CDD: 401

HOW REAL ARE REAL NUMBERS?[1,2]

GREGORY CHAITIN

IBM Watson Research Center
Yorktown Heights
USA
Académie Internationale de Philosophie des Sciences
BELGIQUE

gjchaitin@gmail.com

Abstract: We discuss mathematical and physical arguments against continuity and in favor of discreteness, with particular emphasis on the ideas of Émile Borel (1871–1956).

Keywords: Omega number. Halting probability. Halting problem. Émile Borel. Turing oracles. Irreducible complexity.

I will show you some really weird real numbers.

I'm not trying to undermine what you may have learned in your mathematics classes. I love the real numbers. I have nothing against real numbers. There's even a real number — Ω — that has my name on it.[3] But as you will see, there are some strange things going on with the real numbers.

Let's start by going back to a famous paper by Turing in 1936. This is Turing's famous 1936 paper in the *Proceedings of the London Mathematical Society*; mathematicians proudly claim it creates the computer industry, which is not quite right of course.

[1]Dedicated to Newton da Costa on his 80th birthday.

[2]Adapted from the second chapter of Chaitin's book *Mathematics, Complexity and Philosophy* to be published in a bilingual Spanish/English edition by Midas in Chile.

[3]See the chapter on "Chaitin's Constant" in Finch (2003).

But it does have the idea of a general-purpose computer and of hardware and software, and it is a wonderful paper.

This paper is called "On computable numbers, with an application to the *Entscheidungsproblem*." And what most people forget, and is the subject of my talk today, is that when Turing talks about *computable* numbers, he's talking about computable **real** numbers.

Turing, 1936: *"On computable numbers. . . "*

But when you work on a computer, the last thing on earth you're ever going to see is a real number, because a real number has an infinite number of digits of precision, and computers only have finite precision. Computers don't quite measure up to the exalted standards of pure mathematics.

One of the important contributions of Turing's paper, not to computer technology but to pure mathematics, and even to philosophy and epistemology, is that Turing's paper distinguishes very clearly between real numbers that are computable and real numbers that are **uncomputable**.

What is a real number? It's just a measurement made with infinite precision. So if I have a straight line one unit long, and I want to find out where a point is, that corresponds to a real number. If it is all the way to the left in this unit interval, it's 0.0000... If the point is all the way to the right, it's 1.0000... If it is exactly in the middle, that's .50000... And every point on this line corresponds to precisely one real number. There are no gaps.

0.0 —— 0.5 —— 1.0

So, if you just tell me **exactly** where a point is, that's a real number. From the point of view of geometrical intuition a real number is something very simple: it is just a point on a line. But from an arithmetical point of view, if you want to calculate its numerical value digit by digit

or bit by bit if you're using binary, it turns out that real numbers are **problematical**.

Even though to geometrical intuition points are the most natural and elementary thing you can imagine, if you want to actually calculate the value of a real number with infinite precision, you can get into big trouble. Actually, you never calculate it with infinite precision. What Turing says is that you calculate it with **arbitrary** precision.

His notion of a computable real number is a real number that you can calculate as accurately as you may wish.

I guess he actually says it is an infinite calculation. You start calculating its numerical value, if you're using decimal, digit by digit, or if you're using binary, bit by bit. You have the integer part, the decimal point, and then you have an infinite string of bits or digits, depending on your base, and the computer will grind away gradually giving you more and more bits or more and more digits of the numerical value of the number.

So that's a computable real number. According to Turing, that means it is a real number for which there is an algorithm, a mechanical procedure, for calculating its value with arbitrary accuracy, with more and more precision.

For example, π is a computable real number, $\sqrt{2}$ is a computable real number, and e is a computable real number. In fact, every real number you've ever encountered in your math classes, every individual real number that you've ever encountered, is a computable real number.

$$\text{Computable reals: } \pi,\ \sqrt{2},\ e,\ 1/2,\ 3/4\ldots$$

These are the familiar real numbers, the computable ones, but surprisingly enough, Turing points out that there are also lots of **uncomputable** real numbers.

Dramatically enough, the moment Turing comes up with the computer as a mathematical concept — mathematicians call this *a universal Turing machine* — he immediately points out that there are things

no computer can do. And one thing no computer can do is calculate the value of an uncomputable real number.

How does Turing show that there are uncomputable reals? Well, the first argument he gives goes back to Cantor's theory of infinite sets, which tells us that the set of real numbers is an infinite set that is bigger, that is infinitely more numerous, than the set of computer programs.

The possible computer programs are just as numerous as the positive integers, as the whole numbers 1, 2, 3, 4, 5... but the set of real numbers is a much bigger infinity.

So in fact there are more uncomputable reals than computable reals. From Cantor's theory of infinite sets, we see that the set of uncomputable reals is just as big as the set of all reals, while the set of computable reals is only as big as the set of whole numbers. The set of uncomputable reals is much bigger than the set of computable reals.

$$\#\{\text{uncomputable reals}\} = \#\{\text{all reals}\} = \aleph_1,$$
$$\#\{\text{computable reals}\} = \#\{\text{computer programs}\} =$$
$$\#\{\text{whole numbers}\} = \aleph_0,$$
$$\aleph_1 > \aleph_0.$$

The set of computable reals is as numerous as the computer programs, because each computable real needs a computer program to calculate it. And the computer programs are as numerous as the whole numbers 1, 2, 3, 4, 5... because you can think of a computer program as a very big whole number. In base-two a whole number is just a string of bits, which is all a computer program is.

That most reals are uncomputable was quite a surprise, but Turing doesn't stop with that. In his famous paper he uses a technique from set theory called *Cantor's diagonal argument* to exhibit an individual example of an uncomputable real.

Turing's 1936 paper has an intellectual impact and a technological impact.

From a technological point of view his paper is fantastic because as we all know the computer has changed our lives; we can't imagine living without it. In 1936 Turing has the idea of a flexible machine, of flexible hardware, of what we now call software. A universal machine changes to be like any other machine when you insert the right software. This is a very deep concept: a flexible digital machine. You don't need lots of special-purpose machines, you can make do with only one machine. This is the idea of a general-purpose computer, and it is in Turing's wonderful 1936 paper, before anybody built such a device.

But then immediately Turing points out that there are real numbers that can't be calculated, that no machine can furnish you with better and better approximations for; in fact there are more uncomputable reals than computable reals.

So in a sense this means that most individual real numbers are like mythical beasts, like Pegasus or unicorns — name your favorite mythical object that unfortunately doesn't exist in the real world.

In this talk I will concentrate on the uncomputable reals, not the reals that are familiar to all of us like $\pi, \sqrt{2}$ and $3/4$. I'll tell you about some surprising real numbers, ones that are in the shadow, that we cannot compute, and whose numerical values are quite elusive.

And the first thing I'd like to say on this topic is that there actually was an example of an uncomputable real before Turing's 1936 paper. It was in a short essay by a wonderful French mathematician who is now largely forgotten called Émile Borel. Émile Borel in 1927, without anticipating in any way Turing's wonderful paper, does point out a real number that we can now recognize as an uncomputable real.

Borel's 1927 number is a very paradoxical real number, and I'd like to tell you about it. Let's see if you enjoy it as much as I do!

Borel's idea is to have a know-it-all real number: It's an oracle that knows the answer to every yes/no question.

Borel was a Frenchman, and he imagined writing all possible yes/no questions in French in a numbered list. So each question has its number, and then what you do is consider the real number with no integer part, and whose Nth decimal, the Nth digit after the decimal point of this number, answers the Nth question in the list of all possible questions.

Borel's 1927 know-it-all real: *The Nth digit answers the Nth question.*

If the answer to the Nth question is "yes," then the Nth digit is, say, a 1, and if the answer is "no" then that digit will be a 2. You have an infinite number of digits, so you can pack an infinite number of answers in Borel's number. If you can imagine a list of all possible yes/no questions, this number will be like an oracle that will answer them all.

If you could know the value of this magical number with sufficient accuracy, you could answer any particular, individual question.

Needless to say, this number is a bit fantastic. So why did Borel come up with this crazy example? To show us that there are real numbers that are not for real.

Borel's amazing number will give you the answer to every yes/no question in mathematics and in physics, and about the past and the future.

You can ask Borel's oracle paradoxical questions, like

"Is the answer to this question *no*?"

And then you have a problem, because whether you answer "no" or you answer "yes," it will always be the wrong answer. There is no correct answer.

Another problem is if you ask

"Will I drink coffee tomorrow?"

And depending on what you find in Borel's number, you do the opposite. You will just have to put up with not drinking coffee for one day, perhaps, to refute the know-it-all number.

So these are some paradoxical aspects of Borel's idea. Another problem is, how do you make a list of all possible yes/no questions in order to give a number to each question?

Actually, it is easy to number each question. What you do is make a list of all possible texts drawn from the alphabet of the language you're interested in, and you have ten possibilities for each digit in Borel's oracle number, and so far we've only used the digits 1 and 2. So you can use the other digits to say that that sequence of characters from that particular national alphabet isn't grammatical, or it's not a question, or it's a question but it's not a yes/no question, or it's a yes/no question which has no answer because it's paradoxical, and maybe it's a yes/no question which has an answer but I don't want to tell you about the future because I want to avoid the coffee problem.

Another way to try to fix Borel's number is to restrict it to just give the answer to mathematical questions. You can imagine an infinite list of mathematical questions, you pick a formal language in which you ask yes/no mathematical questions, you pick a notation for asking such questions, and there is certainly no problem with having Borel's know-it-all number answer only mathematical questions. That will get rid of the paradoxes. You can make that work **simply because you can pack an infinite amount of information in a real number**.[4]

And this magical number could be represented, in principle, by two metal rods, if you believe in infinite precision lengths. You have a rod that is exactly one unit long, one meter long, and you have another rod that is smaller, whose length is precisely the know-it-all number. You

[4] As long as we avoid self-reference, i.e., giving this know-it-all number a name and then having a digit ask about itself in a way that requires that very digit to be different. E.g., is the digit of Borel's know-it-all number that corresponds to this very question a 2?

have your standard meter, and you have something less than a meter long whose length is precisely the magical know-it-all number.

If you could measure the size of the smaller rod relative to the standard meter with arbitrary accuracy, you could answer every mathematical question, if somebody gave you this magical metal rod.[5]

Of course, we are assuming that you can make measurements with infinite precision, which any physicist who is here will say is impossible. I think that the most accurate measurement that has ever been made has under twenty digits of precision.

But in theory having these two rods would give us an oracle. It would be like having Borel's know-it-all number.

And now you're not going to be surprised to hear that if you fix Borel's number so that it is at least well-defined, not paradoxical, it is in fact uncomputable. For otherwise you could compute the answer to every question, which is implausible.

But Borel did not really have the notion of computability nor of what a computer is. He was working with the idea of computation intuitively, informally, without defining it. He said that as far as he was concerned this number is conceivable but not really legitimate. He felt that his real number was too wild.

Borel had what is now called a *constructive* attitude. His personal view which he states in that little 1927 essay with the oracle number, is that he believes in a real number if in principle it can be calculated, if in theory there's some way to do that. And he talks about this

[5]This is an argument against infinite divisibility of space. In 1932 Hermann Weyl gave an argument against the infinite divisibility of time. If time is **really** infinitely divisible, then a machine could perform one step of a calculation in one second, then another step in half a second, the next in 1/4 of a second, then in 1/8 of a second, then 1/16 of a second, and in this manner would perform an infinite number of steps in precisely $1 + \frac{1}{2} + \frac{1}{4} + \ldots = 2$ seconds. But no one believes that such so-called *Zeno machines* are actually possible.

counter-example, this number which is conceivable but in his opinion is not really legitimate.

If we remove Borel's number, that will leave a hole in the line, in the unit interval $[0,1] = \{0 \leq x \leq 1\}$. So we've broken the unit interval into two pieces because we just eliminated one point, which is very unfortunate geometrically. But let's go on.

Okay, so this was 1927, this was Émile Borel with his paradoxical know-it-all real number that answers every yes/no question, and it is a mathematical fantasy, not a reality. Now let's go back to Turing.

In 1936 Turing points out that there are more uncomputable reals than computable ones. Now I'd like to tell you something that Turing doesn't point out, which uses ideas from Émile Borel, who was one of the inventors of what's called *measure theory* or probability theory.

It turns out that if you choose a real number x between zero and one, $x \in [0,1]$, and you have uniform probability of picking any real number between zero and one, then the probability is unity that the number x will be uncomputable. The probability is zero that the number will be computable.

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

That's not too difficult to see. Now I'll give you a mathematical proof of this.

You want to **cover** all the computable reals in the unit interval with a covering which can be arbitrarily small. That's the way to prove that the computable reals *have measure zero*, which means that they are an infinitesimal part of the unit interval, that they have zero probability. Technically, you say they have measure zero.

Remember that every computable real corresponds to a program, the program to calculate it, and the programs are essentially positive integers, so there's a first program, a second program, a third program... So you can imagine all the computable reals in a list. There will be a first computable real, a second, a third...

And I cover the first computable real with an interval. I put on top of it an interval of length $\epsilon/2$. And then I cover the second computable real with an interval of length $\epsilon/4$. I cover the third computable real with an interval of length $\epsilon/8\ldots$ then $\epsilon/16$, then $\epsilon/32\ldots$

So the total size of the covering is

$$\frac{\epsilon}{2}+\frac{\epsilon}{4}+\frac{\epsilon}{8}+\frac{\epsilon}{16}+\frac{\epsilon}{32}+\ldots \;=\; \epsilon.$$

Some of these intervals may overlap; it doesn't really matter. What does matter is that the total length of all these intervals is exactly ϵ, and you can make ϵ as small as you want.

So this is just a proof that something is of measure zero. I'm taking the trouble to show that you can corner all the computable real numbers in the unit interval by covering them. You can do it with a covering that you can make as small as you please, which means that the computable reals have zero probability, they occupy zero real-estate in the unit interval.

This is a proof that the computable reals really are exceptional. But they're the exception of our normal, everyday experience. The fact that uncomputable reals have probability unity doesn't help us to find any concrete examples!

To repeat, if I pick a real number at random between zero and one, it is possible to get a computable real, but it is infinitely unlikely. Probability zero in this circumstance doesn't mean impossibility, it just means that it's an infinitesimal probability, it is infinitely unlikely. It is possible. It would be miraculous, but it can happen.

The way mathematicians say this, is that real numbers are *almost surely* uncomputable.

This is a bit discouraging to those of us who prefer computable reals to uncomputable ones. Or maybe it is a bit surprising that all the individual reals in our normal experience are exceptional. It does

make me think that perhaps real numbers are problematic and cannot be taken for granted. What do **you** think?

What's another way to put it? In other words, the real numbers are like a Swiss cheese with a lot of holes. In fact, it's all holes! It's like looking at the night sky. There are stars in the night sky, those are the computable reals, but the background is always black — the stars are the exception.

So that is how the real numbers look!

All the reals we know and love are exceptional.

And Borel goes a little farther in his last book, written when he was in his eighties. In 1952 he published a book called *Les nombres inaccessibles* — Inaccessible Numbers — in which he points out that most real numbers can't even be referred to or named individually in any way. The real numbers that you can somehow name, or pick out as individuals, even without being able to compute them, have to be the exception, with total probability zero.

Most real numbers cannot even be named as individuals in any way, constructive or non-constructive. The way somebody put it is, most reals are wall-flowers, they'll never be invited to dance!

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

Okay, so what I'd like to do in the rest of this talk is to take Borel's crazy, know-it-all, oracle real number, and try to make it as realistic as possible.

We've gotten ourselves into a bit of a quandary. I tend to believe in something if I can calculate it; if so, that mathematical object has concrete meaning for me. So I have a sort of constructive attitude in math.

But there is this surprising fact that in some sense most mathematical facts or objects seem to be beyond our reach. Most real numbers

can never be calculated, they're uncomputable, which suggests that mathematics is full of things that we can't know, that we can't calculate.

This is related to something famous called *Gödel's incompleteness theorem* from 1931, five years before Turing. Gödel's 1931 theorem says that given any finite set of axioms, there will be true mathematical statements that escape, that are true but can't be proven from those axioms. So mathematics resists axiomatization. There is no Theory of Everything for pure mathematics.

Gödel, 1931: *No TOE for pure mathematics!*

And what Turing shows in 1936 is that there are a lot of things in mathematics that you can never calculate, that are beyond our reach because there's no way to calculate the answer. And in fact real numbers are an example: most real numbers, with probability one, cannot be calculated.

So it might be nice to try to come up with an example of a particular real number that can't be calculated, and try to make these strange, mysterious, hidden real numbers as concrete as possible.

I'd like to show you a real number — Ω — which is as real as I can make it, but is nevertheless uncomputable. That's my goal.

In other words, there is what you can calculate or what you can prove in mathematics, and then there is this **vast cloud of unknowing**, of things that you can't know or calculate. And I would like to try to find something right at the border between the knowable and the unknowable. I'm going to make it as real as possible, but it's going to be just beyond our reach, just beyond what we can calculate.

I want to show you a real number that can **almost** be calculated, which is as close as possible to seeming real, concrete, but in fact escapes us and is an example of this ubiquitous phenomenon of uncomputability that Turing discovered in 1936, of numbers which cannot be computed.

How can we come up with a number like this? I'll do it by combining ideas from Turing with ideas from Borel, and then using compression to eliminate all the redundancy. And the result will be my Ω number. This is not how I actually discovered Ω, but I think it is a good way to understand Ω.

In his 1936 paper Turing discovered what's called *the halting problem.* This famous paper took years to digest, and it was a while before mathematicians realized how important the halting problem is. Another important idea in this paper is the notion of a universal Turing machine. Of course, he doesn't call it a Turing machine, that name came later. So if you look there you don't find the words "Turing machine."

Another thing that is in this paper but you won't find it if you look for it, is a very famous result called *the unsolvability of the halting problem,* which I will explain now. If you look at the paper, it's not easy to spot, it's not called that, but the idea is certainly there.

It took years of work on this paper by a community to extract the essential ideas, give them catchy names, and start waving flags with those names on them.

So let me tell you about the halting problem, which is a very fundamental thing that Turing came up with.

Remember that Turing has the idea of a general-purpose computer, and then since he's a pure mathematician, he immediately starts pointing out that there are things that no computer can calculate. There are things that no algorithm can achieve, which there is no mechanical way to calculate.

One of these things is called the halting problem. What is the halting problem? It's a very simple question. Let's say you're given a computer program, and it's a computer program that is self-contained,

so it cannot ask for input, it cannot read in any data. If there is any data it needs all that data has to be included in the program as a constant.

And the program just starts calculating, it starts executing. And there are two possibilities: Does the program go on forever, or at some point does it get a result and say "I'm finished," and halt? That's the question.

Does a self-contained computer program ever halt?

So you're given a program that's self-contained, and want to know what will happen. It's a self-contained program, you just start running it — the process is totally mechanical — and there are two possibilities: The first possibility is that this process will go on forever, that the program is searching for something that it will never find, and is in some kind of an infinite loop.

The other possibility is that the program will eventually find what it is looking for, and maybe produce a result; at any rate, it will halt and stop and it is finished.

And you can find out which is the case by running it. You run the program, and if it stops eventually you are going to discover that, if you are patient enough.

The problem is what if it never stops; running the program cannot determine that. You can give up after running the program for a day or for a week, but you can't be sure it is never going to stop.

So Turing asks the very deep question, "Is there a general procedure, given a program, for deciding in advance, without running it, whether it is going to go on forever or whether it is eventually going to stop?" You want an algorithm for deciding this. You want an algorithm which will take a finite amount of time to decide, and will always give you the correct answer.

And what Turing shows is that there is no general method for deciding, there is no algorithm for doing this; deciding whether a program halts or not isn't a computable function.

This is a very simple question involving computer programs that always has an answer — the program either goes on forever or not — but there's no mechanical procedure for deciding, there's no algorithm which always gives you the correct answer, there's no general way, given a computer program, to tell what is going to happen.

For individual programs you can sometimes decide, you can even settle infinitely many cases, but there's no general way to decide.

This is the famous result called the unsolvability of the halting problem.

Turing, 1936: *Unsolvability of the halting problem!*

However if you are a practical person from the business school, you may say, "What do I care?" And I would have to agree with you. You may well say, "All I care is will the program stop in a reasonable amount of time, say, a year. Who is going to wait more than a year?"

But if you want to know if a program will stop in a fixed amount of time, that's very easy to do, you just run it for that amount of time and see what happens. There is no unsolvability, none at all.

You only get into trouble when there's no time limit.

So you may say that this is sort of a fantasy, because in the real world there is always a time limit: We're not going to live forever, or you're going to run out of power, or the computer is going to break down, or be crushed by glaciers, or the continents will shift and a volcano will melt your computer, or the sun will go nova, whatever horror you want to contemplate!

And I agree with you. The halting problem is a **theoretical** question. It is not a **practical** question. The world of mathematics is a toy world where we ask fantasy questions, which is why we have nice theories that give nice answers. The real world is messy and complicated.

The reason you can use reasoning and prove things in pure mathematics is because it's a toy world, it's much simpler than the real world.

Okay, so this question, the halting problem, is not a real question, it's an abstract, philosophical question. If you suspected that, I agree with you, you were right! But I like to live in the world of ideas. It's a game, you may say, it's a fantasy world, but that's the world of pure mathematics.

So let's start with the question Turing proved in his 1936 paper is unsolvable. There is no general method, no mechanical procedure, no algorithm to answer this question that will always work. So what I do, is I play a trick of the kind that you use in a field called *statistical mechanics,* a field of physics, which is to take an individual problem and imbed it in a space, an ensemble of all possible problems of that type. That's a well-known strategy.

In other words, instead of asking if an individual program halts or not, let's look at the probability that this will happen, taken over the ensemble of all possible programs...

But first let me tell you why it is sometimes very important to know whether a program halts or not. You may say, "Who cares?" Well, in pure mathematics it is important because there are famous mathematical conjectures which it turns out are equivalent to asking whether a program halts or not.

There's a lovely example from ancient Greece. There's something called *a perfect number.* A number is perfect if it is the sum of all its divisors. (Or twice the sum, if you include the number itself as one of the divisors.) So 6 is a perfect number, because its divisors are 1, 2 and 3, and

$$6 = 1 + 2 + 3.$$

That's a perfect number.

If the sum of the divisors is more than the number, then it's *abundant;* if the sum of the divisors is less than the number, then it is

deficient; and if the sum of the divisors is exactly equal to the number, then it is perfect. Furthermore, two numbers are *amicable* if each one is the sum of the divisors of the other.

And there are lots of perfect numbers. The next perfect number is 28.

$$28 = 1 + 2 + 4 + 7 + 14.$$

The question is, are there any odd perfect numbers? This is a question that goes back to ancient Greece, to Pythagoras, Euclid and Plato.

Are there odd perfect numbers?

So the question is, are there any odd perfect numbers? And the answer, amazingly enough, is that nobody knows. It's a very simple question, the concepts go back two millennia, but all the perfect numbers that have been found are even, and nobody knows if there's an odd perfect number.

Now, in principle you could just start a computer program going, have it look at each odd number, find its divisors, add them, and see whether the sum is exactly the number. So if there's an odd perfect number, we're eventually going to find it.

If the program never ends, then all the perfect numbers are even. It searches for an odd perfect number and either it halts because it found one, or goes on forever without ever finding what it is looking for.

It turns out that most of the famous conjectures in mathematics, but not all, are equivalent to asking whether a computer program halts. The general idea is that most down-to-earth mathematical questions are instances of the halting problem. However whether or not there are **infinitely** many perfect numbers — which is also unknown — is **not** a case of the halting problem.

On the other hand, a famous conjecture called *the Riemann hypothesis* is an instance of the halting problem. And there's Fermat's Last

Theorem, actually a three-century old conjecture which has now been proven by Andrew Wiles, stating that there is no solution of

$$x^N + y^N = z^N \quad (x, y, z \text{ integers} > 0, \quad N \text{ integer} \geq 3).$$

These are all conjectures which if false can be refuted by a numerical counter-example. You can search systematically for a counter-example using a computer program, hence that kind of mathematical conjecture is equivalent to asking whether a program halts or not.

There's a program that systematically looks for solutions of $x^N + y^N = z^N$ and there's a program that systematically looks for zeros of the Riemann zeta function that are in the wrong place. The Riemann hypothesis is complicated, but if it's false, there is a finite calculation which refutes it, and you can search systematically for that. (The Riemann hypothesis is important because if it's true, then the prime numbers are smoothly distributed in a certain precise technical sense. This seems to be the case but no one can prove it.)

What I'm trying to say is that a lot of famous mathematical conjectures are equivalent to special cases of the halting problem. If you had a way of solving the halting problem that would be pretty nifty. It would be great to have an *oracle* for the halting problem. Which by the way is Turing's terminology, but not in that famous 1936 paper. In another paper he talks about oracles, which is a lovely term to use in pure mathematics.

Following Borel 1927, we know how to pack the answers to all possible cases of the halting problem into one real number, and this gives us a more realistic version of Borel's magical know-it-all oracle number. You use the successive bits of a real number to give the answer to every individual case of the halting problem.

Remember that you can think of a computer program as a whole number, as an integer. You can number all the programs. In binary

machine language a computer program is just a long bit string, and you can think of it as the base-two numeral for a big whole number. So every program is also a whole number.

And then if a program is the number N, the Nth program in a list of all possible programs, you use the Nth bit of a real number to tell us whether or not that program halts. If the Nth program halts, the Nth bit will be a 1; if it doesn't halt, the Nth bit will be a 0.

Halting-problem oracle number:
The Nth bit answers the Nth case of the halting problem.

This is a more realistic version of Borel's 1927 oracle number. And following Turing's 1936 paper it is uncomputable. Why?

Because if you could compute this real number, you could solve the halting problem, you could decide whether any self-contained program will halt, and this would enable you to settle a lot of famous mathematical conjectures, for instance the Riemann hypothesis. The Clay Mathematics Institute has offered a million dollar prize to the person who settles the Riemann hypothesis, but only if they settle it positively, I think. But it would also be very interesting to refute the Riemann hypothesis.

There is a bit in this real number which corresponds to the program that looks for a refutation of the Riemann hypothesis. If you could know what this particular bit is, that wouldn't actually be worth a million dollars, because it wouldn't give you a proof. Nevertheless this is a piece of information that a lot of mathematicians would like to know because the Riemann hypothesis is a famous problem in pure mathematics having to do with the prime numbers and how smoothly they are distributed.

So a halting-problem oracle would be a valuable thing to have. This number wouldn't tell you about history or the future; it wouldn't answer every yes/no question in French. But Borel's 1927 number is

paradoxical. Our halting-problem oracle is a much more down-to-earth number. In spite of being more down to earth, it is an uncomputable real that would also be very valuable.

But we can do even better! This halting-problem oracle packs a lot of mathematical information into one real number, but it doesn't do it in the best, most economical way. This real number is **redundant**, it repeats a lot of information, it's not the most compact, concise way to give the answer to every case of the halting problem. You're wasting a lot of bits, you're wasting a lot of space in this real number, you're repeating a lot of information.

Let me tell you why. We want to know whether individual programs halt or not. Now I'll give the second and last proof in this talk.

Suppose that we are given a lot of individual cases of the halting problem. Suppose we have a list of a thousand or a million programs, and want to know if each one halts or not. These are all self-contained programs.

If you have a thousand programs or a million programs, you might think that to know whether each of these programs halts or not is a thousand or a million bits of mathematical information. And it turns out that it's not, it's actually only ten or twenty bits of mathematical information.

$$N \text{ cases of halting problem} \quad = \quad \text{only } \log_2 N \text{ bits of information.}$$

Why isn't it a thousand or a million bits of information?

Well, you don't need to know the answer in every individual case. You don't want to ask the oracle too many questions. Oracles should be used sparingly.

Do we really need to ask the oracle about each individual program? Not at all! It is enough to know *how many* of the programs halt; I don't need to know each individual case.

And that's a lot less information. If there are 2^N programs, you only need N bits of information, not 2^N bits. You don't need to know

about each individual case. As I said, you just need to know how many of the programs halt. If there are N programs, that's just $\log_2 N$ bits of information, which is much less than N bits of information.

How come we get this huge savings?

Let's say you are given a finite set of programs, you have a finite collection of programs, and you want to know whether each one halts or not. Why does it suffice to know how many of these programs halt? You just start running all of them in parallel, and they start halting, and eventually all the programs that will ever halt, have halted. And if you know exactly how many that is, you don't have to wait any longer, you can stop at that point. You know that all the other programs will never halt. All the ones that haven't halted yet are never going to halt.

In other words, the answers to individual instances of the halting problem are never independent, they are always correlated. These are not independent mathematical facts. That's why we don't really need to ask an oracle in each individual case whether a program halts. We can compress this information a great deal. This information has a lot of redundancy. There are a lot of correlations in the answers to individual instances of the halting problem.

Okay, so you don't need to use a bit for each program to get a real number that's an oracle for the halting problem. I just told you how to do much better if you are only interested in a finite set of programs. But what if you are interested in all possible programs, what then? Well, here's how you handle this.

You don't ask whether individual programs halt or not; you ask what is the probability that a program chosen at random will halt.

$$\text{Halting probability } \Omega \quad = \quad \text{Prob}\{\text{random program halts}\}.$$

That's a real number between zero and one, and it is a real number I'm very proud of. I like to call it Ω, which is the last letter in the Greek alphabet, because it's sort of a *maximally unknowable real number*.

Let me explain first how you define Ω, and then I'll talk about its remarkable properties.

The idea is this: I'm taking Turing's halting problem and I'm making it into the halting probability. Turing is interested in individual programs and asks whether or not they halt. I take all possible programs, I put them into a bag, a big bag that contains every possible computer program, I close my eyes, I shake the bag, I reach in and pull out a program and ask, "What is the probability that this program will halt?"

If every program halts, this probability would be one. If no program halts, the probability of halting would be zero. Actually some programs halt and some don't, so the halting probability is going to be strictly between zero and one

$$0 < \quad \Omega = .11011100\ldots \quad < 1,$$

with an exact numerical value depending on the choice of programming language.

And it turns out that if you do things properly — there are some technical problems that I don't want to talk about — you don't really need to know for every individual program whether it halts or not. What you really need to know is what is the probability that a program will halt.

And the way it works is this: If I know the numerical value of the halting probability Ω with N bits of precision — I'm writing it in binary, in base two — if I know the numerical value of the halting probability Ω with N bits of precision, then I know for every program up to N bits in size whether or not it halts.

Can you see why? Try thinking about it for a while.

Knowing N bits of $\Omega \quad \Rightarrow \quad$ Knowing which $\leq N$ bit programs halt.

This is a very compact, compressed way — in fact, it is the most compressed, compact way — of giving the answers to Turing's halting problem. You can show this is the best possible compression, the best possible oracle, this is the most economical way to do it, this is the *algorithmic information content* of the halting problem.

Let me try to explain this. Do you know about file compression programs? There are lots of compression programs on your computer, and I'm taking all the individual answers to the halting problem and compressing them.

So whatever your favorite compression program is, let's use it to compress all the answers to the halting problem. If you could compress it perfectly, you'd get something that has absolutely no redundancy, something that couldn't be compressed any more.

So you get rid of all the redundancy in individual answers to the halting problem, and what you get is this number I call the halting probability Ω. This is just the most compact, compressed way to give you the answer to all the individual cases of Turing's famous 1936 halting problem.

Even though Ω is a very valuable number because it solves the halting problem, the interesting thing about it is that it is *algorithmically and logically irreducible.* In other words, Ω looks random, it looks like it has no structure, the bits of its numerical value look like independent tosses of a fair coin.

The bits of Ω are irreducible mathematical information.

Why is this? The answer is, basically, that any structure in something disappears when you compress it. If there were any pattern in the bits of Ω, for example, if 0s and 1s were not equally likely, then Ω would not be maximally compressed. In other words, when you remove all the redundancy from something, what you're left with looks random, but it isn't, because it's full of valuable information.

What you get when you compress Turing's halting problem, Ω, isn't noise, it's very valuable mathematical information, it gives you the answers to Turing's halting problem, but it looks random, accidental, arbitrary, simply because you've removed all the redundancy. Each bit is a complete surprise.

This may seem paradoxical, but it is a basic result in information theory that once you compress something and get rid of all the redundancy in it, if you take a meaningful message and do this to it, afterwards it looks just like noise.

Let me summarize what we've seen thus far.

We have the halting probability Ω that is an oracle for Turing's halting problem. It depends on your programming language and there are technical details that I don't want to go into, but if you do everything properly you get this probability that is greater than zero and less than one. It's a real number, and if you write it in base two, there's no integer part, just a "." and then a lot of bits. These bits look like they have absolutely no structure or pattern; they look random, they look like the typical result of independent tosses of a fair coin. They are sort of maximally unknowable, maximally uncomputable. Let me try to explain what this means.

At this point I want to make a philosophical statement. In pure mathematics all truths are **necessary truths**. And there are other truths that are called *contingent* or *accidental* like historical facts. That Napoleon was the emperor of France is not something that you expect to prove mathematically, it just happened, so it's an accidental or a contingent truth.

And whether each bit of the numerical value of the halting probability Ω is a 0 or a 1 is a necessary truth, but looks like it's contingent. It's a perfect simulation of a contingent, accidental, random truth in pure mathematics, where all truths are necessary truths.

The bits of Ω are necessary but look accidental, contingent.

This is a place where God plays dice. I don't know if any of you remember the dispute many years ago between Neils Bohr and Albert Einstein about quantum mechanics? Einstein said, "God doesn't play dice!", and Bohr said, "Well, He does in quantum mechanics!" I think God also plays dice in pure mathematics.

I do believe that in the Platonic world of mathematics the bits of the halting probability are fully determined. It's not arbitrary, you can't chose them at random. In the Platonic world of pure mathematics each bit is determined. Another way to put it is that God knows what each bit is.

But what can we know down here at our level with our finite means? Well, seen from our limited perspective the bits of Ω are maximally unknowable, they are a worst case.

The precise mathematical statement of why the bits of the numerical value of Ω are difficult to know, difficult to calculate and difficult to prove (to determine what they are by proof) is this: In order to be able to calculate the first N bits of the halting probability you need to use a program that is at least N bits in size. And to be able to prove what each of these N bits is starting from a set of axioms, you need to have at least N bits of axioms.

So the bits of Ω are irreducible mathematical facts, they are computationally and logically irreducible. Essentially the only way to get out of a formal mathematical theory what these bits are, is to put that in as a new axiom. But you can prove anything by adding it as a new axiom.

So this a place where mathematical truth has no structure, no pattern, where logical reasoning doesn't work, because these are sort of accidental mathematical facts.

Let me explain this another way. Leibniz talks about something called the *principle of sufficient reason.* He was a rationalist, and he

believed that if anything is true, it must be true for a reason. In pure math the reason that something is true is called *a proof.* However, the bits of the halting probability Ω are truths that are true for no reason; more precisely, they are true for no reason simpler than themselves. The only way to prove what they are is to take that as a new postulate. They seem to be completely contingent, entirely accidental.

The bits of Ω are mathematical facts that are true for no reason.

They look a lot like independent tosses of a fair coin, even though they are determined mathematically. It's a perfect simulation within pure math of independent tosses of a fair coin.

So to give an example, 0s and 1s are going to be equally likely. If you knew all the even bits, it wouldn't help you to get any of the odd bits. If you knew the first million bits, it wouldn't help you to get the next bit. It's a place where mathematical truth just has no structure or pattern.

But the bits of Ω do have a lot of **statistical** structure. For example, in the limit there will be exactly as many 0s as 1s, the ratio of their occurrences will tend to unity. Also, all blocks of two bits are equally likely. 00, 01, 10 and 11 each have limiting relative frequency exactly 1/4 — you can prove that. More generally, Ω is what Borel called a *normal number*, which means that in each base b, every possible block of K base-b "digits" will have exactly the same limiting relative frequency $1/b^K$. That's provably the case for Ω. Ω is provably Borel normal.

Another thing you can show is that the Ω number is *transcendental*; it's not *algebraic*, it's not the solution of an algebraic equation with integer coefficients. Actually any uncomputable number must be transcendental; it can't be algebraic. But Ω is more than uncomputable, it's maximally uncomputable. This is a place where mathematical truth has absolutely no structure or pattern. This is a place where mathematical truth looks contingent or accidental or random.

Now if I may go one step further, I'd like to end this talk by comparing pure mathematics with theoretical physics and with biology.

Pure mathematics developed together with theoretical physics. A lot of wonderful pure mathematicians of the past were also theoretical physicists, Euler for example, or more recently Hermann Weyl. The two fields are rather similar. And physicists are still hoping for a theory of everything (TOE), which would be a set of simple, elegant equations that give you the whole universe, and which would fit on a T-shirt.

So that's physics. On the other hand we have biology. Molecular biology is a very complicated subject. An individual cell is like a city. Every one of us has 3×10^9 bases in our DNA, which is 6×10^9 bits. There is no simple equation for a human being. Biology is the domain of the complicated.

How does pure mathematics compare with these two other fields? Normally you think pure math is closer to physics, since they grew together, they co-evolved. But what the bits of the halting probability Ω show is that in a certain sense pure math is closer to biology than it is to theoretical physics, because pure mathematics provably contains **infinite irreducible complexity**. Math is even worse than biology, which has very high but only **finite** complexity. The human genome is 6×10^9 bits, which is a lot, but it's finite. But pure mathematics contains the bits of Ω, which is an infinite number of bits of complexity!

$$\text{Human} = 6 \times 10^9 \text{ bits}, \quad \Omega = \text{infinite number of bits.}$$

REFERENCES

FINCH, S. *Mathematical constants.* Cambridge University Press, 2003.

TURING, A. M. "On Computable Numbers, with an Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society*, 2(42), p. 230–65, 1936.