The Perfect Language

Gregory Chaitin

Mathematics / Biography / Vol. 1, No. 3

What follows is based on a talk originally given by the author at the Hebrew University in Jerusalem and then, in expanded form, at the Perimeter Institute in Canada.

I'm going to talk about mathematics, and I'd like to give you a broad overview, most definitely a non-standard view of some intellectual history.

There is a wonderful book by Umberto Eco called *The Search for the Perfect Language*, and I recommend it highly.[1]

In *The Search for the Perfect Language* you can see that Umberto Eco likes the Middle Ages—I think he probably wishes we were still there. The book talks about a dream that Eco believes played a fundamental role in European intellectual history, which is the search for the perfect language.

What is the search for the perfect language? Nowadays a physicist would call this the search for a Theory of Everything (TOE), but in the terms in which it was formulated originally, it was the idea of finding, shall we say, the language of creation, the language before the Tower of Babel, the language that God used in creating the universe, the language whose structure directly expresses the structure of the world, the language in which concepts are expressed in their direct, original format.

You can see that this idea is a little bit like the attempt to find a foundational Theory of Everything in physics.

The crucial point is that knowing this language would be like having a key to universal knowledge. If you're a theologian, it would bring you closer, very close, to God's thoughts, which is dangerous. If you're a magician, it would give you magical powers. If you're a linguist, it would tell you the original, pure, uncorrupted language from which all languages descend.

This very fascinating book is about the quest to find that language. If you find it, you're opening a door to absolute knowledge, to God, to the ultimate nature of reality.

And there are a lot of interesting chapters in this intellectual history, one of them the Catalan, Raymond Lull, who lived in or about 1200.

He was a very interesting man who had the idea of mechanically combining all possible concepts to get new knowledge. So you would have a wheel with different concepts on it, and another wheel with other concepts on it, and you would rotate them to get all possible combinations. This would be a systematic way to discover

new concepts and new truths. If you remember Jonathan Swift's *Gulliver's Travels*, Swift makes fun of an idea like this.

In *The Search for the Perfect Language*, there is an entire chapter about Gottfried Wilhelm Leibniz. Leibniz is wonderful because he is universal. He knew all about Kabbalah, Christian Kabbalah and Jewish Kabbalah, and all kinds of hermetic and esoteric doctrines, and he knew all about alchemy, he actually ghost-authored a book on alchemy. Leibniz knew about all these things, and he knew about ancient philosophy, he knew about scholastic philosophy, and he also knew about what was then called mechanical philosophy, which was the beginning of modern science. And Leibniz saw good in all of this.

Leibniz formulated a version of the search for the perfect language, which was firmly grounded in the magical, theological original idea, but which is also fit for consumption nowadays, that is, acceptable to modern ears, to contemporary scientists. This is a universal language, which he called the *characteristica universalis*, that was supposed to come with a crucial *calculus ratiocinator*.

The idea is to reduce reasoning to calculation, to computation, because the most certain thing is that 2 + 5 = 7, and what is this if not a calculation? If two people have an intellectual dispute, Leibniz remarked, instead of dueling they could just sit down and say, "Gentlemen, let us compute!" and get the correct answer, and find out who was right.

This is Leibniz's version of the search for the perfect language. How far did he get with it? Well, Leibniz was a person who got bored easily and flew like a butterfly from field to field, throwing out fundamental ideas, rarely taking the trouble to develop them fully.

One case of the *characteristica universalis* that Leibniz *did* develop is called the calculus. This is one case where Leibniz worked out his ideas for the perfect language in beautiful detail.

Leibniz's version of the calculus differs from Isaac Newton's precisely because it was part of Leibniz's project for the *characteristica universalis*. Christiaan Huygens hated the calculus. He taught Leibniz mathematics in Paris at a relatively late age, when Leibniz was in his twenties. Most mathematicians start very, very young. And Christiaan Huygens hated Leibniz's calculus because he said that it was mechanical, it was brainless: any fool can just calculate the answer by following the rules, without understanding what he or she is doing.

Huygens preferred the old, synthetic geometry proofs, where you had to be creative and come up with a diagram and some particular reason for something to be true. Leibniz wanted a general method. He wanted to get the formalism, the notation, right, and have a mechanical way to get the answer. Huygens didn't like this, but that was precisely the point. That was precisely what Leibniz was looking for. The idea was that if you get absolute truth, if you have found the truth, it should mechanically enable you to determine what's going on, without creativity. This is good, this is not bad. This is also precisely how Leibniz's version of the calculus differed from Newton's. Leibniz saw clearly the importance of having a formalism that led you automatically to the answer.

Let's now take a big jump, to David Hilbert, about a century ago. No, first I want to tell you about an important attempt to find the perfect language: Georg Cantor's theory of infinite sets. This late nineteenth-century theory is interesting because it's firmly based in the Middle Ages and also, in a way, the inspiration for all of twentieth-century mathematics. This theory of infinite sets was actually theology—*mathematical* theology. Normally you don't mention that fact. The price of admission to the field of mathematics demands that the mathematician throw out all the philosophy, leaving only something technical behind. So all the theology has been thrown out.

But Cantor's goal was to understand God. God is transcendent. The theory of infinite sets has a hierarchy of bigger and bigger infinities, the alephs, the $\aleph$'s. You have $\aleph_0$, $\aleph_1$, the infinity of integers, of real numbers, and you keep going. Each one of these is the set of all subsets of the previous one. And very far out you get mind-boggling infinities like $\aleph_\omega$. This is the first infinity after

$$\aleph_0,\ \aleph_1, \aleph_2, \aleph_3, \aleph_4\ \ldots$$

Then you can continue with

$$\omega + 1,\ \omega + 2,\ \omega + 3\ \ldots\ 2\omega + 1,\ 2\omega + 2,\ 2\omega + 3\ \ldots$$

These so-called ordinal numbers are subscripts for the $\aleph$'s, which are cardinalities. Let's go farther:

$$\aleph_{\omega^2},\ \aleph_{\omega^\omega},\ \aleph_{\omega^{\omega^\omega}}\ \ldots$$

and there's an ordinal called epsilon-nought

$$\epsilon_0 = \omega^{\omega^{\omega^{\omega^{\cdots}}}}$$

which is the smallest solution of the equation

$$x = \omega^x.$$

The corresponding cardinal

$$\aleph_{\epsilon_0}$$

is pretty big!

God is very far off, since God is infinite and transcendent. We can try to go in his direction. But we're never going to get there, because after every cardinal, there's a bigger one, the cardinality of the set of all subsets. And after any infinite sequence of cardinals that you get, you just take the union of all of that, and you get a bigger cardinal than is in the sequence. So this thing is inherently open-ended.

This is absolutely wonderful, breathtaking stuff.

The only problem is that it's contradictory.

The problem is very simple. If you take the universal set, the set of everything, and you consider the set of all its subsets, by Cantor's diagonal argument this should have a bigger cardinality, but how can you have anything bigger than the set of everything?

This is the paradox that Bertrand Russell discovered. Russell looked at this and asked why you get this bad result. And if you look at the Cantor diagonal argument proof that the set of all subsets of everything is bigger than everything, it involves the set of all sets that are not members of themselves,

$$\{\, x \,:\, x \notin x \,\},$$

which can neither be in itself nor not be in itself. This is called the Russell paradox.

Cantor was aware that this happens, but he wasn't bothered by these contradictions, because he was doing theology. We're finite but God is infinite, and it's paradoxical for a finite being to try to comprehend a transcendent, infinite being, so paradoxes are fine. But the mathematical community was not very happy with a theory which leads to contradictions. What mathematicians have done is forget about all this theology and philosophy and try to sweep the contradictions under the rug. There is an expurgated version of all this called Zermelo–Fraenkel set theory, with the axiom of choice, usually designated ZFC.

This is a formal axiomatic theory that you develop using first-order logic, and it is an expurgated version of Cantor's theory believed not to contain any paradoxes.

Bertrand Russell was inspired by all of this to attempt a general critique of mathematical reasoning, and to find a lot of contradictions, a lot of mathematical arguments that lead to contradictions. I already told you about his most famous one, the Russell paradox.

Russell was an atheist who was searching for the absolute, who believed in absolute truth. And he loved mathematics and wanted mathematics to be perfect. Russell went around telling people about these contradictions in order to try to get them fixed.

Besides the paradox that there's no biggest cardinal, and that the set of subsets of everything is bigger than everything, there's also a problem with the ordinal numbers that's called the Burali-Forti paradox, namely that the set of all the ordinals is an ordinal that's bigger than all the ordinals. This works because each ordinal can be defined as the set of all the ordinals that are smaller than it is. (Then an ordinal is less than another ordinal if and only if it is contained in it.)

Russell was going around telling people that reason leads to contradictions. So David Hilbert, about a century ago, proposed a program to put mathematics on a firm foundation. And basically what Hilbert proposed is the idea of a completely formal axiomatic theory, which is a modern version of Leibniz's *characteristica universalis* and *calculus ratiocinator*.

In such a formal axiomatic theory you would have a finite number of axioms, axioms that are not written in an ambiguous natural language. Instead you use a precise artificial language with a simple, regular artificial grammar. You use mathematical logic, not informal reasoning, and you specify the rules of the game

precisely. It should be mechanical to decide whether a proof is correct.

Hilbert was a conservative. He believed that mathematics gives absolute truth, which is an idea from the Middle Ages. You can see evidence of the Middle Ages whenever you mention absolute truth. Nevertheless, modern mathematicians remain enamored with absolute truth. As Kurt Gödel said, we pure mathematicians are the last holdouts of the Middle Ages. We still believe in the Platonic world of ideas, at least mathematical ideas, when everyone else, including philosophers, now laughs at this notion. But pure mathematicians live in the Platonic world of ideas, even though everyone else stopped believing in it a long time ago.

So mathematics gives absolute truth, said Hilbert. Every mathematician somewhere deep inside believes this. Then there ought to exist a finite set of axioms, and a precise set of rules for deduction, for inference, such that all mathematical truth is a consequence of these axioms. You see, if mathematical truth is black or white, and purely objective, then if you fill in all the steps in a proof and carefully use an artificial language to avoid ambiguity, you should be able to have a finite set of axioms we can all agree on, that in principle enables you to deduce all mathematical truth. This is just the notion that mathematics provides absolute certainty.

An important consequence of this idea goes back to the Middle Ages. This perfect language for mathematics, which is what Hilbert was looking for, would in fact give a key to absolute knowledge, because in principle you could mechanically deduce all the theorems from the axioms, simply by running through the tree of all possible proofs. You start with the axioms, then you apply the rules of inference once, and get all the theorems that have one-step proofs; you apply them two times, and you get all the theorems that have two-step proofs; and like that, totally mechanically, you would get all of mathematical truth, by systematically traversing the tree of all possible proofs.

This would not put all mathematicians out of work. In practice this process would take an outrageous amount of time to get to interesting results, and all the interesting theorems would be overwhelmed by uninteresting theorems, such as the fact that $1 + 1 = 2$. It would be hard to find the interesting theorems and to separate the wheat from the chaff. But in principle this would give you all mathematical truths. You wouldn't actually do it, but it would show that mathematics gives absolute certainty.

So this was the idea of putting mathematics on a firm foundation and removing all doubts. This was Hilbert's idea, about a century ago. Meta-mathematics studies a formal axiomatic theory from the outside. Notice that this is a door to absolute truth, following the notion of the perfect language.

What happened? There is some good news and some bad news. Some of the good news I already mentioned. The thing that comes the closest to what Hilbert asked for is Zermelo–Fraenkel set theory, and it is a beautiful axiomatic theory. I want to mention some of the milestones in the development of this theory. One of them is the von Neumann integers, so let me tell you about that. Remember that Baruch Spinoza had a philosophical system in which the world is built out of only one substance, and that substance is God, that's all there is. Zermelo–Fraenkel set theory is similar. Everything is sets, and every set is built out of the empty set. That's all there is: the empty set, and sets built starting with the empty set.

Zero is the empty set $\{\}$, that's the first von Neumann integer, and in general $n + 1$ is defined to be the set of

all integers less than or equal to n:

$$n + 1 = \{0, 1, 2, \ldots, n\}.$$

If you write this out in full, removing all the abbreviations, all you have are curly braces, you have set formation starting with no content, and the full notation for n grows exponentially in n because everything up to that point is repeated in the next number. In spite of this exponential growth, this is a beautiful conceptual scheme.

Then you can define rational numbers as pairs of these integers, you can define real numbers as limit sequences of rational numbers, and you get all of mathematics, starting just with the empty set. So it's a lovely piece of ontology. Here's all of mathematical creation just built out of the empty set.

This is a formal theory that most mathematicians believe enables you to carry out all the arguments that normally appear in mathematics—maybe if you don't include category theory, which is very difficult to formalize, and even more paradoxical than set theory, from what I hear.

So that's some of the positive work on Hilbert's program. Now some of the negative work on Hilbert's program is, of course, Gödel in 1931 and Alan Turing in 1936. What they show is that you can't have a perfect language for mathematics, you cannot have a formal axiomatic theory for all of mathematics because of incompleteness, because no such system will include all of mathematical truth. It will always leave out truths; it will always be incomplete.

This is Gödel's incompleteness theorem of 1931, and Gödel's original proof is very strange. It's basically the paradox of "this statement is false," which is a paradox, of course, because it can be neither true nor false. If it's false that it's false, then it's true, and if it's true that it's false, then it's false. That's just a paradox. But what Gödel does is say "this statement is unprovable." So if the statement says of itself it's unprovable, there are two possibilities: it's provable, or it isn't. If it's provable, then we're proving something that's false, because it says it's unprovable. So we hope that's not the case; by hypothesis, we'll eliminate that possibility. If we prove things that are false, we have a formal axiomatic theory that we're not interested in, because it proves false things. The only possibility left is that it's unprovable. But if it's unprovable then it's true, because it asserts it's unprovable, therefore there's a hole. We haven't captured all of mathematical truth in our theory.

This proof of incompleteness shocked a lot of people.

A better proof of incompleteness, a deeper proof, comes from Turing in 1936. He derived incompleteness from a more fundamental phenomenon, which is uncomputability, the discovery that mathematics is full of stuff that can't be calculated, of things you can define, but which you cannot calculate, because there's no algorithm. And in particular, the uncomputable thing that he discovered was the halting problem, a very simple question: does a computer program that's self-contained halt, or does it go on forever? There is no algorithm to answer this in every individual case, therefore there is no formal axiomatic theory that enables you to always prove in individual cases what the answer is.

So Turing's insight in 1936 was that incompleteness, that Gödel found in 1931, for any formal axiomatic theory, comes from a deeper phenomenon, which is uncomputability. Incompleteness is an immediate corollary of uncomputability, a concept which does not appear in Gödel's 1931 paper.

But Turing's paper has both good and bad aspects. There's a negative aspect of his 1936 paper, which I've just told you about, but there's also a positive aspect. You get another proof, a deeper proof of incompleteness, but you also get a kind of completeness.

You find a perfect language.

There is no perfect language for mathematical reasoning. Gödel showed that in 1931, and Turing showed it again in 1936. But what Turing also showed in 1936 is that there are perfect languages, not for mathematical reasoning, but for computation, for specifying algorithms. What Turing discovered in 1936 is that there's a kind of completeness called universality and that there are universal Turing machines and universal programming languages.

What universal means, what a universal programming language or a universal Turing machine is, is a language in which every possible algorithm can be written. On the one hand, Turing showed us in a deeper way that any language for mathematical reasoning has to be incomplete, but on the other hand, he showed us that languages for computation can be universal, which is just a synonym for completeness. There are perfect languages for computation, for writing algorithms, even though there aren't any perfect languages for mathematical reasoning.

This is the positive side, this is the completeness side, of Turing's 1936 paper.

Now, what I've spent most of my professional life on, is a subject I call algorithmic information theory, which derives incompleteness from uncomputability by taking advantage of a deeper phenomenon, by considering an extreme form of uncomputability, which is called algorithmic randomness or algorithmic irreducibility.

There's a perfect language again, and there's also a negative side, the halting probability Ω, whose bits are algorithmically random, algorithmically irreducible mathematical truths.

$$\Omega = .010010111\ldots$$

This is a place in pure mathematics where there's no structure. If you want to know the bits of the numerical value of the halting probability, this is a well-defined mathematical question, and in the world of mathematics all truths are necessary truths, but these look like accidental, contingent truths. They look random; they have irreducible complexity.

There are actually an infinite number of halting probabilities, depending on your choice of programming language. After you choose a language, then you ask what the probability is that a program generated by coin tossing will eventually halt. And that gives you a different halting probability.

The numerical value will be different; the paradoxical properties are the same.

There are cases for which you can get a few of the first bits. For example, if Ω starts with 1s in binary or 9s in decimal, you can know those bits or digits, if Ω is .11111… base two or .99999… base ten. So you can get a finite number of bits, perhaps, of the numerical value, but if you have an *N*-bit formal axiomatic theory, then you can't get more than *N* bits of Ω. That's sort of the general result. It's irreducible logically and computationally. It's irreducible mathematical information.

That's the bad news. Algorithmic information theory (AIT) goes further than Turing, and picks out, from Turing's universal languages, maximally expressive programming languages—because those are the ones that you have to use to develop this theory where you get to Ω.

AIT has the notion of a maximally expressive programming language in which programs are maximally compact, and deals with a very basic complexity concept, which is the size of the smallest program to calculate something.

Now we have a better notion of perfection. Universal programming languages are not all equally good. We concentrate on a subset, comprising the ones that enable us to write the most concise programs. These are the most expressive languages, the ones with the smallest programs. This definition of complexity is a dry, technical way of expressing an idea in modern terms. But let me put this into medieval terminology, which is much more colorful. What we're asking is, how many yes/no decisions did God have to make to create something?—which is obviously a rather basic question to ask, if you consider that God is calculating the universe. I'm giving you a medieval perspective on these modern developments.

Theology is the fundamental physics, it's the theoretical physics of the Middle Ages.

The notion of the universal Turing machine that is used in AIT is Turing's very basic idea of a flexible machine. It's flexible hardware, which we call software. Now, AIT picks out a particular class of universal Turing machines *U*.

What are the universal computers *U*? A universal computer *U* has the property that, for any other computer *C* and its program *p*, the universal computer *U* will calculate the same result if you give it the original program *p* for *C* concatenated to a prefix π_C which depends only on the computer *C* that you want to simulate. π_C tells *U* which computer to simulate. In symbols,

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

In other words, $\pi_C p$ is the concatenation of two pieces of information. It's a binary string. You take the original program *p*, which is also a binary string, and in front of it you put a prefix that tells you which computer to simulate. This means that these programs $\pi_C p$ for *U* are only a fixed number of bits larger than the programs *p* for any individual machine *C*. These *U* are the universal Turing machines that you use in AIT. These are the most expressive languages. These are the languages in which programs are as concise as possible. This is how you define program-size complexity. God will naturally use the most perfect, most powerful programming languages, when he creates the world, to build everything.

AIT is concerned with particularly efficient ways for U to be universal. Turing's original notion of universality was not this demanding. The fact that you can just add a fixed number of bits to a program for C to get one for U is not completely trivial. Let me tell you why. After you put π_C and p together, you have to know where the prefix ends and the program that is being simulated begins. There are many ways to do this. A very simple way to make the prefix π_C self-delimiting is to have it be a sequence of 0's followed by a 1:

$$\pi_C = 0^k 1$$

And the number k of 0's tells us which machine C to simulate. That's a very wasteful way to indicate this.

The prefix π_C is actually an interpreter for the programming language C. AIT's universal languages U have the property that you give U an interpreter plus the program p in this other language C, and U will run the interpreter to see what p does.

If you think of this interpreter π_C as an arbitrary string of bits, one way to make it self-delimiting is to just double all the bits. 0 goes to 00, 1 goes to 11, and you put a pair of unequal bits 01 as punctuation at the end.

$$\pi_C : 0 \to 00,\ 1 \to 11,\ 01 \text{ at the end.}$$

This is a better way to have a self-delimiting prefix that you can concatenate with p. It only doubles the size, whereas the 0^k1 trick increases the size exponentially. And there are more efficient ways to make the prefix self-delimiting. For example, you can put the size of the prefix in front of the prefix. But it's sort of like Russian dolls, because if you put the size $|\pi_C|$ of π_C in front of π_C, $|\pi_C|$ also has to be self-delimiting:

$$U(\ldots ||\pi_C||\ |\pi_C|\pi_C p) = C(p)$$

Anyway, picking U this way is the key idea in the original 1960s version of AIT that Andrey Kolmogorov, Ray Solomonoff, and I independently proposed. But ten years later I realized that this is not the right approach. You actually want the whole program $\pi_C p$ for U to be self-delimiting, not just the prefix π_C. You want the whole thing to be self-delimiting to get the right theory of program-size complexity.

Let me compare the 1960s version of AIT and the 1970s version of AIT. Let me compare these two different theories of program-size complexity.

In the 1960s version, an N-bit string will in general need an N-bit program, if it's irreducible, and most strings are algorithmically irreducible. Most N-bit strings need an N-bit program. These are the irreducible strings, the ones that have no pattern, no structure. Most N-bit strings need an N-bit program, because there aren't enough smaller programs. But in the 1970s version of AIT, you go from N bits to $N + \log_2 N$ bits, because you want to make the programs self-delimiting. An N-bit string will usually need an $N + \log_2 N$ bit program.

Actually, in 1970s AIT it's N plus $H(N)$, which is the size of the smallest self-delimiting program to calculate N, that is exactly what that logarithmic term is. In other words, in the 1970s version of AIT, the size of the smallest program for calculating an N-bit string is usually N bits plus the size in bits of the smallest self-delimiting program to calculate N, which is roughly

$$\log N + \log\log N + \log\log\log N + \ldots$$

bits long.

That's the Russian dolls aspect of this.

The 1970s version of AIT, which takes the idea of being self-delimiting from the prefix and applies it to the whole program, gives us even better perfect languages. AIT evolved in two stages. First we concentrate on those U with

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

with π_C self-delimiting, and then we insist that the whole thing $\pi_C p$ has also got to be self-delimiting. And when you do that, you get important new results, such as the sub-additivity of program-size complexity,

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

which is not the case if you don't make everything self-delimiting. This just says that you can concatenate the smallest program for calculating x and the smallest program for calculating y to get a program for calculating x and y.

And you can't even define the halting probability Ω in 1960s AIT. If you allow all N-bit strings to be programs, then you cannot define the halting probability in a natural way, because the sum for defining the probability that a program will halt

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

diverges to infinity instead of being between zero and one. This is the key technical point in AIT.

I want the halting probability to be finite. The normal way of thinking about programs is that there are 2^N N-bit programs, and the natural way of defining the halting probability is that every N-bit program that halts contributes $1/2^N$ to the halting probability. The only problem is that for any fixed size N there are roughly on the order of 2^N programs that halt, so if you sum over all possible sizes, you get infinity, which is no good.

In order to get the halting probability to be between zero and one

$$0 < \Omega = \sum_{p \text{ halts}} 2^{-(\text{size in bits of } p)} < 1$$

you have to be sure that the total probability summed over all programs p is less than or equal to one. This happens automatically if we force p to be self-delimiting. How can we do this? Easy! Pretend that you are the universal computer U. As you read the program bit by bit, you have to be able to decide by yourself where the program ends, without any special punctuation, such as a blank, at the end of the program. This implies that no extension of a valid program is itself a valid program, and that the set of valid programs is what's called a prefix-free set. Then the fact that the sum that defines Ω must be between zero and one, is just a

special case of what's called the Kraft inequality in Shannon information theory.

But this technical machinery isn't necessary. That $0 < \Omega < 1$ follows immediately from the fact that as you read the program bit by bit you are forced to decide where to stop without seeing any special punctuation. In other words, in 1960s AIT we were actually using a three-symbol alphabet for programs: 0, 1 and blank. The blank told us where a program ends. But that's a symbol that you're wasting, because you use it very little. As you all know, if you have a three-symbol alphabet, then the right way to use it is to use each symbol roughly one-third of the time. So if you really use only 0s and 1s, then you have to force the Turing machine to decide by itself where the program ends. You don't put a blank at the end to indicate that.

So programs go from N bits in size to $N + \log_2 N$ bits, because you've got to indicate in each program how big it is. On the other hand, you can just take subroutines and concatenate them to make a bigger program, so program-size complexity becomes sub-additive. You run the universal machine U to calculate the first object x, and then you run it again to calculate the second object y, and then you've got x and y, and so

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

These self-delimiting binary languages are the ones that the study of program-size complexity has led us to discriminate as the ideal languages, the most perfect languages. We got to them in two stages, 1960s AIT and 1970s AIT. These are languages for computation, for expressing algorithms, not for mathematical reasoning. They are universal programming languages that are maximally expressive, maximally concise. We already knew how to do that in the 1960s, but in the 1970s we realized that programs should be self-delimiting, which made it possible to define the halting probability Ω.

That's the story, and now maybe I should summarize all of this, this saga of the quest for the perfect language. As I said, the search for the perfect language has some negative conclusions and some positive conclusions.

Hilbert wanted to find a perfect language giving all of mathematical truth, all mathematical knowledge; he wanted a formal axiomatic theory for all of mathematics. This was supposed to be a Theory of Everything for the world of pure mathematics. And this cannot succeed, because we know that every formal axiomatic theory is incomplete, as shown by Gödel, by Turing, and by me. Instead of finding a perfect language, a perfect formal axiomatic theory, we found incompleteness, uncomputability, and even algorithmic irreducibility and algorithmic randomness.

That's the negative side of this story, which is fascinating from an epistemological point of view, because we found limits to what we can know; we found limits of formal reasoning.

Now interestingly enough, the mathematical community couldn't care less. They still want absolute truth! They still believe in absolute truth, and that mathematics gives absolute truth. And if you want a proof of this, just go to the December 2008 issue of the *Notices of the American Mathematical Society*. That's a special issue of the Notices devoted to formal proof.

The technology has been developed to the point where they can run real mathematics, real proofs, through

proof-checkers, and get them checked. A mathematician writes the proof out in a formal language, and fills in the missing steps and makes corrections until the proof-checker can understand the whole thing and verify that it is correct. And these proof-checkers are getting smarter and smarter, so that more and more of the details can be left out. As the technology improves, the job of formalizing a proof becomes easier and easier. The formal-proof extremists are saying that in the future all mathematics will have to be written out formally and verified by proof-checkers.[2]

The position of these extremists is that in the future all mathematics will have to be written out in a formal language, and you will have to get it checked before submitting a paper to a human referee, who will then only have to decide if the proof is worth publishing, not whether the proof is correct. And they want a repository of all mathematical knowledge, which would be a database of checked formal proofs of theorems.

I'm not disparaging this extremely interesting work, but I am saying that there's a wonderful intellectual tension between it and the incompleteness results that I've discussed in this talk. There's a wonderful intellectual tension between incompleteness and the fact that people still believe in formal proof and absolute truth. People still want to go ahead and carry out Hilbert's program and actually formalize everything, just as if Gödel and Turing had never happened!

I think this is an extremely interesting and, at least for me, a quite unexpected development.

These were the negative conclusions from this saga. Now I want to wrap this talk up by summarizing the positive conclusions.

There are perfect languages for computing, not for reasoning. They're computer programming languages. And we have universal Turing machines and universal programming languages, and although languages for reasoning cannot be complete, these universal programming languages are complete. Furthermore, AIT has picked out the most expressive programming languages, the ones that are particularly good to use for a theory of program-size complexity.

So there is a substantial practical spinoff. Furthermore, since I've worked most of my professional career on AIT, I view AIT as a substantial contribution to the search for the perfect language, because it gives us a measure of expressive power, and of conceptual complexity and the complexity of ideas. Remember, I said that from the perspective of the Middle Ages, that's how many yes/no decisions God had to make to create something, which obviously he will do in an optimal manner.[3]

From the theoretical side, however, this quest was disappointing: due to Gödel incompleteness and because there is no Theory of Everything for pure mathematics. In fact, if you look at the bits of the halting probability Ω, they show that pure mathematics contains infinite irreducible complexity, and in this precise sense it is more like biology, the domain of the complex, than like theoretical physics, where there is still hope of finding a simple, elegant TOE.[4]

So this is the negative side of the story, unless you're a biologist. The positive side is we get this marvelous programming technology. So this dream, the search for the perfect language and for absolute knowledge,

ended in the bowels of a computer, it ended in a Golem.

How would all this look to someone from the Middle Ages? This quest, the search for the perfect language, was an attempt to obtain magical, God-like powers.

Let's bring someone from the 1200s here and show them a notebook computer. You have this dead machine, it's a machine, it's a physical object, and when you put software into it, all of a sudden it comes to life!

So from the perspective of the Middle Ages, I would say that the perfect languages that we've found have given us some magical, God-like power, which is that we can breathe life into some inanimate matter. Observe that hardware is analogous to the body, and software is analogous to the soul, and when you put software into a computer, this inanimate object comes to life and creates virtual worlds.

So from the perspective of somebody from the year 1200, the search for the perfect language has been successful and has given us some magical, God-like abilities, except that we take them entirely for granted.

Thanks very much!

Dedicated to Newton da Costa on his 80th birthday

Adapted from the first chapter of Gregory Chaitin's book Mathematics, Complexity and Philosophy.[5] *A version of this essay was published in Portuguese in the magazine* Dicta & Contradicta.[6]

---

1. Umberto Eco, *The Search for the Perfect Language*, trans. James Fentress (London, UK: HarperCollins, 1997).
2. For a discussion of recent developments in this area, the editors suggest the following: Vladimir Voevodsky, "."
3. Note that program-size complexity = size of smallest *name* for something.
4. Incompleteness can be considered good rather than bad: it shows that mathematics is creative, not mechanical.
5. Gregory Chaitin, *Mathematics, Complexity and Philosophy* (Valparaíso, Chile: Midas Ediciones, 2011).
6. Gregory Chaitin, "A busca pela linguagem perfeita," *Dicta & Contradicta* 4 (2009): 26–41.

More From This Author

- ## Doing Mathematics Differently

  A look at complexity using algorithmic information theory.

(Mathematics / Critical Essay / Vol. 2, No. 1)

- **An Algorithmic God**

  On applying software models to biology.

  (Mathematics / Critical Notes / Vol. 1, No. 4)