

# META-MATHEMATICS AND THE FOUNDATIONS OF MATHEMATICS

G. J. Chaitin

*chaitin@us.ibm.com*

*<http://www.cs.auckland.ac.nz/CDMTCS/chaitin>*

## Abstract

This article discusses what can be proved about the foundations of mathematics using the notions of algorithm and information. The first part is retrospective, and presents a beautiful antique, Gödel's proof; the first modern incompleteness theorem, Turing's halting problem; and a piece of postmodern metamathematics, the halting probability  $\Omega$ . The second part looks forward to the new century and discusses the convergence of theoretical physics and theoretical computer science and hopes for a theoretical biology, in which the notions of algorithm and information are again crucial.

## PART I—THREE INCOMPLETENESS THEOREMS

In this article I'm going to concentrate on what we can **prove** about the foundations of mathematics using mathematical methods, in other words, on metamathematics. The current point of departure for metamathematics is that you're doing mathematics using an artificial language and you pick a **fixed** set of axioms and rules of inference (deduction rules), and everything is done so precisely that there is a proof-checking algorithm. I'll call such a formal system a formal axiomatic theory.

Then, as is pointed out in Turing's original paper (1936), and as was emphasized by Post in his *American Mathematical Society Bulletin* paper (1944), the set  $X$  of all theorems, consequences of the axioms, can be systematically generated by running through all possible proofs in size order and mechanically checking which ones are valid. This unending computation, which would be monumentally time-consuming, is sometimes jocularly referred to as the British Museum algorithm.

The size in bits  $H(X)$  of the program that generates the set  $X$  of theorems—that's the program-size complexity of  $X$ —will play a crucial role below. Roughly speaking, it's the number of bits of axioms in the formal theory that we are

considering.  $H(X)$  will give us a way to measure the algorithmic complexity or the algorithmic information content of a formal axiomatic theory.

But first let's retrace history, starting with a beautiful antique, Gödel's incompleteness theorem, the very first incompleteness theorem.

- Alan Turing (1936), "On computable numbers, with an application to the entscheidungsproblem," *Proceedings of the London Mathematical Society*, ser. 2, vol. 42, pp. 230–265. Reprinted in Davis (1965), pp. 115–154.
- Emil Post (1944), "Recursively enumerable sets of positive integers and their decision problems," *American Mathematical Society Bulletin*, vol. 50, pp. 284–316. Reprinted in Davis (1965), pp. 304–337.
- Martin Davis (1965), *The Undecidable*, Raven Press.

## A Beautiful Antique: Gödel's Proof (1931)

Let's fix our formal axiomatic theory as above and ask if "This statement is unprovable!" can be proven within our theory. "This statement is unprovable!" is provable if and only if it's false! "This statement is unprovable!" doesn't sound at all like a mathematical statement, but Gödel shows that it actually is.

Therefore formal axiomatic theories, if they only prove true theorems, are incomplete, because they do not prove all true statements. And so true and provable turn out to be rather different!

How does Gödel's proof work? Gödel cleverly constructs an arithmetical or number-theoretic assertion that refers to itself and its unprovability indirectly, via the (Gödel) numbers of the statements and proofs within the formal theory. In other words, he numbers all the statements and proofs within the formal axiomatic theory, and he can then construct a (very complicated) bona fide mathematical assertion that states that it itself is unprovable. The self-reference is indirect, since Gödel's self-referential statement cannot contain its own Gödel number.

Wonderful as it is, Gödel's proof does not involve three of the big ideas of the 20th century, **algorithm**, **information** and **randomness**. The first step in that direction was taken by Turing only five years later.

But before discussing Turing's work, what is an algorithm?

- Kurt Gödel (1931), "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I" ["On formally undecidable propositions of *Principia Mathematica* and related systems I"], *Monatshefte für Mathematik und Physik*, vol. 38, pp. 173–198. English translation in Davis (1965), pp. 4–38. [Very difficult to understand.]
- Ernest Nagel, James R. Newman (1958), *Gödel's Proof*, New York University Press. [A beautifully clear explanation.]

## What is an Algorithm? [McCarthy (1962), Chaitin (1998, 1999, 2001)]

An algorithm is a mechanical procedure for calculating something, usually formulated in a programming language, for example LISP, which is a computable version of set theory!

In LISP, which is my favorite programming language, applying the function  $f$  to the operands  $x$  and  $y$ ,  $f(x, y)$ , is written `(f x y)`.

Programs and data in LISP are all S-expressions, which are lists with sublists, enclosed in parentheses and with successive elements separated by blanks. For example, `(A BC (123 DD))` is an S-expression. The S in S-expression stands for “symbolic.”

For example, let’s define set membership in LISP.

```
(define (in-set? member set)
  (if (= () set) false
      (if (= member (head set)) true
          (in-set? member (tail set))
          ))
  )
```

This defines `(in-set? member set)` to be `false` if the set is empty, `true` if the member is the first element in the set, and to recursively be `(in-set? member [the rest of the set])` otherwise.

Let me explain some of this: `(if x y z)` yields/picks  $y$  or  $z$  depending on whether or not  $x$  is `true`. And `(= x y)` checks if  $x = y$ , yielding `true` or `false`.

Unfortunately, for historical reasons, `head` is actually written `car` and `tail` is actually written `cdr`.

Then

```
(in-set? (' y) (' (x y z)))
```

yields `true`, and

```
(in-set? (' q) (' (x y z)))
```

yields `false`.

Here `'` or *quote* stops evaluation and contains unevaluated data.

In summary, a LISP program isn’t a list of statements that you execute or run, it’s an expression to be evaluated, and what it does, is it yields a value. In other words, LISP is a functional programming language, not an imperative programming language.

- John McCarthy et al. (1962), *LISP 1.5 Programmer’s Manual*, Massachusetts Institute of Technology Press.

## The First Modern Incompleteness Theorem: Turing's Halting Problem (1936)

At the beginning of his 1936 paper, Turing provides a mathematical definition of the notion of algorithm. He does this using an extremely primitive programming language (now called a Turing machine), not LISP as above, but it is nevertheless a decisive intellectual step forward into the computer age.<sup>1</sup> He then proves that there are things which cannot be computed. Turing does this by making brilliant use of Cantor's diagonal argument from set theory applied to the list of all computable real numbers. This gives Turing an uncomputable real number  $R^*$ , as we explain below, and a completely different source of incompleteness than the one discovered by Gödel in 1931. Real numbers are numbers like 3.1415926...

Imagine a numbered list of all possible computer programs for computing real numbers. That is, a list of all possible computer programs in some fixed language, ordered by size, and within programs of the same size, in some arbitrary alphabetical order. So  $R(N)$  is the real number (if any) that is computed by the  $N$ th program in the list ( $N = 1, 2, 3, \dots$ ).

Let  $R(N, M)$  be the  $M$ th digit after the decimal point of the  $N$ th computable real, that is, the real  $R(N)$  calculated by the  $N$ th program. Define a new real  $R^*$  whose  $N$ th digit after the decimal point,  $R^*(N)$ , is 3 if  $R(N, N)$  is not equal to 3, and otherwise is 2 (including the case that the  $N$ th computer program never outputs an  $N$ th digit). Then  $R^*$  is an uncomputable real, because it differs from the  $N$ th computable real in the  $N$ th digit. Therefore there cannot be any way to decide if the  $N$ th computer program ever outputs an  $N$ th digit, or we could actually compute  $R^*$ , which is impossible.

*Corollary:* No formal axiomatic theory can always enable you to prove whether or not the  $N$ th computer program ever outputs an  $N$ th digit, because otherwise you could run through all possible proofs in size order and compute  $R^*$ , which is impossible.

*Note:* Whether the  $N$ th computer program ever outputs an  $N$ th digit is a special case of the halting problem, which is the problem of determining whether or not a computer program ever halts (with no time limit). If a program does halt, you can eventually determine that, merely by running it. The real problem is to decide that a program will never halt, no matter how much time you give it.

---

<sup>1</sup>In Chaitin (1999) I discuss the halting problem and Gödel's proof using LISP.

## Postmodern Metamathematics: The Halting Probability $\Omega$ [Chaitin (1975, 1987, 1998), Delahaye (2002)]

So that's how Turing brought the notion of **algorithm** into metamathematics, into the discussion about the foundations of mathematics. Now let's find yet another source of incompleteness, and let's bring into the discussion the notions of **information, randomness, complexity** and **irreducibility**. First we need to define a certain kind of computer, or, equivalently, to specify its binary machine language.

What is the "self-delimiting binary universal computer"  $U$  that we use below?

$U$ 's program is a finite bit string, and starts off with a prefix that is a LISP expression. The LISP expression prefix is converted into binary, yielding 8 bits per character, and it's followed by a special punctuation character, 8 more bits, and that is followed by raw binary data. The LISP prefix is evaluated or run and it will read the raw binary data in one bit at a time without ever being allowed to run off the end of the program.

In other words, the LISP prefix must ask for exactly the correct number of bits of raw binary data. If it requests another bit after reading the last one, this does not return a graceful "end of file" condition, it aborts the computation.

The fact that there is no punctuation marking the end of the raw binary data, which is also the end of the entire program, is what forces these machine-language programs to be self-delimiting. In other words, the end of the binary program is like a cliff, and the computer  $U$  must not fall off!

What is the halting probability  $\Omega$  for  $U$ ?

$$\Omega = \sum_{p \text{ halts when run on } U} 2^{-(\text{the number of bits in } p)}.$$

So if the program  $p$  halts and is  $K$  bits long, that contributes  $1/2^K$  to the halting probability  $\Omega$ .

This halting probability can be defined in such a manner that it includes binary programs  $p$  of every possible size precisely because these  $p$  must be self-delimiting. That is to say, precisely because  $U$  must decide **by itself** where to stop reading the program  $p$ .  $U$  must not overshoot, it cannot fall off the cliff, it must read precisely up to the last bit of  $p$ , but not beyond it.

Knowing the first  $N$  bits of the base-two representation of the real number  $\Omega$ , which is a probability and therefore between zero and one, answers the halting problem for all programs up to  $N$  bits in size. So if you knew the first  $N$  bits of  $\Omega$  after the decimal or the binary point, that would theoretically enable you to decide whether or not each binary computer program  $p$  up to  $N$  bits in size halts when run on  $U$ .

Would this be useful? Yes, indeed! Let me give an example showing just how very useful it would be.

Let's consider the Riemann hypothesis, a famous mathematical conjecture that is still open, still unresolved. There is a Riemann-hypothesis testing program that systematically searches for counter-examples and that halts if and only if it finds one and the Riemann hypothesis is false. The size in bits of this program is the program-size complexity of testing the Riemann hypothesis this way. And if this program is  $H(\text{Riemann})$  bits long, knowing that many initial bits of  $\Omega$  would settle the Riemann hypothesis! It would enable you to tell whether or not the Riemann hypothesis is true.

Unfortunately the method required to do this, while theoretically sound, is totally impractical. The computation required, while finite, is much, much too long to actually carry out. The time needed grows much, much faster than exponentially in  $H(\text{Riemann})$ , the number of bits of  $\Omega$  that we are given. In fact, it grows as the time required to simultaneously run all programs on  $U$  up to  $H(\text{Riemann})$  bits in size until all the programs that will ever halt have done so. More precisely, you have to run enough programs for enough time to get the first  $H(\text{Riemann})$  bits of  $\Omega$  right, which because of carries from bits that are further out may actually involve programs that are more than  $H(\text{Riemann})$  bits long.

And precisely because the bits of  $\Omega$  are so useful, it turns out that they are irreducible mathematical information, that they cannot be derived or deduced from any simpler principles. More precisely, we have the following incompleteness result: You need an  $N$ -bit formal axiomatic theory (that is, one that has an  $N$ -bit algorithm to generate all the theorems) in order to be able to determine the first  $N$  bits of  $\Omega$ , or, indeed, the values and positions of any  $N$  bits of  $\Omega$ .

Actually, what I show in Chaitin (1998) is that an  $N$ -bit theory can't determine more than  $N + c$  bits of  $\Omega$ , where the constant  $c$  is 15328.

Let's restate this. Consider a formal axiomatic theory with the set of theorems  $X$  and with algorithmic complexity or algorithmic information content  $H(X)$ . Then if a statement such as

"The 99th bit of  $\Omega$  is 0."

"The 37th bit of  $\Omega$  is 1."

determining the value of a particular bit of  $\Omega$  in a particular place, is in  $X$  only if it's true, then there are at most  $H(X) + c$  such theorems in  $X$ . In other words,  $X$  enables us to determine at most  $H(X) + c$  bits of  $\Omega$ .

We can also describe this irreducibility non-technically, but very forcefully, as follows: Whether each bit of  $\Omega$  is a 0 or a 1 is a mathematical fact that is true for no reason, it's true by accident!

## **What is Algorithmic Information? [Chaitin (1975, 1987, 2001), Calude (2002)]**

$\Omega$  is actually just one piece of my algorithmic information theory (AIT), it's the jewel that I discovered while I was developing AIT. Let me now give some highlights of AIT.

What else can we do using the computer  $U$  that we used to define  $\Omega$ ? Well, you should look at the size of programs for  $U$ .  $U$  is the yardstick you use to measure algorithmic information. And the unit of algorithmic information is the 0/1 bit.

You define the absolute algorithmic *information content*  $H(X)$  of an object (actually, of a LISP S-expression)  $X$  to be the size in bits of the smallest program for  $U$  to compute  $X$ . The *joint* information content  $H(X, Y)$  is defined to be the size in bits of the smallest program for  $U$  to compute the pair  $X, Y$ . (Note that the pair  $X, Y$  is actually  $(X Y)$  in LISP.) The *relative* information content  $H(X|Y)$  is defined to be the size in bits of the smallest program for  $U$  that computes  $X$  from a minimum-size program for  $Y$ , not from  $Y$  directly.

And the complexity  $H(X)$  of a formal axiomatic theory with theorem set  $X$  is also defined using the computer  $U$ . (I glossed over this point before.)  $H(X)$  is defined to be the size in bits of the smallest program that makes  $U$  generate the set of theorems  $X$ . Note that this is an endless computation. You may think of  $H(X)$  as the number of bits of information in the most concise or the most elegant set of axioms that yields the set of theorems  $X$ .

Now here are some of the theorems that you can prove about these concepts.

First of all, let's consider an  $N$ -bit string  $X$ .  $H(X)$  is usually close to  $N + H(N)$ , which is approximately  $N + \log_2 N$ . Bit strings  $X$  for which this is the case are said to be **algorithmically random**, they have the highest possible information content.

On the other hand, an infinite sequence of bits  $X$  is defined to be algorithmically random if and only if there is a constant  $c$  such that  $H$ (the first  $N$  bits of  $X$ ) is greater than  $N - c$  for all  $N$ . And, crucial point, the base-two representation for  $\Omega$  satisfies this definition of algorithmic randomness, which is one of the reasons that  $\Omega$  is so interesting.

Algorithmic information is (sub)additive:

$$H(X, Y) \leq H(X) + H(Y) + c.$$

And the *mutual* information content  $H(X : Y)$  is defined to be the extent to which computing two objects together is better than computing them separately:

$$H(X : Y) = H(X) + H(Y) - H(X, Y).$$

$X$  and  $Y$  are said to be *algorithmically independent* if their mutual information is small compared with their individual information contents, so that

$$H(X, Y) \approx H(X) + H(Y).$$

Finally, here are some subtle results that relate mutual and relative information:

$$H(X, Y) = H(X) + H(Y|X) + O(1),$$

$$H(X : Y) = H(X) - H(X|Y) + O(1),$$

$$H(X : Y) = H(Y) - H(Y|X) + O(1).$$

Here l.h.s. = r.h.s. +  $O(1)$  means that the difference between the left-hand side and the right-hand side of the equation is bounded, it's at most a fixed number of bits. Thus the mutual information is also the extent to which knowing one of a pair helps you to know the other.

These results are quoted here in order to show that  $\Omega$  isn't isolated, it's part of an elegant theory of algorithmic information and randomness, a theory of the program-size complexity for  $U$ .

Now let me tell you what I think is the significance of these incompleteness results and of  $\Omega$ .

## Is Mathematics Quasi-Empirical?

That is, is mathematics more like physics than mathematicians would like to admit? I think so!

I think that incompleteness cannot be dismissed and that mathematicians should occasionally be willing to add new axioms that are justified by experience, experimentally, pragmatically, but are not at all self-evident.<sup>2</sup> Sometimes to prove more, you need to assume more, to add new axioms! That's what my information-theoretic approach to incompleteness suggests to me.

Of course, at this point, at the juncture of the 20th and the 21st centuries, this is highly controversial. It goes against the current paradigm of what mathematics is and how mathematics should be done, it goes against the current paradigm of the nature of the mathematical enterprise. But my hope is that the 21st century will eventually decide that adding new axioms is not at all controversial, that it's obviously the right thing to do! However this radical paradigm shift may take many years of discussion and thought to be accepted, if indeed this ever occurs.

For further discussion of this quasi-empirical, experimental mathematics viewpoint, see Chaitin (1998, 1999, 2002), Tymoczko (1998), Bailey (2002).

For superb histories of many aspects of 20th century thought regarding the foundations of mathematics that we have not touched upon here, see Grattan-Guinness (2000), Tasić (2001).

## General Bibliography for Part I

- David Bailey, Jonathan Borwein, Keith Devlin (2002?), *The Experimental Mathematician*, A. K. Peters. [In preparation.]
- Cristian Calude (2002?), *Information and Randomness*, Springer-Verlag. [In preparation.]
- Gregory Chaitin (1975), "A theory of program size formally identical to information theory," *Association for Computing Machinery Journal*, vol. 22, pp. 329–340.

---

<sup>2</sup>In my opinion  $\mathbf{P} \neq \mathbf{NP}$  is a good example of such a new axiom.

- Gregory Chaitin (1987), *Algorithmic Information Theory*, Cambridge University Press.
- Gregory Chaitin (1998, 1999, 2001, 2002), *The Limits of Mathematics, The Unknowable, Exploring Randomness, Conversations with a Mathematician*, Springer-Verlag.
- Jean-Paul Delahaye (2002), “Les nombres oméga,” *Pour la Science*, mai 2002, pp. 98–103.
- Ivor Grattan-Guinness (2000), *The Search for Mathematical Roots, 1870-1940*, Princeton University Press.
- Vladimir Tasić (2001), *Mathematics and the Roots of Postmodern Thought*, Oxford University Press.
- Thomas Tymoczko (1998), *New Directions in the Philosophy of Mathematics*, Princeton University Press.

## PART II—FUTURE PERSPECTIVES

### Where is Metamathematics Going?

We need a dynamic, not a static metamathematics, one that deals with the evolution of new mathematical concepts and theories. Where do new mathematical ideas come from? Classical metamathematics with its incompleteness theorems deals with a static view of mathematics, it considers a **fixed** formal axiomatic system. But mathematics is constantly evolving and changing! Can we explain how this happens? What we really need now is a new, optimistic **dynamic** metamathematics, not the old, pessimistic **static** metamathematics.

In my opinion the following line of research is relevant and should not have been abandoned:

- Douglas Lenat (1984), “Automated theory formation in mathematics,” in W. W. Bledsoe, D. W. Loveland, *Automated Theorem Proving: After 25 Years*, American Mathematical Society, pp. 287–314.

### Where is Mathematics Going?

Will mathematics become more like biology, more complicated, less elegant, with more and more complicated theorems and longer and longer proofs?

Will mathematics become more like physics, more experimental, more quasi-empirical, with fewer proofs?

For a longer discussion of this, see the chapter on mathematics in the third millennium in Chaitin (1999).

## What is a Universal Turing Machine (UTM)?

As physicists have become more and more interested in complex systems, the notion of algorithm has become increasingly important, together with the idea that what physical systems actually do is computation. In other words, due to complex systems, physicists have begun to consider the notion of algorithm as physics. And the UTM now begins to emerge as a fundamental **physical** concept, not just a mathematical concept [Deutsch (1997), Wolfram (2002)].

To a mathematician a UTM is a formal, artificial, unambiguous language for formulating algorithms, a language that can be interpreted mechanically, and which enables you to specify any algorithm, all possible algorithms. That's why it's called "universal."

What is a UTM to a physicist? Well, it's a physical system whose repertoire of potential behavior is extremely rich, in fact maximally rich, universal, because it can carry out any computation and it can simulate the behavior of any other physical system.

These are two sides of a single coin.

And, in a sense, all of this was anticipated by Turing in 1936 when he used the word **machine**. Also the "halting problem" almost sounds like a problem in physics. It sounds very down-to-earth and concrete. It creates a mental image that is more physical than mathematical, it sounds like you are trying to stop a runaway locomotive!

After all, what you can compute depends on the laws of physics. A different universe might have different computers. So, in a way, Turing's 1936 paper was a physics paper!

- David Deutsch (1997), *The Fabric of Reality*, Penguin.
- Stephen Wolfram (2002), *A New Kind of Science*, Wolfram Media.

## Convergence of Theoretical Physics and Theoretical Computer Science

The fact that "UTM" is now in the mental tool kit of physicists as well as mathematicians is just one symptom. What we are witnessing now is broader than that, it's actually the beginning of an amazing convergence of theoretical physics and theoretical computer science, which would have seemed inconceivable just a few years ago. There are many lines of research, many threads, that now go in that direction. Let me indicate some of these here.

It is sometimes useful to think of physical systems as performing algorithms, and of the entire universe as a single giant computer. Edward Fredkin was one of the earliest proponents of this view. See Wright (1988).

There is an increasing amount of work by physicists that suggests that it is fertile to view physical systems as information-processing systems, and that studies how physical systems process information. The extremely popular field

of research of quantum computation and quantum information [Nielsen (2000)] certainly follows this paradigm.

And there are also suggestions from black hole thermodynamics and quantum mechanics that the physical universe may actually be discrete, not continuous, and that the maximum amount of information contained in a physical system is actually finite. A leading researcher in this area is Jacob Bekenstein, and for more on this topic, see the chapter on the holographic principle in Smolin (2001).

Wolfram (2002) is a treasure-trove of simple combinatorial (symbolic, discrete, non-numerical) algorithms with extremely rich behavior (in fact, universal behavior, equivalent to a UTM, a universal Turing machine, in other words, that can perform an arbitrary computation and simulate an arbitrary physical system). These are superb building blocks that God might well have used in building the universe! Here I refer to high-energy or particle physics. Time will tell—we will see! Wolfram also supports, and himself employs, an experimental, quasi-empirical approach to doing mathematics.

Let me also cite the physicist who might be considered the inventor of quantum computing, and also cite a journalist. See the chapter on “Universality and the limits of computation” in Deutsch (1998), and Siegfried (2000) on the physics of information and the seminal ideas of Rolf Landauer.

I should mention that my own work on AIT in a sense belongs to this school; it can be viewed as an application of the physical notion of entropy (or disorder) to metamathematics. In other words, my work on  $\Omega$  in a sense treats formal axiomatic theories as if they were heat engines. That is how I show that  $\Omega$  is irreducible, using general, “thermodynamical” arguments on the limitations of formal axiomatic theories.

Another example of ideas from physics that are invading computer science are the phase changes that mark a sharp transition from a regime in which an algorithm is fast to a regime in which the algorithm is extremely slow, for instance the situation described in Hayes (2002).

- Robert Wright (1988), *Three Scientists and Their Gods*, Times Books. [On Edward Fredkin.]
- Michael Nielsen, Isaac Chuang (2000), *Quantum Computation and Quantum Information*, Cambridge University Press.
- Tom Siegfried (2000), *The Bit and the Pendulum*, Wiley. [For the work of Rolf Landauer.]
- Lee Smolin (2001), *Three Roads to Quantum Gravity*, Basic Books.
- Brian Hayes (2002), “The easiest hard problem,” *American Scientist*, vol. 90, pp. 113–117.

## To a Theoretical Biology

What is life? Can there be a general, abstract mathematical theory of the origin and the evolution of the complexity of life?

Ergodic theory says that things get washed out and less interesting as time passes. The theory I want would show that interesting things (life! organisms! us!) emerge and evolve spontaneously, and that things get more and more interesting, not less and less interesting.

My old attempt at a theory [Chaitin (1970, 1979)] considered cellular automata and proposed using mutual algorithmic information to distinguish a living organism from its environment. My idea was that the parts of an organism have high mutual information.

Wolfram (2002) sustains the thesis that life is not unusual. He claims that there is no essential difference between us and any other universal Turing machine. Furthermore, according to Wolfram, most non-trivial physical and combinatorial systems are universal Turing machines, UTM's. (A UTM is a physical system whose behavior is as rich as possible because it is a general-purpose computer that can perform an arbitrary computation, in other words, that can simulate any algorithm and any other physical system.) Therefore, according to Wolfram, there is nothing to Darwin, nothing to understand. The evolution of life is a non-issue. According to Wolfram you get there, you get life, right away, all at once.

Wolfram's thesis, while interesting, is not, I believe, the entire story. Universal Turing machines, computation, may be ubiquitous in nature, but the amount of software a UTM has is not taken into account by Wolfram. And that, I believe, is what is actually evolving! After all, DNA is essentially digital software, and we have much more DNA than viruses and bacteria. Our program-size complexity is higher,  $H(\text{human})$  is much greater than  $H(\text{bacteria})$ .

This suggests to me a new toy model of evolution very different from the cellular automata model that I originally considered. My new idea is to model life, an ecology, as a collection of UTM's, and to study how their software evolves in complexity. The problem is how to model the environment, more precisely, the interactions of organisms with each other and with their environment. Let me emphasize that in such a model the problem is not to distinguish an organism from its environment, which before I attempted to do using mutual information, it is to model interactions, so that the organisms are not like Leibniz's windowless monads! And then of course to prove that the software complexity will evolve...

For more on mutual information, see Chaitin (2001). For further discussion of my hopes for a theoretical biology, and for my thoughts on biology in general, see Chaitin (2002). For von Neumann's seminal work in this area, see von Neumann (1966). Two recent books on biology that particularly impressed me are Maynard Smith (1999), and Kay (2000).

- John von Neumann (1966), *Theory of Self-Reproducing Automata*, edited and completed by Arthur Burks, University of Illinois Press.

- Gregory Chaitin (1970, 1979), “To a mathematical definition of ‘life’,” *ACM SICTACT News*, January 1970, pp. 12–18; “Toward a mathematical definition of ‘life’,” in R. D. Levine, M. Tribus, *The Maximum Entropy Formalism*, Massachusetts Institute of Technology Press, pp. 477–498.
- John Maynard Smith, Eörs Szathmáry (1999), *The Origins of Life*, Oxford University Press.
- Lily Kay (2000), *Who Wrote the Book of Life?*, Stanford University Press. [On information theory and molecular biology.]

## To a Theoretical Psychology

What is psychological information? What is thinking? What is the soul? What is intelligence? Is it some kind of information-processing capability? How can we measure it? How can we simulate it? (That’s called AI, artificial intelligence.) And where do new ideas come from? Ideas in general, not just mathematical ideas.

See Nørretranders (1998) for some interesting discussions of precisely these questions.

However, again, I don’t just want an interesting discussion, I want a mathematical theory with beautiful theorems and proofs! Human intelligence may just be a very complicated piece of engineering, or there may be some profound, basic, as yet unknown concepts at play, and a fundamental theory about them. And these two possibilities are not mutually exclusive. Time will tell!

- Tor Nørretranders (1998), *The User Illusion*, Viking. [On information theory and psychology.]

## To the Future!

I trust that a hundred years from now mathematicians will be able to look back on our current mathematics and metamathematics the way we regard the mathematics of 1900—with a justified feeling of superiority! I hope that they will wonder, how we could have been so blind, to miss the simple, wonderful, beautiful new theories that were just around the corner? I hope that they will ask themselves, how come we didn’t see all those beautiful new ideas, when we were almost close enough to touch them and to taste them!

*University of Auckland, May 2002*