. Consider the identity-function computer C with $C(p) = p$ for all programs p. This shows that

$$H_{\mathrm{b}}(s) \leq |s| + \mathrm{sim}_C$$

for all strings s. Do most n-bit s have complexity close to n? Yes, because there are 2^n n-bit strings s but only

$$1 + 2 + 4 + \cdots + 2^{n-k-1} = \sum_{j<n-k} 2^j = 2^{n-k} - 1 < 2^{n-k}$$

programs p for U of size less than $n-k$.

3. The "Halting Probability" $\Omega_{\rm b}$

In Section 2 we saw that most finite bit strings are complex. How about constructing an infinite bit string with complex initial segments?

Let $\sum_n 2^{-f(n)} \leq 1$, e.g., $f(n) = 2\lceil \log_2 n \rceil + 1 = O(\log n)$.

Define

$$\Omega_{\rm b} = \sum_n \#\{\text{programs } p \text{ of size } n \text{ such that } U(p) \text{ halts}\} 2^{-n-f(n)}.$$

Then $0 \leq \Omega_{\rm b} \leq 1$ because there are 2^n n-bit programs p. I.e.,

$$\Omega_{\rm b} \leq \sum_n 2^n \, 2^{-n-f(n)} = \sum_n 2^{-f(n)} \leq 1.$$

Hence the blank-endmarker complexity of the first $n + f(n)$ bits of $\Omega_{\rm b}$ is greater than or equal to $n - c$.

From this it can be shown that $\Omega_{\rm b}$ is what BOREL called a normal real number.

The description in this section is very concise. For more details, see the analogous discussion for LISP in [2, Sections 4, 7 and 9].

4. Proving Lower Bounds on $H_{\rm b}$

Now let's do some metamathematics using the complexity measure $H_{\rm b}$.

We consider a fixed set of rules of inference F to be a recursively enumerable set of ordered pairs of the form $A \vdash T$ indicating that the theorem T follows from the axiom A. (We may consider the axiom A to be represented as a bit string via some standard binary encoding.)

Theorem A: *Consider a formal system F_A consisting of all theorems T derived from a single axiom $A \leq n$ bits in size by applying to it a fixed set of rules of inference. This formal system F_A of size $\leq n$ cannot prove that a specific string has blank-endmarker complexity $> n + c$.*

Proof: Consider a special-purpose computer C that does the following when given the axiom A of a formal system F_A followed by a 1 and any number of 0's:

$$C(A1\overbrace{000\cdots000}^{k\ 0\text{'s}}) = \begin{cases} \text{The first specific string } s^\star \\ \text{that can be shown in } F_A \text{ to have} \\ \text{complexity} > |A| + 2k + 1. \end{cases}$$

How does C accomplish this? Given the program p, C counts the number k of 0's at the right end of p. It then removes them and the rightmost 1 bit in order to obtain the axiom A. Now C knows A and k. C then searches through all proofs derived from the axiom A in size order, and among those of the same size, in some arbitrary alphabetical order, applying the proof-checking algorithm associated with the fixed rules of inference to each proof in turn. (More abstractly, C enumerates $F_A = \{\, T : A \vdash T \,\}$.) In this manner C determines each theorem T that follows from the axiom A. C examines each T until it finds one of the form

$$\text{“}H_{\rm b}(s^\star) > j\text{”}$$

that asserts that a specific bit string $s^\star$ has blank-endmarker complexity greater than a specific natural number j that is greater than or equal to $|A| + 2k + 1$. C then outputs $s^\star$ and halts. This shows that

$$|A| + 2k + 1 < H_{\rm b}(s^\star) \leq |A1\overbrace{000\cdots000}^{k\ 0\text{'s}}| + {\rm sim}_C.$$

I.e.,

$$|A| + 2k + 1 < H_{\rm b}(s^\star) \leq |A| + 1 + k + {\rm sim}_C.$$

This implies

$$k < {\rm sim}_C.$$

Taking $k = {\rm sim}_C$, we have a contradiction. It follows that $s^\star$ cannot exist for this value of k. The theorem is therefore proved with $c = 2k + 1 = 2{\rm sim}_C + 1$. **Q.E.D.**

Here is a stronger version of Theorem A.

Theorem B: *Consider a formal system F_A consisting of all theorems T derived from a single axiom A with blank-endmarker complexity $\leq n$ by applying to it a fixed set of rules of inference. This formal system F_A of blank-endmarker complexity $\leq n$ cannot prove that a specific string has blank-endmarker complexity $> n + c$.*

Proof: Instead of being given A directly, C is given an $H_{\rm b}(A)$ bit program p_A for U to compute A. I.e., C's program p is now of the form

$$p' = p_A 1\overbrace{000\cdots000}^{k\ 0\text{'s}}$$

instead of

$$p = A1\overbrace{000\cdots000}^{k\ 0\text{'s}}.$$

Here

$$|p_A| = H_{\rm b}(A),$$

and

$$U(p_A) = A.$$

So now we have

$$|p'| = H_{\rm b}(A) + 1 + k$$

instead of

$$|p| = |A| + 1 + k.$$

And now

$$C(p') = s^\star$$

and

$$H_{\rm b}(s^\star) > H_{\rm b}(A) + 2k + 1.$$

Hence we have

$$H_{\rm b}(A) + 2k + 1 < H_{\rm b}(s^\star) \leq H_{\rm b}(A) + 1 + k + {\rm sim}_C.$$

This yields a contradiction for $k = {\rm sim}_C$. **Q.E.D.**

Theorems A and B are sharp; here is the converse.

Theorem C: *There is a formal system F_A with n-bit axioms A that enables us to determine:*

(a) which bit strings have blank-endmarker complexity $\geq n$, and

(b) the exact blank-endmarker complexity of each bit string with blank-endmarker complexity $< n$.

Proof: Here are two axioms A from which we can deduce the desired theorems:

(a) The n-bit string that is the base-two numeral for the number of programs p of size $< n$ that halt.

(b) $p1000\cdots000$, the program p of size $< n$ that takes longest to halt (there may be several) padded on the right to exactly n bits with a 1 bit and the required (possibly zero) number of 0 bits.

Q.E.D.

Here is a corollary of the ideas presented in this section. An $n + O(1)$ bit formal system is necessary and sufficient to settle the halting problem for all programs of size less than n bits. For a detailed proof, see [3, Theorem 4.4].

5. Enumeration Complexity $H_{\rm be}$

Following [3], we extend the formalism that we presented in Section 1 from finite computations with a single output to infinite computations with an infinite amount of output. So let's now consider computers that never halt, which we shall refer to as enumeration computers, or e-computers for short. An e-computer C is given by a computable function that maps the program p and the time t into the finite set of bit strings that is the output $C(p,t)$ of C at time t with program p. The total output $C(p)$ produced by e-computer C when running program p is then defined to be

$$C(p) = \bigcup_{t=0}^{\infty} C(p,t).$$

The complexity $H_C(S)$ of the set S based on the e-computer C is the size in bits $|p|$ of the smallest program p for enumerating S with C:

$$H_C(S) = \min_{C(p)=S} |p|.$$

Define a universal e-computer $U_{\rm e}$ as follows:

$$U_{\rm e}(p1\overbrace{000\cdots000}^{i\ 0\text{'s}}) = C_i(p).$$

Here C_i is the e-computer with Gödel number i, i.e., the ith e-computer. Hence

$$H_{U_{\rm e}}(S) \leq H_{C_i}(S) + (i+1)$$

for all S. The general definition of a universal e-computer U is that it is one that has the property that for each e-computer C there is a constant sim_C (the cost in bits of **sim**ulating C) such that

$$H_U(S) \leq H_C(S) + \mathrm{sim}_C$$

for all S. The universal e-computer U_{e} we defined above satisfies this definition with $\mathrm{sim}_{C_i} = i + 1$. We pick this particular universal e-computer U_{e} as our standard one and define the blank-endmarker e-complexity[1] $H_{\mathrm{be}}(S)$ to be $H_{U_{\mathrm{e}}}(S)$:

$$H_{\mathrm{be}}(S) = H_{U_{\mathrm{e}}}(S).$$

In summary, the blank-endmarker e-complexity $H_{\mathrm{be}}(S)$ of a recursively-enumerable set S is the size in bits $|p|$ of the smallest computer program p that makes our standard universal e-computer U_{e} enumerate the set S.

6. Proving Lower Bounds II

Now we reformulate Theorem B using the concepts of Section 5.

Theorem D: *Consider a formal system consisting of a recursively enumerable set T of theorems. Suppose that $H_{\mathrm{be}}(T) \leq n$. This formal system T of blank-endmarker e-complexity $\leq n$ cannot prove that a specific string has blank-endmarker complexity $> n+c$. More precisely, if a theorem of the form "$H_{\mathrm{b}}(s) > n$" is in T only if it is true, then "$H_{\mathrm{b}}(s) > n$" is in T only if $n < H_{\mathrm{be}}(T) + c$.*

Note that $H_{\mathrm{be}}(T)$ combines the complexity of the axioms and the rules of inference in a single number; it is therefore a more natural and straight-forward measure of the complexity of a formal system than the one that we used in Section 4.

Proof: In the proof of Theorem B, let p_A now be p_T, a minimal-size program for enumerating the set T of theorems. To cut a long story short, this time we have $C(p) = s^\star$ and

$$H_{\mathrm{b}}(s^\star) > H_{\mathrm{be}}(T) + 2k + 1$$

[1] In full, this is the "blank-endmarker enumeration complexity."

with

$$p = p_T 1 \overbrace{000\cdots000}^{k \text{ 0's}}$$

of length

$$|p| = H_{\rm be}(T) + 1 + k.$$

Hence

$$H_{\rm be}(T) + 2k + 1 < H_{\rm b}(s^\star) \leq H_{\rm be}(T) + 1 + k + {\rm sim}_C.$$

This yields a contradiction for $k = {\rm sim}_C$. **Q.E.D.**

7. $H_{\rm b}$ versus $H_{\rm be}$

Note that the blank-endmarker e-complexity of a singleton set $\{x\}$ is essentially the same as the blank-endmarker complexity of the string x:

$$H_{\rm be}(\{x\}) = H_{\rm b}(x) + O(1).$$

Proof: There is a special-purpose e-computer C such that

$$C(p) = \{U(p)\}.$$

(Just forget to halt!) This shows that

$$H_{\rm be}(\{x\}) \leq H_{\rm b}(x) + c.$$

On the other hand, there is a special-purpose computer C' such that

$$C'(p) = \text{the first string output by } U_{\rm e}(p).$$

This shows that

$$H_{\rm b}(x) \leq H_{\rm be}(\{x\}) + c'.$$

Q.E.D.

What about finite sets instead of singleton sets? Well, we can define the blank-endmarker complexity of a finite set S as follows:

$$H_{\rm b}(S) = H_{\rm b}(\sum_{x \in S} 2^x).$$

Here we associate the kth string x with the natural number k, i.e., if the kth string is in the set S, then 2^k is in the sum $\sum_{x \in S} 2^x$. This trickery conceals the essential idea, which is that $H_{\rm b}(S)$ is the minimum number of bits needed to tell our standard computer U how to write out S and halt, whereas $H_{\rm be}(S)$ is the minimum number of bits needed to tell our standard computer $U_{\rm e}$ how to enumerate S but never halt, so that we are never sure if the last element of S has been obtained or not.

Enumerating S is easier than computing S, sometimes **much** easier. More precisely,

$$H_{\rm be}(S) \leq H_{\rm b}(S) + c.$$

And

$$\Phi(n) = \max_{\substack{S \text{ finite} \\ H_{\rm be}(S) \leq n}} H_{\rm b}(S),$$

which measures the extent to which it is easier to enumerate finite sets than to compute them, grows faster than any computable function of n. In fact, it is easy to see that Φ grows at least as fast as the function Θ defined as follows:

$$\Theta(n) = \max_{H_{\rm b}(x) \leq n} x.$$

More precisely, there is a c such that for all n,

$$\Phi(n + c) \geq \Theta(n).$$

Proof Sketch: Consider the set

$$\{0, 1, 2, \ldots, \Theta(\Theta(n))\}$$

of all natural numbers up to $\Theta(\Theta(n))$. This set has blank-endmarker complexity roughly equal to $\Theta(n)$, but its e-complexity is very small. In fact, given the natural number n, one can enumerate this set, which shows that its blank-endmarker e-complexity is $\leq \log_2 n + c$. **Q.E.D.**

Here are three related functions Γ_1, Γ_2 and Γ_3 that also grow very quickly [4]:

$$\Gamma_1(n) = \max_{\substack{S \text{ finite} \\ H_{\rm b}(S) \leq n}} \#S.$$

Here, as in Section 3, $\#S$ is the cardinality of the finite set S.

$$\Gamma_2(n) = \max_{\substack{S \text{ finite} \\ H_{\rm be}(S) \le n}} \#S.$$

Last but not least,

$$\Gamma_3(n) = \max_{\substack{\overline{S} \text{ finite} \\ H_{\rm be}(S) \le n}} \#\overline{S}.$$

Here $\overline{S}$ is the complement of the set S, i.e., the set of all bit strings that are not in S.

8. Proving Lower Bounds III

It is possible to sustain the view that e-complexity is more fundamental than normal complexity.[2] If so, here is how to reformulate Theorem D:

Theorem E: *Consider a formal system consisting of a recursively enumerable set T of theorems. Suppose that $H_{\rm be}(T) \le n$. This formal system T of blank-endmarker e-complexity $\le n$ cannot prove that a specific string, considered as a singleton set, has blank-endmarker e-complexity $> n + c$. More precisely, if a theorem of the form "$H_{\rm be}(\{s\}) > n$" is in T only if it is true, then "$H_{\rm be}(\{s\}) > n$" is in T only if $n < H_{\rm be}(T) + c$.*

References

[1] G. J. CHAITIN, "Information-theoretic computational complexity," *IEEE Transactions on Information Theory* **IT–20** (1974), 10–15.

[2] G. J. CHAITIN, "LISP program-size complexity II," *Applied Mathematics and Computation,* in press.

[3] G. J. CHAITIN, "Information-theoretic limitations of formal systems," *Journal of the ACM* **21** (1974), 403–424.

[2] This view is strongly supported by the incompleteness theorems in [5], which use **self-delimiting** e-complexity instead of blank-endmarker e-complexity.

[4] G. J. CHAITIN, "Program size, oracles, and the jump operation," *Osaka Journal of Mathematics* **14** (1977), 139–149.

[5] G. J. CHAITIN, "Information-theoretic incompleteness," *Applied Mathematics and Computation,* in press.

LISP PROGRAM-SIZE COMPLEXITY[1]

Applied Mathematics and Computation 49 (1992), pp. 79–93

G. J. Chaitin

Abstract

A theory of program-size complexity for something close to real LISP is sketched.

[1]This paper should be called "On the length of programs for computing finite binary sequences in LISP," since it closely follows references [3] to [6]. But that's too long!

1. Introduction

The complexity measure used in algorithmic information theory [1] is somewhat abstract and elusive. Instead of measuring complexity via the size in bits of self-delimiting programs for a universal Turing machine as is done in [1], it would be nice to use instead the size in characters of programs in a real computer programming language.

In my book *Algorithmic Information Theory* [1] I make considerable use of a toy version of pure LISP constructed expressly for theoretical purposes. And in Section 5.1 [1], "Complexity via LISP Expressions," I outline a theory of program-size complexity defined in terms of the size of expressions in this toy LISP. However, this toy LISP is rather different from real LISP; furthermore gruesome estimates of the number of S-expressions of a given size (Appendix B [1]) are required.

One can develop a theory of program-size complexity for something close to real LISP; we sketch such a theory here. It was pointed out in [2] that this is a straightforward application of the methods used to deal with bounded-transfer Turing machines in [3–6]. In fact, the results in this paper closely follow those in my two earliest publications on algorithmic information theory, the two AMS abstracts [3, 4] (also reprinted in [2]), but restated for LISP instead of bounded-transfer Turing machines.

2. Précis of LISP

Our basic LISP reference is [7]. So we have a LISP that includes integers. Otherwise, we pretty much restrict ourselves to the pure LISP subset of LISP 1.5, so there are no side-effects. In addition to the usual LISP primitives `CAR`, `CDR`, `CONS`, `EQ`, `ATOM`, `COND`, `QUOTE`, `LAMBDA`, `NIL` and `T`, we need some more powerful built-in functions, which are described below.

We shall principally be concerned with the size in characters of programs for computing finite binary sequences, i.e., bit strings. The convention we shall adopt for representing bit strings and character strings is as follows. A bit string shall be represented as a list of the integers 0 and 1. Thus the bit string 011 is represented by the 7-

character LISP S-expression (0␣1␣1). Similarly, a character string is represented as a list of integers in the range from 0 to 255, since we assume that each character is a single 8-bit byte. Notation: |bit string| or |character string| denotes the number of bits or characters in the string.

LENGTH

Given a list, this function returns the number of elements in the list (an integer).

FLATTEN

This is easy enough to define, but let's simplify matters by assuming that it is provided. This flattens an S-expression into the list of its successive atoms. Thus

```
(FLATTEN(QUOTE(A(BB(CCC)DD)E)))
```

evaluates to

```
(A BB CCC DD E)
```

EVAL

Provides a way to evaluate an expression that has been constructed, and moreover to do this in a clean environment, that is, with the initial association list.

EVLIS

Evaluates a list of expressions and returns the list of values. Applies `EVAL` to each element of a list and `CONS`'s up the results.

TIME-LIMITED-EVAL

This built-in function provides a way of trying to evaluate an expression for a limited amount of time t. In the limit as t goes to infinity, this

will give the correct value of the expression. But if the expression does not have a value, this provides a way of attempting to evaluate it without running the risk of going into an infinite loop. In other words, `TIME-LIMITED-EVAL` is a total function, whereas normal `EVAL` is a partial function (may be undefined and have no value for some values of its argument.)

(`TIME-LIMITED-EVAL` plays a crucial role in [1], where it is used to compute the halting probability Ω.)

`CHARACTER-STRING`

Writes a LISP S-expression into a list of characters, that is, a list of integers from 0 to 255. In other words, this built-in function converts a LISP S-expression into its print representation.

`S-EXPRESSION`

Reads a LISP S-expression from a list of characters, that is, a list of integers from 0 to 255. In other words, this built-in function converts the print representation of a LISP S-expression into the LISP S-expression. `CHARACTER-STRING` and `S-EXPRESSION` are inverse functions.

These two built-in functions are needed to get access to the individual characters in the print representation of an atom. (In [1] and [9, 10] we program out/define both of these functions and do not require them to be built-in; we can do this because in [1] atoms are only one character long.)

3. Definition of Complexity in LISP

$$H_{\text{LISP}}(x) \equiv \min_{\texttt{EVAL}(e)=x} |e|$$

That is, $H_{\text{LISP}}(x)$ is the size in characters $|e|$ of the smallest S-expression e (there may actually be several such smallest expressions) that evaluates to the S-expression x. We usually omit the subscript LISP.

Here e must only be self-contained LISP expressions without permanent side-effects. In other words, any auxiliary functions used must be defined locally, inside e. Auxiliary function names can be bound to their definitions (`LAMBDA` expression) locally by using `LAMDBA` binding. Here is an example of a self-contained LISP expression, one containing definitions of `APPEND` and `FLATTEN`:[2]

```
((LAMBDA (APPEND FLATTEN)
 (FLATTEN (QUOTE (A (BB (CCC) DD) E))))
 (QUOTE (LAMBDA (X Y) (COND ((ATOM X) Y)
                            (T (CONS (CAR X) (APPEND (CDR X) Y))))))
 (QUOTE (LAMBDA (X) (COND ((ATOM X) (CONS X NIL))
                          ((ATOM (CDR X)) X)
                          (T (APPEND (FLATTEN (CAR X)) (FLATTEN (CDR X))))))))
)
```

The value of this expression is `(A BB CCC DD E)`.

4. Subadditivity of Bit String Complexity

One of the most fundamental properties of the LISP program-size complexity of a bit string is that it is subadditive. That is, the complexity of the result of concatenating the non-null bit strings $s_1, \ldots, s_n$ is bounded by the sum of the individual complexities of these strings. More precisely,

$$H(s_1 \cdots s_n) \leq \sum_{k=1}^{n} H(s_k) + c, \tag{4.1}$$

where the constant c doesn't depend on how many or which strings s_k there are.

Why is this so? Simply, because we need only add c more characters in order to "stitch" together the minimal-size expressions for $s_1, \ldots, s_n$ into an expression for their concatenation. In fact, let $e_1, \ldots, e_n$ be the respective minimal-size expressions for $s_1, \ldots, s_n$. Consider the following LISP expression:

$$\texttt{(FLATTEN(EVLIS(QUOTE(}e_1 \cdots e_n\texttt{))))} \tag{4.2}$$

[2] Although we actually are taking `FLATTEN` to be built-in.

Recall that the built-in function `FLATTEN` flattens an S-expression into the list of its successive atoms, and that `EVLIS` converts a list of expressions into the list of their values. Thus this S-expression (4.2) evaluates to the bit string concatenation of $s_1, \ldots, s_n$ and shows that

$$H(s_1 \cdots s_n) \leq \sum_{k=1}^{n} |e_k| + 25 = \sum_{k=1}^{n} H(s_k) + c.$$

This S-expression (4.2) works because we require that each of the e_k in it must be a self-contained LISP S-expression with no side-effect; that is the definition of the LISP program-size complexity H_{LISP}. This S-expression also works because of the self-delimiting LISP syntax of balanced parentheses. That is, the e_k are separated by their delimiting parentheses in (4.2), and do not require blanks to be added between them as separators.

One small but annoying detail is that blanks **would** have to be added to separate the e_k in (4.2) if any two successive e_k were atoms, which would ruin our inequality (4.1). There is no problem if all of the e_k are lists, because then they all begin and end with parentheses and do not require added blanks. Does our LISP initially bind any atoms to bit strings, which are lists of 0's and 1's? Yes, because although the initial association list only binds `NIL` to `NIL` and `T` to `T`, `NIL` is the null bit string! That is why we stipulate that the bit strings s_k in (4.1) are non-null.

5. A Minimal Expression Tells Us Its Size As Well As Its Value

Symbolically,

$$H(x, H(x)) = H(x) + O(1).$$

That is, suppose that we are given a quoted minimal-size expression e for x. Then of course we can evaluate e using `EVAL` to get x. And we can also convert e into the list of characters that is e's print representation and then count the number of characters in e's print representation. This gives us $|e|$, which is equal to the complexity $H(x)$ of x.

More formally, let e be a minimal-size expression for x. Then the following expression, which is only c characters longer than e, evaluates to the pair consisting of x and $H(x)$:

```
((LAMBDA (X) (CONS (EVAL X)
                   (CONS (LENGTH (CHARACTER-STRING X))
                         NIL)))
 (QUOTE e))
```

Thus

$$H(x, H(x)) \leq H(x) + c.$$

Conversely, let e be a minimal-size expression for the pair consisting of x and $H(x)$. Then the following expression, which is only c' characters longer than e, gives us x:

```
(CAR e)
```

Thus

$$H(x) \leq H(x, H(x)) + c'.$$

The fact that an expression can tell us its size as well as its value will be used in Section 9 to show that most n-bit strings have close to the maximum possible complexity $\max_{|s|=n} H(s)$.

6. Lower Bound on Maximum Complexity

$$\max_{|s|=n} H(s) \geq n/8.$$

To produce each of the 2^n bit strings of length n, we need to evaluate the same number of S-expressions, some of which must therefore have at least $n/8$ characters. This follows immediately from the fact that we are assuming that each character in the LISP character set is a single 8-bit byte 0–255.

7. Upper Bounds on Maximum Complexity

Consider an arbitrary n-bit string s whose successive bits are $b_1, \ldots, b_n$. Consider the LISP expression

$$\texttt{(QUOTE(}b_1\sqcup\ldots\sqcup b_n\texttt{))}$$

Here the individual bits b_i in the bit string s of length n are separated by blanks. The value of this expression is of course the list of bits in s. Since this expression is $2n + 8$ characters long, this shows that $H(s) \leq 2n + 8$. Hence

$$\max_{|s|=n} H(s) \leq 2n + c.$$

Let us do slightly better, and change the coefficient from 2 to 1/2. This can be done by programming out hexadecimal notation. That is, we use the digits from 0 to 9 and the letters from A to F to denote the successive quadruples of bits from 0000 to 1111. Thus, for example,

$$\overbrace{\texttt{((LAMBDA(X)...)}}^{\text{big function definition}}\texttt{(QUOTE(F}\sqcup\texttt{F}\sqcup\texttt{F}\sqcup\texttt{F))}$$

will evaluate to the bit string consisting of sixteen 1's. Hence each group of 4 bits costs us 2 characters, and we have

$$\max_{|s|=n} H(s) \leq n/2 + c',$$

but with a much larger constant c' than before.

Final version: let us compress the hexadecimal notation and eliminate the alternating blanks. Form the name of an atom by appending the hexadecimal digits as a suffix to the prefix `HEX`. (The prefix is required because the name of an atom must begin with a letter, not a digit.) Then we can retrieve the hex digits packed into the name of this atom by using the `CHARACTER-STRING` built-in:

$$\texttt{(}\overbrace{\texttt{(LAMBDA(X)...)}}^{\text{bigger function definition}}\texttt{ (CHARACTER-STRING(QUOTE}\sqcup\texttt{HEXFFFF)))}$$

will again evaluate to the bit string consisting of sixteen 1's. Now each group of 4 bits costs us only one character, and we have

$$\max_{|s|=n} H(s) \leq n/4 + c'',$$

with a constant c'' that is even slightly larger than before.

8. Smoothness of Maximum Complexity

Consider an n-bit string S of maximum complexity. Divide S into $\lfloor n/k \rfloor$ successive nonoverlapping bits strings of length k with one string of length $< k$ left over.

Then, by the subadditivity of bit string complexity (4.1), we have

$$H(S) \leq \lfloor n/k \rfloor \max_{|s|=k} H(s) + \max_{|s|<k} H(s) + c,$$

where c is independent of n and k. Hence

$$\max_{|s|=n} H(s) \leq \lfloor n/k \rfloor \max_{|s|=k} H(s) + c'_k,$$

where the constant c' now depends on k but not on n. Dividing through by n and letting $n \to \infty$, we see that

$$\limsup_{n\to\infty} \frac{1}{n} \max_{|s|=n} H(s) \leq \frac{1}{k} \max_{|s|=k} H(s).$$

However, we already know that $\max_{|s|=n} H(s)/n$ is $\geq 1/8$ and $\leq 1/4 + o(1)$. Hence the limit of $\max_{|s|=n} H(s)/n$ exists and is $\geq 1/8$ and $\leq 1/4$. In summary, $\max_{|s|=n} H(s)$ is asymptotic from above to a real constant, which we shall henceforth denote by β, times n:

$$\max_{|s|=n} H(s) \sim \beta n$$

$$\max_{|s|=n} H(s) \geq \beta n \tag{8.1}$$

$$.125 \leq \beta \leq .250$$

9. Most N-bit Strings Have Close to Maximum Complexity

PADDING LEMMA. We shall need the following "padding lemma": *Any S-expression e_0 of size $\leq k$ can be "padded" into an expression e of size $k+18$ that has exactly the same value as e_0.*

PROOF. Here is how to pad e_0 into e:

$$\texttt{((LAMBDA(X}\sqcup\texttt{Y)Y)}\overbrace{\texttt{999...999}}^{\substack{k+1-|e_0| \\ \text{digit 9's}}}\sqcup e_0\texttt{)}$$

Here there are exactly $k+1-|e_0|$ consecutive digit 9's. The constant 18 is because the size in characters of the minimum padding

$$\texttt{((LAMBDA(X}\sqcup\texttt{Y)Y)9}\sqcup\texttt{)}$$

is precisely 18. Correction: The $\sqcup$ next to e_0 becomes a 9 if e_0 is in ()'s—it stays a blank if e_0 is an atom.

Our main result in this section is that the number of n-bit strings x having complexity $H(x)$ less than or equal to

$$\max_{|s|=n} H(s) - \max_{|s|=k} H(s) - c - 18 \tag{9.1}$$

is less than

$$2^{n-k}. \tag{9.2}$$

Assume on the contrary that the number of n-bit strings x having complexity $H(x)$ less than or equal to (9.1) is greater than or equal to (9.2). From this we shall obtain a contradiction by showing that any n-bit string has complexity less than $\max_{|s|=n} H(s)$.

Here is how to produce an arbitrary n-bit string with a LISP expression of size less than $\max_{|s|=n} H(s)$. The LISP expression is made by putting together two pieces: a quoted expression e that was originally of size less than or equal to (9.1) that has been padded to exactly size

$$\max_{|s|=n} H(s) - \max_{|s|=k} H(s) - c$$

and whose value is an n-bit string, and an expression e' of size

$$\leq \max_{|s|=k} H(s)$$

whose value is an arbitrary k-bit string.

From the quoted expression e we immediately obtain its size

$$|e| = \texttt{LENGTH}(\texttt{CHARACTER-STRING}(e))$$

and its value $\texttt{EVAL}(e)$, as in Section 5. Note that $|e| - 18$ is exactly (9.1). Next we generate all character strings of size $\leq |e| - 18$, and use `S-EXPRESSION` to convert each of these character strings into the corresponding LISP S-expression. Then we use `TIME-LIMITED-EVAL` on each of these S-expressions for longer and longer times, until we find the given n-bit string $\texttt{EVAL}(e)$. Suppose that it is the jth n-bit string that we found to be the value of an S-expression of size $\leq |e| - 18$.

Finally we concatenate the jth bit string of size $n-k$ with the k-bit string $\texttt{EVAL}(e')$ produced by e'. The result is an n-bit string S, which by hypothesis by suitable choice of e and e' can be made to be any one of the 2^n possible n-bit strings, which turns out to be impossible, because it gives a LISP expression that is of size less than $\max_{|s|=n} H(s)$ for the n-bit string S.

Why?

The process that we described above can be programmed in LISP and then carried out by applying it to e and e' as follows:

$$\texttt{(}\overbrace{\texttt{(LAMBDA(X Y)...)}}^{\text{very big function definition}}\ \underbrace{\texttt{(QUOTE }e\texttt{)}}_{\substack{\text{eventually}\\ \text{gives arbitrary}\\ n-k \text{ bit string}}}\ \overbrace{e'}^{\substack{\text{directly}\\ \text{gives arbitrary}\\ k \text{ bit string}}}\ \texttt{)}$$

This LISP S-expression, which evaluates to an arbitrary n-bit string S, is a (large) constant number c' of characters larger than $|e| + |e'|$. Thus

$$H(S) \leq |e| + |e'| + c' \leq \left(\max_{|s|=n} H(s) - \max_{|s|=k} H(s) - c\right) + \left(\max_{|s|=k} H(s)\right) + c'.$$

And so

$$H(S) \leq \max_{|s|=n} H(s) - c + c' < \max_{|s|=n} H(s)$$

if we choose the constant c in the statement of the theorem to be $c' + 1$.

10. Maximum Complexity Strings Are Random

Consider a long n-bit string s in which the relative frequency of 0's differs from $1/2$ by more than ϵ. Then

$$H_{\text{LISP}}(s) \leq \beta H\left(\frac{1}{2} + \epsilon, \frac{1}{2} - \epsilon\right) n + o(n).$$

Here

$$H(p, q) \equiv -p \log_2 p - q \log_2 q,$$

where

$$p + q = 1, \quad p \geq 0, \quad q \geq 0.$$

More generally, let the n-bit string s be divided into $\lfloor n/k \rfloor$ successive k-bit strings with one string of $< k$ bits left over. Let the relative frequency of each of the 2^k possible blocks of k bits be denoted by $p_1, \ldots, p_{2^k}$. Then let k and ϵ be fixed and let n go to infinity. If one particular block of k bits has a relative frequency that differs from 2^{-k} by more than ϵ, then we have

$$H_{\text{LISP}}(s) \leq \beta H\left(\frac{1}{2^k} + \epsilon, \frac{1}{2^k} - \frac{\epsilon}{2^k - 1}, \cdots, \frac{1}{2^k} - \frac{\epsilon}{2^k - 1}\right) \frac{n}{k} + o(n).$$

Here

$$H(p_1, \ldots, p_{2^k}) \equiv -\sum_{i=1}^{2^k} p_i \log_2 p_i,$$

where

$$\sum_{i=1}^{2^k} p_i = 1, \quad p_i \geq 0.$$

The notation may be a little confusing, because we are simultaneously using H for our LISP complexity measure and for the Boltzmann–Shannon entropy H! Note that in the definition of the Boltmann–Shannon H, any occurrences of $0 \log_2 0$ should be replaced by 0.

The Boltzmann–Shannon entropy function achieves its maximum value of $\log_2$ of the number of its arguments if and only if the probability distribution p_i is uniform and all probabilities p_i are equal. It follows that a maximum complexity n-bit string s must in the limit as n goes to infinity have exactly the same relative frequency of each possible successive block of k bits (k fixed).

How does one prove these assertions?

The basic idea is to use a counting argument to compress a bit string s with unequal relative frequencies into a much shorter bit string s'. Then the smoothness of the maximum complexity (8.1) shows that the original string s cannot have had maximum complexity.

For the details, see [5, 6]. Here is a sketch.

Most bit strings have about the same number of 0's and 1's, and also about the same number of each of the 2^k possible successive blocks of k bits. Long strings for which this is not true are extremely unusual (the law of large numbers!), and the more unequal the relative frequencies are, the more unusual it is. Since not many strings have this unusual property, one can compactly specify such a string by saying what is its unusual property, and which of these unusual strings it is. The latter is specified by giving a number, the string's position in the natural enumeration of all the strings with the given unusual property. So it boils down to asking how unusual the property is that the string has. That is, how many strings share its unusual property? To answer this, one needs estimates of probabilities obtained using standard probabilistic techniques; for example, those in [8].

11. Properties of Complexity That Are Corollaries of Randomness

Let's now resume the discussion of Section 8. Consider an n-bit string S of maximum complexity $\max_{|s|=n} H(s)$. Divide S into $\lfloor n/k \rfloor$ successive

k-bit strings with one $< k$ bit string left over. From the preceding discussion, we know that if k is fixed and we let n go to infinity, in the limit each of the 2^k possible successive substrings will occur with equal relative frequency 2^{-k}. Taking into account the subadditivity of bit string complexity (4.1) and the asymptotic lower bound on the maximum complexity (8.1), we see that

$$\beta n \leq H(S) \leq \left(\frac{n}{k}\right) \left(\sum_{|s|=k} H(s)/2^k \right) + o(n).$$

Multiplying through by k, dividing through by n, and letting n go to infinity, we see that

$$\beta k \leq \sum_{|s|=k} H(s)/2^k.$$

This piece of reasoning has a pleasant probabilistic flavor.

We can go a bit farther. The maximum $\max_{|s|=n} H(s)$ cannot be less than the average $\sum_{|s|=n} H(s)/2^n$, which in turn cannot be less than βn. Thus if we can find a single string of k bits with less than the maximum possible complexity, it will follow that the maximum is greater than the average, and thus also greater than βk. This is easy to do, for

$$H(\overbrace{000\cdots0}^{k \text{ 0's}}) \leq O(\log k) < \beta k$$

for large k. Thus we have shown that for all large k,

$$\beta k < \max_{|s|=k} H(s).$$

In fact we can do slightly better if we reconsider that long maximum complexity n-bit string S. For any fixed k, we know that for n large there must be a subsequence 0^k of k consecutive 0's inside S. Let's call everything before that subsequence of k 0's, S', and everything after, S''. Then by subadditivity, we have

$$\begin{array}{rcll} \beta n & < & H(S) & \leq & H(S') + H(0^k) + H(S'') + c \\ & & & \leq & H(S') + O(\log k) + H(S'') \end{array}$$

where

$$|S'| + |S''| = n - k.$$

It follows immediately that

$$\left(\max_{|s|=n} H(s)\right) - \beta n$$

must be unbounded.

12. Conclusion

We see that the methods used in references [3–6] to deal with bounded-transfer Turing machines apply neatly to LISP. It is a source of satisfaction to the author that some ideas in one of his earliest papers, ideas which seemed to apply only to a contrived version of a Turing machine, also apply to the elegant programming language LISP.

It would be interesting to continue this analysis and go beyond the methods of references [3–6]. The Appendix is an initial step in this direction.

Appendix

There is an intimate connection between the rate of growth of $\max_{|s|=n} H(s)$ and the number of $\leq n$-character S-expressions.

Why is this?

First of all, if the number of $\leq n$-character S-expressions is $< 2^k$, then clearly $\max_{|s|=k} H(s) > n$, because there are simply not enough S-expressions to go around.

On the other hand, we can use S-expressions as notations for bit strings, by identifying the jth S-expression with the jth bit string. Here we order all S-expressions and bit strings, first by size, and then within those of the same size, lexicographically.

Using this notation, by the time we get to the 2^{k+1}th S-expression, we will have covered all $\leq k$-bit strings, because there are not that many of them. Thus $\max_{|s|=k} H(s) \leq n + c$ if the number of $\leq n$-character S-expressions is $\geq 2^{k+1}$. Here c is the number of characters that must be added to the jth S-expression to obtain an expression whose value is the jth bit string.

So the key to further progress is to study how smoothly the number of S-expressions of size n varies as a function of n; see Appendix B [1] for an example of such an analysis.

One thing that limits the growth of the number of S-expressions of size n, is "synonyms," different strings of characters that denote the same S-expression. For example, `()` and `NIL` denote the same S-expression, and there is no difference between `(A␣B)`, `(A␣␣B)` and `(A␣␣B␣)`. Surprisingly, the fact that parentheses have to balance **does not** significantly limit the multiplicative growth of possibilities, as is shown in Appendix B [1].

References

1 G. J. Chaitin, *Algorithmic Information Theory,* Cambridge University Press, 1987.

2 G. J. Chaitin, *Information, Randomness & Incompleteness—Papers on Algorithmic Information Theory,* Second ed., World Scientific, 1990.

3 G. J. Chaitin, On the length of programs for computing finite binary sequences by bounded-transfer Turing machines, Abstract 66T–26, *AMS Notices* 13:133 (1966).

4 G. J. Chaitin, On the length of programs for computing finite binary sequences by bounded-transfer Turing machines II, Abstract 631–6, *AMS Notices* 13:228–229 (1966).

5 G. J. Chaitin, On the length of programs for computing finite binary sequences, *Journal of the ACM* 13:547–569 (1966).

6 G. J. Chaitin, On the length of programs for computing finite binary sequences: statistical considerations, *Journal of the ACM* 16:145–159 (1969).

7 J. McCarthy et al., *LISP 1.5 Programmer's Manual,* MIT Press, 1962.

8 W. FELLER, *An Introduction to Probability Theory and Its Applications I,* Wiley, 1964.

9 G. J. CHAITIN, *LISP for Algorithmic Information Theory in C,* 1990.

10 G. J. CHAITIN, *LISP for Algorithmic Information Theory in SETL2,* 1991.

LISP PROGRAM-SIZE COMPLEXITY II

Applied Mathematics and Computation 52 (1992), pp. 103–126

G. J. Chaitin

Abstract

We present the information-theoretic incompleteness theorems that arise in a theory of program-size complexity based on something close to real LISP. The complexity of a formal axiomatic system is defined to be the minimum size in characters of a LISP definition of the proof-checking function associated with the formal system. Using this concrete and easy to understand definition, we show (a) that it is difficult to exhibit complex S-expressions, and (b) that it is difficult to determine the bits of the LISP halting probability $\Omega_{\rm LISP}$*. We also construct improved versions* $\Omega'_{\rm LISP}$ *and* $\Omega''_{\rm LISP}$ *of the LISP halting probability that asymptotically have maximum possible LISP complexity.*

1. Introduction

The main incompleteness theorems of my *Algorithmic Information Theory* monograph [1] are reformulated and proved here using a concrete and easy-to-understand definition of the complexity of a formal axiomatic system based on something close to real LISP [2]. This paper is the sequel to [3], and develops the incompleteness results associated with the theory of LISP program-size complexity presented in [3]. Further papers in this series shall study (1) a parenthesis-free version of LISP, and (2) a character-string oriented version of LISP in which the naturally occurring LISP halting probability asymptotically has maximum possible LISP complexity.

In [4] I present the latest versions of my information-theoretic incompleteness theorems; there the complexity of a formal system is defined in terms of the program-size complexity of enumerating its infinite set of theorems. Here the goal has been to make these incompleteness results accessible to the widest possible public, by formulating them as concretely and in as straight-forward a manner as possible. Instead of the abstract program-size complexity measure used in [4], here we look at the size in characters of programs in what is essentially real LISP. The price we pay is that our results are weaker (but much easier to prove and understand) than those in [4].

This paper may also be contrasted with the chapter on LISP in my monograph [1]. There I use a toy version of LISP in which identifiers (atoms) are only one character long. In future papers I shall present two LISP dialects that allow multiple-character LISP atoms, but which share some of the desirable features of the toy LISP in [1].

From a technical point of view, Sections 7 and 8 are of special interest. There two artificial LISP halting probabilities $\Omega'_{\rm LISP}$ and $\Omega''_{\rm LISP}$ are constructed that asymptotically have maximum possible LISP complexity.

At this point it is appropriate to recall LEVIN's delightful and unjustly forgotten book [5] on LISP and metamathematics. (LEVIN was one of the coauthors of the original LISP manual [2].)

Before proceeding, let's summarize the results of [3]. Consider an n-bit string s. Its maximum possible LISP complexity is asymptotic to

a real constant β times n:

$$\max_{|s|=n} H_{\text{LISP}}(s) \sim \beta n.$$

Furthermore, most n-bit strings s have close to this maximum possible LISP complexity; such bit strings are "random." For example, if $H_{\text{LISP}}(s_n) \sim \beta |s_n|$, then as $n \to \infty$ the ratio of the number of 0's to the number of 1's in the bit string s_n tends to the limit unity.

2. Minimal LISP Expressions

Motivation: Imagine a LISP programming class. Suppose the class is given the following homework assignment: find a LISP S-expression for the list of primes less than a thousand. The students might well compete to find the cleverest solution, the smallest LISP S-expression for this list of prime numbers. But one can never be sure that one has found the best solution! As we shall see, here one is up against fundamental limitations on the power of mathematical reasoning! It is this easy to get in trouble!

Consider a minimal-size LISP S-expression p. I.e., p has the value x and no S-expression smaller than p has the same value x. Also, let q be a minimal-size S-expression for p. I.e., the value of q is p and no expression smaller than q has p as value.

$$q \overset{\text{yields value}}{\longrightarrow} p \overset{\text{yields value}}{\longrightarrow} x.$$

Consider the following LISP expression:

```
(quote p)
```

This expression evaluates to p. This shows that

$$H_{\text{LISP}}(p) \leq |p| + 8.$$

Consider the following LISP expression:

```
(eval q)
```

This expression evaluates to x. This shows that

$$|p| = H_{\rm LISP}(x) \leq |q| + 7 = H_{\rm LISP}(p) + 7.$$

Hence

$$H_{\rm LISP}(p) \geq |p| - 7.$$

In summary,

Theorem A: *If p is a minimal expression, it follows that*

$$\Big| H_{\rm LISP}(p) - |p| \Big| \leq 8.$$

Hence the assertion that p is a minimal LISP S-expression is also an assertion about p's LISP complexity. If one can prove that p is a minimal LISP S-expression, one has also shown that $H_{\rm LISP}(p) \geq |p| - 7$.

Anticipation: Where do we go from here? Minimal LISP S-expressions illustrate perfectly the ideas explored in the next section, Section 3.

The following result is a corollary of my fundamental theorem on the difficulty of establishing lower bounds on complexity (Theorem C in Section 3): *A formal system with LISP complexity n can only enable one to prove that a LISP S-expression p is minimal if p's size is $< n + c$ characters.* This is only possible for finitely many p, because there are only finitely many expressions (minimal or not) of size $< n + c$.

Conversely by Theorem D in Section 3, *there are formal systems that have LISP complexity $< n + c'$ in which one can determine each minimal LISP expression p up to n characters in size.* (Basically, the axiom that one needs to know is either the LISP S-expression of size $\leq n$ that takes longest to halt, or the number of LISP S-expressions of size $\leq n$ that halt.)

The details and proofs of these assertions are in Section 3.

3. Exhibiting Complex S-Expressions

Recall the standard LISP convention of having a function return `nil` to indicate "no value"; otherwise the return value is the real value wrapped in a pair of parentheses.

We define an n-character formal system to be an n-character **self-contained** LISP function f that given a purported proof p returns `nil` if the proof is invalid and that returns `(t)` where t is the theorem proved if the proof is valid. I.e., $f(p)$ is always defined and

$$f(p) = \begin{cases} \texttt{nil} & \text{if } p \text{ is an invalid proof;} \\ \texttt{(t)} & \text{if } p \text{ is a valid proof.} \end{cases}$$

Here is an example of a self-contained function definition:

```
(lambda
   (real-argument-of-function)
   (  (lambda
         (main-function
          auxiliary-function-1
          auxiliary-function-2
          auxiliary-function-3
         )
         (main-function real-argument-of-function)
      )
      (quote definition-of-main-function)
      (quote definition-of-auxiliary-function-1)
      (quote definition-of-auxiliary-function-2)
      (quote definition-of-auxiliary-function-3)
   )
)
```

We are interested in the theorems of the following form:

```
(is-greater-than
 (lisp-complexity-of (quote x))
 999999
)
```

This is the LISP notation for

$$H_{\text{LISP}}(x) > 999999.$$

The following theorem works because if we are given a LISP program we can determine its size as well as run it.

Theorem B: *An n-character formal system cannot prove that a specific S-expression has LISP complexity $> n + c$ characters.*

Proof: Suppose we are given a proof-checking algorithm q for a formal system. This is a LISP function of one argument, the putative proof. This function q must always return a value, either `nil` signifying that the proof is incorrect, or a list `(t)` consisting of a single element t, the theorem established by the proof. Given a quoted expression `(quote q)` for the definition of this proof-checking function q, we can do the following two things with it:

1. We can determine the size in characters s of the proof-checker q by converting the S-expression q to a character string[1] and then counting the number of elements in the resulting list of characters.[2] The LISP for doing this is:

$$s = \texttt{(length(character-string q))}$$

2. We can use the proof-checker q to check purported proofs p by forming the expression `(q (quote p))` and then evaluating this expression in a clean environment. The LISP for doing this is:

```
(eval (cons q
      (cons (cons (quote quote)
            (cons p
                  nil))
            nil))
)
```

So we try the given s-character proof checker q on each possible proof p until we find a valid proof p that

[1]The LISP interpreter has to be able to convert S-expressions into character strings in order to print them out. So it might as well make the resulting character strings available internally as well as externally, via a `character-string` built-in function. Characters are integers in the range from 0 to $\alpha - 1$, where α is the size of the LISP alphabet, including the two parentheses and the blank.

[2]Determining the length of a character string, which is just a list of integers, is easily programmed if it is not provided as a built-in function: `(lambda (list) ((lambda (length) (length list)) (quote (lambda (list) (cond ((atom list) 0) (t (plus 1 (length (cdr list))) ))) ))).`

```
(is-greater-than
 (lisp-complexity-of (quote x))
 n
)
```

where the numeral n is $\geq s + k$. I.e., we are searching for a proof that

$$H_{\text{LISP}}(x) > s + k$$

for a specific S-expression x. Then we output the LISP S-expression x and halt. The computation that we have just described in words can be formulated as a LISP expression

```
((lambda(proof-checker lower-bound)...)
 (quote (lambda(purported-proof)...))      k
)
```

(s characters; $\lfloor \log_{10} k \rfloor + 1$ digits)

whose size is $s + \lfloor \log_{10} k \rfloor + c'$ that evaluates to an S-expression x whose LISP character complexity is $> s + k$. Hence

$$s + \lfloor \log_{10} k \rfloor + c' > s + k.$$

This yields a contradiction for a fixed choice c of k that depends only on c' and not on the particular formal proof checker that we are given. **Q.E.D.**

Above we consider an n-character proof-checker or formal system. What if we consider instead a proof-checker whose LISP complexity is n characters? In fact, a slight modification of the above proof shows that

Theorem C: *A formal system with a LISP complexity of n characters cannot prove that a specific S-expression has LISP complexity $> n + c$ characters.*

Proof: The only change is that the very big LISP expression con-

sidered above now becomes:

```
((lambda(proof-checker lower-bound)...)
 (...)
```
minimal-size LISP expression that evaluates to the proof-checker lambda-expression
$\underbrace{k}$
$\lfloor \log_{10} k \rfloor + 1$ digits
```
)
```

I.e., the proof-checker is no longer given as a quoted expression, but is itself computed. **Q.E.D.**

This theorem is sharp; here is the converse.

Theorem D: *There is a formal system with LISP complexity* $< n+c$ *that enables us to determine:*

(a) which LISP S-expressions have LISP complexity $\geq n$,[3] *and*

(b) the exact LISP complexity of each LISP S-expression with LISP complexity $< n$.[4]

Proof: Here are two axioms packed full of information from which we can deduce the desired theorems:

1. Being given the $\leq n$ character LISP S-expression that halts and takes longest to do so (padded to size n).

2. Being given the kth $\leq n$ character LISP S-expression.[5] Here k is the number of $\leq n$ character LISP S-expressions that halt. (Here again, this S-expression must be padded to a fixed size of n characters.)

Here is how to do the padding:

```
(quote((...)xxxxxx))
```

[3] There are infinitely many S-expressions with this property.

[4] There are only finitely many S-expressions with this property.

[5] Pick a fixed ordering of all S-expressions, first by size, then alphabetically among S-expressions of the same size.

The three dots are where the S-expression to be padded is inserted. The `x`'s are the padding. This scheme pads an S-expression e that is $\leq n$ characters long into one `((e)xxxxxx)` that is exactly $n+4$ characters long. (The 4 is the number of added parentheses that glue the expression e to its padding `xxxxxx`.) To retrieve the original expression one takes the `CAR` of the `CAR`, i.e., the first element of the first element. To determine n, one converts the padded S-expression into the corresponding character string, one counts the number of characters in it, and one subtracts 4. **Q.E.D.**

4. The Halting Probability Ω_{LISP}

The first step in constructing an expression for a halting probability for real LISP, is to throw out all atomic S-expressions, S-expressions like `harold`, `big-atom`, etc. Anyway, most atomic S-expressions fail to halt in the sense that they fail to have a value. (Of course, this is when they are considered as self-contained S-expressions, not when they are encountered while evaluating a much larger S-expression with lambda-expressions and bindings.) The usual exceptions are the logical constants `t` and `nil`, which evaluate to themselves.[6] At any rate, the purpose of a halting probability is to help us to decide which S-expressions halt, i.e., have a value. But we don't need any help to decide if an atomic S-expression has a value; this is trivial to do. So let's forget about atomic S-expressions for the moment.

Let's look at all non-atomic S-expressions e, in other words, at S-expressions of the form `(...)`. None of these is an extension of another, because the `()`'s must balance and therefore enable us to decide where a non-atomic S-expression finishes. In other words, non-atomic LISP S-expressions have the vital property that they are what is referred to as "self-delimiting." In a moment we shall show that this self-delimiting property enables us to define a LISP halting probability as follows:

[6] In the LISP dialect in [1], an unbound atom will evaluate to itself, i.e., act as if it were a constant.

$$\begin{aligned}\Omega_{\text{LISP}} &= \sum_{\substack{\text{S-expression } e \\ \text{“halts,”} \\ \text{is defined,} \\ \text{has a value.}}} \alpha^{-[\text{size in characters of S-expression } e]} \\ &= \sum_{(x) \text{ has a value.}} \alpha^{-|(x)|}.\end{aligned}$$

Here α is the number of characters in the LISP alphabet and is assumed to be a power of two. The S-expressions e included in the above sum must not be atomic, **and they must all be different.** If two expressions e and e' differ only in that one contains redundant blanks, then only the one without the redundant blanks is included in the above sum. Similarly, `(())` and `(nil)` are equivalent S-expressions, and are only included once in the sum for Ω_{LISP}. This is a straight-forward definition of a LISP halting probability; we shall see in Sections 7 and 8 that it can be improved.

Ω_{LISP} is also considered to be an infinite bit string, the base-two representation of Ω_{LISP}. It is important to pick the base-two representation **with an infinite number of 1s.** I.e., if it should end with 100000..., pick 011111... instead.[7]

It is crucial that the sum for Ω_{LISP} converges; in fact we have

$$0 < \Omega_{\text{LISP}} < 1.$$

Why is this? The basic reason is that non-atomic LISP S-expressions are self-delimiting because their parentheses must balance. Thus no extension of a non-atomic S-expression is a valid non-atomic S-expression. Extra blanks at the end of an S-expression e are not allowed in the sum for Ω_{LISP}!

Here is a geometrical proof that this works. Associate S-expressions with subsets of the interval of unit length consisting of all real numbers r between zero and one. The S-expression e is associated with all those

[7] We shall see that Ω_{LISP} is highly uncomputable and therefore irrational, so this can't actually occur, but **we don't know that yet!**

real numbers r having that string at the beginning of the fractional part of r's base-α representation:

$$e \overset{\text{is associated with}}{\longleftrightarrow} \left\{ \text{the set of all reals of the form } .e\cdots \text{ in radix } \alpha \text{ notation} \right\}.$$

Then the length of the interval associated with an S-expression e is precisely its probability $\alpha^{-|e|}$. That no extension of a non-atomic S-expression is a valid non-atomic S-expression means that the intervals associated with non-atomic S-expressions do not overlap. Hence the sum of the lengths of these non-overlapping intervals must be less than unity, since they are all inside an interval of unit length. In other words, the total probability that a string of characters picked at random from an alphabet with α characters is a non-atomic S-expression is less than unity, and from these we select those that halt, i.e., evaluate to a value.

Another crucial property of Ω_{LISP} (and of the two halting probabilities Ω'_{LISP} and Ω''_{LISP} that we will construct in Sections 7 and 8) is that it can be calculated in the limit from below. More precisely, Ω_{LISP} can be obtained as the limit from below of a computable monotone increasing sequence[8] of dyadic rational numbers[9] Ω_l:

$$\Omega_0 \le \Omega_1 \le \Omega_2 \le \cdots \le \Omega_{l-1} \le \Omega_l \to \Omega_{\text{LISP}}.$$

This is the case because the set of all S-expressions that halt, i.e., that have a LISP value, is recursively enumerable. In other words, we can eventually discover all S-expressions that halt.

Assume one is given the first $n \log_2 \alpha$ bits of Ω_{LISP}. One then starts to enumerate the set of all S-expressions that halt. As soon as one discovers enough of them to account for the first $n \log_2 \alpha$ bits of Ω_{LISP}, one knows that one has all $\le n$ character non-atomic S-expressions that halt. And there is a trivial algorithm for deciding which $\le n$ character atomic S-expressions halt. One then calculates the set of all the values of $\le n$ character S-expressions that halt, and picks an arbitrary S-expression that is not in this set of values. The result is an S-expression with LISP complexity $> n$. Hence the string of the first $n \log_2 \alpha$ bits of Ω_{LISP} must itself have LISP complexity $> n - c$.

[8] I.e., nondecreasing sequence.

[9] I.e., rationals of the form $i/2^j$.

For more details about the process for using a halting probability to solve the halting problem, see the chapter "Chaitin's Omega" in GARDNER [6], or see [1].

In summary, if one knows the first $n \log_2 \alpha$ bits of $\Omega_{\rm LISP}$, one can determine each LISP S-expression with LISP complexity $\leq n$ characters, and can then produce a LISP S-expression with LISP complexity $> n$ characters. Thus

Theorem E: *The string consisting of the first $n \log_2 \alpha$ bits of $\Omega_{\rm LISP}$ has LISP complexity $> n - c$ characters.*

Using our standard technique for showing that it is difficult to exhibit complex S-expressions (Section 3), it follows immediately that

Theorem F: *To be able to prove what are the values of the first $n \log_2 \alpha$ bits of $\Omega_{\rm LISP}$ requires a formal system with LISP complexity $> n - c$ characters.*

Proof: Suppose we are given a proof-checking algorithm for a formal system. This is a LISP function of one argument, the putative proof. This function must always return a value, either `nil` signifying that the proof is incorrect, or a list `(t)` consisting of a single element t, the theorem established by the proof. Given a quoted expression for the definition of this proof-checking function, we both know its size in characters s, and we can use it to check purported proofs. So we try it on each possible proof until we find a proof that "The first $(s + k) \log_2 \alpha$ bits of $\Omega_{\rm LISP}$ are ..." This would give us a LISP expression with $s + \lfloor \log_{10} k \rfloor + c'$ characters that evaluates to something with LISP complexity $> s + k - c''$ characters. This yields a contradiction for a fixed choice of k that depends only on c' and c'' and not on the particular formal proof checker that we are given. **Q.E.D.**

5. Diophantine Equations for $\Omega_{\rm LISP}$

Now let's convert this incompleteness theorem (Theorem F) into one about diophantine equations.

We arithmetize $\Omega_{\rm LISP}$ in two diophantine equations: one polynomial [7], the other exponential [8]. As we pointed out in Section 4, $\Omega_{\rm LISP}$ can be obtained as the limit from below of a computable monotone

increasing sequence of dyadic rational numbers Ω_l:

$$\Omega_0 \leq \Omega_1 \leq \Omega_2 \leq \cdots \leq \Omega_{l-1} \leq \Omega_l \rightarrow \Omega_{\text{LISP}}.$$

The methods of JONES and MATIJASEVIČ [7, 8, 26] enable one to construct the following:

D_1: A diophantine equation

$$P(k, l, x_1, x_2, x_3, \ldots) = 0$$

that has one or more solutions if the kth bit of Ω_l is a 1, and that has no solutions if the kth bit of Ω_l is a 0.

D_2: An exponential diophantine equation

$$L(k, l, x_2, x_3, \ldots) = R(k, l, x_2, x_3, \ldots)$$

that has exactly one solution if the kth bit of Ω_l is a 1, and that has no solutions if the kth bit of Ω_l is a 0.

Since in the limit of large l the kth bit of Ω_l becomes and remains correct, i.e., identical to the kth bit of Ω_{LISP}, it follows immediately that:

P_1: There are infinitely many values of l for which the diophantine equation

$$P(k, l, x_1, x_2, x_3, \ldots) = 0$$

has a solution iff the kth bit of Ω_{LISP} is a 1.

P_2: The exponential diophantine equation

$$L(k, x_1, x_2, x_3, \ldots) = R(k, x_1, x_2, x_3, \ldots)$$

has infinitely many solutions iff the kth bit of Ω_{LISP} is a 1.

Consider the following questions:

Q_1: For a given value of k, are there infinitely many values of l for which the diophantine equation

$$P(k, l, x_1, x_2, x_3, \ldots) = 0$$

has a solution?

Q_2: For a given value of k, does the exponential diophantine equation

$$L(k, x_1, x_2, x_3, \ldots) = R(k, x_1, x_2, x_3, \ldots)$$

have infinitely many solutions?

As we have seen in Theorem F, to answer the first $n \log_2 \alpha$ of either of these questions requires a formal system with LISP complexity $> n - c$ characters.

In a more abstract setting [4], with diophantine equations constructed from a different halting probability, we can do a lot better. There answering **any** n of these questions requires a formal system whose set of theorems has enumeration complexity $> n - c$ bits.

In Sections 7 and 8 we construct more artificial versions of $\Omega_{\rm LISP}$, $\Omega'_{\rm LISP}$ and $\Omega''_{\rm LISP}$, for which we can show that the LISP complexity of the first n bits is asymptotically the maximum possible, βn characters. By using the method presented in this section, we will automatically get from $\Omega'_{\rm LISP}$ and $\Omega''_{\rm LISP}$ new versions of the diophantine equations D_1 and D_2, new versions of the questions Q_1 and Q_2, and new versions of the corresponding incompleteness theorems. But before diving into these more technical matters, it is a good idea to step back and take a look at what has been accomplished so far.

6. Discussion

The spirit of the results in Section 3 (Theorems B and C) is often expressed as follows:

> "A set of axioms of complexity N cannot yield a theorem of complexity [substantially] greater than N."

This way of describing the situation originated in the introductory discussion of my paper [9]:

> "The approach of this paper... is to measure the power of a set of axioms, to measure the information that it contains. We shall see that there are circumstances in which one only gets out of a set of axioms what one puts in, and in which

> it is possible to reason in the following manner. If a set of theorems constitutes t bits of information, and a set of axioms contains less than t bits of information, then it is impossible to deduce these theorems from these axioms."

This heuristic principle is basically correct, about as correct as any informal explanation of a technical mathematical result can be. But it is useful to point out its limitations.

In fact, **any** set of axioms that yields an infinite set of theorems **must** yield theorems with arbitrarily high complexity! This is true for the trivial reason that there are only finitely many objects of any given complexity. And it is easy to give natural examples. For example, consider the trivial theorems of the form

$$\text{"}N + 1 = 1 + N\text{"}$$

in which the numeral N is, if converted to base-two, a large random bit string, i.e., one with LISP complexity $\sim \beta \log_2 N$. (This will be the case for most large integers N.) This theorem has, if considered as a character string, essentially the same arbitrarily large complexity that the number N has.

So what is to become of our heuristic principle that

> "A set of axioms of complexity N cannot yield a theorem of complexity substantially greater than N" ???

An improved version of this heuristic principle, which is not really any less powerful than the original one, is this:

> "One cannot prove a theorem from a set of axioms that is of greater complexity than the axioms **and know** that one has done this. I.e., one **cannot realize** that a theorem is of substantially greater complexity than the axioms from which it has been deduced, if this should happen to be the case."

Thus even though most large integers N are random bit strings in base-two and yield arbitrarily complex theorems of the form

$$\text{"}N + 1 = 1 + N\text{"},$$

we can never tell **which** N are random and achieve this!

Note that in Section 4 we encountered no difficulty in using our heuristic principle to restrict the ability of formal systems to prove what the value of $\Omega_{\rm LISP}$ is; our general method applies quite naturally in this particular case (Theorem F). This example of the application of our heuristic principle shows that the power of this principle is not restricted by the fact that it really only prevents us from proving theorems that are more complex than our axioms if we can realize that the theorems would be more complex than our axioms are.

Perhaps it is better to avoid all these problems and discussions by rephrasing our fundamental principle in the following totally unobjectionable form:

> "A set of axioms of complexity N cannot yield a theorem that asserts that a specific object is of complexity substantially greater than N."

It was removing the words "asserts that a specific object" that yielded the slightly overly-simplified version of the principle that we discussed above:

> "A set of axioms of complexity N cannot yield a theorem that [asserts that a specific object] is of complexity substantially greater than N."

7. A Second "Halting Probability" $\Omega'_{\rm LISP}$

We shall now "normalize" $\Omega_{\rm LISP}$ and make its information content "more dense." The new halting probability $\Omega'_{\rm LISP}$ in this section has a simple definition, but the proof that it works is delicate. The halting probability in Section 8, $\Omega''_{\rm LISP}$, has a more complicated definition, but it is much easier to see that it works.

Let's pick a total recursive function f such that $\sum 2^{-f(n)} \leq 1$ and $f(n) = O(\log n)$. For example, let $f(n) = 2\lceil \log_2 n \rceil + 1$. This works, because

$$\sum_{n=1}^{\infty} 2^{-2\lceil \log_2 n \rceil - 1} \leq \frac{1}{2} \sum_{n=1}^{\infty} 2^{-2\log_2 n} = \frac{1}{2} \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2/6}{2} \approx \frac{1.644934}{2} < 1.$$

Here we have used the fact discovered by EULER[10] that $1+1/4+1/9+1/16+1/25+\cdots=\pi^2/6$.

Define a new LISP "halting probability" between zero and one as follows:

$$\Omega'_{\rm LISP} = \sum_{n=1}^{\infty} \left[\frac{\text{\# of different} \le n \text{ character LISP S-expressions that halt}}{2^{\lceil \log_2(\text{total \# of different} \le n \text{ character LISP S-expressions}) \rceil}} \right] 2^{-2\lceil \log_2 n \rceil - 1}.$$

The factor of $2^{-2\lceil \log_2 n \rceil - 1}$ is in order to insure convergence. The denominator of (total # of different $\le n$ character LISP S-expressions), or S_n, is increased to the next highest power of two, $2^{\lceil \log_2 S_n \rceil}$, so that we are doing very straightforward binary arithmetic to calculate $\Omega'_{\rm LISP}$.

Why is the LISP complexity of the string of the first n bits of $\Omega'_{\rm LISP}$ asymptotic to the maximum possible, βn?

The main reason is this: Consider all n-bit strings s. From [3] we know that

$$\max_{|s|=n} H_{\rm LISP}(s) \sim \beta n.$$

And below we shall show that it is also the case that

$$n \sim \log_2(\text{\# of different} \le \beta n \text{ character LISP S-expressions}).$$

(This is somewhat delicate.)

So just about when the expression for the denominator in $\Omega'_{\rm LISP}$,

$$\text{“}2^{\lceil \log_2(\text{total \# of different} \le \beta n \text{ character LISP S-expressions}) \rceil}\text{,”}$$

hits 2^n, the numerator,

$$\text{“\# of different} \le \beta n \text{ character LISP S-expressions that halt,”}$$

will include all minimal LISP expressions for n-bit strings. Thus knowing a string consisting of the first $n+o(n)$ bits of $\Omega'_{\rm LISP}$ tells us the LISP complexity of each $\le n$ bit string. Hence the LISP

[10] For EULER's proof that $1+1/4+1/9+1/16+1/25+\cdots=\pi^2/6$, see Section 6 in Chapter II of the first volume of POLYA [10]. (Also see the exercises at the end of Chapter II.)

complexity of the string of the first n bits of $\Omega'_{\rm LISP}$ is asymptotic to $\max_{|s|=n} H_{\rm LISP}(s) \sim \beta n$.

It remains for us to establish the asymptotic expression for the logarithm of S_n, the total number of different $\leq n$ character LISP S-expressions. Let S'_n be the number of different **non-atomic** $\leq n$ character LISP S-expressions. Consider an S-expression that is a list of n elements, each of which is a $\leq k$ character non-atomic S-expression. This shows that we have

$$S'_{nk+2} \geq (S'_k)^n.$$

(The 2 is for the two enclosing parentheses that must be added.) Hence

$$\log_2 S'_{nk+2} \geq n \log_2 S'_k.$$

Dividing through by nk we see that

$$\frac{\log_2 S'_{nk+2}}{nk} \geq \frac{\log_2 S'_k}{k}.$$

From this it is easy to see that

$$\liminf_{n\to\infty} \frac{\log_2 S'_n}{n} \geq \frac{\log_2 S'_k}{k}.$$

On the other hand, the **total** number S_n of different $\leq n$ character LISP S-expressions satisfies:

$$n \log_2 \alpha \geq \log_2 S_n > \log_2 S'_n.$$

Thus $(\log_2 S'_n)/n$ tends to the limit $\gamma \leq \log_2 \alpha$ from below, where

$$\gamma = \sup_{n\to\infty} \frac{\log_2 S'_n}{n} \leq \log_2 \alpha.$$

Furthermore, $\gamma \neq 0$. This can be seen by considering those S-expressions that are a list `(999...999)` containing a single "bignum" or large integer. (For such S-expressions to be different, the first digit must not be 0.) This shows that $9 \times 10^n \leq S'_{n+3}$, and thus $\gamma \geq \log_2 10 > 0$. So the limit γ of $(\log_2 S'_n)/n$ is not equal to zero, and thus $\log_2 S'_n$ is asymptotic to γn:

$$\log_2 S'_n \sim \gamma n.$$

Consider an S-expression that is a list (...) with one element, which may be an atom. This shows that $S_n \leq S'_{n+2}$. On the other hand, $S'_n \leq S_n$, because each S-expression included in S'_n is also included in S_n. Thus we see that

$$S'_n \leq S_n \leq S'_{n+2}.$$

Since $\log_2 S'_n$ is asymptotic to γn, it follows from this inequality that $\log_2 S_n$ is also asymptotic to γn.

So we have shown that

$$\gamma n \sim \log_2(\#\text{ of different} \leq n \text{ character LISP S-expressions})$$

and therefore

$$n \sim \log_2(\#\text{ of different} \leq n/\gamma \text{ character LISP S-expressions}).$$

We finish by showing that $\beta = 1/\gamma$ by using reasoning from the Appendix of [3]. Order the LISP S-expressions, first by their size in characters, and among those of the same size, in an arbitrary alphabetical order.

To get all n-bit strings, one needs to evaluate at least 2^n different LISP S-expressions. Thus if $S_m < 2^n$, then there is an n-bit string s with LISP complexity greater than m. It follows that $\max_{|s|=n} H_{\text{LISP}}(s) > n/\gamma + o(n)$.

On the other hand, we can use the kth S-expression as a notation to represent the kth bit string. This gives us all n-bit strings by the time k reaches 2^{n+1}. And we have to add c characters to indicate how to convert the kth S-expression into the kth bit string. Thus if $S_m \geq 2^{n+1}$, then all n-bit strings s have LISP complexity less than $m+c$. It follows that $\max_{|s|=n} H_{\text{LISP}}(s) < n/\gamma + o(n)$.

So

$$\max_{|s|=n} H_{\text{LISP}}(s) \sim n/\gamma \sim \beta n,$$

and $\beta = 1/\gamma$. That concludes the proof of the following theorem.

Theorem G: *The LISP complexity of the string consisting of the first n bits of Ω'_{LISP} is $\sim \beta n$. In order to answer the first n questions Q_1 or Q_2 for diophantine equations D_1 and D_2 constructed from Ω'_{LISP} as indicated in Section 5, one needs a formal sytem with LISP complexity $> \beta n + o(n)$.*

8. A Third "Halting Probability" $\Omega''_{\rm LISP}$

This time the definition of the "halting probability" is more artificial, but it is much easier to see that the definition does what we want it to do.

As in Section 7, we use the fact that

$$\sum_{i=1}^{\infty} 2^{-2\lceil \log_2 i \rceil - 1} \leq \frac{1}{2} \sum_{i=1}^{\infty} 2^{-2\log_2 i} = \frac{1}{2} \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{12} < 1.$$

Multiplying together two copies of this infinite series, we see that

$$\sum_{i=1}^{\infty} \sum_{j=1}^{\infty} 2^{-2\lceil \log_2 i \rceil - 2\lceil \log_2 j \rceil - 2} < 1^2 = 1.$$

Define a new LISP "halting probability" as follows:

$$\Omega''_{\rm LISP} = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \alpha_{ij}\, 2^{-2\lceil \log_2 i \rceil - 2\lceil \log_2 j \rceil - 2}.$$

Here the dyadic rationals α_{ij}, for which it is always the case that

$$0 \leq \alpha_{ij} \leq 1,$$

are defined as follows:

$$\alpha_{ij} = \left(\frac{\#\text{ of } j\text{-bit strings that have LISP complexity} \leq i}{\#\text{ of } j\text{-bit strings}} \right) \leq 2^j / 2^j.$$

It follows immediately that

$$0 \leq \Omega''_{\rm LISP} \leq 1.$$

A string consisting of the first $n + O(\log n)$ bits of $\Omega''_{\rm LISP}$ tells us the LISP complexity of each $\leq n$ bit string; the most complex have LISP complexity $\sim \beta n$. So

Theorem H: *The LISP complexity of the string consisting of the first n bits of $\Omega''_{\rm LISP}$ is $\sim \beta n$. In order to answer the first n questions Q_1 or Q_2 for diophantine equations D_1 and D_2 constructed from $\Omega''_{\rm LISP}$ as indicated in Section 5, one needs a formal sytem with LISP complexity $> \beta n + o(n)$.*

9. Unpredictability

From the fact that the initial segments of the infinite bit strings $\Omega'_{\rm LISP}$ and $\Omega''_{\rm LISP}$ asymptotically have maximum possible LISP complexity, it follows that their successive bits cannot be predicted using any computable prediction scheme. More precisely,

Theorem I: *Consider a total recursive prediction function F, which given an arbitrary finite initial segment of an infinite bit string, returns either "no prediction", "the next bit is a 0", or "the next bit is a 1". Then if F predicts at least a fixed nonzero fraction of the bits of $\Omega'_{\rm LISP}$ and $\Omega''_{\rm LISP}$, F does no better than chance, because in the limit the relative frequency of correct and incorrect predictions both tend to $\frac{1}{2}$.*

Proof Sketch: We know that $\Omega'_{\rm LISP}$ and $\Omega''_{\rm LISP}$ both have the property that the string Ω_n of the first n bits of each has LISP complexity asymptotic to the maximum possible, which is βn.

The idea is to separate the n-bit string Ω_n consisting of the first n bits of either $\Omega'_{\rm LISP}$ or $\Omega''_{\rm LISP}$ into the substring that is not predicted, which we will leave "as is," and the substring that is predicted, which we will attempt to compress.

Let k be the number of bits of Ω_n that are predicted by F. The $(n-k)$-bit unpredicted substring of Ω_n we are given "as is." This takes $\sim \beta(n-k)$ LISP characters.

The k-bit predicted substring of Ω_n is not given directly. Instead, we calculate the predictions made by F, and are given a k-bit string telling us which predictions are correct. Let l be the number of bits that F predicts correctly. Thus this k-bit string will have l 1 bits, indicating "correct," and $(k-l)$ 0 bits, indicating "incorrect." If l is not about one-half of k, the string of successes and failures of the prediction scheme will be compressible, from the maximum possible of $\sim \beta k$ LISP characters, to only about $\beta k H(\frac{l}{k}, 1-\frac{l}{k})$ LISP characters. Here $H(p,q) = -p\log_2 p - q\log_2 q$ is the Boltzmann-Shannon entropy function. $H(p,q)$ is less than one if p and q are not both equal to a half. (For more details, see [3, Section 10].)

In summary, we use the prediction function F to stitch together the unpredicted substring of Ω_n with the predictions. And we are given a compressed string indicating when the predictions are incorrect.

So we have compressed the n-bit string Ω_n into two LISP expressions

of size totaling about

$$\beta(n-k)+\beta k H(\tfrac{l}{k},1-\tfrac{l}{k}) \ll \beta(n-k)+\beta k=\beta n.$$

This is substantially less than βn characters, which is impossible, unless $l \approx k/2$. Thus about half the predictions are correct. **Q.E.D.**

Consider an F that always predicts that the next bit of $\Omega'_{\rm LISP}$ is a 1. Applying Theorem I, we see that $\Omega'_{\rm LISP}$ has the property that 0's and 1's both have limiting relative frequency $\frac{1}{2}$. Next consider an F that predicts that each 0 bit in $\Omega'_{\rm LISP}$ is followed by a 1 bit. In the limit this prediction will be right half the time and wrong half the time. Thus 0 bits are followed by 0 bits half the time, and by 1 bits half the time. It follows by induction that each of the 2^k possible blocks of k bits in $\Omega'_{\rm LISP}$ has limiting relative frequency 2^{-k}. Thus, to use BOREL's terminology, $\Omega'_{\rm LISP}$ is "normal" in base two; so is $\Omega''_{\rm LISP}$. In fact, $\Omega'_{\rm LISP}$ and $\Omega''_{\rm LISP}$ are BOREL normal in every base, not just base two; we omit the details.

10. Hilbert's 10th Problem

I would now like to discuss HILBERT's tenth problem in the light of the theory of LISP program-size complexity. I will end with a few controversial remarks about the potential significance of these information-theoretic metamathematical results, and their connection with experimental mathematics and the quasi-empirical school of thought regarding the foundations of mathematics.

Consider a diophantine equation

$$P(k,x_1,x_2,\ldots)=0$$

with parameter k. Ask the question, "Does $P(k)=0$ have a solution?" Let

$$q=q_0q_1q_2\cdots$$

be the infinite bit string whose kth bit q_k is a 0 if $P(k)=0$ has no solution, and is a 1 if $P(k)=0$ has a solution:

$$q_k=\begin{cases} 0 & \text{if } P(k)=0 \text{ has no solution,}\\ 1 & \text{if } P(k)=0 \text{ has a solution.}\end{cases}$$

Let

$$q^n = q_0 q_1 \cdots q_{n-1}$$

be the string of the first n bits of the infinite string q, i.e., the string of answers to the first n questions. Consider the LISP complexity of q^n, $H_{\text{LISP}}(q^n)$, the size in characters of the smallest LISP expression whose value is q^n.

If HILBERT had been right and every mathematical question had a solution, then there would be a finite set of axioms from which one could deduce whether $P(k) = 0$ has a solution or not for each k. We would then have

$$H_{\text{LISP}}(q^n) \leq H_{\text{LISP}}(n) + c.$$

The c characters are the finite amount of LISP complexity in our axioms, and this inequality asserts that if one is given n, using the axioms one can compute q^n, i.e., decide which among the first n cases of the diophantine equation have solutions and which don't. Thus we would have

$$H_{\text{LISP}}(q^n) \leq \lfloor \log_{10} n \rfloor + 1 + c = O(\log n).$$

($\lfloor \log_{10} n \rfloor + 1$ is the number of digits in the base-ten numeral for n.) I.e., the LISP complexity $H_{\text{LISP}}(q^n)$ of answering the first n questions would be at most order of $\log n$ characters. We ignore the immense amount of **time** it might take to deduce the answers from the axioms; we are concentrating instead on the **size in characters** of the LISP expressions that are involved.

In 1970 MATIJASEVIČ (see [7]) showed that there is no algorithm for deciding if a diophantine equation can be solved. However, if we are told the number m of equations $P(k) = 0$ with $k < n$ that have a solution, then we can eventually determine which do and which don't. This shows that

$$H_{\text{LISP}}(q^n) \leq H_{\text{LISP}}(n) + H_{\text{LISP}}(m) + c'$$

for some $m \leq n$, which implies that

$$H_{\text{LISP}}(q^n) \leq 2(\lfloor \log_{10} n \rfloor + 1) + c' = O(\log n).$$

I.e., the LISP complexity $H_{\text{LISP}}(q^n)$ of answering the first n questions **is still** at most order of $\log n$ characters. So from the point of view

of the LISP theory of program size, HILBERT's tenth problem, while undecidable, does not look too difficult.

Using the method we presented in Section 5, one can use the "improved" LISP halting probability $\Omega'_{\rm LISP}$ or $\Omega''_{\rm LISP}$ of Sections 7 and 8 to construct an exponential diophantine equation

$$L(k, x_1, x_2, \ldots) = R(k, x_1, x_2, \ldots)$$

with a parameter k. This equation yields randomness and unpredictability as follows. Ask the question, "Does $L(k) = R(k)$ have infinitely many solutions?" Now let

$$q = q_0 q_1 q_2 \cdots$$

be the infinite bit string whose kth bit q_k is a 0 if $L(k) = R(k)$ has finitely many solutions, and is a 1 if $L(k) = R(k)$ has infinitely many solutions:

$$q_k = \begin{cases} 0 & \text{if } L(k) = R(k) \text{ has finitely many solutions,} \\ 1 & \text{if } L(k) = R(k) \text{ has infinitely many solutions.} \end{cases}$$

As before, let

$$q^n = q_0 q_1 \cdots q_{n-1}$$

be the string of the first n bits of the infinite string q, i.e., the string of answers to the first n questions. Consider the LISP complexity of q^n, $H_{\rm LISP}(q^n)$, the size in characters of the smallest LISP expression whose value is q^n. Now we have

$$H_{\rm LISP}(q^n) \sim \beta n,$$

i.e., the string of answers to the first n questions q^n has a LISP complexity that is asymptotic to the maximum possible for an n-bit string. As we discussed in Section 9, it follows that the string of answers $q = q_0 q_1 q_2 \cdots$ is now **algorithmically random**, in the sense that any computable prediction scheme that predicts at least a fixed nonzero fraction of the bits of q^n will do no better than chance.[11]

[11] In the limit exactly half the predictions will be correct and half the predictions will be incorrect.

Surprisingly, HILBERT was wrong to assume that every mathematical question has a solution. The above exponential diophantine equation yields an infinite series of mathematical facts having maximum possible LISP complexity, asymptotically β LISP characters per yes/no fact. It yields an infinite series of questions which reasoning is powerless to answer because their infinite LISP complexity exceeds the finite LISP complexity of any finite set of mathematical axioms! Here one can get out as theorems only as much LISP complexity as one explicitly puts in as axioms, and reasoning is completely useless! I think this approach to incompleteness via program-size complexity makes incompleteness look much more natural and pervasive than has previously been the case. This new approach also provides some theoretical justification for the experimental mathematics made possible by the computer, and for the new quasi-empirical view of the philosophy of mathematics that is displacing the traditional formalist, logicist, and intuitionist positions.

For other discussions of the significance of these information-theoretic incompleteness theorems, see [11–25].

References

[1] G. J. CHAITIN, *Algorithmic Information Theory,* 3rd Printing, Cambridge: Cambridge University Press (1990).

[2] J. MCCARTHY et al., *LISP 1.5 Programmer's Manual,* Cambridge MA: MIT Press (1962).

[3] G. J. CHAITIN, "LISP program-size complexity," *Applied Mathematics and Computation* **49** (1992), 79–93.

[4] G. J. CHAITIN, "Information-theoretic incompleteness," *Applied Mathematics and Computation,* in press.

[5] M. LEVIN, *Mathematical Logic for Computer Scientists,* Report TR–131, Cambridge MA: MIT Project MAC (1974).

[6] M. GARDNER, *Fractal Music, Hypercards and More...,* New York: Freeman (1992).

[7] J. P. JONES and Y. V. MATIJASEVIČ, "Proof of the recursive unsolvability of Hilbert's tenth problem," *American Mathematical Monthly* **98** (1991), 689–709.

[8] J. P. JONES and Y. V. MATIJASEVIČ, "Register machine proof of the theorem on exponential diophantine representation of enumerable sets," *Journal of Symbolic Logic* **49** (1984), 818–829.

[9] G. J. CHAITIN, "Information-theoretic limitations of formal systems," *Journal of the ACM* **21** (1974), 403–424.

[10] G. POLYA, *Mathematics and Plausible Reasoning,* Princeton: Princeton University Press (1990).

[11] G. J. CHAITIN, "Randomness and mathematical proof," *Scientific American* **232**:5 (1975), 47–52.

[12] G. J. CHAITIN, "Gödel's theorem and information," *International Journal of Theoretical Physics* **22** (1982), 941–954.

[13] G. J. CHAITIN, "Randomness in arithmetic," *Scientific American* **259**:1 (1988), 80–85.

[14] G. J. CHAITIN, "Undecidability and randomness in pure mathematics," (transcript of a lecture delivered 28 September 1989 at a SOLVAY conference in Brussels). In: G. J. CHAITIN, *Information, Randomness & Incompleteness—Papers on Algorithmic Information Theory,* 2nd Edition, Singapore: World Scientific (1990), 307–313.

[15] G. J. CHAITIN, "A random walk in arithmetic," *New Scientist* **125**:1709 (1990), 44–46. Reprinted in: N. HALL, *The New Scientist Guide to Chaos,* Harmondsworth: Penguin (1991), 196–202.

[16] J. L. CASTI, *Searching for Certainty,* New York: Morrow (1990).

[17] G. J. CHAITIN, "Number and randomness," (transcript of a lecture delivered 15 January 1991 at the Technical University of Vienna). In: M. E. CARVALLO, *Nature, Cognition and System,* Vol. 3, Dordrecht: Kluwer (1992), in press.

[18] G. J. CHAITIN, "Le hasard des nombres," *La Recherche* **22** (1991), 610–615.

[19] D. RUELLE, *Chance and Chaos,* Princeton: Princeton University Press (1991).

[20] D. RUELLE, *Hasard et Chaos,* Paris: Odile Jacob (1991).

[21] L. BRISSON and F. W. MEYERSTEIN, *Inventer L'Univers,* Paris: Les Belles Lettres (1991).

[22] J. A. PAULOS, *Beyond Numeracy,* New York: Knopf (1991).

[23] J. D. BARROW, *Theories of Everything,* Oxford: Clarendon Press (1991).

[24] T. NØRRETRANDERS, *Mærk Verden,* Denmark: Gyldendal (1991).

[25] P. DAVIES, *The Mind of God,* New York: Simon & Schuster (1992).

[26] C. SMORYŃSKI, *Logical Number Theory I,* Berlin: Springer-Verlag (1991).

LISP PROGRAM-SIZE COMPLEXITY III

Applied Mathematics and Computation 52 (1992), pp. 127–139

G. J. Chaitin

Abstract

We present a "parenthesis-free" dialect of LISP, in which (a) each primitive function has a fixed number of arguments, and (b) the parentheses associating a primitive function with its arguments are implicit and are omitted. The parenthesis-free complexity of an S-expression e is defined to be the minimum size in characters $|p|$ of a parenthesis-free LISP expression p that has the value e. We develop a theory of program-size complexity for parenthesis-free LISP by showing (a) that the maximum possible parenthesis-free complexity of an n-bit string is $\sim \beta n$, and (b) how to construct three parenthesis-free LISP halting probabilities $\Omega_{\rm pf}$, $\Omega'_{\rm pf}$ and $\Omega''_{\rm pf}$.

1. Introduction

In this paper we consider a dialect of LISP half-way between the LISP considered in my paper [1] (which is essentially normal LISP [2]) and that studied in my monograph [3]. The "parenthesis-free" LISP studied in this paper employs multiple-character atoms as in [1], but omits the parentheses associating each primitive function with its arguments as in [3]. Subadditivity arguments as in [1] rather than precise counts of S-expressions as in [3], are used to show that the maximum possible parenthesis-free LISP complexity $H_{\rm pf}$ of an n-bit string is $\sim \beta n$. A particularly natural definition of a parenthesis-free LISP halting probability $\Omega_{\rm pf}$ is presented here. Also two other halting probabilities, $\Omega'_{\rm pf}$ and $\Omega''_{\rm pf}$, that asymptotically achieve maximum possible parenthesis-free LISP complexity βn of the n-bit initial segments of their base-two expansions. We thus show that the entire theory developed in [1, 4] for $H_{\rm LISP}$ can be reformulated in terms of $H_{\rm pf}$.

In the last section we make our parenthesis-free LISP substantially easier to use.[1]

2. Précis of Parenthesis-Free LISP

Let's start with an example! Here is a sample specimen of parenthesis-free LISP. It is a self-contained LISP expression that defines a function `append` for concatenating two lists and then applies this function to the lists `(a b c)` and `(d e f)`.

Old notation [2]: see Figure 1. This expression evaluates to `(a b c d e f)`. New notation: see Figure 2. This expression, which is what we shall call a meta-expression or M-expression, is expanded as it is read by the parenthesis-free LISP interpreter into the corresponding S-expression, which has all the parentheses: see Figure 3. This expression also evaluates to `(a b c d e f)`.

In parenthesis-free LISP, each LISP primitive function must have a fixed number of arguments. So we have to fix `cond, define, plus,`

[1] Two decades ago the author wrote an interpreter for a similar LISP dialect. At that time he did not realize that a mathematical theory of program size could be developed for it.

```
(
 (lambda (append)
  (append (quote (a b c)) (quote (d e f)))
 )
 (quote
  (lambda (x y)
   (cond ((atom x) y)
         (t (cons (car x) (append (cdr x) y)))
   )
  )
 )
)
```

Figure 1. Old Notation

```
(fnc (app) (app '(a b c) '(d e f))
 'fnc (x y)
  if at x y
     jn hd x (app tl x y)
)
```

Figure 2. M-expression

```
((fnc (app) (app ('(a b c)) ('(d e f))))
 ('(fnc (x y)
   (if (at x) y
       (jn (hd x) (app (tl x) y)))))
)
```

Figure 3. S-expression

and `times`. We replace conditional expressions with an indefinite number of arguments by if-then-else expressions with three arguments. And `define`, `plus`, and `times` now always have two arguments. As long as we're at it, we also shorten the names of the primitive functions. See the table summarizing parenthesis-free LISP on page 87.

Functions dealing with integers are the same as in C. `mexp` converts an S-expression into the list of characters in the smallest possible equivalent M-expression.[2] This will be a list of integers in the range from 0 to $\alpha - 1$ (α = the size of the alphabet). `sexp` converts the list of characters in an M-expression into the corresponding S-expression.[3] `$` indicates that there are no implicit parentheses in the immediately following S-expression. (`$` also loses any special meaning within the range of a `$`.) Thus

```
'$(hd tl jn $)
```

evaluates to `(hd tl jn $)`. Another detail: `'` and `$` do not need a blank before the next character, i.e., no other atoms can start with the characters `'` or `$`.

For this parenthesis-free approach to work, it is important that

(a) Every S-expression can be written in this notation.

(b) It should be possible given a parenthesis-free LISP S-expression to calculate the equivalent M-expression[4] **of smallest size.** Here is an example of the synonym problem: `'$(eq x at y)` and `'($eq x $at y)` are the same.

(c) And one must be able to calculate the size in characters of the M-expression that corresponds to a given S-expression as in (b).

3. Parenthesis-Free LISP Complexity

Previously we had two LISP theories of program-size complexity: one for real LISP [1, 4], and one for a toy LISP [3]. In this section we

[2] This is the inverse of what the LISP interpreter's read routine does.

[3] This is an internally available version of the read routine used by the LISP interpreter.

[4] This is the inverse of the input parse that puts in the implicit parentheses.

Old Name	**New Name**	**Read As**	**Number of Arguments**
car	hd	head	1
cdr	tl	tail	1
cons	jn	join	2
atom	at	atom predicate	1
eq	eq	equal predicate	2
quote	'	quote	1
lambda	fnc	function	2
(cond (p x) (t y))	if p x y	if-then-else	3
	$	no implicit ()'s	1
define	def	define	2
eval	val	value-of	1
	valt	time-limited eval	1
	sexp	m-expr to s-expr	1
	mexp	s-expr to m-expr	1
numberp	#	number predicate	1
plus	+	plus	2
difference	−	minus	2
times	*	times	2
expt	^	raised-to-the	2
quotient	/	divided-by	2
remainder	%	remainder	2
equal	=	equal predicate	2
	!=	not-equal predicate	2
lessp	<	less-than predicate	2
greaterp	>	greater-than predicate	2
	<=	not-greater predicate	2
	>=	not-less predicate	2

Summary of Parenthesis-Free LISP

present a method for getting two new theories of LISP program-size complexity: a theory of program size for the LISP presented in Section 2, which is the subject of this paper, and a theory of parenthesis-free program size for the toy LISP in [3], which we shall say no more about. So we have **four** LISP complexity theories altogether.[5]

It is straightforward to apply to parenthesis-free LISP the techniques I used to study bounded-transfer Turing machines [6–9]. Let us define $H_{\rm pf}(x)$ where x is a bit string to be the size in characters of the smallest parenthesis-free LISP M-expression whose value is the list x of 0's and 1's. Consider the self-contained defined-from-scratch parenthesis-free version of `(append p q)`:

```
(fnc(app)(app p q)'fnc(x y)if at x y jn hd x(app tl x y))
|                                                       |
12345678901234567890123456789012345678901234567890123456
         |         |         |         |         |
         10        20        30        40        50
```

Here p is a minimal parenthesis-free LISP M-expression for the bit string x, and q is a minimal parenthesis-free LISP M-expression for the the bit string y. I.e., the value of p is the list of bits x and p is $H_{\rm pf}(x)$ characters long, and the value of q is the list of bits y and q is $H_{\rm pf}(y)$ characters long. `(append p q)` evaluates to the concatenation xy of the bit strings x and y and is

$$H_{\rm pf}(x) + H_{\rm pf}(y) + 57 - 2$$

characters long. Hence

$$H_{\rm pf}(xy) \leq H_{\rm pf}(x) + H_{\rm pf}(y) + 55.$$

Adding 55 to both sides of this inequality, we have

$$H_{\rm pf}(xy) + 55 \leq [H_{\rm pf}(x) + 55] + [H_{\rm pf}(y) + 55].$$

Therefore, let us define $H'_{\rm pf}$ as follows:

$$H'_{\rm pf}(x) = H_{\rm pf}(x) + 55.$$

[5] The next paper in this series [5], on a character-string oriented dialect of LISP, will add one more LISP complexity theory to the list, for a grand total of five!

$H'_{\rm pf}$ is subadditive just like $L(S)$, the maximum bounded-transfer Turing machine state complexity of an n-bit string:

$$H'_{\rm pf}(xy) \le H'_{\rm pf}(x) + H'_{\rm pf}(y).$$

The discussion of bounded-transfer Turing machines in [6–9] therefore applies practically word for word to $H'_{\rm pf}$. In particular, let $B(n)$ be the maximum of $H'_{\rm pf}(s)$ taken over all n-bit strings s:

$$B(n) = \max_{|s|=n} H'_{\rm pf}(s) = \max_{|s|=n} H_{\rm pf}(s) + 55.$$

Consider a string s that is $n+m$ bits long and that has the maximum complexity $H'_{\rm pf}(s)$ possible for an $(n+m)$-bit string, namely $B(n+m)$. This maximum complexity $(n+m)$-bit string s can be obtained by concatenating the string u of the first n bits of s with the string v of the last m bits of s. Therefore we have

$$B(n+m) = B(|uv|) = H'_{\rm pf}(uv) \le H'_{\rm pf}(u) + H'_{\rm pf}(v) \le B(|u|) + B(|v|) = B(n) + B(m).$$

Thus B is subadditive:

$$B(n+m) \le B(n) + B(m).$$

This extends to three or more addends. For example:

$$B(n+m+l) \le B(n+m) + B(l) \le B(n) + B(m) + B(l).$$

In general, we have:

$$B(n+m+l+\cdots) \le B(n) + B(m) + B(l) + \cdots.$$

From this subadditivity property it follows that if we consider an arbitrary n and k:

$$B(n) \le \left\lfloor \frac{n}{k} \right\rfloor B(k) + \max_{i<k} B(i).$$

Hence

$$B(n) \le \left(\frac{n}{k} + O(1) \right) B(k) + O(1).$$

[Recall that in this context $O(1)$ denotes a bounded term and $o(1)$ denotes a term that tends to the limit 0.[6]] Dividing through by n and letting $n \to \infty$, we see that

$$\frac{B(n)}{n} \leq \left(\frac{1}{k} + o(1)\right) B(k) + o(1).$$

[Recall that

$$\begin{aligned} \limsup_{n\to\infty} \varphi(n) &= \lim_{n\to\infty} \sup\{\varphi(k) : k \geq n\}, \\ \liminf_{n\to\infty} \varphi(n) &= \lim_{n\to\infty} \inf\{\varphi(k) : k \geq n\}. \end{aligned}$$

The **sup**remum/**inf**inum of a set of reals is the l.u.b./g.l.b. (least upper bound/greatest lower bound) of the set. This extends the **max**imum and **min**imum from finite sets to infinite sets of real numbers.[7]] Therefore

$$\limsup_{n\to\infty} \frac{B(n)}{n} \leq \frac{B(k)}{k}.$$

Since this holds for any k, it follows that in fact

$$\limsup_{n\to\infty} \frac{B(n)}{n} = \inf_k \frac{B(k)}{k} = \beta.$$

This shows that as n goes to infinity, $B(n)/n$ tends to the finite limit β from above. Now we shall show that this limit β is greater than zero. It is easy to see that because shorter LISP M-expressions may be extended with blanks,

$$\alpha^{\max_{|s|=n} H_{\rm pf}(s)} \geq 2^n.$$

Here α is the number of characters in the parenthesis-free LISP alphabet. (This inequality merely states that LISP M-expressions of at least this size are needed to be able to produce all 2^n n-bit strings.[8]) In other

[6] See HARDY and WRIGHT [10, p. 7].

[7] See HARDY [11].

[8] If it weren't for the fact that all shorter expressions are already included in this count, this inequality would have the following slightly more cumbersome form:

$$\sum_{k \leq \max_{|s|=n} H_{\rm pf}(s)} \alpha^k \geq 2^n.$$

words,

$$2^{(\log_2 \alpha) \max_{|s|=n} H_{\rm pf}(s)} \geq 2^n.$$

Thus

$$(\log_2 \alpha) \max_{|s|=n} H_{\rm pf}(s) \geq n.$$

I.e.,

$$\max_{|s|=n} H_{\rm pf}(s) \geq \frac{n}{\log_2 \alpha}.$$

Hence

$$\frac{\max_{|s|=n} H_{\rm pf}(s)}{n} \geq \frac{1}{\log_2 \alpha}.$$

Therefore

$$\frac{B(n)}{n} = \frac{\max_{|s|=n} H_{\rm pf}(s) + 55}{n} \geq \frac{1}{\log_2 \alpha} + \frac{O(1)}{n} = \frac{1}{\log_2 \alpha} + o(1).$$

Thus we see that for all sufficiently large n, $B(n)/n$ is bounded away from zero:

$$0 < \frac{1}{\log_2 \alpha} \leq \liminf_{n \to \infty} \frac{B(n)}{n}.$$

Hence the finite limit

$$\lim_{n \to \infty} \frac{B(n)}{n} = \beta,$$

which we already know exists, must be greater than zero. We can thus finally conclude that $B(n)$ is asymptotic from above to a nonzero constant β times n:

$$\begin{aligned} B(n) &\sim \beta n, \\ &\geq \beta n. \end{aligned}$$

I.e., the "adjusted by +55" maximum parenthesis-free LISP complexity $H'_{\rm pf}(s)$ of an n-bit string s is asymptotic from above to a nonzero constant β times n:

$$\begin{aligned} \max_{|s|=n} H'_{\rm pf}(s) &\sim \beta n, \\ &\geq \beta n. \end{aligned}$$

In other words, the maximum parenthesis-free LISP complexity $H_{\rm pf}(s)$ of an n-bit string s is asymptotic to a nonzero constant β times n:

$$\begin{aligned} \max_{|s|=n} H_{\rm pf}(s) &\sim \beta n, \\ &\geq \beta n - 55. \end{aligned}$$

4. The Halting Probabilities $\Omega_{\rm pf}, \Omega'_{\rm pf}, \Omega''_{\rm pf}$

In Section 3 we showed that the maximum possible parenthesis-free LISP complexity $H_{\rm pf}$ of an n-bit string is asymptotic to βn, just as was the case with $H_{\rm LISP}$ in [1, Section 8], even though we no longer count all parentheses as part of the complexity of a LISP S-expression. With this basic result in hand, one can immediately rework all of [1] for this new complexity measure $H_{\rm pf}$.

What about reworking the sequel [4], which studies the corresponding incompleteness theorems and halting probabilities? Everything is straightforward and immediate. The only problems that arise in reworking the discussion in [4] to use $H_{\rm pf}$ instead of $H_{\rm LISP}$, are with halting probabilities. We must figure out (a) how to define a new halting probability $\Omega_{\rm pf}$ to replace $\Omega_{\rm LISP}$, and (b) how to prove that the initial n-bit segment of the new version $\Omega'_{\rm pf}$ of $\Omega'_{\rm LISP}$ has $H_{\rm pf} \sim \beta n$.

$\Omega_{\rm pf}$

Problem: How do we define a new halting probability $\Omega_{\rm pf}$ to replace $\Omega_{\rm LISP}$?

As was discussed in [4, Section 4], the sum

$$\Omega_{\rm LISP} = \sum_{(e)\ \text{halts}} \alpha^{-|(e)|}$$

is ≤ 1 and converges because non-atomic LISP S-expressions are self-delimiting. I.e., no extension of an (e) included in $\Omega_{\rm LISP}$ is included in $\Omega_{\rm LISP}$. However, extensions of non-atomic parenthesis-free LISP M-expressions may yield other valid M-expressions.

There is a simple general solution to this problem: In our imagination we add a blank at the end of each parenthesis-free LISP M-expression to cut off possible extensions. With this imaginary modification, it is again the case that no extension of a parenthesis-free LISP M-expression is another valid parenthesis-free LISP M-expression, just as was the case with non-atomic S-expressions in [4, Section 4]. In other words, we define the parenthesis-free LISP halting probability as follows:

$$\Omega_{\rm pf} = \sum_{e\ \text{halts}} \alpha^{-|e|-1}.$$

This is summed over all parenthesis-free LISP M-expressions e that have a value or are defined. In [4, Section 4] this sum is taken over all **non-atomic** LISP S-expressions e that have a value or are defined. Here we do not have to restrict the expressions e included in the sum to be non-atomic. We have $\Omega_{\rm pf} \leq 1$, as desired.

$\Omega''_{\rm pf}$

The exact same method used to produce the halting probability in [4, Section 8], $\Omega''_{\rm LISP}$, immediately yields a corresponding $\Omega''_{\rm pf}$.

$$\Omega''_{\rm pf} = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \alpha_{ij}\, 2^{-2\lceil \log_2 i \rceil - 2\lceil \log_2 j \rceil - 2}.$$

Here

$$\alpha_{ij} = \frac{\text{\# of } j\text{-bit strings that have parenthesis-free LISP complexity} \leq i}{\text{\# of } j\text{-bit strings}}.$$

Just as in [4, Section 8], $\Omega''_{\rm pf}$ has the property that the string consisting of the first n bits of the base-two expansion of $\Omega''_{\rm pf}$ asymptotically has maximum possible parenthesis-free LISP complexity βn. Thus we can follow [4, Section 5] and construct from $\Omega''_{\rm pf}$ diophantine equations D_1 and D_2 with the following property: To answer either the first n cases of the yes/no question Q_1 in [4, Section 5] about equation D_1 or the first n cases of the yes/no question Q_2 in [4, Section 5] about equation D_2 requires a formal system with parenthesis-free LISP complexity $> \beta n + o(n)$. I.e., the proof-checking function associated with a formal system that enables us to determine the first n bits of the base-two expansion of $\Omega''_{\rm pf}$ must have parenthesis-free LISP complexity $> \beta n + o(n)$.

$\Omega'_{\rm pf}$

Following [4, Section 7], $\Omega'_{\rm pf} = \sum_{n=1}^{\infty}$ of

$$\frac{\text{\# of different} \leq n \text{ character parenthesis-free LISP M-expressions that halt}}{2^{\lceil \log_2(\text{total \# of different} \leq n \text{ character parenthesis-free LISP M-expressions}) \rceil + 2\lceil \log_2 n \rceil + 1}}.$$

The proof in [4, Section 7] that an initial n-bit segment of $\Omega'_{\rm LISP}$ asymptotically has maximum possible complexity βn no longer works for $\Omega'_{\rm pf}$. The problem is showing that there is a real number γ such that

$$\gamma n \sim \log_2 S_n$$

where

$$S_n = \#\text{ of different} \le n \text{ character parenthesis-free LISP M-expressions.}$$

As was the case in defining $\Omega_{\rm pf}$, the trick is to imagine an extra blank at the end of each M-expression. In other words, everywhere "$\le n$ character M-expressions" appears, one must change this to "$< n$ character M-expressions." So the fix is that now we consider instead

$$S_n = \#\text{ of different} < n \text{ character parenthesis-free LISP M-expressions.}$$

As was the case with the definition of $\Omega_{\rm pf}$, this trick automatically takes care of the fact that atomic M-expressions are not self-delimiting. For $\Omega'_{\rm pf}$ it is not necessary to follow the proof in [4, Section 7] and consider S'_n, which is the number of expressions counted in S_n that are non-atomic. Instead one works directly with S_n, and it is now the case that $S_{nk+3} \ge (S_k)^n$.

Thus $\Omega'_{\rm pf}$ has the property that the string consisting of the first n bits of the base-two expansion of $\Omega'_{\rm pf}$ asymptotically has maximum possible parenthesis-free LISP complexity βn. Thus we can follow [4, Section 5] and construct from $\Omega'_{\rm pf}$ diophantine equations D_1 and D_2 with the following property: To answer either the first n cases of the yes/no question Q_1 in [4, Section 5] about equation D_1 or the first n cases of the yes/no question Q_2 in [4, Section 5] about equation D_2 requires a formal system with parenthesis-free LISP complexity $> \beta n + o(n)$. I.e., the proof-checking function associated with a formal system that enables us to determine the first n bits of the base-two expansion of $\Omega'_{\rm pf}$ must have parenthesis-free LISP complexity $> \beta n + o(n)$.

In summary, we see that all of [1, 4] carries over from LISP complexity to parenthesis-free LISP complexity.

5. Improving Parenthesis-Free LISP

Following [3], here is an improvement to parenthesis-free LISP.

Let's add `let`, read as "let... be...". `let` has three arguments. If the first argument is an atom, the M-expression

```
let x v e
```

stands for the S-expression

```
((fnc (x) e) v)
```

I.e., evaluate `e` with `x` bound to the value of `v`. On the other hand, the M-expression

```
let (f x y...) d e
```

stands for the S-expression

```
((fnc (f) e) ('(fnc (x y...) d)))
```

I.e., evaluate `e` with `f` bound to the definition of a function whose formal parameters are `x y...` and having `d` as the body of its definition.

Here is an example, a **single** M-expression:

```
let (app x y) if at x y
                 jn hd x (app tl x y)
let x '(a b c)
let y '(d e f)
(app x y)
```

The value of this expression is:

```
(a b c d e f)
```

Evaluating the following **four** M-expressions gives the same final value, but leaves `app, x,` and `y` defined.

```
def (app x y) if at x y
                 jn hd x (app tl x y)
def x '(a b c)
def y '(d e f)
(app x y)
```

The `let` notation makes our parenthesis-free LISP more convenient to use, and all the proofs in Sections 3 and 4 go through without change with `let` added.

References

[1] G. J. CHAITIN, "LISP program-size complexity," *Applied Mathematics and Computation* **49** (1992), 79–93.

[2] J. MCCARTHY et al., *LISP 1.5 Programmer's Manual,* Cambridge MA: MIT Press (1962).

[3] G. J. CHAITIN, *Algorithmic Information Theory,* 3rd Printing, Cambridge: Cambridge University Press (1990).

[4] G. J. CHAITIN, "LISP program-size complexity II," *Applied Mathematics and Computation,* in press.

[5] G. J. CHAITIN, "LISP program-size complexity IV," *Applied Mathematics and Computation,* in press.

[6] G. J. CHAITIN, "On the length of programs for computing finite binary sequences by bounded-transfer Turing machines," Abstract 66T–26, *AMS Notices* **13** (1966), 133.

[7] G. J. CHAITIN, "On the length of programs for computing finite binary sequences by bounded-transfer Turing machines II," Abstract 631–6, *AMS Notices* **13** (1966), 228–229.

[8] G. J. CHAITIN, "On the length of programs for computing finite binary sequences," *Journal of the ACM* **13** (1966), 547–569.

[9] G. J. CHAITIN, "On the length of programs for computing finite binary sequences: Statistical considerations," *Journal of the ACM* **16** (1969), 145–159.

[10] G. H. HARDY and E. M. WRIGHT, *An Introduction to the Theory of Numbers,* Oxford: Clarendon Press (1990).

[11] G. H. HARDY, *A Course of Pure Mathematics,* Cambridge: Cambridge University Press (1952).

LISP PROGRAM-SIZE COMPLEXITY IV

Applied Mathematics and Computation 52 (1992), pp. 141–147

G. J. Chaitin

Abstract

We present a new "character-string" oriented dialect of LISP in which the natural LISP halting probability asymptotically has maximum possible LISP complexity.

1. Introduction

This paper continues the study of LISP program-size complexity in my monograph [1, Chapter 5] and the series of papers [2–4].

In this paper we consider a dialect of LISP half-way between the

LISP considered in my paper [2] (which is essentially normal LISP [5]) and that studied in my monograph [1]. The "character-string" LISP studied in this paper employs multiple-character atoms as in [2], but allows all possible character strings as S-expressions subject only to the requirement that parentheses balance as in [1]. Precise counts of S-expressions as in [1] rather than subadditivity arguments as in [2], are used to show that the maximum possible character-string LISP complexity $H_{\rm cs}(s)$ of an n-bit string s is $\beta n + O(\log n)$, where $\beta = 1/\log_2 \alpha$ and α = the number of characters in the alphabet. A particularly natural definition of a character-string LISP halting probability $\Omega_{\rm cs}$ is presented here. The n-bit initial segment of the base-two expansion of $\Omega_{\rm cs}$ asymptotically achieves maximum possible character-string LISP complexity βn. Indeed, the entire theory developed in [1, Section 5.1] for toy LISP and in [2, 3] for $H_{\rm LISP}$ can be reformulated in terms of $H_{\rm cs}$.

2. Précis of Character-String LISP

Let's put the LISP of [5] on the operating table and examine it from an information-theoretic point of view. The problem is that sometimes different looking S-expressions produce the same internal structure when read in by the LISP interpreter and look the same if then written out. In other words, the problem is **synonyms!** Information is being wasted because not all different strings of characters in the external representation of an S-expression lead to different S-expressions in the internal representation, which consists of binary trees of pointers.

In my monograph [1] I fixed this by using drastic surgery. First of all, blanks are eliminated and all atoms are exactly one character long. Next, `()` and `nil` are no longer synonyms, because the only way to denote the empty list is via `()`. And "true" and "false" are `1` and `0` instead of `t` and `nil`.

The illness is serious, but the cure in [1] is rather drastic. Here we present another way to eliminate synonyms and improve the expressive power of LISP from an information-theoretic point of view. This time the aim is to keep things as much as possible the way they are in normal LISP [5]. So each extra, not-strictly-necessary blank is now a "blank

atom," which prints as a blank or ␣ when we want to show it. And both `nil` and `()` denote an empty list, but will print out differently and compare different internally using the LISP primitive function `eq`.

Also we allow **any** succession of characters in a legal S-expression between an opening left parenthesis and a closing right parenthesis, and consider that each such S-expression **is different**. The only rule is that parentheses must balance.

I think that a good way to express this idea is to call it a "character-string" oriented version of LISP. It might also be called a "wysiwyg" ("what you see is what you get") version of LISP. The external representation of S-expressions is now taken seriously; before only the internal representation of S-expressions really counted.

In summary, internal and external format are now precisely equivalent, and reading an S-expression in and then writing it out gives exactly the same thing back that was read in. There are no synonyms: `()` and `nil` are different, `00022` and `22` are different, and `(x␣y)` and `(x␣␣␣y)` are different. Each different string of characters is a different S-expression and blanks within an S-expression are respected. `0x0x` is a valid atom.

For example, the following S-expressions **are all different**:

`(a␣b␣nil)`

`(a␣b␣())`

`(a␣b())`

`(a␣b␣(␣))`

`(a␣b(␣))`

`(a␣b␣()␣)`

`(a␣b()␣)`

`(␣a␣b␣())`

`(␣a␣b())`

`(␣a␣b␣nil)`

(a␣␣␣b␣␣␣nil)

(a␣␣␣b␣␣␣())

In normal LISP, these would all be the same S-expression. And the LISP primitive-function `eq` is defined to be character-string equality, so that it can see that all these S-expressions are different.

But what do these S-expressions with blanks mean? Consider the following character-string LISP S-expression:

(␣␣␣abc␣␣␣def␣␣␣)

The first three blanks denote three blank atoms. The second three blanks denote only two blank atoms. And the last three blanks denote three blank atoms. Blank atoms are always **a single blank**.

In general, n consecutive blanks within an S-expression will either denote n blank atoms or $n-1$ blank atoms; it will be $n-1$ blank atoms iff the blanks separate two consecutive atoms $\neq$`()`. In other words, n consecutive blanks denote n blank atoms except when they separate two characters that are neither parentheses nor blanks:

...abc␣␣␣def...

In this situation, and only this situation, $n+1$ blanks denote n blank atoms. E.g., a single blank separating two atoms that are $\neq$`()` does not entail any blank atoms:

...abc␣def...

On the other hand, here there is no blank atom:

...(abc)(def)...

And here there is exactly one blank atom:

...(abc)␣(def)...

With this approach, `append` of two S-expressions will carry along all the blanks in its operands, and will add a blank if the first list ends with an atom $\neq$`()` and the second list begins with an atom $\neq$`()`:

$$\texttt{\$(append('(}\sqcup\texttt{a}\sqcup\texttt{b}\sqcup\texttt{c}\sqcup\texttt{))('(}\sqcup\texttt{d}\sqcup\texttt{e}\sqcup\texttt{f}\sqcup\texttt{)))} \to \texttt{(}\sqcup\texttt{a}\sqcup\texttt{b}\sqcup\texttt{c}\sqcup\sqcup\sqcup\texttt{d}\sqcup\texttt{e}\sqcup\texttt{f}\sqcup\texttt{)}$$

$$\texttt{\$(append('(}\sqcup\texttt{a}\sqcup\texttt{b}\sqcup\texttt{c}\sqcup\texttt{))('(}\sqcup\texttt{(d)}\sqcup\texttt{e}\sqcup\texttt{f}\sqcup\texttt{)))} \to \texttt{(}\sqcup\texttt{a}\sqcup\texttt{b}\sqcup\texttt{c}\sqcup\sqcup\texttt{(d)}\sqcup\texttt{e}\sqcup\texttt{f}\sqcup\texttt{)}$$

Thus the result of appending a list of n elements and a list of m elements always has $n+m$ elements as usual. $ starts a mode in which all blanks are significant in the next complete S-expression. In the normal mode, excess blanks are removed, as is usual in LISP.

A few more words on $: Normally, extra blanks are helpful in writing S-expressions, to indicate their structure, but we don't really want the blank atoms. So we use the LISP input "meta-character" $ to indicate that in the next complete S-expression extra blanks should not be removed. For example, $\texttt{\$}\sqcup$ is a single blank atom. And $\texttt{\$(}\sqcup\sqcup\sqcup\texttt{)}$ is a list of three blank atoms. But the input expression

$$\texttt{(a}\sqcup\sqcup\sqcup\texttt{\$(}\sqcup\sqcup\sqcup\texttt{)}\sqcup\sqcup\sqcup\texttt{b)}$$

denotes

$$\texttt{(a(}\sqcup\sqcup\sqcup\texttt{)b)}$$

because the extra blanks next to the `a` and the `b` are outside the range of the $. The use of $ as a meta-character makes it impossible to have $'s in the name of an atom.

Note that when taking `car`, `cdr` and `cons`, blanks are usually just carried along, but sometimes a single blank must be stripped off or added. The following examples explain how this all works:

- $\texttt{\$(car('(}\sqcup\texttt{a}\sqcup\sqcup\texttt{b}\sqcup\sqcup\texttt{c)))} \to \sqcup$

  $\texttt{\$(cdr('(}\sqcup\texttt{a}\sqcup\sqcup\texttt{b}\sqcup\sqcup\texttt{c)))} \to \texttt{(a}\sqcup\sqcup\texttt{b}\sqcup\sqcup\texttt{c)}$

  $\texttt{\$(cons('}\sqcup\texttt{)('(a}\sqcup\sqcup\texttt{b}\sqcup\sqcup\texttt{c)))} \to \texttt{(}\sqcup\texttt{a}\sqcup\sqcup\texttt{b}\sqcup\sqcup\texttt{c)}$

- $\texttt{\$(cons('a)('(b}\sqcup\texttt{c)))} \to \texttt{(a}\sqcup\texttt{b}\sqcup\texttt{c)}$
  (adds one blank after the `a`)

  $\texttt{\$(cons('a)('(}\sqcup\texttt{b}\sqcup\sqcup\texttt{c)))} \to \texttt{(a}\sqcup\sqcup\texttt{b}\sqcup\sqcup\texttt{c)}$
  (adds one blank after the `a`)

  $\texttt{\$(cdr('(a}\sqcup\sqcup\texttt{b}\sqcup\sqcup\texttt{c)))} \to \texttt{(}\sqcup\texttt{b}\sqcup\sqcup\texttt{c)}$
  (eliminates one blank after the `a`)

- `$(cons('a)('((b)␣␣c)))` → `(a(b)␣␣c)`
  (doesn't add one blank after the `a`)

  `$(cons('a)('(␣(b)␣␣c)))` → `(a␣(b)␣␣c)`
  (doesn't add one blank after the `a`)

  `$(cdr('(a␣(b)␣␣c)))` → `(␣(b)␣␣c)`
  (doesn't eliminate one blank after the `a`)

- `$(cons('a)('nil))` → `(a)`

  `$(cons('a)('()))` → `(a)`

  `$(cdr('(a)))` → `nil`
  (`cdr` never yields `()`)

- `$(eq('nil)('()))` → `nil`

The primitive functions `car`, `cdr` and `cons` are defined so that if there are no extra blanks, they give exactly the same value as in normal LISP. The primitive-function `eq` is defined to be character-string equality of entire S-expressions, not just atoms. Of course, the convert S-expression to character-string primitive function (see [2, Section 2]) gives each repeated blank; this and `eq` are a way to extract all the information from an S-expression. And the convert character-string to S-expression primitive function (see [2, Section 2]) is able to produce all possible S-expressions.

In summary, this new approach avoids the fact that `()` is the same as `nil` and blanks are often ignored in LISP. We now allow and distinguish all combinations of blanks; this makes it possible to pack much more information in an S-expression. We must also allow names of atoms with any possible mix of characters, as long as they are neither blanks nor parentheses; we cannot outlaw, as normal LISP does, the atom `9xyz`. In normal LISP [5], if an atom begins with a digit, it must **all** be digits. In our LISP, `9xyz` is allowed as the name of an atom. Thus our definition of an integer is that it is an atom that is all digits, possibly preceded by a minus sign (hyphen), not an atom that begins with a digit. Also we must allow numbers with 0's at the left like `00022`, and `eq` must consider `00022` and `022` to be different. There is a different equality predicate "`=`" that is for numbers.

Note that it is still possible to implement character-string LISP efficiently via binary trees of pointers as is done in normal LISP. Efficient implementations of normal LISP are based on cells containing two pointers, to the `car` and the `cdr` of an S-expression (see [5]).

3. The Halting Probability Ω_{cs}

Of course the character-string LISP complexity $H_{\text{cs}}(x)$ of an S-expression x is now defined to be the minimum size in characters $|e|$ of a character-string LISP S-expression e that evaluates to x. Here e is in the "official" character-string LISP notation in which every blank is significant, not in the "meta-notation" used in Section 2 in which only blanks in the range of a $ are significant.

The new idea that we presented in Section 2 is to think of LISP S-expressions as character strings, to allow **any** succession of characters in a legal S-expression between an opening left parenthesis and a closing right parenthesis, and to consider that each such S-expression **is different**. The only rule is that parentheses must balance. From the discussion of toy LISP in [1], we know that having parentheses balance does not significantly decrease the multiplicative growth of the number of possibilities. I.e., the number of S-expressions with n characters has a base-two logarithm that is asymptotic to $n \log_2 \alpha$, where α is the number of characters in the LISP alphabet including the blank and both parentheses. (See [1, Appendix B].)

More precisely, the analysis of [1, Appendix B] gives the exact number and the asymptotics of the **non-atomic** character-string LISP S-expressions of size n. There are also $(\alpha - 3)^n$ **atomic** character-string LISP S-expressions of size n, which is negligible in comparison. Thus the total number S_n of character-string LISP S-expressions with exactly n characters is asymptotic to

$$S_n \sim \frac{\alpha^{n-2}}{2\sqrt{\pi}(n/\alpha)^{1.5}}.$$

Hence

$$\log_2 S_n = n \log_2 \alpha + O(\log n).$$

From this it is easy to see that the maximum possible character-string LISP complexity $H_{\rm cs}(s)$ of an n-bit string s is $\beta n + O(\log n)$, where $\beta = 1/\log_2 \alpha$ and $\alpha =$ the number of characters in the alphabet (including the blank and both parentheses).

We see that the rules in Section 2 remove almost all the redundancy in normal LISP S-expressions.[1] Also, because no extension of a non-atomic S-expression (e) is a valid non-atomic S-expression, we can define a character-string LISP halting probability $\Omega_{\rm cs}$ as follows:

$$\Omega_{\rm cs} = \sum_{(e) \text{ has a value}} \alpha^{-[\text{size of } (e)]} = \sum_{(e) \text{ halts}} \alpha^{-|(e)|}.$$

Just as in [3, Section 4], we see that being given the first $n + O(\log n)$ bits of the base-two expansion of $\Omega_{\rm cs}$ would enable one to determine the character-string LISP complexity $H_{\rm cs}$ of each $\leq n$ bit string. The maximum of $H_{\rm cs}(s)$ taken over all $\leq n$ bit strings s is asymptotic to βn. As in [3, Section 4], it follows that the string consisting of the first n bits of the base-two expansion of $\Omega_{\rm cs}$ itself asymptotically has maximum possible character-string LISP complexity $H_{\rm cs} \sim \beta n$.[2] Thus we can follow [3, Section 5] and construct from $\Omega_{\rm cs}$ diophantine equations D_1 and D_2 with the following property: To answer either the first n cases of the yes/no question Q_1 in [3, Section 5] about equation D_1 or the first n cases of the yes/no question Q_2 in [3, Section 5] about equation D_2 requires a formal system with character-string LISP complexity $> \beta n + o(n)$. I.e., the proof-checking function associated with a formal system that enables us to determine the first n bits of the base-two expansion of $\Omega_{\rm cs}$ must have character-string LISP complexity $> \beta n + o(n)$.

References

[1] G. J. CHAITIN, *Algorithmic Information Theory,* 3rd Printing, Cambridge: Cambridge University Press (1990).

[1] Hence minimal character-string LISP S-expressions for a given result are essentially unique, and can also be proved to be "normal" (i.e., in the limit, all α possible characters occur with equal relative frequency). See [1, Section 5.1].

[2] From this it is not difficult to show that $\Omega_{\rm cs}$ is BOREL normal in every base. See [3, Section 9].

[2] G. J. CHAITIN, "LISP program-size complexity," *Applied Mathematics and Computation* **49** (1992), 79–93.

[3] G. J. CHAITIN, "LISP program-size complexity II," *Applied Mathematics and Computation,* in press.

[4] G. J. CHAITIN, "LISP program-size complexity III," *Applied Mathematics and Computation,* in press.

[5] J. MCCARTHY et al., *LISP 1.5 Programmer's Manual,* Cambridge MA: MIT Press (1962).

INFORMATION-THEORETIC INCOMPLETENESS

Applied Mathematics and Computation 52 (1992), pp. 83–101

G. J. Chaitin

Abstract

We propose an improved definition of the complexity of a formal axiomatic system: this is now taken to be the minimum size of a self-delimiting program for enumerating the set of theorems of the formal system. Using this new definition, we show (a) that no formal system of complexity n can exhibit a specific object with complexity greater than $n + c$, and (b) that a formal system of complexity n can determine at most $n + c$ scattered bits of the halting probability Ω. We also present a short, self-contained proof of (b).

1. Introduction

The main incompleteness theorems of my *Algorithmic Information Theory* monograph [16] are reformulated and proved here using a new and improved definition of the complexity of a formal axiomatic system. This new approach is the self-delimiting version of that used in my long 1974 paper [4], which may be contrasted with my short 1974 paper [3]. In other words, this paper and my monograph [16] stand in the same relation as my papers [4] and [3] do.

The new idea is to measure the complexity of a formal system in terms of the program-size complexity of enumerating its infinite set of theorems, not in terms of the program-size complexity of the finite string of the axioms.

This new approach combines in a single number the complexity of the axioms and the rules of inference, and the new complexity κ' is never more than c greater and can sometimes be up to $\approx \log_2 \kappa$ less than the old complexity κ. Thus the incompleteness results given here are never weaker and are sometimes somewhat stronger than the incompleteness results in [16].

In addition, this new approach led me to a short, self-contained proof (presented in Section 9) that it is hard to determine scattered bits of the halting probability Ω. While the general theory developed in my monograph [16] is still necessary to substantiate my thesis that there is randomness in arithmetic, there is now a short-cut to the result on the difficulty of determining scattered bits of Ω.

2. Bit String Complexity

Following CHAITIN [6, 16], a computer C is a partial recursive function that maps a program p (a bit string) into an output $C(p)$, which is also a bit string. C is not given the entire program p immediately. Instead, C must request each bit of p, one bit at a time [6]. If a bit is requested it is always provided, so that in a sense programs are initial segments of infinite bit strings. The blank-endmarker approach in [3, 4] may also be considered to request one bit at a time, but differs from the self-delimiting approach used here because requesting a bit may also yield

a blank, indicating that the program has ended.

A more abstract formulation of this self-delimiting program approach, but one that can be proved [6] to be entirely equivalent, is to stipulate that a computer C has the property that no extension of a valid program is a valid program. I.e., if p' is an extension of p, then $C(p')$ cannot be defined if $C(p)$ is defined. In other words, the set of valid programs, which is the domain of definition of the partial recursive function C, is a so-called "prefix-free set."

The complexity $H_C(x)$ of the string x based on the computer C is the size in bits $|p|$ of the smallest program p for computing x with C:

$$H_C(x) = \min_{C(p)=x} |p|.$$

In addition to this complexity measure, there are related probabilities that take into account **all** programs that produce a given result, not just the smallest ones. The probability $P_C(x)$ of the string x based on the computer C is the probability that C computes x if each bit of the program p is the result of an independent toss of a fair coin:

$$P_C(x) = \sum_{C(p)=x} 2^{-|p|}.$$

It is easy to see that this sum converges and must be between zero and one, because the p that are summed are a prefix-free set. I.e., if a program p is included in this sum, then no extension of p is included in this sum.

Define a universal computer U as follows:

$$U(\overbrace{000\cdots000}^{i\ 0\text{'s}}1p) = C_i(p).$$

Here C_i is the computer with GÖDEL number i, i.e., the ith computer. Hence

$$H_U(x) \leq H_{C_i}(x) + (i+1)$$

and

$$P_U(x) \geq P_{C_i}(x)\, 2^{-(i+1)}$$

for all strings x. The general definition of a universal computer U is that it has the property that for each computer C there is a prefix σ_C such that

$$U(\sigma_C\, p) = C(p)$$

for all p. I.e., the prefix σ_C tells U how to simulate C. Hence for each computer C there is a constant $\mathrm{sim}_C = |\sigma_C|$ (the cost in bits of **sim**ulating C) such that

$$H_U(x) \le H_C(x) + \mathrm{sim}_C$$

and

$$P_U(x) \ge P_C(x)\, 2^{-\mathrm{sim}_C}$$

for all x. The universal computer U we defined above satisfies this definition with $\mathrm{sim}_{C_i} = i+1$. We pick this particular universal computer U as our standard one and define the complexity $H(x)$ to be $H_U(x)$, and the algorithmic probability $P(x)$ to be $P_U(x)$:

$$H(x) = H_U(x), \qquad P(x) = P_U(x).$$

The halting probability Ω is the total algorithmic probability:

$$\Omega = \sum_x P(x) = \sum_{U(p) \text{ is defined}} 2^{-|p|}.$$

Ω is a real number between zero and one, and we also think of it as the infinite bit string of its binary digits. In [16] it is shown that Ω satisfies three different but equivalent definitions of randomness: the constructive measure-theoretic definitions of MARTIN-LÖF and SOLOVAY, and the complexity-theoretic definition of CHAITIN. (An alternative:

$$0 < \Omega' = \sum_x 2^{-H(x)} < 1.$$

This often works just as well.)

3. Discussion

A fundamental theorem shows that the complexity measure H and the algorithmic probability P are closely related:

$$H(x) = -\log_2 P(x) + O(1).$$

Another basic fact is that most bit strings s of length n have close to the maximum possible complexity

$$\max_{|s|=n} H(s) = n + H(n) + O(1) = n + O(\log n).$$

How close are the complexities of most n-bit strings s to the maximum possible? The number of n-bit strings s with complexity k less than the maximum drops off exponentially:

$$\#\left\{|s| = n : H(s) < n + H(n) - k\right\} < 2^{n-k+c}.$$

The reason for picking a computer U with self-delimiting programs to measure program-size complexity, is that self-delimiting programs can be concatenated. For example, if $U(p) = x$ and $U(q) = y$, then we can compute x and y if we are given pq, the concatenation of the programs p and q.

4. Exhibiting Complex Strings

Now to metamathematics! First we do things the old way.

Following CHAITIN [3, 8, 16], the rules of inference F are a recursively enumerable set of ordered pairs of the form $\langle A, T\rangle$ indicating that the theorem T follows from the axiom A:

$$F = \{\langle A_1, T_1\rangle, \langle A_2, T_2\rangle, \langle A_3, T_3\rangle, \dots\}.$$

Instead of $\langle A, T\rangle \in F$, one often writes $A \vdash_F T$. (The axiom A is represented as a bit string via some standard binary encoding.) F is fixed, and A varies.

Theorem A (CHAITIN [8][1]): *Consider a formal system F_A consisting of all theorems derived from an axiom A by applying the rules of inference F. The formal system F_A cannot exhibit a specific string with complexity $> H(A) + c_F$. More precisely, if $A \vdash_F H(s) > n$ only if $H(s) > n$, then $A \vdash_F H(s) > n$ only if $n < H(A) + c_F$.*

[1]See [1–5] for early versions of this information-theoretic incompleteness theorem.

Proof: Consider a special-purpose computer C that does the following when given a minimal-size program p_k for the natural number k followed by a minimal-size program p_A for the axiom A:

$$C(p_k\, p_A) = \left\{ \begin{array}{l} \text{The first specific string } s^\star \\ \text{that can be shown in } F_A \text{ to} \\ \text{have complexity} > k + |p_A|. \end{array} \right.$$

Here

$$U(p_k) = k, \qquad |p_k| = H(k),$$

and

$$U(p_A) = A, \qquad |p_A| = H(A).$$

How does C accomplish this? First C simulates running p_k on U to determine the natural number k. Then C simulates running p_A on U to determine the axiom A. Now C knows A and k. C then searches through all proofs derived from the axiom A, searching through the possible proofs in size order, and among those of the same size, in some arbitrary alphabetical order, applying the proof-checking algorithm associated with the fixed rules of inference F to each proof in turn. (More abstractly, C enumerates the set of theorems

$$F_A = \{\, T : A \vdash_F T \,\} = \{\, T : \langle A, T \rangle \in F \,\}.$$

) In this manner C determines each theorem that follows from the axiom A. C examines each of these theorems until it finds the first one of the form

$$\text{“}H(s^\star) > j\text{”}$$

that asserts that a specific bit string $s^\star$ has complexity greater than a specific natural number j that is greater than or equal to k plus the complexity $|p_A|$ of the axiom A:

$$j \geq k + |p_A|.$$

C then outputs the string $s^\star$ and halts. Hence

$$H_C(s^\star) \leq |p_k\, p_A|,$$

and

$$k + |p_A| < H(s^\star) \leq |p_k\, p_A| + \mathrm{sim}_C.$$

We therefore have the following crucial inequality:

$$k + H(A) < H(s^\star) \leq H(k) + H(A) + \mathrm{sim}_C.$$

This implies

$$k < H(k) + \mathrm{sim}_C = O(\log k),$$

which can only be true for finitely many values of k. Pick c_F to be a k that violates this inequality. It follows that $s^\star$ cannot exist for $k = c_F$. The theorem is therefore proved. **Q.E.D.**

5. Set Enumeration Complexity

Following CHAITIN [7, 8], we extend the formalism of Section 2 from finite computations with a single output to infinite computations with an infinite amount of output. Consider a new class of computers, computers that never halt, and which we shall refer to as enumeration computers, or e-computers for short. An e-computer C is given by a total recursive function that maps its program p into the recursively-enumerable set of bit strings $C(p)$. C must request each bit of the program p, and cannot run off the end of p, so p is self-delimiting. But p's total extent now only emerges in the limit of infinite time.

The complexity $H_C(S)$ of the set S based on the e-computer C is the size in bits $|p|$ of the smallest program p for enumerating S with C:

$$H_C(S) = \min_{C(p)=S} |p|.$$

The probability $P_C(S)$ of the set S based on the e-computer C is the probability that C enumerates S if each bit of the program p is produced by an independent toss of a fair coin:

$$P_C(S) = \sum_{C(p)=S} 2^{-|p|}.$$

Define a universal e-computer U_{e} as follows:

$$U_{\mathrm{e}}(\overbrace{000\cdots000}^{i\ 0\text{'s}}1p) = C_i(p).$$

Here C_i is the e-computer with GÖDEL number i, i.e., the ith e-computer. Hence

$$H_{U_e}(S) \leq H_{C_i}(S) + (i+1)$$

and

$$P_{U_e}(S) \geq P_{C_i}(S)\, 2^{-(i+1)}$$

for all sets S. The general definition of a universal e-computer U_e is that it has the property that for each e-computer C there is a prefix σ_C such that

$$U_e(\sigma_C\, p) = C(p)$$

for all p. I.e., the prefix σ_C tells U_e how to simulate C. Hence for each e-computer C there is a constant $\mathrm{sim}_C = |\sigma_C|$ (the cost in bits of **sim**ulating C) such that

$$H_{U_e}(S) \leq H_C(S) + \mathrm{sim}_C$$

and

$$P_{U_e}(S) \geq P_C(S)\, 2^{-\mathrm{sim}_C}$$

for all sets S. The universal e-computer U_e we defined above satisfies this definition with $\mathrm{sim}_{C_i} = i + 1$. We pick this particular universal e-computer U_e as our standard one and define the e-complexity[2] $H_e(S)$ to be $H_{U_e}(S)$, and the enumeration probability $P_e(S)$ to be $P_{U_e}(S)$:

$$H_e(S) = H_{U_e}(S), \qquad P_e(S) = P_{U_e}(S).$$

In summary, the e-complexity $H_e(S)$ of a recursively-enumerable set S is the size in bits $|p|$ of the smallest computer program p that makes our standard universal e-computer U_e enumerate the set S. $P_e(S)$ is the probability that our standard universal e-computer U_e enumerates the set S if each bit of the program p is produced by an independent toss of a fair coin.

6. Discussion

The programs p_A and p_B for enumerating two sets A and B can be concatenated. More precisely, the bits in the two programs p_A and p_B

[2] In full, the "enumeration complexity."

can be **intertwined** or merged in the order that they are read by two copies of the universal e-computer U_e running in parallel and sharing a single program bit stream. Thus e-complexity is additive, because the size of the intertwined bit string $p_A \oplus p_B$ is the sum of the sizes of the original strings p_A and p_B.

Let's show some applications of intertwining. Define the e-complexity of a function f to be the e-complexity of the graph of f, which is the set of all ordered pairs $\langle x, f(x)\rangle$. By intertwining,

$$H_e(\Big\{ f(x) : x \in X \Big\}) < H_e(f) + H_e(X) + c.$$

Here is the cartesian product of two sets:

$$H_e(\Big\{ \langle x, y\rangle : x \in A, y \in B \Big\}) < H_e(A) + H_e(B) + c.$$

Two other examples of intertwining:

$$H_e(A \cap B) < H_e(A) + H_e(B) + c,$$

and

$$H_e(A \cup B) < H_e(A) + H_e(B) + c.$$

Here is a horse of a different color:

$$H(\varphi(\psi(x))) < H_e(\varphi) + H_e(\psi) + H(x) + c.$$

7. Exhibiting Complex Objects

While a minimal-size program for the computer U tells us its size as well as its output

$$H(x, H(x)) = H(x) + O(1),$$

this is not the case with a minimal-size program for the e-computer U_e. Instead we only get its size in the limit from below:

$$H_e(X, \{0, 1, 2, \ldots, H_e(X)\}) = H_e(X) + O(1).$$

(It is annoying to have to define H and $H_{\rm e}$ for multiple arguments. Intuitively, one simply computes several outputs simultaneously.[3]) In the proof that

$$A \vdash_F H(s) > n \quad \Longrightarrow \quad n < H(A) + c_F$$

in Section 4, it is important that C knows $|p_A| = H(A)$ as well as A. So it looks like we cannot prove that

$$\text{“}H(s) > n\text{”} \in T \quad \Longrightarrow \quad n < H_{\rm e}(T) + c.$$

Surprisingly, everything works anyway.

Theorem B: *Consider a formal system consisting of a recursively enumerable set T of theorems. The formal system T cannot exhibit a specific string with complexity $> H_{\rm e}(T)+c$. More precisely, if a theorem of the form “$H(s) > n$” is in T only if it is true, then “$H(s) > n$” is in T only if $n < H_{\rm e}(T) + c$.*

Proof: Consider a special-purpose computer C that is given as its program a minimal-size program p_k for the singleton set $\{k\}$ of the natural number k appropriately intertwined with a minimal-size program p_T for the set of theorems T. Here

$$U_{\rm e}(p_k) = \{k\}, \qquad |p_k| = H_{\rm e}(\{k\}),$$

and

$$U_{\rm e}(p_T) = T, \qquad |p_T| = H_{\rm e}(T).$$

When C is given p_k intertwined with p_T it does the following. C runs p_k and p_T in parallel on $U_{\rm e}$ to determine k and enumerate the set of theorems T. As C enumerates T, C keeps track of the number ρ of bits of p_T that it has read. At some point C will find k. Thereafter, C continually checks T until C finds a theorem of the form

$$\text{“}H(s^\star) > j\text{”}$$

[3]Here are some more formal definitions. For H, one can compute a tuple $\langle x, y, z, \ldots\rangle$. The tuple mapping is a computable one-to-one correspondence between bit strings and singletons, pairs, triples, ... of bit strings. For $H_{\rm e}$, one can enumerate several sets $A, B, C, \ldots$ by prefixing the elements of each with a different prefix:

$$\{1x : x \in A\} \cup \{01x : x \in B\} \cup \{001x : x \in C\} \cup \cdots$$

asserting that a specific string $s^\star$ has complexity greater than a specific natural number j that is greater than or equal to k plus the current value of ρ:

$$j \geq k + \rho.$$

C then outputs the string $s^\star$ and halts. It is possible that not all bits of p_k and p_T have been read by C; unread bits of p_k and p_T are not actually included in the intertwined program for C. Let p'_k and p'_T be the portions of p_k and p_T that are actually read. Thus the final value of ρ must satisfy

$$\rho = |p'_T| \leq H_{\mathrm{e}}(T).$$

In summary,

$$C(p'_k \oplus p'_T) = s^\star, \qquad \text{“}H(s^\star) > j\text{”} \in T, \qquad j \geq k + \rho.$$

Thus $s^\star$ must have the property that

$$H_C(s^\star) \leq |p'_k| + |p'_T| \leq H_{\mathrm{e}}(\{k\}) + \rho.$$

We therefore have the following crucial inequality:

$$k + \rho < H(s^\star) \leq H_{\mathrm{e}}(\{k\}) + \rho + \mathrm{sim}_C.$$

Note that

$$H_{\mathrm{e}}(\{k\}) + \rho + \mathrm{sim}_C < O(\log k) + \rho.$$

Hence

$$k + \rho < H(s^\star) < O(\log k) + \rho.$$

This implies

$$k < O(\log k),$$

which can only be true for finitely many values of k. Pick c to be a value of k that violates this inequality. For $k = c$, $s^\star$ cannot exist and C can never halt. Thus T cannot contain any theorem of the form

$$\text{“}H(s^\star) > j\text{”}$$

with

$$j \geq H_{\mathrm{e}}(T) + k,$$

because

$$H_e(T) + k \geq k + \rho$$

and C would halt. The theorem is therefore proved. **Q.E.D.**

Recall the setup in Theorem A: the fixed rules of inference F and the variable axiom A yield the set of theorems F_A. Let's apply Theorem B to the set of theorems F_A so that we can compare how powerful Theorems A and B are. Here is what we get:

$$\begin{array}{lll} \text{Theorem A:} & A \vdash_F H(s) > n & \Longrightarrow \quad n < H(A) + c_F. \\ \text{Theorem B:} & \text{“}H(s) > n\text{”} \in F_A & \Longrightarrow \quad n < H_e(F_A) + c. \end{array}$$

The e-complexity $H_e(F_A)$ is never more than c bits larger and can sometimes be up to $\approx \log_2 H(A)$ bits smaller than the complexity $H(A)$. This is because it is sometimes much easier to give the size $|p|$ of a program in the limit from below than to give the size of a program and then halt. It all boils down to the fact that $H_e(\{0, 1, 2, \ldots, |p|\})$ can be insignificant in comparison with $H(|p|)$ (see [7, Section 3]). Thus Theorem B is never weaker and sometimes is a little stronger than Theorem A.

Let's look at some other consequences of the method used to establish Theorem B.

We have seen that one can't exhibit complex strings. What about sets? One can't exhibit e-complex sets:

$$\text{“}H_e(U_e(p)) > n\text{”} \in T \quad \Longrightarrow \quad n < H_e(T) + c.$$

Here is a bound on what can be accomplished with a single axiom A and the rules of inference F:

$$A \vdash_F H(s) > n \quad \Longrightarrow \quad n < H(A) + H_e(F) + c.$$

Recall that Theorem A only asserted that

$$A \vdash_F H(s) > n \quad \Longrightarrow \quad n < H(A) + c_F.$$

Consider the set of theorems T derived from a set of axioms A using the rules of inference F. (Now F is a recursively enumerable

set of ordered pairs each consisting of a finite set of axioms and a consequence.) We have

$$H_{\rm e}(T) \leq H_{\rm e}(A) + H_{\rm e}(F) + c.$$

And we get the following bound on what can be accomplished with the set of axioms A and the rules of inference F:

$$A \vdash_F H(s) > n \quad \Longrightarrow \quad n < H_{\rm e}(A) + H_{\rm e}(F) + c.$$

8. Determining Bits of Ω

In the same spirit as Theorem B in Section 7, here is a new and improved version of the key theorem in [16, Chapter 8] that one can determine at most $n + c$ bits of the halting probability Ω with a formal system of complexity n.

Theorem C: *Consider a formal system consisting of a recursively enumerable set T of theorems. If the formal system T has the property that a theorem of the form*

"The nth bit of Ω is 0/1."

is in T only if it is true, then at most $H_{\rm e}(T)+c$ theorems of this form are in T. In other words, if the e-complexity of T is n, then T can enable us to determine the positions and values of at most $n + c$ scattered bits of Ω.

Proof: (By reductio ad absurdum.) Suppose on the contrary that for each k there is a formal system T that enables us to determine $H_{\rm e}(T) + k$ bits of Ω. We shall show that this contradicts the fact (see [16]) that Ω is a MARTIN-LÖF random real number.

Here is a way to produce a set of intervals A_k that covers Ω and has measure $\mu(A_k) \leq 2^{-k}$. I.e., a way that given k, we can enumerate a set of intervals A_k that includes Ω and whose total length is $\leq 2^{-k}$.

Start running for more and more time all possible programs p on the standard universal e-computer $U_{\rm e}$. If at any point we have read $\rho \leq |p|$ bits of a program p while enumerating its output $U_{\rm e}(p)$ and this output includes $\rho + k$ theorems of the form

"The nth bit of Ω is 0/1."

determining $\rho + k$ bits of Ω, then we do the following:

1. The $\rho + k$ theorems in $U_e(p)$ give us a set of intervals of total measure $2^{-(\rho+k)}$ that covers Ω. More precisely, this set of intervals covers Ω if $U_e(p)$ is a truthful formal system. We add these intervals to the set of intervals A_k.

2. We stop exploring this subtree of the tree of all possible programs p. In other words, we don't continue with the current program p nor do we examine any program whose first ρ bits are the same as the first ρ bits of p. This removes from further consideration all programs in an interval of length $2^{-\rho}$, i.e., a set of programs of measure $2^{-\rho}$.

For each k, the set of intervals A_k will have total measure $\mu(A_k) \leq 2^k$. Ω cannot be in all the A_k or Ω would not be MARTIN-LÖF random, which it most certainly is [16]. Therefore Ω is in only finitely many of the A_k. Let c be the first k for which Ω is not in A_k, i.e., such that we never find a way of determining $\rho + k$ bits of Ω with only ρ bits of axioms. **Q.E.D.**

9. A Different Proof

In Section 8 it was shown that a formal system of e-complexity n cannot determine the positions and values of more than $n+c$ bits of Ω (Theorem C). The proof is like an iceberg, because it depends on the theory systematically developed in the second half of my monograph [16]. Here is a rather different proof that is essentially self-contained.[4]

Theorem C: *A formal system T can enable us to determine the positions and values of at most $H_e(T) + c$ bits of Ω. In other words, if a theorem of the form*

"The nth bit of Ω is 0/1."

[4] An analogous situation occurs in elementary number theory. See the remarkably simple ZERMELO–HASSE–LORD CHERWELL proof of the unique prime factorization theorem in RADEMACHER and TOEPLITZ [17, p. 200].

is in T only if it is true, then at most $H_{\rm e}(T)+c$ theorems of this form are in T.

Proof #2: Consider the following special-purpose computer C. Given a program

$$\overbrace{000\cdots0001}^{k\text{ bits}}\,p_T\,x,$$

C does the following. First C reads the initial run of 0 bits and the 1 bit at its end. The total number of bits read is k. Then C starts running on $U_{\rm e}$ the remainder of its program, which begins with a minimal-size program p_T for enumerating T:

$$U_{\rm e}(p_T)=T, \qquad |p_T|=H_{\rm e}(T).$$

As C enumerates T, C counts the number of bits

$$\rho \le H_{\rm e}(T)$$

of p_T that it has read. The moment that C has enumerated enough theorems in the formal system T to find the values and positions of $\rho+2k$ bits of Ω, C stops enumerating the theorems of the formal system T. (Unread bits of p_T are not actually included in the program for C. Let p'_T be the portion of p_T that is actually read. Thus the final value of ρ equals $|p'_T|$ and may actually be **less than** $H_{\rm e}(T)$ if not all bits of the minimal-size program for enumerating T are needed.) Now C knows $\rho+2k$ bits of Ω. Next C determines the position n of the bit of Ω that it knows that is farthest from the decimal point. In other words, C finds the largest n in the first $\rho+2k$ theorems of the form

"The nth bit of Ω is 0/1."

in T, where $\rho=|p'_T|$ is the number of bits of p_T that are read. Consider the string Ω_n of the first n bits of Ω:

$$\Omega_n=\beta_1\beta_2\beta_3\cdots\beta_n.$$

From T, C has determined β_n and $\rho+2k-1$ other bits of Ω_n. To fill in the gaps, the remaining $n-\rho-2k$ bits of Ω_n are provided to C as the remainder of its program, x, which is exactly $n-\rho-2k$ bits long.

(For C's program to be self-delimiting, it is crucial that at this point C already knows exactly how long x is.) Now C knows the first n bits of Ω, and it outputs them and halts:

$$C(\overbrace{000\cdots0001}^{k \text{ bits}} p'_T\, x) = \Omega_n.$$

Note that the size of C's program is

$$|\overbrace{000\cdots0001}^{k \text{ bits}}| + |p'_T| + |x| = k + \rho + (n - \rho - 2k).$$

This of course simplifies as follows:

$$k + \rho + (n - \rho - 2k) = n - k.$$

Hence

$$H_C(\Omega_n) \leq n - k.$$

We therefore have the following crucial inequality:[5]

$$n - c' < H(\Omega_n) \leq n - k + \mathrm{sim}_C.$$

Hence

$$k < c' + \mathrm{sim}_C.$$

Taking

$$k = c' + \mathrm{sim}_C$$

we get a contradiction. Thus T cannot yield

$$\rho + 2k = \rho + 2(c' + \mathrm{sim}_C) \leq H_{\mathrm{e}}(T) + 2(c' + \mathrm{sim}_C)$$

bits of Ω. The theorem is proved with

$$c = 2(c' + \mathrm{sim}_C).$$

Q.E.D.

[5] It is easy to see that there is a computer C' such that

$$U(p) = \Omega_n \quad \Longrightarrow \quad C'(p) = \text{the first string } x \text{ with } H(x) > n.$$

Hence

$$H(\Omega_n) > n - \mathrm{sim}_{C'} = n - c'.$$

I.e., Ω is a CHAITIN random real. For the details, see the chapter "Chaitin's Omega" in GARDNER [29], or the proof that Ω is CHAITIN random in [16].

10. Diophantine Equations

Now let's convert Theorem C into an incompleteness theorem about diophantine equations.[6]

We arithmetize Ω in two diophantine equations: one polynomial, the other exponential. Ω can be obtained as the limit of a computable sequence of rational numbers Ω_l:

$$\Omega = \lim_{l\to\infty} \Omega_l.$$

(Careful: in Section 9, Ω_l meant something else.) For example, let Ω_l be the sum of $2^{-|p|}$ taken over all programs p of size $\leq l$ that halt in time $\leq l$:

$$\Omega_l = \sum_{\substack{|p| \leq l \\ U(p) \text{ halts in time} \leq l}} 2^{-|p|}.$$

The methods of JONES and MATIJASEVIČ [10, 22, 31] enable one to construct the following:

1. A diophantine equation

$$P(k, l, x_1, x_2, x_3, \ldots) = 0$$

that has one or more solutions if the kth bit of Ω_l is a 1, and that has no solutions if the kth bit of Ω_l is a 0.

2. An exponential diophantine equation

$$L(k, l, x_2, x_3, \ldots) = R(k, l, x_2, x_3, \ldots)$$

that has exactly one solution if the kth bit of Ω_l is a 1, and that has no solutions if the kth bit of Ω_l is a 0.

Since in the limit of large l the kth bit of Ω_l becomes and remains correct, i.e., identical with the kth bit of Ω, it follows immediately that:

[6] My first information-theoretic incompleteness theorem about diophantine equations is stated without proof in the introduction of my 1974 paper [4]. A better one is mentioned in my 1982 paper [9, Section 4]. Finally, two papers [11, 12] and a book [16] give number theory their undivided attention.

1. There are infinitely many values of l for which the diophantine equation
$$P(k, l, x_1, x_2, x_3, \ldots) = 0$$
has a solution iff the kth bit of Ω is a 1.

2. The exponential diophantine equation
$$L(k, x_1, x_2, x_3, \ldots) = R(k, x_1, x_2, x_3, \ldots)$$
has infinitely many solutions iff the kth bit of Ω is a 1.

Consider the following questions:

1. For a given value of k, are there infinitely many values of l for which the diophantine equation
$$P(k, l, x_1, x_2, x_3, \ldots) = 0$$
has a solution?

2. For a given value of k, does the exponential diophantine equation
$$L(k, x_1, x_2, x_3, \ldots) = R(k, x_1, x_2, x_3, \ldots)$$
have infinitely many solutions?

By Theorem C, to answer any n cases of the first question or any n cases of the second question requires a formal system of e-complexity $> n - c$.

For discussions of the significance of these information-theoretic incompleteness theorems, see [13, 15, 18–21, 23–30].

References

[1] G. J. Chaitin, "Computational complexity and Gödel's incompleteness theorem," *AMS Notices* **17** (1970), 672.

[2] G. J. Chaitin, "Computational complexity and Gödel's incompleteness theorem," *ACM SIGACT News* **9** (1971), 11–12.

[3] G. J. CHAITIN, "Information-theoretic computational complexity," *IEEE Transactions on Information Theory* **IT–20** (1974), 10–15.

[4] G. J. CHAITIN, "Information-theoretic limitations of formal systems," *Journal of the ACM* **21** (1974), 403–424.

[5] G. J. CHAITIN, "Randomness and mathematical proof," *Scientific American* **232**:5 (1975), 47–52.

[6] G. J. CHAITIN, "A theory of program size formally identical to information theory," *Journal of the ACM* **22** (1975), 329–340.

[7] G. J. CHAITIN, "Algorithmic entropy of sets," *Computers & Mathematics with Applications* **2** (1976), 233–245.

[8] G. J. CHAITIN, "Algorithmic information theory," *IBM Journal of Research and Development* **21** (1977), 350–359, 496.

[9] G. J. CHAITIN, "Gödel's theorem and information," *International Journal of Theoretical Physics* **22** (1982), 941–954.

[10] J. P. JONES and Y. V. MATIJASEVIČ, "Register machine proof of the theorem on exponential diophantine representation of enumerable sets," *Journal of Symbolic Logic* **49** (1984), 818–829.

[11] G. J. CHAITIN, "Randomness and Gödel's theorem," *Mondes en Développement* **54–55** (1986), 125–128.

[12] G. J. CHAITIN, "Incompleteness theorems for random reals," *Advances in Applied Mathematics* **8** (1987), 119–146.

[13] G. J. CHAITIN, "Randomness in arithmetic," *Scientific American* **259**:1 (1988), 80–85.

[14] G. J. CHAITIN, *Information, Randomness & Incompleteness—Papers on Algorithmic Information Theory,* 2nd Edition, Singapore: World Scientific (1990).

[15] G. J. CHAITIN, "Undecidability and randomness in pure mathematics," (transcript of a lecture delivered 28 September 1989 at a SOLVAY conference in Brussels), in [14, pp. 307–313].

[16] G. J. CHAITIN, *Algorithmic Information Theory,* 3rd Printing, Cambridge: Cambridge University Press (1990).

[17] H. RADEMACHER and O. TOEPLITZ, *The Enjoyment of Mathematics,* New York: Dover (1990).

[18] G. J. CHAITIN, "A random walk in arithmetic," *New Scientist* **125**:1709 (1990), 44–46. Reprinted in N. HALL, *The New Scientist Guide to Chaos,* Harmondsworth: Penguin (1991).

[19] J. L. CASTI, *Searching for Certainty,* New York: Morrow (1990).

[20] G. J. CHAITIN, "Number and randomness," (transcript of a lecture delivered 15 January 1991 at the Technical University of Vienna). In M. E. CARVALLO, *Nature, Cognition and System,* Vol. 3, Dordrecht: Kluwer (1992), in press.

[21] G. J. CHAITIN, "Le hasard des nombres," *La Recherche* **22**:232 (1991), 610–615.

[22] J. P. JONES and Y. V. MATIJASEVIČ, "Proof of the recursive unsolvability of Hilbert's tenth problem," *American Mathematical Monthly* **98** (1991), 689–709.

[23] D. RUELLE, *Chance and Chaos,* Princeton: Princeton University Press (1991).

[24] D. RUELLE, *Hasard et Chaos,* Paris: Odile Jacob (1991).

[25] L. BRISSON and F. W. MEYERSTEIN, *Inventer L'Univers,* Paris: Les Belles Lettres (1991).

[26] J. A. PAULOS, *Beyond Numeracy,* New York: Knopf (1991).

[27] J. D. BARROW, *Theories of Everything,* Oxford: Clarendon Press (1991).

[28] T. NØRRETRANDERS, *Mærk Verden,* Denmark: Gyldendal (1991).

[29] M. GARDNER, *Fractal Music, Hypercards and More...,* New York: Freeman (1992).

[30] P. DAVIES, *The Mind of God,* New York: Simon & Schuster (1992).

[31] C. SMORYŃSKI, *Logical Number Theory I,* Berlin: Springer-Verlag (1991).

Part II

Non-Technical Discussions

"A mathematical theory is not to be considered complete until you have made it so clear that you can explain it to the first man whom you meet on the street."

"This conviction of the solvability of every mathematical problem is a powerful incentive to the worker. We hear within us the perpetual call: There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no *ignorabimus* [we shall never know]."

—David Hilbert,
International Congress of Mathematicians,
Paris, 1900

"Wir müssen wissen! Wir werden wissen!"
[We **must** know! We **will** know!]

—David Hilbert,
Naturerkennen und Logik,
Königsberg, 1930

"One does not immediately associate with Hilbert's name any definite and important metamathematical result. Nevertheless, Hilbert will deservedly be called the father of metamathematics. For he is the one who created metamathematics as an independent being; he fought for its right to existence, backing it with his whole authority as a great mathematician. And he was the one who mapped out its future course and entrusted it with ambitions and important tasks."

—Alfred Tarski in
Constance Reid, *Hilbert*

ARENA PROGRAM ON 'NUMBERS' (BBC TV)

Produced by Fisher Dilke
Finished 19 April 1989
Broadcast Friday 19 January 1990

CHAITIN

Most people think that a computer is absolutely mechanical, reliable—it goes from step to step in a completely mechanical fashion. This may seem like a very surprising place to come up with unpredictability and randomness. Computers to be useful have to be as predictable, as unrandom, as possible.

There's an absolutely fundamental famous problem called the halting problem. The problem is to decide whether a computer program will ever halt.

Most people don't understand why this is a problem at first. If you take a computer program and you put it into a computer, and it halts, you know it's halted. If you want to decide if a program will halt in an hour, you run it for an hour, and it's either halted or it hasn't. If you want to decide whether it halts in a day, you run it for a day, and it either halts or it doesn't.

What turns out to be a tremendously fundamental conceptual problem—and this has been known since the 30's—is to decide if a

program will *ever* halt, where there's *no* limit on the time it takes.

Of course if a program does halt eventually, if we're very very patient we can find that out, by just running it. Maybe in a million years or in a billion years (I'm speaking now as a mathematician—this is all rather theoretical) we'll see that it halted.

What turns out to be the absolutely fundamental problem is to decide that a program that *doesn't* halt will never do it.

And then, instead of asking whether or not a program halts, you ask what is the probability that a program chosen at *random* will halt. That's when you get complete randomness. That's when I've shown you get complete absolute randomness, unpredictability and incomprehensibility.

DILKE

Is this in the ordinary arithmetic that people learn at school?

CHAITIN

That's a very good question.

Clearly, there's nothing more certain than the fact that two plus two is equal to four. I'm not saying that sometimes it will come out five and sometimes it's going to come out three. I'm only dealing with the whole numbers. Questions like this are clearly very easy to settle. This is probably the most solid and concrete part of mathematics.

Instead the first step is to mirror the halting problem. The same way that one asks whether or not a program ever halts, one can look at equations involving whole numbers and ask whether or not they have a solution.

That's the first step. That's a more abstract question.

If there is a solution for an equation, one can eventually discover that, by experimenting and trying different possibilities for the solution. The problem is to prove that there is no solution. That's equivalent to the halting problem, and escapes the power of mathematics in some cases.

But it doesn't give complete randomness.

What I've done is to go to a slightly more abstract question. That question is, to ask about an equation involving whole numbers, not whether or not it has a solution, but does it have an infinity of solutions or only a finite number of solutions (and no solution is a finite number of solutions).

If you construct the equations in the right way, and then you ask whether the number of solutions is finite or infinite, I can show that you get complete randomness. You get something that is completely incomprehensible, that is completely unpredictable, and that no matter how much cleverness a mathematician will apply, will forever be incomprehensible and show absolutely *no* pattern or structure.

Since this is rather unbelievable, I thought that it was important to actually write the equations down and show them to people, to make this randomness as tangible as possible. These equations turn out to be enormous. In fact the first one is two hundred pages long. I had to use a computer to write it out.

DILKE

So this calls for pessimism?

CHAITIN

No, I think it's wonderful! Who would have thought that the whole numbers had it in them to behave in this fascinating, rich, unexpected fashion! Who knows what else they're capable of doing! I think this is very exciting.

A RANDOM WALK IN ARITHMETIC

New Scientist 125, No. 1709 (24 March 1990), pp. 44–46. Reprinted in N. Hall, The New Scientist Guide to Chaos, Penguin, 1991.

Gregory Chaitin

God not only plays dice in physics but also in pure mathematics. Mathematical truth is sometimes nothing more than a perfect coin toss.

THE NOTION of randomness obsesses physicists today. To what extent can we predict the future? Does it depend on our own limitations? Or is it in principle impossible to predict the future? The question of predictability has a long history in physics. In the early 19th century, the classical deterministic laws of Isaac Newton led Pierre Simon de Laplace to believe that the future of the Universe could be determined

forever.

Then quantum mechanics came along. This is the theory that is fundamental to our understanding of the nature of matter. It describes very small objects, such as electrons and other fundamental particles. One of the controversial features of quantum mechanics was that it introduced probability and randomness at a fundamental level to physics. This greatly upset the great physicist Albert Einstein, who said that God did not play dice.

Then surprisingly, the modern study of nonlinear dynamics showed us that even the classical physics of Newton had randomness and unpredictability at its core. The theory of chaos, as the series of articles in *New Scientist* last year described, has revealed how the notion of randomness and unpredictability is beginning to look like a unifying principle.

It seems that the same principle even extends to mathematics. I can show that there are theorems connected with number theory that cannot be proved because when we ask the appropriate questions, we obtain results that are equivalent to the random toss of a coin.

My results would have shocked many 19th-century mathematicians, who believed that mathematical truths could always be proved. For example, in 1900, the mathematician, David Hilbert, gave a famous lecture in which he proposed a list of 23 problems as a challenge to the new century. His sixth problem had to do with establishing the fundamental universal truths, or axioms, of physics. One of the points in this question concerned probability theory. To Hilbert, probability was simply a practical tool that came from physics; it helped to describe the real world when there was only a limited amount of information available.

Another question he discussed was his tenth problem, which was connected with solving so-called "diophantine" equations, named after the Greek mathematician Diophantus. These are algebraic equations involving only whole numbers, or integers. Hilbert asked: "Is there a way of deciding whether or not an algebraic equation has a solution in whole numbers?"

Little did Hilbert imagine that these two questions are subtly related. This was because Hilbert had assumed something that was so basic to his thinking that he did not even formulate it as a question in

his talk. That was the idea that every mathematical problem has a solution. We may not be bright enough or we may not have worked long enough on the problem but, in principle, it should be possible to solve it—or so Hilbert thought. For him, it was a black or white situation.

It seems now that Hilbert was on shaky ground. In fact, there is a connection between Hilbert's sixth question dealing with probability theory and his tenth problem of solving algebraic equations in whole numbers that leads to a surprising and rather chilling result. That is: randomness lurks at the heart of that most traditional branch of pure mathematics, number theory.

Clear, simple mathematical questions do not always have clear answers. In elementary number theory, questions involving diophantine equations can give answers that are completely random and look grey, rather than black or white. The answer is random because the only way to prove it is to postulate each answer as an additional independent axiom. Einstein would be horrified to discover that not only does God play dice in quantum and classical physics but also in pure mathematics.

Where does this surprising conclusion come from? We have to go back to Hilbert. He said that when you set up a formal system of axioms there should be a mechanical procedure to decide whether a mathematical proof is correct or not, and the axioms should be consistent and complete. If the system of axioms is consistent, it means that you cannot prove both a result and its contrary. If the system is complete, then you can also prove any assertion to be true or false. It follows that a mechanical procedure would ensure that all mathematical assertions can be decided mechanically.

There is a colourful way to explain how this mechanical procedure works: the so-called "British Museum algorithm." What you do—it cannot be done in practice because it would take forever—is to use the axiom system, set in the formal language of mathematics, to run through all possible proofs, in order of their size and lexicographic order. You check which proofs are correct—which ones follow the rules and are accepted as valid. In principle, if the set of axioms is consistent and complete, you can decide whether any theorem is true or false. Such a procedure means that a mathematician no longer needs ingenuity or inspiration to prove theorems. Mathematics becomes mechanical.

Of course, mathematics is not like that. Kurt Gödel, the Austrian logician, and Alan Turing, the father of the computer, showed that it is impossible to obtain both a consistent and complete axiomatic theory of mathematics and a mechanical procedure for deciding whether an arbitrary mathematical assertion is true or false, or is provable or not.

Gödel was the first to devise the ingenious proof, couched in number theory, of what is called the incompleteness theorem (see "The incompleteness of arithmetic," *New Scientist,* 5 November 1987). But I think that the Turing version of the theorem is more fundamental and easier to understand. Turing used the language of the computer—the instructions, or program, that a computer needs to work out problems. He showed that there is no mechanical procedure for deciding whether an arbitrary program will ever finish its computation and halt.

To show that the so-called halting problem can never be solved, we set the program running on a Turing machine, which is a mathematical idealisation of a digital computer with no time limit. (The program must be self-contained with all its data wrapped up inside the program.) Then we simply ask: "Will the program go on forever, or at some point will it say 'I'm finished' and halt?"

Turing showed that there is no set of instructions that you can give the computer, no algorithm, that will decide if a program will ever halt. Gödel's incompleteness theorem follows immediately because if there is no mechanical procedure for deciding the halting problem, then there is no complete set of underlying axioms either. If there were, they would provide a mechanical procedure for running through all possible proofs to show whether programs halt—although it would take a long time, of course.

To obtain my result about randomness in mathematics, I simply take Turing's result and just change the wording. What I get is a sort of a mathematical pun. Although the halting problem is unsolvable, we can look at the probability of whether a randomly chosen program will halt. We start with a thought experiment using a general purpose computer that, given enough time, can do the work of any computer—the universal Turing machine.

Instead of asking whether or not a specific program halts, we look at the ensemble of all possible computer programs. We assign to each computer program a probability that it will be chosen. Each bit of

information in the random program is chosen by tossing a coin, an independent toss for each bit, so that a program containing so many bits of information, say, N bits, will have a probability of 2^{-N}. We can now ask what is the total probability that those programs will halt. This halting probability, call it Ω, wraps up Turing's question of whether a program halts into one number between 0 and 1. If the program never halts, Ω is 0; if it always halts, Ω is 1.

In the same way that computers express numbers in binary notation, we can describe Ω in terms of a string of 1s and 0s. Can we determine whether the Nth bit in the string is a 0 or a 1? In other words, can we compute Ω? Not at all. In fact, I can show that the sequence of 0s and 1s is random using what is called algorithmic information theory. This theory ascribes a degree of order in a set of information or data according to whether there is an algorithm that will compress the data into a briefer form.

For example, a regular string of 1s and 0s describing some data such as 0101010101 ... which continues for 1000 digits can be encapsulated in a shorter instruction "repeat 01 500 times." A completely random string of digits cannot be reduced to a shorter program at all. It is said to be algorithmically incompressible.

My analysis shows that the halting probability is algorithmically random. It cannot be compressed into a shorter program. To get N bits of the number out of a computer, you need to put in a program at least N bits long. Each of the N bits of Ω is an irreducible independent mathematical fact, as random as tossing a coin. For example, there are as many 0s in Ω as 1s. And knowing all the even bits does not help us to know any of the odd bits.

My result that the halting probability is random corresponds to Turing's assertion that the halting problem is undecidable. It has turned out to provide a good way to give an example of randomness in number theory, the bedrock of mathematics. The key was a dramatic development about five years ago. James Jones of the University of Calgary in Canada and Yuri Matijasevič of the Steklov Institute of Mathematics in Leningrad discovered a theorem proved by Edouard Lucas in France a century ago. The theorem provides a particularly natural way to translate a universal Turing machine into a universal diophantine equation that is equivalent to a general purpose computer.

I thought it would be fun to write it down. So with the help of a large computer I wrote down a universal-Turing-machine equation. It had 17 000 variables and went on for 200 pages.

The equation is of a type that is referred to as "exponential diophantine." All the variables and constants in it are non-negative integers, 0, 1, 2, 3, 4, 5, and so on. It is called "exponential" because it contains numbers raised to an integer power. In normal diophantine equations the power has to be a constant. In this equation, the power can be a variable. So in addition to having X^3, it also contains X^Y.

To convert the assertion that the halting probability Ω is random into an assertion about the randomness of solutions in arithmetic, I need only to make a few minor changes in this 200-page universal-Turing-machine diophantine equation. The result, my equation exhibiting randomness, is also 200 pages long. The equation has a single parameter, the variable N. For any particular value of this parameter, I ask the question: "Does my equation have a finite or infinite number of whole-number solutions?" Answering this question turns out to be equivalent to calculating the halting probability. The answer "encodes" in arithmetical language whether the Nth bit of Ω is a 0 or a 1. If the Nth bit of Ω is a 0, then my equation for that particular value of N has a finite number of solutions. If the Nth bit of the halting probability Ω is a 1, then this equation for that value of the parameter N has an infinite number of solutions. Just as the Nth bit of Ω is random—an independent, irreducible fact like tossing a coin—so is deciding whether the number of solutions of my equation is finite or infinite. We can never know.

To find out whether the number of solutions is finite or infinite in particular cases, say, for k values of the parameter N, we would have to postulate the k answers as k additional independent axioms. We would have to put in k bits of information into our system of axioms, so we would be no further forward. This is another way of saying that the k bits of information are irreducible mathematical facts.

I have found an extreme form of randomness, of irreducibility, in pure mathematics—in a part of elementary number theory associated with the name of Diophantus and which goes back 2000 years to classical Greek mathematics. Hilbert believed that mathematical truth was black or white, that something was either true or false. I think that my

work makes things look grey, and that mathematicians are joining the company of their theoretical physics colleagues. I do not think that this is necessarily bad. We have seen that in classical and quantum physics, randomness and unpredictability are fundamental. I believe that these concepts are also found at the very heart of pure mathematics.

Gregory Chaitin is a member of the theoretical physics group at the Thomas J. Watson Research Center in Yorktown Heights, New York, which is part of IBM's research division.

Further Reading G. J. Chaitin, *Information, Randomness & Incompleteness,* Second Edition, World Scientific, Singapore, 1990; G. J. Chaitin, *Algorithmic Information Theory,* third printing, Cambridge University Press, Cambridge, 1990.

NUMBER AND RANDOMNESS Algorithmic Information Theory— Latest Results on the Foundations of Mathematics

In M. E. Carvallo, Nature, Cognition and System, Vol. 3, Kluwer, 1993

Gregory J. Chaitin
IBM Research Division, New York

Lecture given Tuesday 15 January 1991 in the Technical University of Vienna, at a meeting on "Mathematik und Weltbild," immediately following a lecture by Prof. Hans-Christian Reichel on "Mathematik

und Weltbild seit Kurt Gödel." The lecture was videotaped; this is an edited transcript.

It is a great pleasure for me to be speaking today here in Vienna. It's a particularly great pleasure for me to be here because Vienna is where the great work of Gödel and Boltzmann was done, and their work is a necessary prerequisite for my own ideas. Of course the connection with Gödel was explained in Prof. Reichel's beautiful lecture. What may be a bit of a surprise is the name of Boltzmann. So let me talk a little bit about Boltzmann and the connection with my own work on randomness in mathematics.

You see, randomness in mathematics sounds impossible. If anything, mathematics is where there is least randomness, where there is most certainty and order and pattern and structure in ideas. Well, if you go back to Boltzmann's work, Boltzmann also put together two concepts which seem contradictory and invented an important new field, statistical mechanics.

I remember as a student reading those two words "statistical mechanics," and thinking how is it possible—aren't these contradictory notions? Something mechanical is like a machine, predictable. What does statistics have to do with mechanics? These seem to be two widely separate ideas. Of course it took great intellectual courage on Boltzmann's part to apply statistical methods in mechanics, which he did with enormous success.

Statistical mechanics now is a fundamental part of physics. One forgets how controversial Boltzmann's ideas were when they were first proposed, and how courageous and imaginative he was. Boltzmann's work in many ways is closely connected to my work and to Gödel's work, which may be a little surprising.

I'm trying to understand Gödel's great incompleteness theorem, I'm obsessed with that. I believe that the full meaning of Gödel's result can be obtained by taking Boltzmann's ideas and applying them to mathematics and to mathematical logic. In other words, I propose a thermodynamical approach, a statistical-mechanics approach, to understand-

ing the foundations of mathematics, to understanding the limitations and possibilities of mathematical reasoning.

Thermodynamics and statistical mechanics talk about what can be accomplished by machines, by heat engines, by steam engines, by physical systems. My approach to understanding the full implications of Gödel's work is mathematically analogous to the ideas of thermodynamics and Boltzmann and statistical mechanics. You might say, not completely seriously, that what I'm proposing is "thermodynamical epistemology!"

What led me to all this? Well, I was absolutely fascinated by Gödel's theorem. It seemed to me that this had to be the most profound result, the most mysterious result, in mathematics. And I think that a key question that one should ask when one reads Gödel's enormously surprising result, is, well, how seriously should one take it?! It's clearly an enormously startling and unexpected result, but consider the mathematician working on normal mathematical questions. What is the meaning of Gödel for daily work in mathematics? That's the question I'd like to ask.

Gödel explicitly constructed an arithmetical assertion that is true but not provable within the system of *Principia Mathematica* of Russell and Whitehead. It's a very strange assertion. It's an enormously clever assertion: It says of itself, "I'm unprovable!" This is not the kind of assertion that one normally is interested in as a working mathematician. But of course a great part of Gödel's genius was to take such a bizarre question very seriously and also to clothe it as an arithmetical question. With the years this has led to the work on Hilbert's tenth problem, which is an even more straight-forward arithmetical incompleteness result inspired by Gödel's fundamental path-breaking work.

Let me make my question more explicit. There are many problems in the theory of numbers that are very simple to state. Are there an infinity of twin primes, primes that are two odd numbers separated by one even number? That question goes back a long way. A question which goes back to the ancient Greeks is, are there infinitely many even

perfect numbers, and are there any odd perfect numbers?

Is it possible that the reason that these results have not been proven is because they are unprovable from the usual axioms? Is the significance of Gödel's incompleteness theorem that these results, which no mathematician has been able to prove, but which they believe in, should be taken as new axioms? In other words, how pervasive, how common, is the incompleteness phenomenon?

If I have a mathematical conjecture or hypothesis, and I work for a week unsuccessfully trying to prove it, I certainly do not have the right to say, "Well obviously, invoking Gödel's incompleteness theorem, it's not my fault: Normal mathematical reasoning cannot prove this—we must add it as a new axiom!" This extreme clearly is *not* justified.

When Gödel produced his great work, many important mathematicians like Hermann Weyl and John von Neumann took it as a personal blow. Their faith in mathematical reasoning was severely questioned. Hermann Weyl said it had a negative effect on his enthusiasm for doing mathematics. Of course it takes enormous enthusiasm to do good research, because it's so difficult. With time, however, people have gone to the other extreme, saying that in practice incompleteness has nothing to do with normal, every-day mathematics.

So I think it's a very serious question to ask, "How common is incompleteness and unprovability?" Is it a very bizarre pathological case, or is it pervasive and quite common? Because if it is, perhaps we *should* be doing mathematics quite differently.

One extreme would be experimental number theory, to do number theory as if it were physics, where one looks for conjectures by playing with prime numbers with a computer. For example, a physicist would say that the Riemann ζ hypothesis is amply justified by experiment, because many calculations have been done, and none contradicts it. It has to do with where the zeros of a function called the Riemann ζ function are. Up to now all the zeros are where Riemann said they were, on a certain line in the complex plane.

This conjecture has rich consequences. It explains a lot of empirically verified properties of the distribution of prime numbers. So it's a very useful conjecture. Now in physics, to go from Newtonian physics to relativity theory, to go from relativity theory to quantum mechanics, one adds new axioms. One needs new axioms to understand new fields

of human experience.

In mathematics one doesn't normally think of doing this. But a physicist would say that the Riemann hypothesis should be taken as a new axiom because it's so rich and fertile in consequences. Of course, a physicist has to be prepared to throw away a theory and say that even though it looked good, in fact it's contradicted by further experience. Mathematicians don't like to be put in that position.

These are very difficult questions: How should one do mathematics? Should number theory be considered an experimental science like physics? Or should we forget about Gödel's result in our everyday work as mathematicians? There are many possibilities in this spectrum.

I think these are very difficult questions. I think it will take many years and many people to understand this fully. But let me tell you my tentative conclusion based on my "thermodynamical" approach. It's really an information-theoretic approach: The work of Boltzmann on statistical mechanics is closely connected intellectually with the work of Shannon on information theory and with my own work on *algorithmic* information theory. There's a clear evolutionary history connecting these ideas.

My approach is to measure how much information there is in a set of axioms, to measure how much information there is in a theorem. In certain circumstances I can show that if you have five pounds of axioms, only five pounds, but here is a ten-pound theorem, well this theorem is too big, it weighs too much to get from only five pounds of axioms.

Of course, I actually use an information-theoretic measure related to the Boltzmann entropy concept. Boltzmann would recognize some of the formulas in my papers, amazingly enough, because the interpretation is quite different: it involves computers and program size. But some of the formulas are identical. In fact, I like to use H for the same reason that Shannon used H, in honor of the Boltzmann H function, the H function dear to the heart of statistical physicists. (Of course, there's also a Hamiltonian H function, which is something else.)

The incompleteness phenomenon that Gödel discovered seems very

natural from my information-theoretic point of view. You see, there is no self-reference. Gödel's incredibly clever proof skirts very very close to paradox. I was fascinated by it. I was also very disturbed by it as a child when I started thinking about all this.

If one measures information, then it seems natural to think, that if you want to get more information out, sometimes you have to put more information in. A physicist would say that it's natural that if one wants to encompass a wider range of mathematical experience, one needs to add additional axioms. To a physicist that doesn't seem outrageous. To a mathematician it's quite questionable and controversial.

So the point of view of algorithmic information theory suggests that what Gödel found is not an isolated singularity. The information-theoretic point of view suggests that Gödel's incompleteness phenomenon is very natural, pervasive and widespread. If this is true, perhaps we *should* be doing mathematics a little bit differently and a little bit more like physics is done.

Physicists always seem very pleased when I say this, and mathematicians don't seem at all pleased.

These are very difficult questions. I'm proposing this point of view, but by no means is it established. I think that one needs to study all this a lot more.

In summary, let me tell a story from ten years ago, from 1979, which was the centenary of Einstein's birth. There were many meetings around the world celebrating this occasion. And at one of them in New York I met a well-known physicist, John Wheeler. I went up to Wheeler and I asked him, "Prof. Wheeler, do you think there's a connection between Gödel's incompleteness theorem and the Heisenberg uncertainty principle?" Actually, I'd heard that he *did,* so I asked him, "What connection do you think there is between Gödel's incompleteness theorem and Heisenberg's uncertainty principle?"

This is what Wheeler answered. He said, "Well, one day I was at the Institute for Advanced Study, and I went to Gödel's office, and there was Gödel..." I think Wheeler said that it was winter and Gödel

had an electric heater and had his legs wrapped in a blanket.

Wheeler said, "I went to Gödel, and I asked him, 'Prof. Gödel, what connection do you see between your incompleteness theorem and Heisenberg's uncertainty principle?' " I believe that Wheeler exaggerated a little bit now. He said, "And Gödel got angry and threw me out of his office!" Wheeler blamed Einstein for this. He said that Einstein had brain-washed Gödel against quantum mechanics and against Heisenberg's uncertainty principle!

In print I recently saw a for-the-record version of this anecdote,[1] which probably is closer to the truth but is less dramatic. It said, not that Wheeler was thrown out of Gödel's office, but that Gödel simply did not want to talk about it since he shared Einstein's disapproval of quantum mechanics and uncertainty in physics. Wheeler and Gödel then talked about other topics in the philosophy of physics, and about cosmology.

There is some little-known work of Gödel connected with general relativity, some very interesting work, about universes where the past and the future is a loop, and you can travel into your past by going around. That's called a Gödel universe. It's a little-known piece of work that shows the stamp of Gödel's originality and profundity.

Okay, so what was the final conclusion of all this? I went up to Wheeler at this Einstein centenary meeting, and I asked him this question. Wheeler told me that he asked Gödel the same question, and Gödel didn't answer Wheeler's question, and Wheeler never answered my question! So I'm going to answer it!

I'll tell you what I think the connection really is between Gödel's incompleteness theorem and Heisenberg's uncertainty principle. To answer the question I want to make it a broader question. I would like to tell you what I think the connection is between incompleteness and *physics.*

I think that at the deepest level the implication of Gödel's incompleteness theorem is as I said before that mathematics should be pursued more in the spirit of physics, that that's the connection. I see some negative reactions from the audience! Which doesn't surprise me!

[1] Jeremy Bernstein, *Quantum Profiles,* Princeton University Press, 1991, pp. 140–141.

Of course this is a difficult question and it's quite controversial. But that's what my work using an information-theoretic approach to Gödel suggests to me.

Number theory has in fact been pursued to a certain extent in the spirit of an experimental science. One could almost imagine a journal of experimental number theory. For example, there are papers published by number theorists which are, mathematicians say, "modulo the Riemann hypothesis." That is to say, they're taking the Riemann hypothesis as an axiom, but instead of calling it a new axiom they're calling it a hypothesis.

There are many examples of how this information-theoretic point of view yields incompleteness results. I think the most interesting one is my recent work on randomness in arithmetic, which I haven't really referred to yet in my talk.

A fundamental question that many of us wonder about, especially as teenagers—that's an age particularly well-suited for fundamental questions—is the question, "To what extent can the universe be comprehended by the human mind?" Is the universe ordered? Is there chaos and randomness? Are there limits in principle to what we will ever be able to understand?

Hilbert stated very beautifully that he didn't believe that there were limits to what the human mind could accomplish in mathematics. He believed that every question could be resolved: either shown to be true or false. We might not be able to ever do it, but he believed that in principle it was possible. Any clear mathematical question would have a clear resolution via a mathematical proof. Of course, Gödel showed that this is not the case.

But it's really a more general question. Can the universe be comprehended, the physical universe as well as the universe of mathematical experience? That's a broader question.

To what extent can all this be comprehended by the human mind? We know that it cannot be completely comprehended because of Gödel's work. But is there some way of getting a feeling for how much can be

comprehended? Again it boils down to that.

When I was a student at the university, I totally believed in science. But my faith in science was tried by the work I had to do in experimental physics laboratories. The experiments were difficult. It was hard for me to get good results. I'm sure some of you are excellent experimentalists. There are people who have a natural talent for doing physics experiments like there are people who have a natural talent for growing flowers. But for me, the physics laboratory was a difficult experience and I began to marvel that scientists had been able to create modern science in spite of the fact that Nature does not give a clear answer to questions that we ask in the laboratory. It's very difficult to get a clear answer from Nature as to how the world works.

So I asked myself, what is it that is the most convincing evidence, in our normal daily experience, that the universe can be comprehended, that there is law and order and predictability rather than chaos and arbitrary things which cannot be predicted and cannot be comprehended? In my experience I would say that what most convinces me in science and predictability and the comprehensibility of the universe is, you'll laugh, *the computer!*

I'm not referring now to the computer as an industrial gadget. I think the computer is really amazing not because of its practical usefulness, but because of the fact that *it works!* To get a physical system to behave so predictably over such long periods, over very extended calculations, is amazing when one thinks about it.

I've done calculations which involved billions[2] of successive operations each of which had to be accurately derived from the preceding ones. Billions of steps each of which depended on the preceding ones. I had ways of suspecting or predicting the final result or some characteristic of it, and it worked! It's really rather amazing. Of course, it doesn't always work, because the machine breaks down, or the programmer makes a mistake. But it works a lot of the time. And if one runs a program several times one usually gets the same answers.

It's really amazing when one thinks how many steps the machine is doing and how this chain of causal events *is* predictable and *is* understandable. That's the job of the computer engineer, to find physical

[2] 10^9

principles that are as predictable as possible, that give him a physical way to model the predictability of mathematics. Because computers are actually mathematical machines, that is what they really are. At least a mathematician might say that.

So the computer is a wonderful example of predictability and a case where the physical behavior of a big chunk of the universe is very understandable and very predictable and follows definite laws. I don't know the detailed laws of how a transistor works. But the overall behavior of the system is amazingly comprehensible and predictable. Otherwise one would not use computers. They would be absolutely useless.

Now it may seem strange that starting with the computer one can construct what I believe to be a very dramatic example of randomness. This is an idea I got from the work of Turing, which in turn was inspired by the work of Gödel, both of which of course were responses to questions that Hilbert asked.

Turing asks, can one decide if a computer program will ever halt, if it will ever stop running? Turing took Cantor's diagonal argument from set theory and used it to show that there is no mechanical procedure for deciding if a computer program will ever halt.

Well, if one makes a small change in this, in Turing's theorem that the halting *problem* is undecidable, one gets my result that the halting *probability* is algorithmically random or irreducible mathematical information. It's a mathematical pun!

The problem with this theorem is of course that in doing everyday mathematics one does not worry about halting probabilities or halting problems. So I had the same problem that Gödel had when he was thinking about mathematical assertions which assert of themselves that they're unprovable. My problem was how to take this bizarre notion of a halting probability and convert it into an arithmetical assertion.

It turns out that one *can* do this: One can exhibit a way to toss a coin with whole numbers, with the integers, which are the bedrock of mathematics. I can show that in some areas of arithmetic there is

complete randomness!

Don't misunderstand. I was interviewed on a BBC TV program. A lot of people in England think I said that $2 + 2$ is sometimes 4, sometimes 5, and sometimes 3, and they think it's very funny! When I say that there is randomness in arithmetic I'm certainly not saying that $2 + 2$ is sometimes 3 and sometimes 5. It's not that kind of randomness. That is where mathematics is as certain and as black and white as possible, with none of the uncertainties of physics.

To get complete randomness takes two steps.

The first step was really taken by Turing and is equivalent to Hilbert's tenth problem posed in 1900. One doesn't ask if $2 + 2 = 4$ (we know the answer!). One asks if an algebraic equation involving only whole numbers, integers, has a solution or not.

Matijasevič showed in 1970 that this problem, Hilbert's tenth problem, is equivalent to Turing's theorem that the halting problem is undecidable: Given a computer program one can construct a diophantine equation, an algebraic equation in whole numbers, that has a solution if and only if the given computer program halts. Conversely, given a diophantine equation, an algebraic equation involving only whole numbers, one can construct a computer program that halts if and only if the given diophantine equation has a solution.

This theorem was proven by Matijasevič in 1970, but intellectually it can be traced directly back to the 1931 incompleteness theorem of Gödel. There were a number of people involved in getting this dramatic 1970 result. It may be viewed as Gödel's original 1931 result restated in much simpler arithmetical terms.

Unfortunately it turns out that this doesn't give complete randomness; it only gives partial randomness.

I'll now speak information-theoretically. Consider N cases of Hilbert's tenth problem. You ask, does the equation have a solution or not for N different equations? The worst would be if that were N bits of information, because each answer is independent. It turns out that it is only order of $\log_2 N$ bits of information, because the answers are not at all independent. That's very easy to see, but I can't go into it.

So what does one do to get completely independent mathematical facts in elementary arithmetic? It's very simple. One goes a step farther: Instead of taking the halting problem and making it into the

question of whether a diophantine equation has a solution or not, one takes my halting probability, and makes it into the question of whether a diophantine equation has a finite or an infinite number of solutions.

If the equations are constructed properly, whether they have a finite or an infinite number of solutions is completely random. In fact, a single equation with a parameter will do. One takes the parameter to be 1, 2, 3, 4, 5, ... and one gets a series of derived equations from the original equation by fixing the value of the parameter. For each of these derived equations one asks: "Is there a finite or an infinite number of solutions?" I can construct this equation in such a way that the answers to this question are independent irreducible mathematical facts.

So that is how you use arithmetic to toss a coin, to give you randomness.

By the way, this equation turns out to be about 200 pages long and has 17,000 variables, and it's fun to calculate it. But one doesn't do it by hand! One does it with a computer. A computer is essential to be able to exhibit this equation.

It is an infinite series of equations really, each of which has a different value of the parameter. We ask whether each of the equations has a finite or an infinite number of solutions. Exactly what does it mean to say that these are irreducible mathematical facts?

Well, how does one reduce mathematical facts? To *axioms,* to postulates! And the inverse of the reduction is to prove a theorem, I mean, to expand axioms into theorems. The traditional notion of mathematics is that a small finite set of axioms can give us all of mathematics, all mathematical truths. That was the pre-Gödel notion that Hilbert believed in.

So in a sense what we're doing is we're compressing a lot of mathematical facts enormously, into a small set of axioms. Or actually, we're expanding a finite set of axioms into individual mathematical facts.

I'm asserting that I've constructed irreducible mathematical facts. What does this mean? It means that you cannot shrink them any more, you cannot squeeze them into axioms. In fact, that these are irreducible

mathematical assertions means that essentially the only way to prove them is if we directly take each individual assertion that we wish to prove as an axiom! That's *cheating!*

Yes, one can always prove an assertion by putting the assertion itself as a new axiom, but then we're not using reasoning. Picking new axioms is not deduction; it's the kind of thing that physicists worry about.

It is surprising that we can have an infinite number of independent mathematical facts that can only be proven by taking them as axioms. But if we think about coin tossing this is not at all surprising. You see, the notion of independent coin tosses is *exactly* like that.

Each time one tosses a fair coin, whether the outcome of that particular toss is head or tails, tells us absolutely nothing about the outcome of any future toss, and absolutely nothing about the outcome of any previous toss. That's how casinos make money: There is no way to predict from what has happened at a roulette wheel what is going to happen. Well, there is if the roulette wheel isn't balanced, and of course the casino works hard to make sure that the roulette wheel is working properly.

Let's go back to coin tossing, to the notion that a series of tosses has no structure. Even if one knew all the even results, it wouldn't help us predict any of the odd results. Even if one knew the first thousand tosses, that wouldn't help us predict the thousand-first toss.

Well, it's the same with using my equation to get randomness. Even if somehow one were told for all the even cases, whether there are a finite or an infinite number of solutions, this would be absolutely no help in getting the odd cases. Even if one were told the first thousand cases, whether there are a finite or an infinite number of solutions, it would be no help in getting the thousand-first case.

In fact I don't see how one could ever get *any* of the cases. Because there is absolutely no structure or pattern, and as I said these are irreducible mathematical facts. Essentially the only way to prove them is to directly assume them, which is not using reasoning at all.

So we've gone a long way in less than a hundred years: From Hilbert's conviction that every mathematical problem can be settled decisively by mathematical reasoning, to Gödel's surprising discovery that any finite set of axioms for elementary arithmetic is incomplete, to a new extreme, areas of arithmetic where reasoning is totally impotent and totally irrelevant.

Some people were depressed by Gödel's result. You might say, "This is all rather upsetting; should I switch fields and stop studying mathematics?" I certainly don't think you should!

You see, even though there is no pattern or structure in the question of whether *individual* cases of my equation have a finite or an infinite number of solutions, one *can* deal with it statistically: It turns out that in half the cases there's a finite number of solutions, and in half the cases there's an infinite number of solutions.

It's exactly like coin tosses, independent fair coin tosses. One can use statistical methods and prove theorems about the statistical patterns and properties of the answers to the question, which cannot be answered in each particular case, of whether there are a finite or an infinite number of solutions.

Let me repeat that the answers have a very simple statistical structure, that of independent tosses of a fair coin. So half the cases are heads and half are tails, one-fourth are a head followed by a head, one-fourth a head followed by a tail, one-fourth tail-head, one-fourth tail-tail, and so on for larger blocks and all the other statistical properties that one would like.

This kind of situation is not new; it's happened before, in physics. In quantum mechanics the Schrödinger equation shows this very clearly. The Schrödinger equation does not directly predict how a physical system will behave. The Schrödinger ψ function is only a probability. We can solve the Schrödinger equation to determine the *probability* that a physical system will behave in a certain way. The equation does not tell us what the system will do, it tells us the probability that it will do certain things.

In the 1920's and 1930's, this was very controversial, and Einstein hated it. He said, "God doesn't play dice!" But as you all know and

as Prof. Reichel explained, in recent times this lack of predictability has spread outside quantum mechanics. It turns out that even classical physics, Newtonian physics, contains unpredictability and randomness.

This is the field of non-linear dynamics or "deterministic chaos." It occurs in situations where small changes can produce big effects, in non-linear situations, very unstable situations, like the weather. It turns out that the weather is unpredictable, even in principle, as Prof. Casti discusses in his forthcoming book.[3] He studies the question of predictability and comprehensibility in a very broad context, including mathematics, the weather, and economics.

So it begins to look now like randomness is a unifying principle. We not only see it in quantum mechanics and classical physics, but even in pure mathematics, in elementary number theory. As I said before, I don't think that this should be viewed pessimistically. What it suggests to me, is that pure mathematics has much closer ties with physics than one suspected. Perhaps Plato's universe of mathematical ideas and the physical universe that we live in when we're not doing mathematics, perhaps these are closer to each other than has hitherto been suspected.

Thank you.

[3]John L. Casti, *Searching for Certainty—What Scientists Can Know About the Future,* William Morrow, New York, 1991.

RANDOMNESS IN ARITHMETIC

In M. E. Carvallo, Nature, Cognition and System, Vol. 3, Kluwer, 1993

G. J. Chaitin

Lecture given 16 January 1991 in the Gödel Room of the Mathematical Institute of the University of Vienna.

History

- HILBERT: Math is consistent, complete and decidable?
- GÖDEL 1931: Math is incomplete!
- TURING 1936: Math is undecidable!
- Gödel's & Turing's results superseded by stronger result:

- CHAITIN 1987: Math is random!
- Random?
- Answering Einstein, "God not only plays dice in physics, but even with the whole numbers!"

How to toss a coin?

- Exponential diophantine equation $E(\vec{X}, K) = 0$
- 200 pages long!
- With 17,000 integer variables $\vec{X}$!
- For $K = 0, 1, 2, 3, \ldots$ we ask "Does $E(K) = 0$ have finitely or infinitely many solutions $\vec{X}$?"
- The answers cannot be distinguished from independent tosses of a fair coin!
- The answers are irreducible mathematical information!
- The only way to prove them is to explicitly assume each of the answers to these questions as a new mathematical axiom/postulate!
- Cheating!

Information theory

- Classical information theory:

  $$H(p_1, \ldots, p_n) \equiv -\sum p_i \log_2 p_i$$

- Algorithmic information theory:

  $H(X) \equiv$ size in bits $|P|$ of smallest program P for computing X

- I.e.,

$$H(X) \equiv \min_{C(P)=X} |P|$$

- Choice of computer C used as measuring stick doesn't matter very much.
- Programs P must be syntactically self-delimiting:
- No extension of a valid program is a valid program.

Undecidability of halting problem (Turing 1936)

- Assume halting problem is decidable.
- Then $\log_2 N + c$ bit program can do following:
- Contains base-two numeral for N. ($\log_2 N$ bits)
- Remaining c bits:
- Consider each program P of size $\leq N$.
- Does P halt?
- If so, run P, and get largest integer printed.
- Maximize over all integers printed by programs that halt.
- Add one to result, print this, and halt.
- So $\log_2 N + c$ bit program prints integer greater than any program up to size N can print.
- Eventually, $\log_2 N + c$ is much less than N.
- Contradiction!

The halting probability Ω

- $\Omega \equiv \sum_{\text{program } P \text{ halts}} 2^{-|P|}$
- $\Omega < 1$ because no extension of a valid program is a valid program.
- Ω is an algorithmically irreducible or random real number.
- I.e., to produce any N bits of base-two expansion of Ω would take an N-bit program.
- Implies statistical randomness.
- Thus, e.g., Ω is "absolutely normal" in sense of Borel.
- I.e., in any base, all blocks of digits of the same length have equal limiting relative frequency.

"Computing" Ω in the limit

- $\Omega_N \equiv N$th approximation to halting probability Ω
- $\Omega_N \equiv \sum_{|P| \le N \ \&\ P \text{ halts in time } \le N} 2^{-|P|}$
- Ω_N is computable function of N (slow!)
- $\Omega_1 \le \Omega_2 \le \Omega_3 \le \cdots \uparrow \Omega$
- Ω_N converges to Ω very, very slowly!

Equation for Ω

- Write pure LISP program for Kth bit of $\Omega_N \equiv (N$th approximation to $\Omega)$.
- Only three pages long.
- Halts if Kth bit of Ω_N is 1.
- Doesn't halt if Kth bit of Ω_N is 0.

- Plug this LISP program into a 200-page universal exponential diophantine equation.
- (A universal diophantine equation is one that "simulates" a universal Turing machine.)
- Resulting equation:
- has exactly one solution if Kth bit of Ω_N is 1;
- has no solution if Kth bit of Ω_N is 0.
- Since $\Omega_N \uparrow \Omega$, equation:
- has finitely many solutions if Kth bit of Ω is 0;
- has infinitely many solutions if Kth bit of Ω is 1.

Solvable/unsolvable?

- Hilbert's 10th problem:
- "Does a diophantine equation have a solution?"
- Hilbert's 10th problem is undecidable (Matijasevič 1970).
- Proof:
- $P(\vec{X}, K) = 0$ is solvable iff Kth program halts.
- "Is $P(K) = 0$ solvable/unsolvable?" $(0 \leq K < N)$
- Answers not independent.
- If know how many solvable, can determine which.
- So N answers are only $O(\log N)$ bits of information.

Finitely/infinitely many solutions?

- Now consider $E(\vec{X}, K) = 0$ with infinitely many solutions iff Kth bit of Ω is 1.
- (Exponential diophantine equation $E = 0$ instead of polynomial diophantine equation $P = 0$.)
- "Does $E(K) = 0$ have finitely/infinitely many solutions?" $(0 \leq K < N)$
- Now N answers are essentially N bits of information.
- Answers independent!
- Irreducible mathematical information!

Universal diophantine equation

- Simulates universal Turing machine ≡ general-purpose computer.
- Parameter of equation is program to execute.
- If program halts, equation has exactly one solution ("singlefoldness").
- If program doesn't halt, equation has no solution.
- Solutions to universal exponential diophantine equation are history vectors giving contents of each machine register as a function of time.
- Each "digit" in very large base is current register contents.

Constructing the equation

- Program parameter is written in toy pure LISP.

- Solution of equation is computation of interpreter EVAL for toy pure LISP.
- Very little number theory involved!
- Kth binomial coefficient of order N is odd iff $K \Rightarrow N$ bit by bit in base-two (Lucas, late 19th century).
- "Kth digit of $(11)^N$ base 2^N is odd" can be expressed as singlefold exponential diophantine equation (Jones & Matijasevič 1984).
- Combine equations by using $X = 0$ & $Y = 0$ iff $X^2 + Y^2 = 0$.

Toy pure LISP

- Pure functional language:
- Evaluate expressions, no side-effects.
- Data and function definitions are S-expressions:
- Character strings with balanced ()'s.
- Head of ((abc)de(fg(h))) is (abc)
- Tail of ((abc)de(fg(h))) is (de(fg(h)))
- Join of (abc) and (de(fg(h))) is ((abc)de(fg(h)))
- Define LISP functions recursively as in Gödel's 1931 paper.

LISP register machine

- LISP S-expressions are character strings in machine registers.
- (Register machine, not Turing machine as in Turing's 1936 paper.)
- Register modulo 256 is first character of S-expression.
- Register/256 is remaining characters of S-expression.

- Interpreter EVAL for toy pure LISP is 300 register machine instructions.
- Apply techniques of Jones & Matijasevič 1984.
- EVAL expands into 200-page universal exponential diophantine equation.

Summary

- Exponential diophantine equation $E(\vec{X}, K) = 0$
- 200 pages long!
- With 17,000 integer variables $\vec{X}$!
- For $K = 0, 1, 2, 3, \ldots$ we ask "Does $E(K) = 0$ have finitely or infinitely many solutions $\vec{X}$?"
- The answers cannot be distinguished from independent tosses of a fair coin!
- The answers are irreducible mathematical information!
- The only way to prove them is to explicitly assume each of the answers to these questions as a new mathematical axiom/postulate!
- Cheating!

References

- I. Stewart, "The ultimate in undecidability," *Nature,* 10 March 1988.
- J.P. Delahaye, "Une extension spectaculaire du théorème de Gödel: l'équation de Chaitin," *La Recherche,* juin 1988, *AMS Notices,* October 1989.

- G.J. Chaitin, "Randomness in arithmetic," *Scientific American,* July 1988.
- G.J. Chaitin, "A random walk in arithmetic," *New Scientist,* 24 March 1990.
- G.J. Chaitin, "Le hasard des nombres," *La Recherche,* mai 1991.
- G.J. Chaitin, *Algorithmic Information Theory,* third printing, Cambridge University Press, England, 1990.
- G.J. Chaitin, *Information, Randomness and Incompleteness—Papers on Algorithmic Information Theory,* second edition, World Scientific, Singapore, 1990.

LE HASARD DES NOMBRES

La Recherche 22, N° 232 (mai 1991), pp. 610–615

Gregory J. Chaitin

Les mathématiques passent pour l'incarnation de la rigueur logique et de l'exactitude. Peut-on néanmoins y déceler du désordre ? Dans les années 1930, les travaux du logicien Kurt Gödel, en démontrant l'≪ incomplétude ≫ des mathématiques, avaient déjà ébranlé quelques solides certitudes. Plus récemment, G.J. Chaitin a proposé une définition du hasard faisant appel à la théorie algorithmique de l'information. Moyennant cette définition, l'auteur a pu montrer que certaines questions en théorie des nombres ont des réponses tout aussi aléatoires que le résultat d'un jeu de pile ou face. Les mathématiques joueraient-elles donc aux dés ?

Le concept de hasard intrigue depuis longtemps les physiciens, et même l'humanité en général. Quelle est l'origine du hasard ? Dans quelle mesure le futur peut-il être prédit ? Notre incapacité à prédire l'avenir est-elle la conséquence de nos limites humaines ou plutôt la conséquence d'une impossibilité de principe ? Ces questions ont, en physique, une longue histoire. La physique classique héritée d'Isaac Newton était complètement déterministe : la connaissance parfaite de l'état d'un système à un instant donné permettait en principe la détermination de son état à tout instant ultérieur. Puis vint au début de ce siècle la mécanique quantique, où probabilités et hasard interviennent au niveau le plus fondamental de la théorie ; en effet, celle-ci ne peut fournir que des probabilités de mesurer telle ou telle valeur de la position, de l'énergie, etc. La mécanique quantique introduisit donc une indétermination fondamentale dans la nature, chose qu'Einstein lui-même ne voulut jamais accepter, comme en témoigne son célèbre ≪ Dieu ne joue pas aux dés ≫. Puis, assez récemment, on s'aperçut avec l'étude des ≪ systèmes dynamiques ≫, qu'après tout, la physique classique avait aussi du hasard, ou plus exactement de l'imprévisibilité, enfouis en son sein, puisque certains systèmes même très simples, comme un système de pendules, peuvent exhibir un comportement imprévisible (voir ≪ Le chaos déterministe ≫ dans *La Recherche* d'octubre 1990). Le hasard, souvent associé au désordre, est ainsi devenu, du moins en physique, une notion pleine de contenu.

Et en mathématiques, discipline souvent perçue comme la certitude par excellence ? Le hasard y a-t-il une place ? La question est déjà un peu surprenante. La réponse l'est encore plus ! Différents travaux, notamment les miens, ont montré que la situation en mathématiques est apparentée à celle qui prévaut en physique : le hasard apparaît en mathématiques à un niveau fondamental. Je ne fais pas ici allusion à la théorie des probabilités, qui fait partie intégrante des mathématiques et qui est un outil pour décrire et étudier des phénomènes aléatoires, sans se préoccuper de l'origine de leur caractère aléatoire. La problématique qui nous occupe ici est tout autre et, d'un certain point de vue, beaucoup plus profonde : supposons que nous, mathématiciens, n'arrivions pas à démontrer un théorème, que nous ne discernions pas une struc-

ture ou une loi portant sur des nombres ou d'autres entités (cela arrive très souvent, et même dans la plupart des cas !). Plusieurs questions sont alors possibles : est-ce de notre faute, est-ce parce que nous ne sommes pas assez astucieux, parce que nous n'avons pas assez travaillé ? Ou bien est-ce plutôt parce qu'il n'y a pas de loi mathématique à trouver, pas de théorème, pas de réponse à la question mathématique que nous nous sommes posée ? C'est en liaison avec cette dernière question, comme nous le verrons, qu'apparaissent les thèmes du hasard et de l'imprévisibilité en mathématiques.

Une façon d'orienter notre réflexion sur cette question est de nous rappeler la célèbre liste des vingt-trois problèmes que le mathématicien allemand David Hilbert (fig. 1) avait proposée en 1900 en tant que défi au XX^{e} siècle naissant[(1)]. L'un des problèmes, le sixième de la liste, était d'axiomatiser la physique, c'est-à-dire d'exprimer toutes les lois fondamentales de la physique sous forme de règles mathématiques formelles ; cette question englobait l'axiomatisation de la théorie des probabilités car, pour Hilbert, les probabilités concernaient le monde réel et étaient donc du ressort de la physique. Son dixième problème avait rapport, lui, aux équations $\ll$ diophantiennes $\gg$, c'est-à-dire les équations algébriques où l'on cherche des solutions sous forme de nombres entiers. La question posée par Hilbert était : $\ll$ Y a-t-il un moyen de décider si une équation algébrique possède ou non une solution en nombres entiers ? $\gg$. Hilbert était loin de soupçonner une quelconque relation entre ces deux problèmes, alors que nous verrons plus loin qu'il y en a bel et bien une.

Gödel : il n'existe pas de système axiomatique complet et cohérent qui puisse contenir l'arithmétique

Pour Hilbert et pour la plupart des mathématiciens de l'époque, l'idée que tout problème mathématique possède une solution était un principe qui allait de soi. Ce n'est que par la suite qu'Hilbert a reconnu qu'il y avait là un thème à explorer. De cette exploration, il s'est avéré qu'une question mathématique simple et claire ne possède pas toujours

de réponse tranchée ; de plus, une certaine forme de hasard intervient même en mathématiques pures, et se rencontre au travers des équations diophantiennes, objet du dixième problème de Hilbert. En effet, nous verrons que certaines questions assez simples d'arithmétique, liées aux équations diophantiennes, ont — dans un sens bien déterminé — une réponse complètement aléatoire. Et cela, non pas parce que nous ne pourrons y répondre demain, dans cent ou mille ans, mais parce que la réponse est aléatoire quel que soit le raisonnement utilisé.

Comment en est-on arrivé là ? Un premier point a rapport avec la notion de raisonnement axiomatique, c'est-à-dire de raisonnement mathématique fondé sur des règles formelles. Le système géométrique d'Euclide est un exemple simple de système axiomatique, mais depuis la fin du XIX^{e} siècle divers systèmes d'axiomes ont été proposés afin de formaliser complètement les mathématiques ainsi que la logique sur laquelle tout raisonnement humain repose. L'axiomatique et le fondement des mathématiques ont été étudiés par de nombreux chercheurs, y compris par Hilbert lui-même. En particulier, ce dernier avait formulé une exigence : pour qu'un système d'axiomes soit satisfaisant, il doit exister une ≪ procédure mécanique ≫, c'est-à-dire une suite d'opérations logiques en nombre fini, permettant de décider si une démonstration mathématique quelconque vérifie ou non les règles formelles fixées. C'est là une exigence de clarté et d'objectivité, qui semble parfaitement naturelle. Le point important pour la suite est que si l'on bâtit un système d'axiomes *cohérent* (c'est-à-dire tel qu'on ne peut y prouver un résultat et son contraire simultanément) et *complet* (c'est-à-dire tel que toute assertion y est soit vraie, soit fausse), alors il découle immédiatement qu'il existe une procédure mécanique permettant — en principe — de trancher toute question qui puisse être formulée dans le cadre de cette théorie.

Une telle procédure consisterait (du moins en principe, car en pratique le temps nécessaire serait prohibitif) à faire la liste de toutes les démonstrations possibles écrites dans le langage formel, c'est-à-dire dans le système d'axiomes choisi, par ordre de taille et par ordre alphabétique des symboles employés. C'est ce qu'on peut appeler de manière imagée l'≪ algorithme du British Museum ≫, pour faire allusion au gigantisme de l'≪ inventaire ≫ à effectuer. Autrement dit, on énumère toutes les démonstrations possibles, et on vérifie si elles

découlent bien des règles formelles du système axiomatique. De cette façon, on obtient en principe tous les théorèmes, tout ce qui peut être prouvé dans le cadre du système d'axiomes. Et si celui-ci est cohérent et complet, toute affirmation pourra être confirmée (si elle est démontrée) ou infirmée (son contraire est alors démontré). Ainsi, cela fournit une procédure mécanique permettant de décider si une assertion est vraie ou non.

Malheureusement, la situation s'est avérée beaucoup moin simple. On sait, depuis les travaux fondamentaux de l'Autrichien Kurt Gödel en 1931 et du Britannique Alan Turing en 1936 (fig. 2), que cette entreprise est vaine : il n'existe pas de système axiomatique cohérent et complet pour l'arithmétique, et de plus il ne peut y avoir de procédé mécanique permettant de déterminer, pour toute assertion mathématique, si elle est vraie ou fausse.

De ce résultat qui a profondément marqué la pensée mathématique, Gödel a fourni une très ingénieuse démonstration : c'est son célèbre ≪ théorème d'incomplétude ≫. Mais l'approche de Turing me semble, d'une certaine manière, plus fondamentale et plus facile à comprendre. Je me réfère ici au théorème de Turing, affirmant qu'il n'existe pas de procédure mécanique pouvant déterminer, pour un programme informatique arbitraire, s'il s'exécutera en un temps fini ou non une fois mis en route. Le théorème d'incomplétude de Gödel en découle immédiatement : s'il n'existe pas de procédure mécanique pour déterminer si un programme s'arrête en un temps fini ou non, alors il ne peut non plus exister de système d'axiomes permettant de le faire.

Turing : il n'existe pas d'algorithme général permettant de savoir si un programme s'exécutera en un temps fini ou non

Sans entrer dans les détails, on peut esquisser une façon de démontrer que le problème de l'arrêt d'un programme est insoluble, en faisant un raisonnement par l'absurde. Supposons qu'il existe une procédure mécanique permettant de savoir, pour tout programme, si celui-ci s'exécutera en un temps fini. Cela implique alors qu'il est possible

de construire un programme (P) incorporant la donnée d'un nombre entier N, et effectuant les tâches suivantes : d'abord, examiner *tous* les programmes possibles de taille inférieure ou égale à N bits (tout programme informatique pouvant être traduit en une suite de chiffres binaires, 0 ou 1, appelés bits, et constituant chacun une unité d'$\ll$ information $\gg$) et déterminer lesquels d'entre eux s'arrêtent en un temps fini. Ensuite, simuler l'exécution de *tous* ces derniers et considérer leurs résultats. Supposons que les résultats soient des nombres entiers positifs ou nuls, ce que l'on peut faire sans perte de généralité puisque tout programme produit comme résultat une suite de 0 ou 1, laquelle peut toujours être interprétée comme représentant un entier positif ou nul. La dernière tâche que l'on assigne alors au programme (P) est de prendre le résultat le plus élevé produit par *tous* les programmes qui s'arrêtent en un temps fini et dont la taille ne dépasse pas N bits, et de calculer le double (par exemple) de ce résultat maximal.

Examinons maintenant la situation à laquelle on aboutit. Le nombre N est l'*essentiel* de l'information incluse dans le programme (P) que l'on vient de décrire. Par conséquent, la taille de ce programme est de l'ordre de $\log_2 N$ bits, puisque pour exprimer le nombre N, il n'est besoin que de $\log_2 N$ bits dans le système binaire (par exemple, le nombre 109 s'écrit 1101101 dans le système binaire, ce qui nécessite donc $7 \approx \log_2 109$ bits). Bien sûr, le programme (P) doit aussi contenir d'autres instructions permettant d'énumérer et de simuler l'exécution de tous les programmes de taille inférieure à N bits, mais le résultat n'est pas fondamentalement modifié : le programme (P) a bien une taille d'ordre $\log_2 N$ bits (donc inférieure à N bits). Ce point nécessite peut être un peu plus d'éclaircissements : naïvement, on aurait tendance à penser que (P) doit contenir en lui tous les programmes de moins de N bits. Mais ce n'est pas parce que (P) simule leur exécution qu'il doit les contenir ! Pour donner une image, un programme chargé d'effectuer la somme de tous les entiers de 1 à 1 000 n'a pas besoin de contenir en mémoire tous les entiers de 1 à 1 000 : il les produit successivement au fur et à mesure du calcul de la somme. Cela pour faire comprendre que N est bien l'ingrédient principal du programme (P). Mais revenons à notre propos ; par construction, ce programme produit un résultat qui est au moins deux fois plus grande que celui produit par tout programme dont la taille est inférieure à N bits : il

y a contradiction, puisque (P) fait lui-même partie de ces programmes et qu'il donnerait donc un résultat au moins deux fois plus grand que celui qu'il fournit lui-même... L'hypothèse de départ (l'existence de (P)) est alors fausse. Le problème de l'arrêt d'un programme est donc insoluble, ce que nous venons de montrer en utilisant un point de vue de la théorie de l'information.

Partons de ce résultat fondamental de Turing ; afin d'obtenir mon résultat établi en 1987[(2)] sur le hasard en mathématiques, il suffit de modifier le vocabulaire. C'est une sorte de calembour mathématique. De l'insolubilité du problème de l'arrêt, on passe au hasard lié à la *probabilité d'arrêt.* Qu'est donc cette dernière ? Au lieu de se demander si un programme donné va s'arrêter ou non au bout d'un temps fini, on considère l'ensemble de tous les programmes informatiques possibles, ce qui peut se faire en principe à l'aide d'un ordinateur idéalisé, appelé dans le jargon ≪ calculateur universel de Turing ≫. A chaque programme possible, on associe une probabilité (à ne pas confondre avec la probabilité d'arrêt que l'on va bientôt définir). Comme tout programme est finalement équivalent à une suite de bits, on choisit chaque bit au hasard, par exemple en tirant à pile ou face : à un programme de N bits on associera donc la probabilité $1/2^N$. En fait, on se limite aux programmes bien structurés dont on suppose qu'ils se terminent par l'instruction ≪ fin de programme ≫, laquelle ne peut apparaître en début ou en milieu de programme ; autrement dit, aucun programme bien structuré ne constitue l'extension d'un autre programme bien structuré. Cette hypothèse est technique mais essentielle, car en son absence le total des probabilités $1/2^N$ serait supérieur à 1 (et même infini). On définit alors la probabilité d'arrêt Ω (oméga) : c'est la probabilité pour que, ayant tiré au hasard un programme, celui-ci s'exécute en un nombre fini d'étapes. Ce nombre Ω vaut $\sum_N (a_N/2^N)$, où a_N est le nombre de programmes bien structurés de N bits qui s'exécutent en un temps fini. Ω est une probabilité, donc un nombre compris entre 0 et 1 ; si l'on trouvait $\Omega = 0$, cela signifierait qu'aucun programme ne s'arrête, et si l'on trouvait $\Omega = 1$, qu'ils s'arrêtent tous. Cette probabilité peut être exprimée en diverses bases ; une base particulièrement commode est la base binaire, dans laquelle le nombre Ω est une suite de 0 ou 1, par exemple 0,111010001101.... La question que l'on peut alors se poser est : ≪ Quel est le N-ième bit de la probabilité

d'arrêt Ω ? ≫

L'assertion de Turing (≪ le problème de l'arrêt est indécidable ≫) mène à mon résultat établissant que la probabilité d'arrêt est aléatoire ou plus exactement constitue de l'*information mathématique irréductible.* En d'autres termes, chaque bit de la représentation binaire de Ω est un fait mathématique qui est logiquement et statistiquement indépendant des autres : savoir si un bit donné de Ω est un 0 ou un 1 est un fait mathématique irréductible, qui ne peut être davantage condensé ou réduit. Une manière plus précise de le dire est que la probabilité d'arrêt est *algorithmiquement aléatoire,* c'est-à-dire que pour calculer N bits de la représentation binaire de Ω, il faut un programme informatique dont la taille est d'au moins N bits (voir l'encadré 1). Une façon résumée d'exprimer cela est : ≪ L'assertion que le N-ième bit de Ω est un 0 ou un 1, pour un N donné, est un fait mathématique aléatoire, analogue au résultat d'un jet de pile ou face ≫.

On rétorquera immédiatement que ce n'est pas le genre d'assertions que l'on rencontre habituellement en mathématiques pures. On aimerait bien pouvoir traduire cet énoncé dans le langage de la théorie des nombres, laquelle constitue le soubassement des mathématiques. En fait, Gödel était confronté au même problème. L'assertion vraie mais indémontrable qu'il avait construite était bizarre, elle disait : ≪ je suis indémontrable ! ≫. Gödel a déployé énormément d'ingéniosité et utilisé des raisonnements très sophistiqués afin de transformer ≪ je suis indémontrable ≫ en un énoncé portant sur les nombres entiers. Les travaux de Gödel ont donné lieu à de nombreuses recherches, dont la conclusion finale est que le dixième problème d'Hilbert est insoluble : il n'existe pas d'algorithme pouvant déterminer, en un nombre fini d'opérations, si une équation diophantienne arbitraire possède une solution. En fait, ce problème s'avère équivalent à celui de Turing sur l'arrêt d'un programme : étant donné un programme informatique, on peut construire une équation diophantienne qui a une solution si et seulement si ce programme s'exécute en un temps fini ; réciproquement, étant donnée une équation diophantienne, on peut construire un programme qui s'arrête si et seulement si cette équation possède une solution.

Particulièrement spectaculaires sont dans ce contexte les travaux des mathématiciens James P. Jones, de l'université de Calgary au Canada, et Yuri V. Matijasevič, de l'institut Steklov à Léningrad, publiés il y a

environ six ans[3]. Ces deux mathématiciens ont remarqué qu'il existait un théorème très simple, démontré par le Français Edouard Lucas, il y a plus d'un siècle, et qui résoud le dixième problème de Hilbert assez facilement s'il est utilisé de façon appropriée. Le théorème de Lucas concerne la parité des coefficients du binôme : demandons-nous si le coefficient de X^K dans le développement de $(1+X)^N$ est pair ou impair, c'est-à-dire quelle est la parité du K-ième coefficient binomial d'ordre N (pour $K = 0, 1, 2,..., N$) ; le théorème de Lucas répond que ce coefficient est impair si et seulement si $\ll K$ implique N en tant que suites de bits $\gg$. Cela signifie que ce coefficient est impair si à chaque $\ll 1 \gg$ de la représentation binaire de K correspond un $\ll 1 \gg$ à la même place dans la représentation binaire de N (fig. 3). Dans le cas contraire, le coefficient binomial est pair.

En utilisant la technique de Jones et Matijasevič (voir l'encadré 2), qui se fonde sur ce remarquable théorème de Lucas, j'ai mis au point un ensemble de programmes, écrits dans le langage C (et très récemment, dans le langage SETL2[4]), et que j'ai fait $\ll$ tourner $\gg$ sur un ordinateur IBM RISC System/6000 (les lecteurs intéressés par le logiciel peuvent me contacter[5]). Pour obtenir quoi ? Une équation diophantienne, plus exactement une équation diophantienne exponentielle. Les équations de ce type ne comportent que des additions, des multiplications et des exponentiations, les constantes et les inconnues considérées étant des nombres entiers positifs ou nuls. Contrairement à une équation diophantienne classique, on admet que la puissance à laquelle est élevée une inconnue puisse être aussi une inconnue. Ainsi, par exemple, une telle équation peut contenir non seulement des termes comme X^2 ou X^3, mais aussi des termes comme X^Y ou Y^X.

L'équation diophantienne que j'ai obtenue comporte près de 17 000 variables et occupe plus de 200 pages (voir $\ll$ Une extension spectaculaire du théorème de Gödel : l'équation de Chaitin $\gg$ dans *La Recherche* de juin 1988) ! Quelle est sa signification ? Elle contient un paramètre unique, le nombre N. Pour toute valeur donnée de ce paramètre, posons-nous la question suivante : $\ll$ Cette équation a-t-elle un nombre fini ou infini de solutions en nombres entiers (c'est-à-dire un nombre fini ou infini de listes de 17 000 nombres entiers, chaque liste étant une solution de l'équation) ? $\gg$. La réponse à cette question s'avère être un fait arithmétique aléatoire, analogue à un tirage à pile

ou face. Elle est une transcription arithmétique du fait mathématique irréductible que le N-ième bit de la probabilité d'arrêt Ω est 0 ou 1 : si cette équation diophantienne (de paramètre N) a un nombre fini de solutions, alors ce N-ième bit est 0, et si l'équation possède un nombre infini de solutions, ce bit est 1 (soulignons en passant que s'il n'y a pas de solution, le nombre de solutions est fini et vaut 0). La réponse à la question ne peut donc pas être *calculée,* et le N-ième bit de Ω non plus. Cela ne veut pas dire que les bits de Ω ne sont pas définis et déterminés mathématiquement, mais plutôt qu'il n'existe pas d'algorithme à nombre fini d'étapes pour les calculer, et que la connaissance des N premiers bits de Ω n'aide strictement en rien à la détermination des suivants.

La différence par rapport au problème posé par Hilbert est double ; d'une part, Hilbert ne pensait qu'aux équations diophantiennes classiques, non exponentielles ; d'autre part, la question qu'il avait posée était : $\ll$ Y a-t-il une solution à l'équation ? $\gg$ Cette question est indécidable, mais la réponse n'est pas totalement aléatoire, elle ne l'est que dans une certaine mesure. Les réponses ne sont pas indépendantes les unes des autres ; en effet, on sait qu'étant donné un nombre fini d'équations diophantiennes, il est possible de déterminer lesquelles ont une solution si l'on sait combien d'entre elles en possèdent. Pour obtenir un hasard vraiment total, semblable à celui associé à un tirage à pile ou face, la question adéquate que l'on doit poser est : $\ll$ Y a-t-il un nombre fini ou infini de solutions ? $\gg$ Mon assertion est que l'on ne pourra jamais le savoir, car décider si le nombre de solutions est fini ou infini, pour chaque valeur de N, est un fait mathématique irréductible. La réponse est, au sens algorithmique, aléatoire. La seule façon d'aller de l'avant est de considérer les réponses comme des axiomes. Si l'on cherche à résoudre M fois la question de savoir si le nombre de solutions est fini pour M valeurs données du paramètre N, alors il faudra incluire M bits d'information dans les axiomes de notre système formel. C'est en ce sens précis que l'on peut dire que les mathématiques contiennent du $\ll$ hasard $\gg$.

La probabilité d'arrêt est algorithmiquement aléatoire

Dans le sixième problème que Hilbert avait proposé, l'axiomatisation de la physique devait selon lui englober l'axiomatisation de la théorie des probabilités. Au fil des ans, cependant, la théorie des probabilités est devenue une branche des mathématiques à part entière. Mais d'après ce qui précède, une forme extrême de ≪ hasard ≫ — plus précisément, d'irréductibilité — apparaît dans un autre contexte, en mathématiques pures, en théorie élémentaire des nombres. Les recherches aboutissant à ces conclusions prolongent les travaux de Gödel et Turing, qui ont réfuté l'hypothèse de base faite par Hilbert et d'autres, selon laquelle toute question mathématique possède une réponse univoque.

Cela fait maintenant près d'un siècle que la philosophie et les fondements des mathématiques suscitent un grand intérêt. Auparavant, de nombreux efforts ont été consacrés à rendre rigoureuse l'analyse mathématique (la notion de nombre réel, de limite, etc.). L'examen moderne des mathématiques a réellement débuté, je pense, avec la théorie de l'infini de G. Cantor et les paradoxes et les surprises qu'elle a engendrés, et avec les efforts fournis par des mathématiciens comme Peano, Russell et Whitehead pour donner aux mathématiques des fondements solides et rigoureux. On avait placé beaucoup d'espoir en la théorie des ensembles. On avait ainsi cherché à définir de façon rigoureuse les nombres entiers 0, 1, 2, 3,... en termes d'ensembles. Mais il s'est avéré que la notion d'ensemble peut engendrer toutes sortes de paradoxes (Bertrand Russell en a donné un exemple célèbre : ≪ L'ensemble de tous les ensembles qui ne font pas partie d'eux-mêmes ≫ ; cet ensemble fait-il partie de lui-même ?). La théorie des ensembles est une partie fascinante et vitale des mathématiques ; néanmoins il me semble qu'il y a eu un certain désabusement vis-à-vis d'elle et qu'un retour aux 0, 1, 2, 3,... intuitifs s'est opéré. Malheureusement, les travaux que j'ai mentionnés, et en particulier mon propre travail, font que l'édifice des nombres entiers paraît moins solide qu'on ne le pensait. J'ai toujours cru, et probablement la plupart des mathématiciens y croient aussi, en un sorte d'univers platonicien où règne une ≪ réalité mathématique ≫ indépendante de la réalité physique. Ainsi, la ques-

tion de savoir si une équation diophantienne a un nombre fini ou infini de solutions a très peu de sens concret, mais j'ai toujours pensé en mon for intérieur que même si nous ne pourrons jamais y répondre, Dieu, lui, le pouvait.

Avec ces découvertes, les mathématiciens sont en train de rejoindre, en un sens, leurs collègues de la physique théorique. Ce n'est pas nécessairement une mauvaise chose. Dans la physique moderne, le hasard et l'imprévisibilité jouent un rôle fondamental ; la reconnaissance et la caractérisation de ce fait, lequel pouvait être perçu *a priori* comme une limitation, sont un progrès. J'ai la conviction qu'il en sera de même en mathématiques pures. □

Gregory J. Chaitin travaille au centre de recherches Thomas J. Watson d'IBM à Yorktown Heights aux Etats-Unis. Ses recherches ont trait à la théorie algorithmique de l'information, dont il a posé les bases vers le milieu des années 1960.

Pour en savoir plus :

□ G. J. Chaitin, *Algorithmic information theory,* Cambridge University Press, 1990 (troisième impression).

□ G. J. Chaitin, *Information, randomness and incompleteness — Papers on algorithmic information theory,* World Scientific, 1990 (seconde édition).

□ E. Nagel, J.R. Newman, K. Gödel et J.-Y. Girard, *Le théorème de Gödel,* Seuil, 1989.

Notes

(1) Voir par exemple l'article ≪ Hilbert (problémes de) ≫ de Jean-Michel Kantor dans *Encyclopedia Universalis,* vol. 9, 300, 1985.

(2) G.J. Chaitin, *Advances in Applied Mathematics, 8,* 119, 1987;

G.J.Chaitin, *Algorithmic information theory,* Cambridge University Press, 1990 (troisième impression).

(3) J.P. Jones et Y.V. Matijasevič, *Journal of Symbolic Logic, 49,* 818, 1984.

(4) SETL2 est un nouveau langage de programmation, permettant d'écrire ces programmes de façon plus courte et plus facile à comprendre (mais ils sont plus lents). Ce langage se fonde sur un idée de J.T.Schwartz de l'institut Courant à New York, selon laquelle la théorie des ensembles peut être convertie directement dans un langage de programmation (voir W.K. Snyder, *The SETL2 programming language,* Courant Institute, 1990; J.T. Schwartz *et al., Programming with sets — An introduction to SETL,* Springer-Verlag, 1986).

(5) G.J. Chaitin, *LISP for ≪ Algorithmic information theory ≫ in C,* août 1990.

Encadré 1. Les décimales de π forment-elles une suite aléatoire?

En quel sens la suite de chiffres qui composent un nombre peut-elle être qualifiée d'≪ aléatoire ≫ ? La question est moins simple qu'elle ne paraît. Il y a près d'un siècle, le mathématicien français Emile Borel (voir cliché) avait défini dans ce contexte la notion de nombre ≪ normal ≫ et avait démontré que presque tous les nombres sont normaux.

Qu'est-ce qu'un nombre normal ? Un nombre est dit normal dans une base b si chacun des b chiffres possibles apparaît, dans le développement du nombre selon cette base, avec la même fréquence $1/b$, si chacun des b^2 groupes de deux chiffres successifs apparaît avec la même fréquence $1/b^2$, et de même avec les groupes de trois chiffres, de quatre chiffres, etc. Par exemple, un nombre est normal dans le système binaire ($b = 2$) si dans son développement binaire les chiffres 0 et 1 apparaissent avec las même fréquence limite 1/2, si les séquences 00, 01, 10, 11 apparaissent avec la même fréquence limite 1/4, etc. Un

nombre est dit *absolument normal* s'il est normal quelle que soit la base b dans laquelle il est exprimé.

E. Borel montra en 1909 que *presque tous* (expression qui a un sens mathématique précis) les nombres réels sont absolument normaux. En d'autres termes, choisissons un nombre en tirant à pile ou face chacun de ses bits constituant son développement infini dans le système binaire. Alors, le nombre compris entre 0 et 1 choisi de cette façon est absolument normal, et cela $\ll$ presque sûrement $\gg$, c'est-à-dire avec une probabilité égale à 1.

Si la non-normalité est l'exception à la règle, on serait tenté de penser qu'il est facile de trouver des exemples de nombres normaux. Qu'en est-il par exemple de $\sqrt{2}$, π ou e ? Sont-ils normaux ? Chose étonnante, on n'en sait rien ! De nombreux calculs par ordinateurs ont été effectués afin d'obtenir les chiffres successifs formant ces nombres et de déterminer leur fréquence d'apparition ; tout se passe comme s'ils étaient normaux, mais personne n'a pu à ce jour le démontrer rigoureusement. En fait, il a été extrêmement difficile d'exhibir un exemple de nombre normal. En 1933, D.G. Champernowne parvint à exhiber un nombre qu'il put démontrer être normal dans le système décimal ; ce nombre s'écrit :

0, 0 1 2 3 4 5 6 7 8 9
10 11 12...98 99 100 101 102...
998 999 1000 1001 1002....

Mais on ne sait pas si ce nombre est normal absolument, c'est-à-dire normal dans toute base.

Néanmoins, on dispose à présent d'un exemple naturel de nombre absolument normal : la probabilitié d'arrêt Ω dont il est question dans l'article. En effet, on peut facilement démontrer que Ω est absolument normal à partir du fait qu'il est algorithmiquement aléatoire. Un nombre est algorithmiquement aléatoire si, pour déterminer N bits de son développement binaire, il faut un programme dont la taille est d'au moins N bits. Pour donner un contre-exemple, les N décimales des nombres 0,1111111111... ou 0,110110110110... peuvent être calculés très facilement par un programme dont la taille est très inférieure à N (pour N pas trop petit) ; en effet, il suffit de traduire les ordres $\ll$

répéter N fois le chiffre 1 $\gg$ ou $\ll$ répéter N' fois la séquence 110 $\gg$ en langage binaire. Ces nombres ne sont donc pas du tout algorithmiquement aléatoires. Même si $\sqrt{2}$, π ou e sont normaux (ce qui reste à prouver), ils ne peuvent être algorithmiquement aléatoires, puisqu'il existe des algorithmes de taille finie pour calculer leurs chiffres successifs. Le nombre de Champernowne est même pire à cet égard : non seulement ses chiffres sont calculables et prévisibles, mais en plus il est très facile de le faire. On voit donc clairement que la notion de nombre algorithmiquement aléatoire est beaucoup plus forte que celle de nombre normal. De la même façon, la grande importance pratique des algorithmes produisant des nombres pseudo-aléatoires (utilisés dans les jeux informatiques ou dans certains méthodes de calcul numérique) réside précisément dans le fait que les suites de nombres produites sont extrêmement compressibles, algorithmiquement parlant. (Cliché Harlingue-Viollet)

Encadré 2. Tirer à pile ou face à l'aide d'une équation diophantienne

Comment traduire en équation algébrique la détermination des bits de la probabilité d'arrêt Ω dont il est question dans l'article ? La méthode utilise une technique développée par Jones et Matijasevič, qui elle-même s'appuie sur le théorème de Lucas. Celui-ci affirme (fig. 3) que $K \ll$ implique $\gg N$ bits à bit si et seulement si le K-ième coefficient binomial d'ordre N est impair. Jones et Matijasevič montrent que cela est équivalent à dire que dans la base $b = 2^N$, le K-ième chiffre de 11^N est impair.

Mathématiquement, cela s'exprime ainsi : $K \ll$ implique $\gg N$ si et seulement s'il existe des entiers positifs ou nuls uniques b, x, y, z, u, v, w tels que

$$b = 2^N$$
$$(b+1)^N = xb^{K+1} + yb^K + z$$
$$z + u + 1 = b^K$$
$$y + v + 1 = b$$

$y = 2w + 1$

Pour obtenir une équation diophantienne, il suffit de réécrire ces cinq équations pour que leur membre de droite soit 0, de les élever au carré et de les ajouter. On obtient l'équation

$$
\begin{aligned}
&[b - 2^N]^2 + \\
&[(b+1)^N - xb^{K+1} - yb^K - z]^2 + \\
&[z + u + 1 - b^K]^2 + \\
&[y + v + 1 - b]^2 + \\
&[y - 2w - 1]^2 = 0
\end{aligned}
$$

L'équation de 200 pages que j'ai obtenue a été construite en utilisant de façon répétée cette technique, afin d'exprimer en une équation diophantienne le calcul du N-ième bit d'une K-ième approximation de Ω. Cette équation possède exactement une solution si ce bit est 1, et n'en possède pas si ce bit est 0. On change alors le point de vue, et on considère K non pas comme un paramètre mais comme une inconnue supplémentaire. La même équation aura alors, pour une valeur donnée du paramètre N, un nombre fini ou infini de solutions selon que le N-ième bit de Ω est 0 ou 1 (la valeur de K peut différer d'une solution à l'autre). Pour K assez grand, l'approximation de Ω est suffisament bonne pour que le N-ième bit de la K-ième approximation de Ω soit le bon. Mais il est impossible de calculer, pour N donné, la valeur de K à partir de laquelle le bit a la bonne valeur, car la probabilité d'arrêt Ω est algorithmiquement aléatoire.

Figure 1.

En 1900, lors d'un congrès tenu à Paris, le grand mathématicien allemand David Hilbert (1862–1943) a énoncé une liste restée célèbre de 23 problèmes ouverts. Ses travaux et ses réflexions ont considérablement influencé les recherches mathématiques du vingtième siècle. Son dixième problème concernait la résolubilité des équations diophantiennes : existe-t-il une procédure permettant de déterminer en un nombre fini d'opérations si une équation diophantienne arbitraire

possède une solution ? Y.V. Matijasevič a pu montrer en 1970 que la réponse était négative. De nombreux problèmes de mathématiques peuvent être traduits en termes de non-résolubilité d'une certaine équation diophantienne ; c'est le cas de la question si oui ou non un programme informatique s'exécute en un temps fini. (Cliché Bildarchiv Preussischer Kulturbesitz)

Figure 2.

Le logicien autrichien K. Gödel (1906–1978) (A) a ébranlé en 1931 la conviction intime de la quasi-totalité des mathématiciens, conviction selon laquelle il était possible de construire des systèmes formels d'axiomes qui soient complets et cohérents. Il a pu démontrer que tout système formel comporte des énoncés qui sont indécidibles, c'est-à-dire qui ne peuvent être confirmés ou infirmés en utilisant uniquement les axiomes du système. Le Britannique A.M. Turing (1912–1954) (B) a formalisé les notions de calculabilité et d'algorithmique qui sont les fondements théoriques de l'informatique. Il a notamment montré en 1936 qu'il n'existe pas de procédure mécanique permettant de savoir si un programme arbitraire s'exécutera en un temps fini ou non. (Clichés AFP et E.T. ARCHIVE)

Figure 3.

Une programme informatique étant choisi au hasard, la probabilité Ω pour qu'il s'exécute en un temps fini peut s'écrire dans le système binaire sous forme d'une suite de 0 ou 1, appelés bits. Pour obtenir une équation déterminant les bits de la probabilité d'arrêt d'un programme choisi au hasard, G.J.Chaitin a utilisé des techniques qui s'appuient sur un théorème simple dû à un mathématicien français du siècle dernier, Edouard A. Lucas (1842–1891). Ce théorème affirme que le coefficient de X^K dans le développement de $(1+X)^N$ est impair si et seulement si les $\ll 1 \gg$ de l'écriture binaire de K se retrouvent à la même place dans l'écriture binaire de N.

COMPLEXITY AND RANDOMNESS IN MATHEMATICS

"Pensar la complexitat" Symposium, Barcelona, 4 November 1991

G. J. Chaitin

Abstract

I have developed a mathematical theory of complexity, which I call "algorithmic information theory." I have applied this theory to mathematics itself, and have shown that mathematics is not as simple as had been thought, and indeed that arithmetic contains infinite complexity and complete randomness. Here I shall give examples of my mathematical concept of complexity and how it is measured, outline its main properties, and discuss what it says about the limitations of mathematics.

To be published in Spanish by Tusquets Editores, S.A.

1. Introduction

Mathematics is much simpler than physics or biology, because it does not have to deal with the real world. I have developed a very theoretical mathematical theory of complexity, which I call "algorithmic information theory." Hopefully it may help to suggest to physicists and biologists what to do to develop their own more practical theories of complexity.

I have used algorithmic information theory to clarify the meaning of randomness, patternlessness, and lack of structure, and to show that some arithmetical questions, mathematical questions involving whole numbers or integers, have answers which completely escape the power of mathematical reasoning because they are completely random, patternless, and unstructured.

Formerly it was widely assumed by mathematicians, and emphasized by the famous mathematician David Hilbert, that mathematics was simple in the sense that all questions could be resolved by reasoning based on a small finite set of mathematical axioms that all could agree upon. The surprising results obtained by Kurt Gödel and Alan Turing, the incompleteness theorem and the undecidability of the halting problem, showed that mathematics is not that simple.

I have greatly extended the work of Gödel and Turing by showing that there are infinite classes of mathematical questions which are infinitely and irreducibly complex, in that the answers exhibit absolutely no structure or pattern that we could ever perceive using **any** finite set of mathematical axioms. Thus mathematics, and in fact even elementary arithmetic, far from being simple, is infinitely complex!

A good way to summarize this situation, is to connect it with parallel developments in modern physics. Classical physics has given way to quantum physics and now to chaos theory, and it certainly appears that, to use Einstein's words, God plays dice with the physical universe, that physics is complex and harbors randomness and unpredictability. My work shows that God also plays dice with the whole numbers, in arithmetic, because there are arithmetical questions involving the whole numbers whose answers are completely random.

Here I shall attempt to outline these developments and give an overall feel for the character of my theory, without giving any proofs

or going into the technical details. I shall illustrate the discussion with many examples and try to keep everything as concrete as possible.

2. Complexity

Let's start by looking at some examples of how to measure complexity. We shall measure complexity in binary digits or bits.

Here is an n-bit string which is random and has complexity n:

$$\overbrace{10\cdots1}^{n \text{ bits}}$$

This sequence of n bits is produced by tossing a coin n times. The coin must be fair, that is, the probability of producing a head and a tail must both be equal to $1/2$. The tosses must also be independent. If we get heads, we add a 1 to the sequence. If we get tails, we add a 0 to the sequence. We do this n times. There is no pattern or structure in the resulting sequence of 0's and 1's.[1]

Here are three examples of n-bit strings which only have complexity $n/2$:

$$\overbrace{10\cdots1}^{n/2 \text{ bits}}\overbrace{10\cdots1}^{n/2 \text{ bits}}$$

In the above sequence, the first half is produced by tossing a coin $n/2$ times. The second half is a copy of the first half.

$$\overbrace{10\cdots1}^{n/2 \text{ bits}}\overbrace{1\cdots01}^{n/2 \text{ bits}}$$

In the above sequence, the first half is produced by tossing a coin $n/2$ times. The second half is the bit string reversal of the first half.

$$\overbrace{\underbrace{11}_{1}\underbrace{00}_{0}\cdots\underbrace{11}_{1}}^{n/2 \text{ pairs}}$$

This time we produce an n-bit sequence of complexity $n/2$ by tossing a coin $n/2$ times to produce an $n/2$-bit string, and then doubling each bit.

[1] More precisely, this is highly probable.

Here are two examples of n-bit strings which have complexity $n/3$:

$$\overbrace{10\cdots1}^{n/3\text{ bits}}\overbrace{10\cdots1}^{n/3\text{ bits}}\overbrace{10\cdots1}^{n/3\text{ bits}}$$

In the above sequence, the first third is produced by tossing a coin $n/3$ times. This sequence is then copied twice.

$$\overbrace{\underbrace{111}_{1}\underbrace{000}_{0}\cdots\underbrace{111}_{1}}^{n/3\text{ triples}}$$

This time we produce an n-bit sequence of complexity $n/3$ by tossing a coin $n/3$ times to produce an $n/3$-bit string, and then tripling each bit.

Now let's look at some examples of bit strings with much lower complexity:

$$\overbrace{111\cdots\cdots111}^{n\text{ 1's}}$$

$$\overbrace{\cdots\cdots\cdots\cdots}^{\text{first } n \text{ bits of fractional part of } \sqrt{2} \text{ in binary}}$$

$$\overbrace{\cdots\cdots\cdots\cdots}^{\text{first } n \text{ bits of fractional part of } \pi \text{ in binary}}$$

To produce each of these sequences, one really only needs to know n, which is usually about $\log_2 n$ bits,[2] so these are n-bit strings which only have complexity $\log_2 n$. All the information is really in their size, not in their content!

These examples all illustrate the following

Idea: the complexity of an object is the number of bits required to specify/describe how to construct/compute/ calculate it.

[2] "Usually," because some n have a special form, e.g., are a power of two.

Here is a more advanced example:

$$\overbrace{75\%\ 1\text{'s},\quad 25\%\ 0\text{'s}.}^{n\ \text{bits}}$$

For example, if 75% of the bits are 1's and 25% are 0's, then this n-bit sequence has its complexity reduced to 80% of what it would normally be, about $.80\,n$. Here we assume that the sequence has 1's and 0's in the proportion of 3 to 1 but is otherwise random. That is to say, it has been produced by n independent tosses of a coin which has probability .75 of producing heads and probability .25 of producing tails.

More generally, consider the following n-bit string:

$$\overbrace{\alpha n\ 1\text{'s},\quad \beta n\ 0\text{'s}.}^{n\ \text{bits}}$$

In this string the relative frequency of 1's is α and the relative frequency of 0's is β. It was produced by flipping a coin which produces 1's with probability α and 0's with probability β. In this case the maximum possible complexity, which is n bits, is reduced by a factor known as the Boltzmann–Shannon entropy, given by the following formula:

$$-\alpha \log_2 \alpha - \beta \log_2 \beta.$$

3. Is Complexity Additive?

How can we combine two bit strings into a single bit string? This turns out to be a real problem!

What about the complexity of a pair of strings $\langle x, y\rangle$? Is this always less than the sum of their individual complexities? It would seem so. If x and y have nothing in common, first tell me how to compute x, then tell me how to compute y. If they are related, one can combine parts of the computation and do even better.

But this doesn't work. The problem is telling where the description for x ends and that for y begins.

Here is an example. Let's try to combine the strings 101 and 000.

$$\langle 101, 000\rangle$$

This is easy to do because the strings are the same size

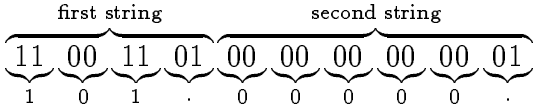
$$101000 \overset{\text{divide in middle}}{\longrightarrow} \langle 101, 000 \rangle$$

So this special case is easy.

But what if the strings that we wish to combine have different sizes?

$$\langle 101, 00000 \rangle$$

One straight-forward approach is to double each bit and use a pair of unequal bits as an endmarker:



This shows that the complexity $H(x, y)$ of two strings x and y is bounded by twice the sum of their individual complexities. Symbolically,

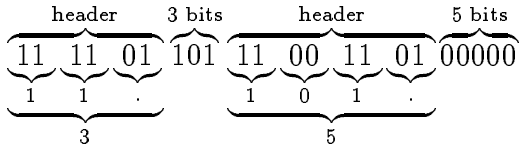
$$H(x, y) \leq 2H(x) + 2H(y).$$

This approach also works if we are combining three or more strings:

$$H(x, y, z) \leq 2H(x) + 2H(y) + 2H(z).$$

But this is very wasteful! We can do much better!

Here is a more clever approach. Let's just concatenate the given strings, and put in front of each string its size in binary. This is called a "header". But how do we know where the header ends and the data begins? Well, we double each bit in the header and use 01 as punctuation. Here is an example:



This shows that the complexity of two strings is bounded by the sum of their individual complexities plus twice the logarithms of their complexities:

$$H(x, y) \leq H(x) + H(y) + 2 \log_2 H(x) H(y).$$

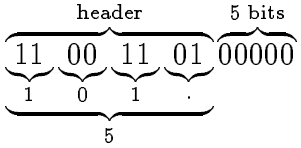
And similarly if there are three or more strings:

$$H(x,y,z) \leq H(x) + H(y) + H(z) + 2\log_2 H(x)H(y)H(z).$$

★ **Now change the point of view: require descriptions to be self-delimiting, so each H gets bigger and H is additive[3].**

4. Self-Delimiting Complexity

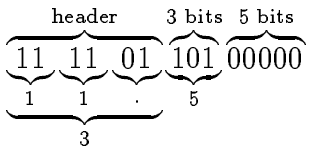
So here is our initial approach for making descriptions self-delimiting:



In general, this will make an n-bit string into an $(n+2\log n)$-bit string:

$$n \longrightarrow n + 2\log n.$$

We can improve this if instead of doubling each bit in the header to make it self-delimiting, we precede the header itself by a second header!



This will make an n-bit string into an $(n+\log n+2\log\log n)$-bit string:

$$n \longrightarrow n + \log n + 2\log\log n.$$

And with headers of headers of headers, we make an n-bit string into an $(n+\log n+\log\log n+2\log\log\log n)$-bit string:

$$n \longrightarrow n + \log n + \log\log n + 2\log\log\log n.$$

What about strings that are 2^n bits of data? For such strings, it is cheaper to use a header which is a program for computing the length of

[3]Technically, "**sub**additive."

the data, rather than to give the length of the data directly. And instead of having a header on the header etc., let's break the infinite regress by stipulating that the header which is a program for computing the length of the data *must itself be self-delimiting*. Thus the most compact header preceding n bits of data has as many bits as the **self-delimiting** complexity of the number n. This is usually about $\log n + \log\log n + \cdots$, but can be much much shorter if the number of bits of data, considered as a bit string, is highly non-random, i.e., has complexity much smaller than its size.

In summary, the new viewpoint which automatically makes complexity additive works like this:

	Random N-bit String
Old Complexity Measure	N
New Complexity Measure	N + complexity of N

Once we make this fundamental conceptual shift, a lot changes. The most important change is that no extension of a valid program is a valid program, since we know how many bits it has from its header. This implies that we can now talk about the *probability* of computing something as well as the minimal-program complexity.

The program-size complexity is defined as follows:

$$H(x) \equiv \min_{C(p)=x} |p|.$$

That is to say, $H(x)$ is the minimum size in bits $|p|$ that a program p must have in order to be able to compute a given object x using our standard computer C.[4]

Now let's generate the program p at random. That is to say, we toss a fair coin, one independent toss for each bit of p, and the probability of computing x using our standard computer C is simply the sum of

[4]The choice of C is not very important. Technically speaking, C must be a "universal Turing machine," i.e., sufficiently general-purpose that it can simulate any other computer. Moreover, it must be able to do this economically, that is to say, by adding a fixed number of bits to each program to indicate which computer is to be simulated.

the probability of each program p that computes x. The probability of a program p is merely

$$2^{-(\text{the number of bits } |p| \text{ in the program } p)}.$$

Adding these probabilities for all p that compute x, we see that the probability $P(x)$ of computing x is precisely

$$P(x) = \sum_{C(p)=x} 2^{-|p|}.$$

Moreover, if we add the probabilities that our computer C computes anything at all, this gives the total *halting probability* Ω.

$$\Omega = \sum_{C(p) \text{ halts}} 2^{-|p|}.$$

None of these probabilistic notions could work before, when our computer programs/descriptions were not self-delimiting. The reason is that before we had 2^n n-bit programs, and if each has probability 2^{-n}, then summing the probabilities of all programs of all sizes that do something particular will give infinity. When programs are self-delimiting this no longer happens, because no extension of a valid program is a valid program. I.e., if p is a valid program, then $p0$, $p1$, $p00$, $p01$, ... cannot be.

There is a geometrical proof that this works. Associate bit strings with subsets of the interval of unit length consisting of all real numbers r between zero and one. The string b is associated with all those real numbers r having that string at the beginning of the fractional part of r's base-two representation:

$$b \overset{\text{is associated with}}{\longleftrightarrow} \{\text{the set of all reals of the form } .b\cdots\}.$$

Then the length of the interval associated with a program is precisely its probability. That no extension of a valid program is a valid program means that the intervals associated with valid programs do not overlap. Hence the sum total of the lengths of these non-overlapping intervals must be less than unity, since they are all inside an interval of unit length. In other words, the total probability that a program is valid is less than unity, as it should be, and not infinite.

5. Joint, Relative and Mutual Complexity

In addition to the plain complexity we have discussed up to now, there are some important variations called the *joint, relative,* and the *mutual* complexity, which involve two or more objects instead of a single object. And these composite complexities have some important and elegant properties.

5.1. Joint Complexity

We have actually already seen the first of these composite complexities, the *joint* complexity $H(x, y, \ldots)$ of two or more objects $x, y, \ldots$. The key property of the joint complexity is (sub)additivity:

$$H(x, y, \ldots) \leq H(x) + H(y) + \cdots,$$

the joint complexity is bounded by the sum of the individual complexities. In fact, we changed our definition of complexity and demanded that descriptions be self-delimiting precisely so that this inequality would hold.

A more esoteric and subtle property is that the complexity of an object x is equal to that of the pair $\langle x, H(x)\rangle$ consisting of the object x and its complexity $H(x)$:

$$H(x, H(x)) = H(x).$$

This is the case because a minimum-size program tells us its size as well as its output. In other words, in addition to interpreting a minimum-size description of an object to determine the object, we can also see the size of the description, and this can be very useful.

This is a rather technical point, and I'd best not say any more about this equation, except that it indicates most clearly where the formalism of my algorithmic information theory differs from the original Shannon formalism of classical ensemble information theory. It also leads to my next subject, the *relative* complexity of two objects.

5.2. Relative Complexity

The *relative* complexity $H(x \mid y)$ of x given y is the size of the shortest/smallest description of how to obtain x from y. However, and this is a

key point, this is not quite right. It is actually necessary to define the relative complexity $H(x \mid y)$ of x given y as the size of the shortest/smallest description of how to obtain x from the pair $\langle y, H(y)\rangle$. This is somewhat subtle and technical, and very closely related to the fact that $H(y, H(y)) = H(y)$.

When relative complexity is correctly defined, it turns out that the following fundamental decomposition holds:

$$H(x, y) = H(x) + H(y \mid x).$$

In other words, the joint complexity of x and y is equal to the sum of the complexity of x and the relative complexity of y given x. Let me repeat that here in $H(y \mid x)$ we are not given x directly, we are given a minimum-size description of x, which is equivalent to knowing x and $H(x)$.

5.3. Mutual Complexity

This leads us to the *mutual complexity* $H(x : y)$, which measures the extent to which it is cheaper to compute x and y together rather than to compute them separately. In other words, this is the extent to which two objects appear less complex when seen together than when seen separately. $H(x : y)$ measures the extent to which x and y are related, i.e., how much they have in common.

It is an important theorem of mine that $H(x : y)$ is also equal to the extent to which knowing x helps one to know y, and vice versa. I.e., the extent to which each of two objects seems less complex given the other, turns out to be equal to the extent to which it is less complex to compute them together than separately:

$$\begin{aligned} H(x : y) &\equiv H(x) + H(y) - H(x, y), \\ &= H(x) - H(x \mid y), \\ &= H(y) - H(y \mid x). \end{aligned}$$

This is very important because it shows that my definitions of self-delimiting complexity and of the relative complexity of one object given a minimal-description of another object are the correct definitions. Many variations of these definitions seem plausible but reveal their inadequacy by failing to have this property.

Here are two examples of the mutual complexity.

At one extreme, consider two n-bit strings x and y chosen at random. They have only their length in common, not their content, so their mutual complexity $H(x : y)$ is equal to $H(n)$, which is usually $\approx \log_2 n$ unless n is of a special form.

At the other extreme, $H(x : x) = H(x)$, that is say, if one considers two identical objects, all their complexity is mutual.

6. Complexity of Axiomatic Theories

Let's look at the limitations of mathematics, more precisely, at the limitations of the axiomatic method, since Euclid the basis for mathematics. A formal axiomatic theory is one that is specified so precisely that a computer can print out all the theorems by running through all possible proofs in size-order and checking which are valid.

$$\boxed{\text{Axioms}} \overset{\text{deduction}}{\longrightarrow} \boxed{\text{Theorems}}$$

If we know the complexity $H(\text{axioms})$ of a formal axiomatic theory, what does this tell us about its limitations?

Recall the halting probability I call Ω that we encountered in Section 4:

$$\Omega = \sum_{C(p)\ \text{halts}} 2^{-|p|}.$$

Since Ω is a probability, it is a real number between zero and one. It is an extremely basic fact that if Ω is written in binary notation, it is a random infinite bit string. More precisely,

$$\lim_{n\to\infty} H(\Omega_n) - n = \infty.$$

In other words, the complexity of Ω_n the first n bits of the base-two notation for the halting probability Ω, becomes and stays arbitrarily greater than n as n gets larger and larger.

My theorem that Ω is random is closely related to Turing's famous theorem on the undecidability of the halting problem, because if one knew the first n bits of Ω it would in principle enable one to solve the halting problem for all programs p of size $|p|$ less than n. This

implies that Ω is a highly uncomputable real number. In fact it is even extremely hard to *prove* whether a *particular* bit of Ω is a 0 or a 1.

More precisely, the randomness of the halting probability Ω implies that one cannot deduce what the first n bits of Ω are using a formal axiomatic theory whose complexity $H(\text{axioms})$ is less than n bits. In fact, I can show that the randomness of the halting probability Ω implies that one cannot even deduce what are the values and positions of n **scattered** bits of Ω using a formal axiomatic theory whose complexity $H(\text{axioms})$ is less than n bits.

In other words, essentially the only way to determine whether particular bits of Ω are 0 or 1 using reasoning based on axioms, is if the particular results that one wishes to prove are axioms. I.e., the value of each bit of Ω is an irreducible independent mathematical fact, and each bit is analogous to the result of an independent coin toss. In this domain, reasoning is completely impotent and completely irrelevant.

One can however make *statistical* statements about the bits of Ω. For example, 0's and 1's both have limiting relative frequency precisely $1/2$. If this were not the case, Ω would be compressible, as we saw in Section 2.

The real number Ω may seem to be somewhat exotic, but it actually even appears in elementary number theory, in the arithmetic of whole numbers. I have constructed a two-hundred page equation with seventeen-thousand variables which has the remarkable property that it captures Ω arithmetically.

My equation is what is called an *exponential diophantine equation.* That is to say, it is an algebraic equation involving only unsigned whole-number constants and variables, and is built up using only the operations of addition $x+y$, multiplication $x \times y$, and integer exponentiation x^y. Such equations are named after the ancient Greek Diophantos who first studied them.

How does my equation "capture" Ω?

One of the seventeen-thousand variables in my equation is the variable n. For each particular value of $n = 1, 2, 3, \ldots$, let's ask whether my monster equation has a finite or an infinite number of solutions. It must be one or the other, because no solution is a finite number of solutions. My equation is craftily constructed so that the answer to this question turns out to depend on whether the nth bit of Ω is a 0 or

a 1. If the nth bit of Ω is 0, then my equation has only finitely many solutions. If the nth bit of Ω is 1, then my equation has infinitely many solutions.

Thus it is just as impossible to prove whether my equation has finitely or infinitely many solutions for a particular value of the parameter n, as it is to prove whether the nth bit of Ω is a 0 or a 1! So as far as deciding whether my equation has finitely or infinitely many solutions is concerned, mathematical truth is infinitely complex and has absolutely no structure or pattern and appears to be completely random! No set of axioms of finite complexity can cope with the infinite complexity of Ω embodied in my equation!

7. Conclusions

I have outlined the major features of my algorithmic information theory. It provides a mathematical definition of complexity that has elegant formal properties. My theory also throws a devastating new light on Gödel's incompleteness theorem and on the limitations of the axiomatic method. Algorithmic information theory does this by providing information-theoretic incompleteness theorems based on measuring the complexity of the axioms of formal mathematical theories.

My complexity based approach to incompleteness suggests that incompleteness is natural and pervasive. To prove more one must sometimes assume more. In some cases the complexity of the axioms must be increased to be commensurate with the complexity of the theorems to be derived from them.

Further Reading

[1] CHAITIN, G.J., "Randomness and mathematical proof," *Scientific American* 232, No. 5 (May 1975), pp. 47–52. Also published in the French, Italian and Japanese editions of *Scientific American.*

[2] CHAITIN, G.J., "Randomness in arithmetic," *Scientific American* 259, No. 1 (July 1988), pp. 80–85. Also published in the

French, German, Italian, Japanese and Spanish editions of *Scientific American.*

[3] CHAITIN, G.J., "A random walk in arithmetic," *New Scientist* 125, No. 1709 (24 March 1990), pp. 44–46. Reprinted in N. Hall, *The New Scientist Guide to Chaos,* Penguin, 1991. Catalan translation: "Un passeig aleatori a l'aritmètica," *Butlletí de la Societat Catalana de Matemàtiques,* Número 5, Setembre 1990, pp. 9–13.

[4] CHAITIN, G.J., "Le hasard des nombres," *La Recherche* 22, N° 232 (mai 1991), pp. 610–615. Also published in the Spanish edition of *La Recherche.*

[5] CHAITIN, G.J., Lectures on "Number and randomness" and "Randomness in arithmetic," in M.E. Carvallo, *Nature, Cognition and System,* Vol. 3, Kluwer, 1992.

[6] CHAITIN, G.J., *Algorithmic Information Theory,* 3rd Printing, Cambridge University Press, 1990.

[7] CHAITIN, G.J., *Information, Randomness & Incompleteness—Papers on Algorithmic Information Theory,* 2nd Edition, World Scientific, 1990.

BOOK REVIEW

The Mathematical Intelligencer
14, No. 4 (Fall 1992), pp. 69–71

A Diary on Information Theory by Alfréd Rényi
Chichester: John Wiley & Sons, 1987; ix + 125 pp.
Hardcover, US$54.95 (ISBN 0–471–90971–8)
Reviewed by Gregory J. Chaitin

> Can the difficulty of an exam be measured by how many bits of information a student would need to pass it? This may not be so absurd in the encyclopedic subjects but in mathematics it doesn't make any sense since things follow from each other and, in principle, whoever knows the bases knows everything. All of the results of a mathematical theorem are in the axioms of mathematics in embryonic form, aren't they? *I will have to think this over some more.*
>
> A. Rényi (*A Diary on Information Theory*, p. 31)

This remarkable quotation comes from Rényi's unfinished 1969 manuscript, written in the form of a fictitious student's diary. This "diary" comprises the bulk of Rényi's posthumous work, *A Diary on Information Theory,* a stimulating introduction to information theory and an

essay on the mathematical notion of information, a work left incomplete at Rényi's death in 1970 at the age of 49.

Alfréd Rényi was a member of the Hungarian Academy of Sciences. The *Diary,* as well as the material on information theory in his two books on probability theory [1, 2], attest to the importance he attached to the idea of information. This *Diary* also illustrates the importance that Rényi ascribed to wide-ranging nontechnical discussions of mathematical ideas as a way to interest students in mathematics. He believed the discussions served as vital teaching tools and stimuli for further research.

Rényi was part of the tidal wave of interest in information theory provoked by Claude Shannon's publications in the 1940s. The many papers with titles like "Information Theory, Photosynthesis, and Religion" actually published illustrate the tremendous and widespread initial interest in information theory.

When Rényi wrote his *Diary,* the initial wave of interest in information theory was dying out. In fact, Rényi was unaware of a second major wave of interest in information theory slowly beginning to gather momentum in the 1960s. At that time, Andrei Kolmogorov and I independently proposed a new **algorithmic** information theory to capture mathematically the notion of a random, patternless sequence as one that is algorithmically incompressible.

The development of this new information theory was not as dramatically abrupt as was the case with Shannon's version. It was not until the 1970s that I corrected the initial definitions. The initial definitions Kolmogorov and I proposed had serious technical deficiencies which led to great mathematical awkwardness. It turned out that a few changes in the definitions led to a revised algorithmic information theory whose elegant formulas closely mirror those in Shannon's original theory in a radically altered interpretation [3].

In the 1970s I also began to apply algorithmic information theory to extend and broaden Gödel's incompleteness theorem, culminating in the 1980s in an explicit constructive proof that there is randomness in arithmetic [4]. (For recent discussions of algorithmic information theory directed to the general scientific public, see [5–16].)

Rényi's *Diary* stops at the brink between Shannon's ensemble information theory and the newer algorithmic information theory applying

to individual sequences. With the benefit of hindsight, one can detect the germ of ideas that, if Rényi had pursued them properly, might have led him in the direction of algorithmic information theory.

Let us take the quotation at the head of this review. If Rényi had developed it properly, it might have led him to my insight that incompleteness can be obtained very naturally via metatheorems whose spirit can be summarized in the phrase, "a theorem cannot contain more information than the axioms from which it is deduced." I think this new information-theoretic viewpoint makes incompleteness seem a much more menacing barrier than before.

A second instance occurs later in Rényi's *Diary,* p. 41:

> Therefore, the method of investigating the redundancy of a text by erasing and reconstruction is not appropriate. By this method, we would get a correct estimation of the real redundancy only if the reconstruction could be done by a computer. In that case, the meaning of the text wouldn't be a factor because a computer wouldn't understand it and could reconstruct it only by means of a dictionary and grammatical rules.

If Rényi could have formalized this, perhaps he might have discovered the complexity measure used in algorithmic information theory. (In algorithmic information theory, the complexity of a string or sequence of symbols is defined to be the size of the smallest computer program for calculating that string of symbols.)

So Rényi's *Diary* balances on the edge between the old and the new versions of information theory. It also touches on connections between information theory and physics and biology that are still the subject of research [7, 8].

In what remains of this review, I would like to flesh out the above remarks by discussing Hilbert's tenth problem in the light of algorithmic information theory. I will end with a few controversial remarks about the potential significance of these information-theoretic metamathematical results, and their connection with experimental mathematics and the quasi-empirical school of thought regarding the foundations of mathematics.

Consider a diophantine equation

$$P(k, x_1, x_2, \ldots) = 0$$

with parameter k. Ask the question, "Does $P(k) = 0$ have a solution?" Let

$$q = q_0 q_1 q_2 \cdots$$

be the infinite bit string with

$$q_k = \begin{cases} 0 & \text{if } P(k) = 0 \text{ has no solution} \\ 1 & \text{if } P(k) = 0 \text{ has a solution.} \end{cases}$$

Let

$$q^n = q_0 q_1 \cdots q_{n-1}$$

be the string of the first n bits of the infinite string q, that is, the string of answers to the first n questions. Let $H(q^n)$ be the complexity of q^n, that is, the size in bits of the smallest program for computing q^n.

If Hilbert had been right and every mathematical question had a solution, then there would be a finite set of axioms from which one could deduce whether $P(k) = 0$ has a solution or not for each k. We would then have

$$H(q^n) \leq H(n) + c.$$

The c bits are the finite amount of information in our axioms, and this inequality asserts that if one is given n, using the axioms one can compute q^n, that is, decide which among the first n cases of the diophantine equation have solutions and which do not. Thus, the complexity $H(q^n)$ of answering the first n questions would be at most order of $\log n$ bits. We ignore the immense **time** it might take to deduce the answers from the axioms; we are concentrating on the amount of **information** involved.

In 1970, Yuri Matijasevič showed that there is no algorithm for deciding if a diophantine equation can be solved. However, if we are told the number m of equations $P(k) = 0$ with $k < n$ that have a solution, then we can eventually determine which do and which do not. This shows that

$$H(q^n) \leq H(n) + H(m) + c'$$

for some $m \leq n$, which implies that the complexity $H(q^n)$ of answering the first n questions **is still** at most order of $\log n$ bits. So from an information-theoretic point of view, Hilbert's tenth problem, while undecidable, does not look too difficult.

In 1987, I explicitly constructed [4] an exponential diophantine equation

$$L(k, x_1, x_2, \ldots) = R(k, x_1, x_2, \ldots)$$

with a parameter k. This equation gives complete randomness as follows. Ask the question, "Does $L(k) = R(k)$ have infinitely many solutions?" Now let

$$q = q_0 q_1 q_2 \cdots$$

be the infinite bit string with

$$q_k = \begin{cases} 0 & \text{if } L(k) = R(k) \text{ has finitely many solutions} \\ 1 & \text{if } L(k) = R(k) \text{ has infinitely many solutions.} \end{cases}$$

As before, let

$$q^n = q_0 q_1 \cdots q_{n-1}$$

be the string of the first n bits of the infinite string q, that is, the string of answers to the first n questions. Let $H(q^n)$ be the complexity of q^n, that is, the size in bits of the smallest program for computing q^n. Now we have

$$H(q^n) \geq n - c'',$$

that is, the string of answers to the first n questions q^n is irreducible mathematical information and the infinite string of answers $q = q_0 q_1 q_2 \cdots$ is now algorithmically random.

Surprisingly, Hilbert was wrong to assume that every mathematical question has a solution. The above exponential diophantine equation yields an infinite series of independent irreducible mathematical facts. It yields an infinite series of questions which reasoning is impotent to answer because the only way to answer these questions is to assume each individual answer as a new axiom! Here one can get out as theorems only what one explicitly puts in as axioms, and reasoning is completely useless! I think this information-theoretic approach to incompleteness makes incompleteness look much more natural and pervasive than has previously been the case. Algorithmic information theory

provides some theoretical justification for the experimental mathematics made possible by the computer and for the new quasi-empirical view of the philosophy of mathematics that is displacing the traditional formalist, logicist, and intuitionist positions [5].

References

1. Alfréd Rényi, Introduction to information theory, *Probability Theory,* Amsterdam: North-Holland (1970), 540–616.

2. Alfréd Rényi, Independence and information, *Foundations of Probability,* San Francisco: Holden-Day (1970), 146–157.

3. Gregory J. Chaitin, A theory of program size formally identical to information theory, *Information, Randomness & Incompleteness—Papers on Algorithmic Information Theory,* Second Edition, Singapore: World Scientific (1990), 113–128.

4. Gregory J. Chaitin, *Algorithmic Information Theory,* Cambridge: Cambridge University Press (1987).

5. John L. Casti, Proof or consequences, *Searching for Certainty,* New York: Morrow (1990), 323–403.

6. Gregory J. Chaitin, A random walk in arithmetic, *The New Scientist Guide to Chaos* (Nina Hall, ed.), Harmondsworth: Penguin (1991), 196–202.

7. David Ruelle, Complexity and Gödel's theorem, *Chance and Chaos,* Princeton: Princeton University Press (1991), 143–149.

8. David Ruelle, Complexité et théorème de Gödel, *Hasard et Chaos,* Paris: Odile Jacob (1991), 189–196.

9. Luc Brisson and F. Walter Meyerstein, Que peut nous apprendre la science?, *Inventer L'Univers,* Paris: Les Belles Lettres (1991), 161–197.

10. Gregory J. Chaitin, Le hasard des nombres, *La Recherche* 22 (1991) no. 232, 610–615.

11. John A. Paulos, Complexity of programs, Gödel and his theorem, *Beyond Numeracy,* New York: Knopf (1991), 47–51, 95–97.

12. John D. Barrow, Chaotic axioms, *Theories of Everything,* Oxford: Clarendon Press (1991), 42–44.

13. Tor Nørretranders, Uendelige algoritmer, *Mærk Verden,* Denmark: Gyldendal (1991), 65–91.

14. Martin Gardner, Chaitin's omega, *Fractal Music, Hypercards and More...,* New York: Freeman (1992), 307–319.

15. Paul Davies, The unknowable, *The Mind of God,* New York: Simon & Schuster (1992), 128–134.

16. Gregory J. Chaitin, Zahlen und Zufall, *Naturwissenschaft und Weltbild* (Hans-Christian Reichel and Enrique Prat de la Riba, eds.), Vienna: Verlag Hölder–Pichler–Tempsky (1992), 30–44.

IBM Research Division
Yorktown Heights, NY 10598 USA

Epilogue
The Challenge for the Future

"This is my hand. I can move it, feel the blood pulsing through it. The sun is still high in the sky and I, Antonius Block, am playing chess with Death."

—The KNIGHT in
INGMAR BERGMAN's 1956 film
The Seventh Seal

COMPLEXITY AND BIOLOGY

New Scientist 132, No. 1789 (5 October 1991), p. 52

Information and the Origin of Life
by Bernd-Olaf Küppers, *MIT Press, pp 215, £20.25*

Gregory Chaitin

BERND-OLAF Küppers' *Information and the Origin of Life,* originally published in German several years ago, belongs to that handful of books, including Erwin Schrödinger's 1944 classic *What is Life?,* that confront the most fundamental conceptual problems of biology.

What are these fundamental problems? In a more practical domain, Sydney Brenner, I believe, put it succinctly. "Genetic engineering," he said, "is being able to build a centaur!" At a more conceptual level, the problem, a physicist might say, is to develop a thermodynamic or statistical mechanics theory of the origin and evolution of life; while a mathematician would say that it is to prove when life must arise and evolve, and what its rate of evolution is.

Such a theory would have to tell us how likely life is to appear and

evolve, to give us a feel for how common life is in the Universe, and whether it is ubiquitous or extremely rare.

This book discusses the connection between biological information, the mathematical theory of information and the newer algorithmic version of information theory. I think it is fair to say that, in spite of the interesting points of contact between biology and information theory, much remains to be done and we are far from being able to answer the fundamental questions.

From where is progress likely to come?

On the one hand, rapid advances in understanding the molecular biology of embryogenesis and development may suggest new versions of algorithmic information theory more in tune with the actual "computer programming" language used by DNA to describe how to build organisms.

And I hope that one day we will visit other planets and other solar systems and get a feel for whether life is common or rare, so that even if theoreticians make no progress space exploration will eventually give us the answer. In the short term, I expect experiments with "artificial life" on massively parallel computers will lead to theoretical developments. [See STEVEN LEVY, *Artificial Life,* New York: Pantheon Books (1992).]

In summary, I would like to repeat a story from Abraham Pais's forthcoming book *Niels Bohr's Times* (Oxford University Press, pp 565, £25). According to Pais, Bohr told the following story: "Once upon a time a young rabbinical student went to hear three lectures by a famous rabbi. Afterwards he told his friends: 'The first talk was brilliant, clear and simple. I understood every word. The second was even better, deep and subtle. I didn't understand much, but the rabbi understood all of it. The third was by far the finest, a great and unforgettable experience. I understood nothing and the rabbi didn't understand much either.' "

Information and the Origin of Life belongs to the latter class. It reminds us that in spite of the splendid achievements of molecular biology, there is still much that we do not understand and much to be done. □

Gregory Chaitin is at IBM's Thomas J. Watson Research Center in New York.

AFTERWORD

A life is a whirlwind of experiences. Here are some intense experiences that stick out in my mind:

Making love.
Proving a significant theorem.
Hiking up a mountain all morning in fog, mud and drizzle then suddenly on rock at summit above cloud in dazzling sunshine (upstate New York).
Inventing a new mathematical concept.
Crossing a snow bridge in the Argentine Andes, unroped.
Swimming from one British Virgin Island to another.
Making it back to a rowing club sculling against the current after a long day in the Tigre river delta in Argentina.
Writing a large computer program.
Getting a large computer program to work.

Dancing in the street in Rio de Janiero at Carnival, with la porta-bandeira of an escola de samba.
Writing my Cambridge University Press book.
Lecturing on randomness in arithmetic in GÖDEL's classroom in the Mathematical Institute of the University of Vienna.
Meeting the KING and QUEEN of Belgium and MR HONDA at a SOLVAY conference in Brussels.

Finding the Ring nebula in Lyra with an 8" f/6 Newtonian reflector that I built myself.
The aurora shimmering in the frosty sky of northern Quebec the first week in September.

The zodiacal light[1] after sunset in the British Virgin Islands.

The brilliant Milky Way high in the Andes on the Argentine-Chile border.

A total eclipse of the sun breaking the searing noonday heat in southern India.

The white nights and the snow and ice in early May north of the arctic circle in Sweden.

Hearing GOD's thoughts in BACH's *The Art of the Fugue.*

The suffering and madness in SHOSTAKOVICH's First Violin Concerto.

A sumptuous rather than stark performance of JACQUES OFFENBACH's *The Tales of Hoffmann* at the Colon Opera House in Buenos Aires.

To Life!

GREGORY CHAITIN
April 1992

[1]Sunlight scattered from dust particles in the plane of the solar system.

Bibliography

SOURCES OF QUOTATIONS

The quotations are from:

1. Paul Arthur Schilpp, *Albert Einstein: Philosopher–Scientist,* Library of Living Philosophers, Volume VII, La Salle: Open Court (1988).

2. Constance Reid, *Hilbert,* New York: Springer-Verlag (1970).

3. Felix E. Browder, *Mathematical Developments Arising from Hilbert Problems,* Proceedings of Symposia in Pure Mathematics, Volume XXVIII, Providence: American Mathematical Society (1976).

4. Constance Reid, *Hilbert–Courant,* New York: Springer-Verlag (1986).

5. G. H. Hardy, *A Mathematician's Apology,* Cambridge: Cambridge University Press (1989).

6. J. E. Littlewood, *A Mathematician's Miscellany,* London: Methuen (1963).

7. Béla Bollobás, *Littlewood's Miscellany,* Cambridge: Cambridge University Press (1988).

8. G. Polya, *How to Solve It—A New Aspect of the Mathematical Method,* Princeton: Princeton University Press (1988).

9. Eric Temple Bell, *Mathematics: Queen & Servant of Science,* Redmond: Microsoft Press (1987).

10. Mark Kac, *Enigmas of Chance—An Autobiography,* Berkeley: University of California Press (1987).

11. Ingmar Bergman, *Four Screenplays of Ingmar Bergman,* New York: Simon & Schuster (1989).

12. Norman C. Chaitin, *The Small Hours,* Film Library of the Museum of Modern Art, New York.

CLASSICS

Here is a list of some of the favorite books from my childhood, books that are especially valuable for a young self-taught mathematician:

1. George Gamow, *One, Two, Three, Infinity,* New York: Dover (1988).

2. George Gamow, *Mr Tompkins in Paperback,* Cambridge: Cambridge University Press (1987).

3. Albert Einstein and Leopold Infeld, *The Evolution of Physics from Early Concepts to Relativity and Quanta,* New York: Simon & Schuster (1966).

4. W. W. Sawyer, *Mathematician's Delight,* Harmondsworth: Penguin (1991).

5. Tobias Dantzig, *Number, the Language of Science,* New York: Macmillan (1954).

6. Eric Temple Bell, *Mathematics: Queen & Servant of Science,* Redmond: Microsoft Press (1987).

7. Richard Courant and Herbert Robbins, *What is Mathematics? An Elementary Approach to Ideas and Methods,* Oxford: Oxford University Press (1978).

8. Hans Rademacher and Otto Toeplitz, *The Enjoyment of Mathematics—Selections from Mathematics for the Amateur,* New York: Dover (1990).

9. G. POLYA, *How to Solve It—A New Aspect of the Mathematical Method,* Princeton: Princeton University Press (1988).

10. G. POLYA, *Mathematics and Plausible Reasoning,* (two volumes), Princeton: Princeton University Press (1990).

11. G. H. HARDY, *A Mathematician's Apology,* Cambridge: Cambridge University Press (1989).

12. G. H. HARDY and E. M. WRIGHT, *An Introduction to the Theory of Numbers,* Oxford: Clarendon Press (1990).

13. G. H. HARDY, *A Course of Pure Mathematics,* Cambridge: Cambridge University Press (1952).

14. JAMES R. NEWMAN, *The World of Mathematics,* (four volumes), Redmond: Microsoft Press (1988).

About the author

Gregory J Chaitin *is a member of the Computer Science Department at the IBM Thomas J Watson Research Center in Yorktown Heights, New York. He created algorithmic information theory in the mid 1960's when he was a teenager. In the quarter century since he has been the principal architect of the theory. His contributions include: the definition of a random sequence via algorithmic incompressibility, the reformulation of program-size complexity in terms of self-delimiting programs, the definition of the relative complexity of one object given a minimal-size program for another, the discovery of the halting probability Omega and its significance, the information-theoretic approach to Gödel's incompleteness theorem, the discovery that the question of whether an exponential diophantine equation has finitely or infinitely many solutions is in some cases absolutely random, and the theory of program size for Turing machines and for LISP. He is the author of the monograph "Algorithmic Information Theory" published by Cambridge University Press in 1987.*

INFORMATION-THEORETIC INCOMPLETENESS

World Scientific Series in Computer Science — Vol. 35

by Gregory J Chaitin (IBM)

In this mathematical autobiography Gregory Chaitin presents a technical survey of his work and a nontechnical discussion of its significance. The volume is an essential companion to the earlier collection of Chaitin's papers *INFORMATION, RANDOMNESS & INCOMPLETENESS,* also published by World Scientific.

The technical survey contains many new results, including a detailed discussion of LISP program size and new versions of Chaitin's most fundamental information-theoretic incompleteness theorems. The nontechnical part includes the lecture given by Chaitin in Gödel's classroom at the University of Vienna, a transcript of a BBC TV interview, and articles from *NEW SCIENTIST, LA RECHERCHE,* and *THE MATHEMATICAL INTELLIGENCER.*

Readership: Computer scientists, mathematicians, physicists, philosophers and biologists.