

2. Complexity, Randomness¹

In 1931 Gödel discovered incompleteness by considering “This statement is unprovable!” As we said in the previous chapter, the work of Turing in 1936 revealed that incompleteness is a corollary of a deeper phenomenon, uncomputability. In this chapter we’re going to go beyond Turing by considering an extreme form of uncomputability, namely algorithmic irreducibility and randomness.

We’ll study the halting probability Ω , an extreme case of maximal uncomputability, a place in pure math where God seems to play dice. Imagine writing the numerical value of Ω in base-two:

$$\Omega = .1101011 \dots$$

The 0, 1 bits of the numerical value of Ω are algorithmically and logically irreducible, they appear to be completely haphazard, without any structure. Each bit of Ω has got to be a 0 or a 1, but these are mathematical facts that are true for no reason, more precisely, for no reason simpler than themselves. They seem more like contingent truths than like necessary truths. And, as we shall see, Ω ’s irreducible complexity can be found throughout pure mathematics, even in such traditional fields as number theory and algebra.

In fact, as is discussed in this chapter, the reason that the bits of Ω seem to be completely accidental and random is because they are an irredundant compression of the answers to all possible instances of the halting problem. Once all the redundancy has been compressed out of something, what is left seems completely random, but is in fact packed full of useful information. Such is the case with the bits of Ω . Ω is the best way to pack all the answers to individual cases of the halting problem into a single real number.

The halting probability Ω plays a leading role in this chapter, but our goals are actually much broader. In this chapter we present an information-theoretic perspective on epistemology using software models. We shall use the notion of algorithmic information to discuss what is a physical law, to determine the limits of the axiomatic method, and to analyze Darwin’s theory of evolution.

And the best way to understand the deep concepts of conceptual complexity and algorithmic information, which are the basis for everything in this chapter, is to see how they evolved, to know their long history. Let’s start with Hermann Weyl and the great philosopher/mathematician G. W. Leibniz. That everything that is true is true for a reason is rationalist Leibniz’s famous *principle of sufficient reason*. The bits of Ω seem to refute this fundamental principle and also the idea that everything can be proved starting from self-evident facts.

¹Portions of this chapter first appeared in G. J. Chaitin, “Metaphysics, metamathematics and metabiology,” in H. Zenil, *Randomness Through Computation*, World Scientific, 2011.

Weyl, Leibniz, complexity and the principle of sufficient reason

What is a scientific theory?

The starting point of algorithmic information theory, which is the subject of this chapter, is this toy model of the scientific method:

theory/program/010 → **Computer** → experimental data/output/110100101.

A scientific theory is a computer program for producing exactly the experimental data, and both theory and data are a finite sequence of bits, a bit string. Then we can define the complexity of a theory to be its size in bits, and we can compare the size in bits of a theory with the size in bits of the experimental data that it accounts for.

That the simplest theory is best, means that we should pick the smallest program that explains a given set of data. Furthermore, if the theory is the same size as the data, then it is useless, because there is always a theory that is the same size as the data that it explains. In other words, a theory must be a compression of the data, and the greater the compression, the better the theory. Explanations are compressions, comprehension is compression!

Furthermore, if a bit string has absolutely no structure, if it is completely random, then there will be no theory for it that is smaller than it is. Most bit strings of a given size are incompressible and therefore incomprehensible, simply because there are not enough smaller theories to go around.

This software model of science is not new. It can be traced back via Hermann Weyl (1932) to G. W. Leibniz (1686)! Let's start with Weyl. In his little book on philosophy *The Open World: Three Lectures on the Metaphysical Implications of Science*, Weyl points out that if arbitrarily complex laws are allowed, then the concept of law becomes vacuous, because there is always a law! In his view, this implies that the concept of a physical law and of complexity are inseparable; for there can be no concept of law without a corresponding complexity concept. Unfortunately he also points out that in spite of its importance, the concept of complexity is a slippery one and hard to define mathematically in a convincing and rigorous fashion.

Furthermore, Weyl attributes these ideas to Leibniz, to the 1686 *Discours de métaphysique*. What does Leibniz have to say about complexity in his *Discours*? The material on complexity is in Sections V and VI of the *Discours*.

In Section V, Leibniz explains why science is possible, why the world is comprehensible, lawful. It is, he says, because God has created the best possible, the most perfect world, in that the greatest possible diversity of phenomena are governed by the smallest possible set of ideas. God simultaneously maximizes the richness and diversity of the world and minimizes the complexity of the ideas, of the mathematical laws, that determine this world. That is why science is possible!

A modern restatement of this idea is that science is possible because the world seems very complex but is actually governed by a small set of laws having low conceptual complexity.

And in Section VI of the *Discours*, Leibniz touches on randomness. He points out that any finite set of points on a piece of graph paper always seems to follow a law, because there is always a mathematical equation passing through those very points. But there is a law only if the equation is simple, not if it is very complicated. This is the idea that impressed Weyl, and it becomes the definition of randomness in algorithmic information theory.²

Finding elegant programs

So the best theory for something is the smallest program that calculates it. How can we be sure that we have the best theory? Let's forget about theories and just call a program *elegant* if it is the smallest program that produces the output that it does. More precisely, a program is elegant if no smaller program written in the same language produces the same output.

So can we be sure that a program is elegant, that it is the best theory for its output? Amazingly enough, we can't: It turns out that any formal axiomatic theory A can prove that at most finitely many programs are elegant, in spite of the fact that there are infinitely many elegant programs. More precisely, it takes an N -bit theory A , one having N bits of axioms, having complexity N , to be able to prove that an individual N -bit program is elegant. And we don't need to know much about the formal axiomatic theory A in order to be able to prove that it has this limitation.

What is a formal axiomatic theory?

All we need to know about the axiomatic theory A , is the crucial requirement emphasized by David Hilbert that there should be a proof-checking algorithm, a mechanical procedure for deciding if a proof is correct or not. It follows that we can systematically run through all possible proofs, all possible strings of characters in the alphabet of the theory A , in size order, checking which ones are valid proofs, and thus discover all the theorems, all the provable assertions in the theory A .³

That's all we need to know about a formal axiomatic theory A , that there is an algorithm for generating all the theorems of the theory. This is the software model of the axiomatic method studied in algorithmic information theory. If the software for producing all the theorems is N bits in size, then the complex-

²*Historical Note:* Algorithmic information theory was first proposed in the 1960s by R. Solomonoff, A. N. Kolmogorov, and G. J. Chaitin. Solomonoff and Chaitin considered this toy model of the scientific method, and Kolmogorov and Chaitin proposed defining randomness as algorithmic incompressibility.

³*Historical Note:* The idea of running through all possible proofs, of creativity by mechanically trying all possible combinations, can be traced back through Leibniz to Ramon Llull in the 1200s.

ity of our theory A is defined to be N bits, and we can limit A 's power in terms of its complexity $H(A) = N$. Here's how:

Why can't you prove that a program is elegant?

Suppose that we have an N -bit theory A , that is, that $H(A) = N$, and that it is always possible to prove that individual elegant programs are in fact elegant, and that it is never possible to prove that inelegant programs are elegant. Consider the following paradoxical program P :

P runs through all possible proofs in the formal axiomatic theory A , searching for the first proof in A that an individual program Q is elegant for which it is also the case that the size of Q in bits is larger than the size of P in bits. And what does P do when it finds Q ? It runs Q and then P produces as its output the output of Q .

In other words, the output of P is the same as the output of the first provably elegant program Q that is larger than P . But this contradicts the definition of elegance! P is too small to be able to calculate the output of an elegant program Q that is larger than P . We seem to have arrived at a contradiction!

But do not worry; there is no contradiction. What we have actually proved is that P can never find Q . In other words, there is no proof in the formal axiomatic theory A that an individual program Q is elegant, not if Q is larger than P . And how large is P ? Well, just a fixed number of bits c larger than N , the complexity $H(A)$ of the formal axiomatic theory A . P consists of a small, fixed main program c bits in size, followed by a large subroutine $H(A)$ bits in size for generating all the theorems of A .

The only tricky thing about this proof is that it requires P to be able to know its own size in bits. And how well we are able to do this depends on the details of the particular programming language that we are using for the proof. So to get a neat result and to be able to carry out this simple, elegant proof, we have to be sure to use an appropriate programming language. This is one of the key issues in algorithmic information theory, which programming language to use, and I'll talk about that later.⁴

Farewell to reason: The halting probability Ω ⁵

So there are infinitely many elegant programs, but there are only finitely many provably elegant programs in any formal axiomatic theory A . The proof of this is rather straightforward and short. Nevertheless, this is a fundamental information-theoretic incompleteness theorem that is rather different in style

⁴See also the chapter on "The Search for the Perfect Language" in Chaitin, *Mathematics, Complexity and Philosophy*, in press.

⁵*Farewell to Reason* is the title of a book by Paul Feyerabend, a wonderfully provocative philosopher. We borrow his title here for dramatic effect, but he does not discuss Ω in this book or any of his other works.

from the classical incompleteness results of Gödel, Turing and others that were discussed in the previous chapter.

An even more important incompleteness result in algorithmic information theory has to do with the halting probability Ω , the numerical value of the probability that a program p whose successive bits are generated by independent tosses of a fair coin will eventually halt:

$$\Omega = \sum_{p \text{ halts}} 2^{-(\text{size in bits of } p)}.$$

To be able to define this probability Ω , it is also very important how you chose your programming language. If you are not careful, this sum will diverge instead of being ≤ 1 like a well-behaved probability should.

Turing's fundamental result is that the halting problem is unsolvable. In algorithmic information theory the fundamental result is that the halting probability Ω is algorithmically irreducible or random. It follows that the bits of Ω cannot be compressed into a theory less complicated than they are. They are irreducibly complex. It takes N bits of axioms to be able to determine N bits of the numerical value

$$\Omega = .1101011 \dots$$

of the halting probability. If your formal axiomatic theory A has $H(A) = N$, then you can determine the values and positions of at most $N + c$ bits of Ω .

In other words, the bits of Ω are logically irreducible, they cannot be proved from anything simpler than they are. Essentially the only way to determine what are the bits of Ω is to add these bits to your theory A as new axioms. But you can prove anything by adding it as a new axiom. That's not using reasoning!

So the bits of Ω refute Leibniz's principle of sufficient reason: they are true for no reason. More precisely, they are not true for any reason simpler than themselves. This is a place where mathematical truth has absolutely no structure, no pattern, for which there is no theory!

*Adding new axioms: Quasi-empirical mathematics*⁶

So incompleteness follows immediately from fundamental information - theoretic limitations. What to do about incompleteness? Well, just add new axioms, increase the complexity $H(A)$ of your theory A ! That is the only way to get around incompleteness.

In other words, do mathematics more like physics, add new axioms not because they are self-evident, but for pragmatic reasons, because they help mathematicians to organize their mathematical experience just like physical theories

⁶The term *quasi-empirical* is due to the philosopher Imre Lakatos, a friend of Feyerabend. For more on this school, including the original article by Lakatos, see the collection of quasi-empirical philosophy of math papers edited by Thomas Tymoczko, *New Directions in the Philosophy of Mathematics*.

help physicists to organize their physical experience. After all, Maxwell's equations and the Schrödinger equation are not at all self-evident, but they work! And this is just what mathematicians have done in theoretical computer science with the hypothesis that $P \neq NP$, in mathematical cryptography with the hypothesis that factoring is hard, and in abstract axiomatic set theory with the new axiom of projective determinacy.⁷

This concludes the introductory historical overview. Now let's start over and do things a little more carefully.

Defining information content and conceptual complexity

Here is how we define *information content* or *conceptual complexity* for finite and infinite objects.

For the sake of brevity, I will call a Turing machine a *computer*, and will use $C(p)$ to denote the output produced by running the program p on the computer C . We are only interested in *binary* computers, so the p that we consider will always be bit strings, and we shall try to minimize the size in bits $|p|$ of p .

Pick an *optimal self-delimiting universal binary computer* U to use as our standard, official programming language. U has the property that for each self-delimiting binary computer C there exists a prefix π_C such that

$$U(\pi_C p) = C(p),$$

that is to say, U carries out the same computation that C does if U is given the program p for C preceded by a fixed prefix π_C that depends on C but not on p . Thus, programs for U are at most a fixed number of bits larger than programs for C , at most $|\pi_C|$ bits longer, in fact, which is the size in bits of a self-delimiting interpreter π_C for U to simulate C .

Self-delimiting means that π_C indicates within itself how long it is, so that when it is concatenated with the program p it is nevertheless possible for U to decide where π_C ends and p begins. The entire programs $\pi_C p$ and p for U and C respectively, are similarly required to be self-delimiting, in that U and C can realize where $\pi_C p$ and p end without have to read beyond the ends of their respective binary programs.

Henceforth we use a particular U to measure the size of programs, to define the program-size or conceptual complexity H . The choice of U does not affect things too much, since two optimal self-delimiting universal binary computers U and U' require programs of almost the same size.

For individual digital objects, $H(x) \equiv$ the size in bits of the smallest self-delimiting program p that calculates x and then halts. I.e.,

$$H(x) \equiv \min_{U(p)=x} |p|.$$

⁷See the article on "The Brave New World of Bodacious Assumptions in Cryptography" in the March 2010 issue of the *AMS Notices*, and the article by W. Hugh Woodin on "The Continuum Hypothesis" in the June/July 2001 issue of the *AMS Notices*.

For infinite sets X of digital objects, $H(X) \equiv$ the size in bits of the smallest self-delimiting program p that runs forever outputting all the elements of X (and only the elements of X). So $H(X)$ is the size in bits of the smallest self-delimiting program p such that $U(p) = X$.

As we said before, most bit strings and binary sequences have maximal complexity and are called *algorithmically irreducible* or *algorithmically random*. Most N -bit strings x have

$$H(x) \approx N + H(N).$$

And if the bits of an infinite binary sequence X are chosen using independent tosses of a fair coin, then with probability one there exists a constant c such that

$$H(\text{the first } N \text{ bits of } X) > N - c$$

for all N . This implies Martin-Löf statistical randomness, namely that X is not contained in any *constructive measure zero set*.⁸ Amazingly enough, it turns out that Martin-Löf randomness is actually equivalent to algorithmic irreducibility, the fact that

$$H(\text{the first } N \text{ bits of } X) > N - c,$$

even though these concepts are defined completely differently. There is also a lesser-known but technically superior measure-theoretic definition of statistical randomness due to Robert Solovay, which also turns out to be equivalent to algorithmic irreducibility. For a proof of these equivalence theorems, see Chaitin, *Exploring Randomness*.

Furthermore, the fact that programs are self-delimiting enables us to define the halting probability Ω as a sum over all programs that halt when run on U :

$$0 < \Omega \equiv \sum_{p \text{ halts}} 2^{-|p|} < 1.$$

Each program p that halts contributes $1/2^{(p\text{'s size in bits})}$ to the halting probability Ω , since each bit of p is chosen using an independent toss of a fair coin. Note that the precise numerical value of $\Omega = \Omega_U$ depends on the choice of universal machine U , but Ω 's remarkable properties do not. If U were not self-delimiting, this sum would diverge. If U were not optimal, Ω would not be maximally complex and statistically random.

Since Ω is algorithmically irreducible, which we shall show later in this chapter, it is also Martin-Löf random and therefore Borel normal. In other words, written in any base b , the b different base- b digits of Ω all provably have equal limiting relative frequency $1/b$, and each block of k consecutive base- b digits has limiting relative frequency $1/b^k$.⁹ Ω is the only natural example of a real number that is provably Borel normal for every base b and

⁸A set of real numbers is a constructive measure zero set if it can be covered by an arbitrarily small covering and this can be done algorithmically. The infinite sequences X correspond to the reals $x \in [0, 1] \equiv \{x : 0 \leq x \leq 1\}$; simply place a decimal point in front of the bits of X to get the base-two representation for x .

⁹For a self-contained proof that Ω is Borel normal, see Chaitin, *Algorithmic Information Theory*.

for blocks of every size k , even though Borel proved that for a real number $x \in [0,1] \equiv \{x : 0 \leq x \leq 1\}$ this is true with probability one. It is conjectured that the real number $\pi = 3.1415926\dots$ also has this property, but π is not algorithmically irreducible. In fact,

(The first N digits of π) is a computable function f_π of N .

And so

$$H(\text{the first } N \text{ digits of } \pi) = H(f_\pi(N)) \leq H(f_\pi) + H(N) = c + H(N) \approx \log_2 N.$$

Why theories? Subadditivity and mutual information

It is crucial that because we are using self-delimiting programs, the information content of two objects (of computing the pair) is bounded by the sum of the individual information contents.

For x and y individual digital objects:

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

Here we can simply concatenate the two self-delimiting programs, and then run one and then the other. The constant c is the size in bits of the prefix π_C that makes U do this. In other words, there is a C such that

$$C(x^*y^*) = \text{the pair } (x, y).$$

Here x^* is a minimum-size program for x and y^* is a minimum-size program for y . Therefore

$$U(\pi_C x^* y^*) = \text{the pair } (x, y).$$

And

$$H(x, y) \leq |\pi_C x^* y^*| = |\pi_C| + H(x) + H(y) = H(x) + H(y) + c.$$

Similarly, for infinite sets of digital objects X and Y :

$$H(X \cup Y) \leq H(X) + H(Y) + c'.$$

Here we have to imagine both of these infinite computations proceeding in parallel, and interleave the bits of their respective programs in the order that they are required.

In particular, consider formal mathematical theories, that is, infinite sets of theorems, X and Y . Then $H(X)$ is the number of bits of axioms in the theory X , in other words, the size in bits of the proof-checking algorithm, more precisely, the size in bits of the smallest program for running through all possible proofs checking which ones are correct and finding all the theorems. $H(X)$ is the best possible way of doing this.

Consider a formal axiomatic theory X . Our crucial theorem on proving that programs are elegant can now be stated more precisely like this:

If X has the property that “ p is elegant” $\in X$ only if p is elegant, then “ p is elegant” $\in X$ only if $|p| \leq H(X) + c$.

In other words, to prove that an N -bit program is elegant you need at least N bits of axioms. Similarly:

If X has the property that “The k th bit of Ω is a 0” and “The k th bit of Ω is a 1” are in X only if they are true, then X can determine at most $H(X) + c'$ bits of Ω .

In other words, an N -bit theory can determine at most N bits of Ω .¹⁰

How much information do x and y or X and Y have in common?

Here is how we measure the *mutual information* for individual objects:

$$H(x : y) \equiv H(x) + H(y) - H(x, y),$$

and for infinite sets of objects:

$$H(X : Y) \equiv H(X) + H(Y) - H(X, Y).$$

Low mutual information is *algorithmic independence*:

$$H(x, y) \approx H(x) + H(y), \quad H(X, Y) \approx H(X) + H(Y).$$

The mutual information is the extent to which it is simpler to compute them together than to compute them separately, and in the case of independent objects there is no essential difference between computing them together and computing them separately.

For example, if you choose two N -bit strings x, y at random, with high probability all they have in common is their length:

$$H(x : y) \approx H(N) \approx \log_2 N.$$

This information-theoretic notion of independence is connected both with statistical and with logical independence.

Let me show you the connection with logical independence.

The reason that theories work is that things are not all independent; there is a lot of mutual information, there are a lot of common principles. In particular, let's consider mathematical theories:

The reason that theories work is that there are common principles, that there is a lot of mutual information between the theorems. Otherwise we would just have to add each theorem as an independent axiom!

¹⁰We shall not prove this here, but you can see this by combining ideas from our proof that only finitely many programs are provably elegant with the fact that Ω is algorithmically irreducible, which we shall prove later in this chapter. For a complete proof that is worked out in detail using LISP, see Chaitin, *The Limits of Mathematics*.

So our information-theoretic, complexity viewpoint suggests that physics and mathematics are not that different. In both cases, theories work for roughly the same reason: they are compressions, they find common principles. And this suggests a quasi-empirical view of mathematics, and adding non-self-evident new principles for pragmatic reasons, because they help us to organize our mathematical experience, just like scientific theories help us to organize our physical experience.

Combining theories and making conjectures

Following the late Ray Solomonoff, one of the founders of algorithmic information theory, let's show how to combine alternative physical or mathematical theories in order to make predictions.

Say we have several, for example, two theories, which explain what we have already seen and predict different future events. E.g., say we have seen 0110 and we want to know the next bit, and we have two theories, two computer programs that output 0110 and that then continue with other bits. At the start of this chapter, we said the simplest theory is best, but what if we have several theories and want to take them all into account, what then?

So we have two p with $U(p) = 0110x$ and how much weight should we put on each prediction x ? According to Solomonoff, each p gets weight $2^{-|p|}$, $1/(\text{two raised to the size in bits of } p)$. So if one p is three bits long, and the other is five bits long, and both explain what we have already seen and predict different continuations, then these weights will be $1/8$ and $1/32$. So if the first predicts a 0 and the second theory predicts a 1, then to get probabilities from these weights we must normalize them:

$$\frac{1/8}{1/8 + 1/32} \quad \frac{1/32}{1/8 + 1/32}$$

are the respective weights for the 0 and 1 predictions.¹¹

Similarly in the case of multiple mathematical theories. Say we have two math theories p that both explain a desired set of theorems X and avoid an undesired set of theorems Y , and we want to guess if a new theorem z is true or not. I.e.,

$$X \subset U(p), U(p) \cap Y = \emptyset, z \in U(p) ?$$

Suppose that our first theory p is three bits long, our second theory is five bits long, and that the first includes z while the second doesn't. Then as before we get the respective weights

$$\frac{1/8}{1/8 + 1/32} \quad \frac{1/32}{1/8 + 1/32}$$

¹¹At first Solomonoff had some trouble making this method converge; he was not aware of the crucial idea of self-delimiting programs, which makes it easy to sum over all possible programs for something. That algorithmic information theory should be based on self-delimiting programs was independently realized in the 1970s by L. A. Levin and Chaitin.

in favor of theorem z and against theorem z .

These are *a priori* estimates based only on considering conceptual complexity, on the fact that we believe in theories because they are compressions. Ideally, we should sum over *all* theories that match what we already know to make our predictions, but this cannot be done in practice. We can only make rough estimates of this ideal prediction.

Examples of randomness in real mathematics

After Turing discovered the halting problem, there was a lot of work that revealed that the halting problem occurs in many fields of mathematics. This work was unfortunately never collected in a single place, it was never put together in a definitive treatise. Nevertheless we can use it to find the halting probability in many fields of pure mathematics.

In quantum physics you can predict probabilities, not individual events. In pure math, if you can show that something encodes the bits of the halting probability, then you immediately get very precise statistical information about it, even though individual cases are impossible to determine. And the halting problem and halting probability are ubiquitous in pure mathematics, because *universal computation is ubiquitous*. You can easily build a universal computer using combinatorial components from just about any field of discrete pure math.¹² And as we will show in the next two chapters, this can also be done with continuous mathematics, with classical questions in mathematical analysis and mathematical physics.

The fundamental question then becomes, how natural are these examples of uncomputability and even algorithmic irreducibility and randomness in many different fields of mathematics? Are these examples artificial, degenerate cases, or are they natural and symptomatic? With elegant programs, we saw a situation in which only finitely many truths of a certain kind are provable. When we study computable reals later in this chapter, we will see a situation in which the difficulties are pervasive. Most reals are uncomputable, with probability one. There are of course many computable reals, but they nevertheless have total probability zero.

Because of examples like these we believe that uncomputability, undecidability and incompleteness are pervasive, that these are not isolated phenomena. And we are doing our best to also convince you of this!

Now we're going to show in some detail that the algorithmic randomness and irreducibility of the bits of Ω is also found in real mathematics, namely in number theory and algebra. Let's start with number theory.

Universal diophantine equations

A diophantine equation is an equation involving only whole numbers.

¹²See Stephen Wolfram, *A New Kind of Science* and Chaim Goodman-Strauss "Can't Decide? Undecide!", *AMS Notices*, March 2010, pp. 343–356.

There is a single polynomial diophantine equation which is a universal Turing machine:

$$P_U(i, k, n, x_1, x_2, \dots, x_m) = 0$$

has a solution with non-negative integers x_1, \dots, x_m iff the universal Turing machine eventually halts when started with program i and inputs k and n . Actually, if the program i halts on inputs k and n , then P_U has *infinitely many solutions*, and if i fails to halt on inputs k and n , then P_U has no solutions.

Instead of an infinite number of solutions, we would prefer there to be a single solution if i halts on inputs k and n . This can be achieved by using an exponential diophantine equation instead of a polynomial diophantine equation.

An exponential diophantine equation is a diophantine equation in which we allow x^y as well as x^3 , i.e., variables can also appear in exponents, which is not the case in polynomial diophantine equations, where exponents must always be constants.

There is a single exponential diophantine equation which is a universal Turing machine:

$$E_U(i, k, n, x_1, x_2, \dots, x_m) = 0$$

has a solution with non-negative integers x_1, \dots, x_m iff the universal Turing machine eventually halts when started with program i and inputs k and n . Furthermore if the program i halts on inputs k and n , then E_U has *exactly one solution*, and if i fails to halt on inputs k and n , then E_U has no solutions.

Let's approximate Ω , more precisely, let's get better and better lower bounds on Ω . Consider this computable sum:

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

Ω_n is computable and tends to Ω in the limit from below:

$$\Omega_1 \leq \Omega_2 \leq \Omega_3 \dots \rightarrow \Omega.$$

This converges to Ω , but this happens immensely slowly, and there is no way to know how far out to go to get a given degree of accuracy.

To use our universal diophantine equations P_U and E_U , we consider a program i that computes the k th bit of Ω_n and then loops forever if this bit is a 0 and halts if this bit is a 1. Note that since $\Omega_n \rightarrow \Omega$, for all sufficiently large n the k th bit of Ω_n will be correct.¹³ Plug into our universal diophantine equations the program i . This turns P_U into P_Ω and E_U into E_Ω . More precisely, this choice of i makes

$$P_U(i, k, n, x_1, x_2, \dots, x_m) = 0$$

into

$$P_\Omega(k, n, x_1, x_2, \dots, x_m) = 0,$$

¹³This works because Ω must be irrational, but one can avoid this issue by stipulating that the base-two representation for Ω must have infinitely many 1s. In other words, if there are only finitely many 1s, take the last 1 in Ω , which comes before an infinite list of 0s, and replace it by a 0 followed by an infinite list of 1s.

and makes

$$E_U(i, k, n, x_1, x_2, \dots, x_m) = 0$$

into

$$E_\Omega(k, n, x_1, x_2, \dots, x_m) = 0.$$

Now let's ask:

Does a diophantine equation have finitely or infinitely many solutions?

In 1900, David Hilbert asked for a general method to determine whether or not a diophantine equation has a solution. In 1970 Yuri Matiyasevich showed that there is no general method, since this is equivalent to solving the halting problem. He did this by constructing the universal diophantine equation given above. In 1984 he and James Jones came up with a particularly elegant construction for a universal exponential diophantine equation. We can do even better: Instead of just getting the halting problem, we can get the halting probability by using an individual diophantine equation.

Here is how you get the bits of Ω from P_Ω . If we fix k , the number of n such that

$$P_\Omega(k, n, x_1, x_2, \dots, x_m) = 0$$

has a solution will be finite or infinite depending on whether the k th bit of Ω is, respectively, a 0 or a 1.

And here is how you get the bits of Ω from E_Ω . If we fix k , the number of solutions of

$$E_\Omega(k, n, x_1, x_2, \dots, x_m) = 0$$

will be finite or infinite depending on whether the k th bit of Ω is, respectively, a 0 or a 1.¹⁴

Does a diophantine equation have an even/odd number of solutions?

Following Ord and Kieu (2003), we can change P_Ω to another polynomial P'_Ω with the following property: If we fix k , the number of n such that

$$P'_\Omega(k, n, x_1, x_2, \dots, x_m) = 0$$

has a solution will always be finite and it will be even or odd depending on whether the k th bit of Ω is, respectively, a 0 or a 1. To see how to produce P'_Ω , which is more complicated than P_Ω , see Chaitin, *Meta Math!*.

¹⁴In Chaitin, *Algorithmic Information Theory*, 1987, this equation is worked out in detail using LISP and the method of Matijasevic and Jones, which is based on a beautiful lemma of Édouard Lucas that the coefficient of x^k in the expansion of $(1+x)^n$ is odd iff each bit in k bitwise implies the corresponding bit in n . The result is a 200-page exponential diophantine equation in 20,000 unknowns, parts of which are printed in *Algorithmic Information Theory*.

Similarly, we can change $E_\Omega = 0$ into another exponential diophantine equation $E'_\Omega = 0$ with the following property: If we fix k , the number of solutions of

$$E'_\Omega(k, n, x_1, x_2, \dots, x_m) = 0$$

will always be finite and it will be even or odd depending on whether the k th bit of Ω is, respectively, a 0 or a 1.

The word problem for semi-groups

Enough of number theory! Now for some algebra.

Given a finite set of generators (letters), e.g., a, b, c , and a finite set of relations between them, e.g.,

$$aab = ca, ab = ba, acb = c,$$

are two words equivalent? Questions of this kind were originally studied by the Norwegian mathematician Axel Thue.

Following Emil Post (1947), we can represent what is called the *instantaneous description* of a Turing machine by a word bounded on both sides by the special symbol λ , and giving the current contents of the Turing machine tape with a symbol denoting the current state q inserted just to the left of the current position of the read/write head. Then for each Turing machine there is a finite set of relations over a finite alphabet such that

$$\lambda q_{\text{initial}}xyz \dots \lambda = \lambda q_{\text{final}}\lambda$$

if and only if the Turing machine eventually halts when started with $xyz \dots$ on the tape and with the read/write head scanning x .

To get each of the bits of Ω we pick a specific Turing machine very carefully,¹⁵ we convert it into a word problem, and then we ask the following questions. For each k , are there finitely or infinitely many n such that

$$\lambda q_{\text{initial}}a^k b^n \lambda = \lambda q_{\text{final}}\lambda ?$$

The answer determines the k th bit of Ω . Alternatively, for a different choice of alphabet and relations, again for each k , are there an even or an odd number of n such that this equality holds? If the alphabet and finite set of relations are chosen properly, the answer again determines the k th bit of Ω .¹⁶

So Ω 's irreducible complexity can be found in many (most?) fields of pure mathematics. God plays dice everywhere in pure math!¹⁷

¹⁵As before, this Turing machine computes the k th bit of Ω_n and then loops forever if this bit is a 0 and halts if this bit is a 1.

¹⁶For an extremely clear explanation of the method used by Post in 1947, see the article by Martin Davis "What is a Computation?" in Cristian Calude, *Randomness and Complexity, from Leibniz to Chaitin*.

¹⁷For a work of fiction reacting to this, see Arturo Sangalli, *Pythagoras' Revenge*, reviewed in the May 2010 issue of the *AMS Notices*.

How real are the real numbers? Borel 1927, 1952 and Turing 1936 revisited

Now we'd like to consider a path leading to Ω that makes clearer how it works, why Ω has the properties that it does, and provides some historical context for appreciating Ω . We'll also consider a domain in which the solvable problems have probability zero and the unsolvable problems have probability one, which is even worse than what happens with elegant programs, for which only finitely many are provably elegant and infinitely many are unprovably elegant.

Turing, 1936: There are more uncomputable reals than computable reals

It is not usually remembered that Turing's famous 1936 paper "On Computable Numbers..." deals with computable and uncomputable infinite-precision real numbers, something which one never sees in a modern computer, where all reals are finite precision. A real is computable if there is an un-ending algorithm that will compute it digit by digit with arbitrarily high precision, and a real is uncomputable if this is impossible to do.

Turing immediately notes that the computable reals are only as numerous as the set of possible computer programs, which is only a countable set, while the uncomputable reals are just as numerous as the set of all reals. So:

$$\#\{\text{computable reals}\} = \aleph_0, \quad \#\{\text{uncomputable reals}\} = 2^{\aleph_0}.$$

Uncomputable reals have probability one, computable reals have probability zero

In fact, a simple measure-theoretic or probabilistic argument shows that if you pick a real x at random between 0 and 1, it is possible but infinitely unlikely to be computable.

In other words, consider a real $x \in [0, 1] \equiv \{x : 0 \leq x \leq 1\}$ with the uniform probability distribution. Then

$$\mathbf{Prob}\{\text{computable reals}\} = 0, \quad \mathbf{Prob}\{\text{uncomputable reals}\} = 1.$$

Here is a proof that the computable reals have measure zero. Cover the first computable real with an interval of size $\epsilon/2$, the second computable real with an interval of size $\epsilon/4$, etc. The total size of the covering is

$$\leq \frac{\epsilon}{2} + \frac{\epsilon}{4} + \frac{\epsilon}{8} + \frac{\epsilon}{16} + \frac{\epsilon}{32} + \dots = \epsilon,$$

which we can make as small as desired. Hence the computable reals are a set of measure zero and have zero probability, and the uncomputable reals have probability one.

This proof uses ideas that were pioneered by Borel. But Borel never states this result. Instead he proves a much stronger, a much stranger result:

Borel 1952: Un-nameable reals have probability one, nameable reals have probability zero

In his last book, published when he was in his 80s, Émile Borel points out that most real numbers cannot even be individually named, constructively or non-constructively:

$$\mathbf{Prob}\{\text{nameable reals}\} = 0, \quad \mathbf{Prob}\{\text{un-nameable reals}\} = 1.$$

This follows immediately just as before, since the set of all possible names for reals is a subset of the set of all possible strings over a finite alphabet, and is therefore countable, just like the set of all possible computer programs and the set of all possible algorithms for computing a computable real.

Borel's 1927 oracle number: Nth bit answers the Nth yes/no question

Borel felt more comfortable with countable sets than with uncountable sets. He also preferred computable functions to uncomputable functions. In 1927 he gave a wonderful example of how unreal a real number could be: his amazing know-it-all oracle number written in base-two whose N th bit after the decimal point answers the N th yes/no question. Here we take advantage of the fact that the set of all possible yes/no questions is included in the set of all possible strings over a finite alphabet, and is therefore countable.

Unfortunately, Borel's oracle real is hard to define rigorously. For example, consider the following question:

Is the answer to this question "no"?

There is no valid answer.

Or we can ask about our future behavior and then do the opposite.

However, just number all the possible computer programs. Then it is easy to make a rigorous version of the Borel oracle number that answers all instances of Turing's halting problem:

"Borel-Turing" oracle number: Nth bit tells us if the Nth program halts

Unfortunately this Borel-Turing oracle number — by the way, Borel and Turing seemed to be totally unaware of each other's work — is extremely redundant; it repeats a lot of information.

Why?

Because N instances of the halting problem is only equal to $\log_2 N$ bits of mathematical information, not to N bits of mathematical information. The Borel-Turing number is highly redundant.

Here is a proof of this. Consider N bits of the Borel-Turing number, i.e., N cases of the halting problem. You can tell which of these N programs halt if you know *how many of them* halt. Just run the N programs in parallel until that many have halted. The remaining programs will never halt.

By using a slightly more sophisticated version of this idea, we finally arrive at the halting probability

$$\Omega \equiv \sum_{U(p) \text{ halts}} 2^{-|p|},$$

which contains no redundancy and is the best possible way to pack the answers to every possible case of the halting problem in a single oracle real.

Here is how this works. It works because:

First N bits of Ω tell us which $\leq N$ bit programs halt

Why is this? Well, do you remember those lower bounds Ω_k on Ω that we considered earlier in this chapter:

$$\Omega_k = \sum_{|p| \leq k \text{ \& } U(p) \text{ halts in time } \leq k} 2^{-|p|} ?$$

Ω_k is computable and tends to Ω in the limit from below:

$$\Omega_1 \leq \Omega_2 \leq \Omega_3 \dots \rightarrow \Omega.$$

So if you are given a minimum-size program Ω^* that calculates the first N bits of Ω , we can concatenate in front of Ω^* a prefix π_Ω that does the following: First π_Ω runs Ω^* to calculate the first N bits of Ω . Then π_Ω calculates better and better lower bounds on Ω , Ω_k , for $k = 1, 2, 3, \dots$ until the first N bits of Ω_k are correct, i.e., the same as the first N bits of Ω .

This is possible because Ω must be irrational, but just in case we happen not to know this, just do all of this with a base-two representation for Ω in which there are infinitely many 1s. In other words, if there are only finitely many 1s, take the last 1, which comes before an infinite list of 0s, and replace it by a 0 followed by an infinite list of 1s.

Once we have found a value of k for which the first N bits of Ω_k are correct, we know every program that is $\leq N$ bits in size that ever halts (each of them halts in time $\leq k$), because if another $\leq N$ bit program were then to halt, the value of Ω_k would become greater than Ω , which contradicts the fact that Ω_k is always $\leq \Omega$.

Once we have found all $\leq N$ bit programs that halt, we can run them all to see what they produce, and then pick as our very own output the first positive integer M that does not have $H(M) \leq N$. So finally π_Ω produces as its output the first positive integer M with $H(M) > N$.

In summary,

$$U(\pi_\Omega \Omega^*) = M$$

and

$$H(M) > N,$$

and so

$$|\pi_\Omega \Omega^*| = c + H(\text{the first } N \text{ bits of } \Omega) > N$$

and

$$H(\text{the first } N \text{ bits of } \Omega) > N - c,$$

which was to be proved.

$H(\text{First } N \text{ bits of } \Omega) > N - c$, and Ω is irreducible

This shows that unlike the Borel-Turing oracle number, Ω is an oracle for the halting problem that is incompressible. From this fact, using the ideas in our proof that it takes an N -bit theory to prove that an N -bit program is elegant, it is not difficult to show that it takes an N -bit theory to determine N bits of Ω , which is the *logical irreducibility* of Ω , and has a much greater epistemological impact than the mere fact that Ω is algorithmically irreducible and algorithmically random.

On the other hand, it is the fact that Ω is algorithmically irreducible that makes it look random, accidental, undistinguished, typical, even though it is jam-packed with useful information about the halting problem. For if Ω had any redundancy, then it would not be the best possible oracle for the halting problem. Any time we eliminate all the redundancy from something, we get randomness! And that's why pure math is full of randomness.

And most real numbers are rather unreal. Ω is a violently unreal real number that we have tried to make as real as possible. In other words, Ω is a specific real that lies just on the border between the computable and the violently uncomputable. As the series of approximations Ω_k show, Ω is almost real, but not quite. And that is why it is so very interesting. Reals are un-nameable with probability one, but we can never exhibit a specific example, because then we would actually be naming such a real. But in the case of Ω , there it is, a specific, fairly natural example of algorithmic irreducibility and randomness.

Mathematics, biology and metabiology

We've discussed physical and mathematical theories in this chapter; now let's turn to biology, the most exciting field of science at this time, but one where mathematics is not very helpful. Biology is very different from physics. There is no simple equation for your spouse. Biology is the domain of the complex. There are not many universal rules. There are always exceptions. Math is very important in theoretical physics, but there is no fundamental mathematical theoretical biology.

This is unacceptable. The honor of mathematics requires us to come up with a mathematical theory of evolution and either prove that Darwin was wrong or right! We want a general, abstract theory of evolution, not an immensely complicated theory of actual biological evolution. And we want proofs, not computer simulations! So we've got to keep our model very, very simple.

That's why this proposed new field is *metabiology*, not biology.

What kind of math can we use to build such a theory? Well, it's certainly not going to be differential equations. Don't expect to find the secret of life in a

differential equation; that's the wrong kind of mathematics for a fundamental theory of biology.

In fact a universal Turing machine has much more to do with biology than a differential equation does. A universal Turing machine is a very complicated new kind of object compared to what came previously, compared with the simple, elegant ideas in classical mathematics like analysis. And there are self-reproducing computer programs, which is an encouraging sign.

There are in fact three areas in our current mathematics that do have some fundamental connection with biology, that show promise for math to continue moving in a biological direction:

Computation, Information, Complexity.

DNA is essentially a programming language that computes the organism and its functioning; hence the relevance of the theory of computation for biology.

Furthermore, DNA contains biological information. Hence the relevance of information theory. There are in fact at least four different theories of information:

- Boltzmann statistical mechanics and Boltzmann entropy,
- Shannon communication theory and coding theory,
- algorithmic information theory (Solomonoff, Kolmogorov, Chaitin), which is the subject of this chapter, and
- quantum information theory and qubits.

Of the four, AIT (algorithmic information theory) is closest in spirit to biology. AIT studies the size in bits of the smallest program to compute something. And the complexity of a living organism can be roughly (very roughly) measured by the number of bases in its DNA, in the biological computer program for calculating it.

Finally, let's talk about complexity. Complexity is in fact the most distinguishing feature of biological as opposed to physical science and mathematics. There are many computational definitions of complexity, usually concerned with computation times, but again AIT, which concentrates on program size or conceptual complexity, is closest in spirit to biology.

Let's emphasize what we are not interested in doing. We are certainly not trying to do systems biology: large, complex realistic simulations of biological systems. And we are not interested in anything that is at all like Fisher-Wright population genetics that uses differential equations to study the shift of gene frequencies in response to selective pressures.

We want to use a sufficiently rich mathematical space to model the space of all possible designs for biological organisms, to model biological creativity. And the only space that is sufficiently rich to do that is a software space, the space of all possible algorithms in a fixed programming language. Otherwise we have limited ourselves to a fixed set of possible genes as in population

genetics, and it is hopeless to expect to model the major transitions in biological evolution such as from single-celled to multicellular organisms, which is a bit like taking a main program and making it into a subroutine that is called many times.

Recall the cover of Stephen Gould's *Wonderful Life* on the Burgess shale and the Cambrian explosion? Around 250 primitive organisms with wildly differing body plans, looking very much like the combinatorial exploration of a software space. Note that there are no intermediate forms; small changes in software produce vast changes in output.

So to simplify matters and concentrate on the essentials, let's throw away the organism and just keep the DNA. Here is our proposal:

Metabiology: a field parallel to biology that studies the random evolution of artificial software (computer programs) rather than natural software (DNA), and that is sufficiently simple to permit rigorous proofs or at least heuristic arguments as convincing as those that are employed in theoretical physics.

This analogy may seem a bit far-fetched. But recall that Darwin himself was inspired by the analogy between artificial selection by plant and animal breeders and natural selection imposed by malthusian limitations.

Furthermore, there are many tantalizing analogies between DNA and large, old pieces of software. Remember *bricolage*, that Nature is a cobbler, a tinkerer? In fact, a human being is just a very large piece of software, one that is 3×10^9 bases = 6×10^9 bits \approx one gigabyte of software that has been patched and modified for more than a billion years: a tremendous mess, in fact, with bits and pieces of fish and amphibian design mixed in with that for a mammal.¹⁸ For example, at one point in gestation the human embryo has gills. As time goes by, large human software projects also turn into a tremendous mess with many old bits and pieces.

The key point is that you can't start over, you've got to make do with what you have as best you can. If we could design a human being from scratch we could do a much better job. But we can't start over. Evolution only makes small changes, incremental patches, to adapt the existing code to new environments.

So how do we model this? Well, the key ideas are:

Evolution of mutating software,

and:

Random walks in software space.

That's the general idea. And here are the specifics of our current model, which is quite tentative.

We take an organism, a single organism, and perform random mutations on it until we get a fitter organism. That replaces the original organism, and

¹⁸See Neil Shubin, *Your Inner Fish: A Journey into the 3.5-Billion-Year History of the Human Body*.

then we continue as before. The result is a random walk in software space with increasing fitness, a hill-climbing algorithm in fact.¹⁹

Finally, a key element in our proposed model is the definition of fitness. For evolution to work, it is important to keep our organisms from stagnating. It is important to give them something challenging to do.

The simplest possible challenge to force our organisms to evolve is what is called the Busy Beaver problem, which is the problem of providing concise names for extremely large integers. Each of our organisms produces a single positive integer. The larger the integer, the fitter the organism.²⁰

The Busy Beaver function of N , $BB(N)$, that is used in AIT is defined to be the largest positive integer that is produced by a program that is less than or equal to N bits in size. $BB(N)$ grows faster than any computable function of N and is closely related to Turing's famous halting problem, because if $BB(N)$ were computable, the halting problem would be solvable.²¹

Doing well on the Busy Beaver problem can utilize an unlimited amount of mathematical creativity. For example, we can start with addition, then invent multiplication, then exponentiation, then hyper-exponentials, and use this to concisely name large integers:

$$N + N \rightarrow N \times N \rightarrow N^N \rightarrow N^{N^N} \rightarrow \dots$$

There are many possible choices for such an evolving software model: You can vary the computer programming language and therefore the software space, you can change the mutation model, and eventually you could also change the fitness measure. For a particular choice of language and probability distribution of mutations, and keeping the current fitness function, it is possible to show that in time of the order of 2^N the fitness will grow as $BB(N)$, which grows faster than any computable function of N and shows that genuine creativity is taking place, for mechanically changing the organism can only yield fitness that grows as a computable function.²²

So with random mutations and just a single organism we actually do get evolution, unbounded evolution, which was precisely the goal of metabiology!

This theorem may seem encouraging, but it actually has a serious problem. The times involved are so large that our search process is essentially *ergodic*,

¹⁹In order to avoid getting stuck on a local maximum, in order to keep evolution from stopping, we stipulate that there is a non-zero probability to go from any organism to any other organism, and $-\log_2$ of the probability of mutating from A to B defines an important concept, the *mutation distance*, which is measured in bits.

²⁰*Alternative formulations:* The organism calculates a total function $f(n)$ of a single non-negative integer n and $f(n)$ is fitter than $g(n)$ if $f(n)/g(n) \rightarrow \infty$ as $n \rightarrow \infty$. Or the organism calculates a (constructive) Cantor ordinal number and the larger the ordinal, the fitter the organism.

²¹Consider $BB'(N)$ defined to be the maximum run-time of any program that halts that is less than or equal to N bits in size.

²²Note that to actually simulate our model an oracle for the halting problem would have to be employed to avoid organisms that have no fitness because they never calculate a positive integer. This also explains how the fitness can grow faster than any computable function. In our evolution model, implicit use is being made of an oracle for the halting problem, which answers questions whose answers cannot be computed by any algorithmic process.

which means that we are doing an exhaustive search. Real evolution is not at all ergodic, since the space of all possible designs is much too immense for exhaustive search.

It turns out that with this same model there is actually a much quicker *ideal evolutionary pathway* that achieves fitness $BB(N)$ in time of the order of N . This path is however unstable under random mutations, plus it is much too good: Each organism adds only a single bit to the preceding organism, and immediately achieves near optimal fitness for an organism of its size, which doesn't seem to at all reflect the haphazard, frozen-accident nature of what actually happens in biological evolution.²³

So that is the current state of metabiology: a field with some promise, but not much actual content at the present time. The particular details of our current model are not too important. Some kind of mutating software model should work, should exhibit some kind of basic biological features. The challenge is to identify such a model, to characterize its behavior statistically,²⁴ and to *prove* that it does what is required.

Post Scriptum

After this chapter was completed the mathematical structure of metabiology fell into place. The crucial step was to permit *algorithmic* mutations: If a mutation M is a K -bit program that takes the original organism A as input and produces the mutated organism $A' = M(A)$ as output, then this mutation M has probability 2^{-K} . Then it becomes possible to show that evolution rapidly takes place in the Busy Beaver model of evolution discussed above. In fact, better and better lower bounds on the halting probability Ω will rapidly evolve.

For the mathematical details, see G. J. Chaitin, "Life as evolving software," to be published in H. Zenil, *A Computable Universe*, World Scientific, 2012. For a non-technical book-length treatment, see G. Chaitin, *Proving Darwin: Making Biology Mathematical* to be published by Pantheon in 2012. For an overview of this book, a lecture entitled "Life as evolving software," go to www.youtube.com and search for *chaitin ufryg*.

²³The N th organism in this ideal evolutionary pathway is essentially just the first N bits of the numerical value of the halting probability Ω . Can you figure out how to compute $BB(N)$ from this?

²⁴For instance, will some kind of hierarchical structure emerge? Large human software projects are always written that way.

References

- [1] É. Borel, *Leçons sur la théorie des fonctions*, Gauthier-Villars (1950), Gabay (2003), pp. 275.
- [2] É. Borel, *Les Nombres inaccessibles*, Gauthier-Villars (1952), p. 21.
- [3] É. Borel, *Space and Time*, Dover (1960), pp. 212–214.
- [4] C. Calude, *Randomness and Complexity, from Leibniz to Chaitin*, World Scientific (2007).
- [5] G. J. Chaitin, *Algorithmic Information Theory*, Cambridge University Press (1987).
- [6] G. J. Chaitin, *The Limits of Mathematics*, Springer (1998).
- [7] G. J. Chaitin, *Exploring Randomness*, Springer (2001).
- [8] G. J. Chaitin, *Meta Math!*, Pantheon (2005).
- [9] G. J. Chaitin, *Mathematics, Complexity and Philosophy*, Midas, in press (2010).
- [10] J. Fresán, “Review of *Pythagoras’ Revenge*,” *AMS Notices* **57**, 637–638 (2010).
- [11] C. Goodman-Strauss, “Can’t decide? Undecide!,” *AMS Notices* **57**, 343–356 (2010).
- [12] S. Gould, *Wonderful Life*, Norton (1989).
- [13] N. Koblitz and A. Menezes, “The brave new world of bodacious assumptions in cryptography,” *AMS Notices* **57**, 357–365 (2010).
- [14] G. W. Leibniz, *Discours de métaphysique, suivi de Monadologie*, Gallimard (1995).
- [15] A. Sangalli, *Pythagoras’ Revenge*, Princeton University Press (2009).
- [16] N. Shubin, *Your Inner Fish*, Pantheon (2008).
- [17] T. Tymoczko, *New Directions in the Philosophy of Mathematics*, Princeton University Press (1998).
- [18] H. Weyl, *The Open World*, Yale University Press (1932).
- [19] S. Wolfram, *A New Kind of Science*, Wolfram Media (2002).
- [20] W. H. Woodin, “The continuum hypothesis, Part I,” *AMS Notices* **48**, 567–576 (2001).