FIVE GOLDEN RULES

there anyone left who still uses such things?) Like a Turing machine, a typewriter is also a primitive device, allowing us to print sequences of symbols on a piece of paper that is potentially infinite in extent. A typewriter also has only a finite number of states that it can be in: uppercase and lowercase letters, red or black ribbon, different symbol balls, and so on. Yet despite these limitations, any typewriter can be used to type The Canterbury Tales, Alice in Wonderland, or any other string of symbols. Of course, it might take a Chaucer or a Lewis Carroll to be the first to tell the machine what to do, but it can be done. By way of analogy, it might take a very skilled programmer to tell the Turing machine how to solve difficult computational problems. However, the Turing–Church Thesis states, the basic model—the Turing machine—suffices for every type of question that is at all answerable by carrying out a computation.

It has probably not escaped the reader’s attention that there is a quite evident parallel between the actions taken by a Turing machine as it goes about performing a computation and the steps a mathematician follows in proving a theorem using a chain of logical inferences. So let’s now shift our attention for a while from the computational to the purely logical, our goal being to show that they are actually the same thing.

Form and Content

During the 1928 International Congress of Mathematicians (ICM) in Bologna, Italy, the famed German mathematician David Hilbert threw down a challenge that would ultimately change forever the way we think about the relationship between what is logically provable and what is actually true.

At stake in Hilbert’s 1928 address was the issue of whether or not it is possible to prove every true mathematical statement. What Hilbert was looking for was a kind of Truth Machine capable of settling every possible mathematical statement. Just feed the statement in at one end, turn the crank, and out the other end pops the answer: TRUE or FALSE. Ideally, in this setup the original statement would be either a true mathematical fact and, hence, logically deducible from the given assumptions and thus a theorem, or it would be false and, consequently, not a theorem, that is, its negation would be a theorem. In short, Hilbert’s Truth Machine would give a complete account of every possible mathematical assertion. In his Bologna talk Hilbert laid down the requirements for such a Truth Machine, or what’s more pedantically termed an axiomatic, or formal, logical system along with the conviction that his research “Program” would ultimately yield a complete axiomatization of all mathematics.

With this challenge to the mathematical world, Hilbert was reemphasizing a different aspect of another problem he had posed at an earlier ICM gathering in Paris in 1900. With the conviction that unsolved problems are the lifeblood of any field of intellectual activity—and to mark the turn of the century—Hilbert listed 23 problems whose resolution he felt was of central importance for the development of mathematics. The second problem on this list involved proving that mathematical reasoning is reliable. In other words, by following the rules of mathematical reasoning, one should not be able to arrive at mutually contradictory statements; a proposition and its negation should not both be theorems. Of course, this self-consistency requirement is a necessary condition for any axiomatic system of the sort Hilbert had in mind to lead to logically coherent results, since Aristotle had shown long ago that if the system is inconsistent any assertion can be proved true or false as we please. And this is hardly a secure basis for any kind of reliable knowledge, mathematical or otherwise.

Less than 3 years after Hilbert’s Bologna address, the young Austrian logician Kurt Gödel astonished the mathematical world by publishing a revolutionary paper turning Hilbert’s fondest dream into his wildest nightmare. In his 1931 paper, Gödel showed that there exist true but unprovable mathematical statements. Put more prosaically, there is an eternally unbridgeable gap between what can be proved and what’s true. The idea of establishing an axiomatic framework for all of mathematics—put forth as a primary goal of mathematics by Hilbert—was the starting point for Gödel’s assault on proof. So we begin our story with a bit of background on Hilbert’s Program for axiomatizing mathematical truth.

Hilbert believed that the way to eliminate the possibility of paradoxes like “This sentence is false” (that is, inconsistencies) arising in mathematics was to create an essentially "meaningless," purely syntactic framework within which to speak about the truth or falsity of mathematical statements. This would be a framework in which the mathematical
statements are all expressed using purely abstract symbols having no intrinsic meaning other than what's given to them by definition in the framework itself. Such a framework is now termed a formal system, and it constitutes the historical jumping-off point for investigations of the gap between what can be proved and what is actually true in the universe of mathematics.

The "meaningless statements" of a formal system are composed of finite sequences of abstract symbols. The symbols are often termed the alphabet of the system, while the "words" of the system are usually called symbol strings. The symbols might be objects like ★, ●, and ★, or they might even be signs like 0 and 1. But in the latter case, it's absolutely essential to recognize that we're not talking yet about the numbers 0 and 1, but simply about the numerals 0 and 1. It's only when these symbols are given meaning as numbers that they acquire the properties we usually associate with the numbers 0 and 1. We'll come back to this point with a vengeance shortly. In a formal system, a finite number of these symbol strings are taken as the axioms of the system. To round things out, the system also has a finite number of transformation rules, or what are often called rules of logical inference. These rules specify how a given string of symbols can be converted into a new string of symbols.

The general idea of proof within a formal system is to start from one of the axioms and apply a finite sequence of transformations, thereby converting the axiom into a succession of new strings, where each string is either one of the given axioms of the system or is derived from its predecessors by application of the transformation rules. The last string in such a sequence is called a theorem of the system. The totality of all theorems constitutes what can be proved within the system. But note carefully that these so-called statements don't actually say anything; they are just strings of abstract symbols. We'll get to how the theorems acquire meaning in a moment. But first let's see how this setup works with a simple example.

Suppose the symbols of our system are the three objects ★ (star), ✧ (Maltese cross), and ● (sunburst). Let the two-element string ✧ ★ be the sole axiom of the system. Letting x denote an arbitrary finite string of stars, crosses, and sunbursts, we take the transformation rules of our system to be:

The Halting Theorem

| Rule I:         | x ★ → x ★★         |
| Rule II:       | ✧ x → ✧ x          |
| Rule III:      | ★ ★ ★ → ★          |
| Rule IV:       | x ★★ x → xx        |

In these rules, → means "is replaced by." So, for instance, Rule I says that we can form a new string by appending a star to any string that ends in a sunburst. The interpretation of Rule IV is that any time two stars appear together in a string, they can be dropped to form a new string. Now let's see how these rules can be used to prove a theorem.

Starting with the single axiom ✧ ★, we can deduce that the string ✧ ★ ★ ★ is a theorem by applying the transformation rules in the following order:

∀ (Axiom) (Rule II) (Rule II) (Rule III)

Such a sequence of steps, starting from an axiom and ending at a statement like ✧ ★ ★ ★, is termed a proof sequence for the theorem represented by the last string in the sequence. Observe that when applying Rule III at the final step, we could have replaced the last three ★s from the preceding string rather than the first three, thereby ending up with the theorem ✧ ★ ★ instead of ✧ ★ ★. The perspicacious reader will have also noted that all the intermediate strings obtained in moving from the axiom to the theorem begin with ✧. It's fairly evident from the axiom and the action of the transformation rules for this system that every string will have this property. This is a metamathematical property of the system, since it's a statement about the system rather than one made in the system itself. The distinction between what the system can say from the inside (its strings) and what we can say about the system from the outside (properties of the strings) is of the utmost importance for Godel's results.

Upon comparing the workings of a Turing machine program and the operations we just went through using the transformation rules of a formal system, one might say that there's no essential difference between the two. And so it is. The matchups showing what amounts to a perfect correspondence between Turing machines and formal systems are shown in Table 4.3.

We spoke earlier of Hilbert's famous Entscheidungsproblem, or Decision Problem, which asked if there is any algorithmic procedure for
deciding if a given symbol string is or is not a theorem of a particular formal system. Using the “isomorphism” in Table 4.3 between Turing machines and formal systems, Turing was able to translate the Decision Problem involving theorems in a formal system into its equivalent expression in the language of machines. We have already seen that this computing equivalent is the Halting Problem, whose negative solution implies the same sad answer to the Decision Problem. Since about now the right question to be asking yourself is: What does all this meaningless symbol manipulation have to do with everyday reality?, let’s quickly turn our attention from matters of form to those of content.

The answer to how we get from form to content can be given in one word: *interpretation*. Let’s focus our interest right now on the slice of everyday reality consisting of mathematical facts. Depending on the kind of mathematical structure under consideration (for example, euclidean geometry, elementary arithmetic, calculus, topology, and so forth), we have to make up a dictionary by which we can match up (that is, interpret) the objects constituting that mathematical structure, things like points, lines, and numbers, with the abstract symbols, strings, and rules of the formal system that we want to employ to represent the structure. By this dictionary construction step, we attach meaning, or semantic content, to the abstract, purely syntactic strings formed from the symbols of the formal system. Thereafter, all the theorems of the formal system can be interpreted as true statements about the associated mathematical objects. Figure 4.4 illustrates this crucial distinction between the purely syntactic world of formal systems and the meaningful world of mathematics.

Once this dictionary has been written and the associated interpretation established, then we can hope along with Hilbert that there will be a perfect, one-to-one correspondence between the true facts of the mathematical structure and the theorems of the formal system. Loosely speaking, Hilbert’s dream was to find a formal system in which every mathematical truth translates into a theorem, and conversely. Such a system is termed *complete*. Moreover, if a mathematical structure is to avoid contradiction, a mathematical truth and its negation should never both translate to theorems, that is, be provable in the formal system. Such a system in which no contradictory statements can be proved is termed *consistent*. With these preliminaries in hand, we can finally describe Gödel’s wreckage of Hilbert’s Program.

**The Undecidable**

By the time of Hilbert’s 1928 Bologna lecture, mathematicians had already established that geometrical propositions as well as all other types of mathematical assertions could be recast as assertions about numbers. Thus, the problem of the consistency of mathematics as a whole was reducible to the determination of the consistency of arithmetic. That is, to the properties and relations among the natural numbers (the positive integers $1, 2, 3, \ldots$). So the problem became to give a “theory of arithmetic,” that is, a formal system that was (1) finitely describable, (2) consistent, (3) complete, and (4) sufficiently powerful to represent all the statements that can be made about the natural numbers. By the term *finitely describable* what Hilbert meant was not only that the number and length of the axioms and rules of the system should be constructible in a finite number of steps, but also that every provable statement in the system—every theorem—should also be provable in a finite number of steps. This condition seems reasonable enough, since you don’t really
FIVE GOLDEN RULES

have a theory at all unless you can tell other people about it. And you certainly can’t tell them about it if there are an infinite number of axioms, rules, and/or steps in a proof sequence.

A central question that arises in connection with any such formalization of arithmetic is to ask if there is a finite procedure by which we can decide the truth or falsity of every arithmetical statement. Thus, for example, if we make the statement “The sum of two odd numbers is always an even number,” we want a finite procedure—essentially a computer program—that halts after a finite number of steps, telling us whether that statement is provable or not in some formal system powerful enough to encompass ordinary arithmetic. For example, in the $\mathcal{S} \cdot \mathcal{S}$-system considered above, such a decision procedure is given by the not-entirely-obvious conditions: “A string is a theorem if and only (1) it begins with a $\mathcal{S}$, (2) the remainder of the string consists solely of $\mathcal{S}$s and $\mathcal{S}$s, and (3) the number of $\mathcal{S}$s is not a multiple of 3.”

Hilbert was convinced that a formalization of arithmetic satisfying the foregoing desiderata was possible, and his Bologna manifesto challenged the international mathematical community to find or create it. But in 1931, less than 3 years after Hilbert’s Bolognese call to arms, Kurt Gödel published the following metamathematical fact, perhaps the most famous mathematical (and philosophical) result of this century:

---

**GÖDEL’S THEOREM—INFORMAL VERSION** Arithmetic is not completely formalizable.

---

Remember that for arithmetic there are an infinite number of ways we can choose a finite set of axioms and rules of inference in a formal system so as to attempt to mirror syntactically the mathematical truths about numbers. Gödel’s result says that none of these choices will work; there does not and cannot exist a formal system satisfying all the requirements of Hilbert’s Program. In short, there are no rules for generating all the truths about the natural numbers.

In arriving at his proof of the incompleteness of arithmetic, Gödel’s first crucial observation was to recognize the importance of Hilbert’s insight that every formalization of a branch of mathematics is itself a mathematical object in its own right, since what we mean when we say we have “formalized” something is that we have created a mathematical framework within which to speak about whatever it is we wanted to formalize. So if we create a formal system intended to capture the truths of arithmetic, that formal system can be studied not just as a set of mindless rules for manipulating symbols, but also as an object possessing mathematical, that is, semantic, as well as syntactic properties. In particular, since Gödel was interested in the relationships between numbers, his goal was to represent any formal system purporting to encompass arithmetic within arithmetic itself. Basically, this involved showing how to code any statement about numbers and their relationships by a unique number itself. Thus, what Gödel saw was a way to mirror all statements about relationships between the natural numbers by using these very same numbers themselves.

This mirroring idea is probably more familiar in the context of ordinary language where we use words in the English language to speak about language. For example, we use words to describe properties of words like whether they are nouns or verbs, and we discuss the structure of, say, a treatise on English grammar, which consists of words, by employing other words of the English language. Thus, in both cases we are making use of language in two different ways: (1) as a collection of uninterpreted strings of alphabetic symbols that are manipulated according to the rules of English grammar and syntax, and (2) as a set of interpreted strings having a meaning within the context under discussion. So the key notion is that the very same objects can be considered in two quite distinct ways, opening up the possibility for that object to speak about itself. In passing, let me note that the very same dual-level idea pertains to the symbols and their interpretations in the genetic material (the DNA) of every living cell. The nucleotide bases A, G, C, and T on the DNA strand can either be interpreted as instructions for building the proteins from which every living organism is formed, or they can simply be copied without interpretation as, for example, when the DNA is replicated in the process of cell division. Gödel discovered how to do this same trick with mirrors using the natural numbers.

To see how Gödel’s method works, let’s consider a somewhat streamlined version of the language of symbolic logic as found in the monumental treatise *Principia Mathematica* by Bertrand Russell and Alfred North Whitehead. This slimmed-down version is due to Ernest Nagel and James R. Newman. In this toy version of the language of logic

---

158

---

THE HALTING THEOREM
there are elementary signs and variables. To follow Gödel's scheme, suppose there are the 10 logical signs shown in Table 4.4, each with its Gödel code number, an integer between 1 and 10.

In addition to the elementary signs, the language of the *Principia Mathematica* contains logical variables that are linked through the signs. These variables come in three different flavors, representing a kind of hierarchical ordering that depends upon the exact role the variable plays in the overall logical expression. Some variables are *numerical*, meaning that they can take on numerical values. For other variables we can substitute entire logical expressions or formulas (*sentential* variables). Finally, we have what are called *predicate* variables, which express properties of numbers or numerical expressions like *prime*, *odd*, or *less than*. All the logical expressions and provability relations in *Principia Mathematica* can be written using combinations of these three types of variables, connecting them via the logical signs. For our streamlined version of *Principia*, there are only 10 logical signs, although in the real case there are quite a few more. In this scaled-down version of *Principia Mathematica*, Gödel's numbering system would code numerical variables by prime numbers greater than 10, sentential variables by the squares of primes greater than 10, and predicate variables by the cubes of primes greater than 10.

To see how this numbering process works, consider the logical formula \((\exists x)(x = sy)\), which translated into plain English reads: "There exists a number \(x\) that is the immediate successor of the number \(y\)." Since \(x\) and \(y\) are numerical variables, the Gödel coding rules dictate that we make the assignment \(x \rightarrow 11\), \(y \rightarrow 13\), since 11 and 13 are the first two prime numbers larger than 10. The other symbols in the formula can be coded by substituting numbers using the correspondences in Table 4.4. Carrying out this coding yields the sequence of numbers \(8, 4, 11, 9, 8, 11, 5, 7, 13, 9\), formed by reading the logical expression symbol by symbol and substituting the appropriate number according to the coding rule. This sequence of 10 numbers pins down the logical formula uniquely. But since number theory—that is, arithmetic—is about the properties of single numbers, not sequences of numbers, one would like to be able to represent the formula in an unambiguous way by a single number. Gödel's procedure for doing this is to take the first 10 prime numbers (since there are 10 symbols in the formula) and multiply them together, each prime number being raised to a power equal to the Gödel number of the corresponding element in the formula. Since the first 10 prime numbers in order are 2, 3, 5, 7, 11, 13, 17, 19, 23, and 29, we make the substitutions \((\rightarrow 2^8, 3 \rightarrow 3^4, x \rightarrow 5^1, \text{and so on. The final Gödel number for the above formula is then})\)

\[(\exists x)(x = sy) \rightarrow 2^8 \times 3^4 \times 5^{11} \times 7^9 \times 11^{8} \times 13^{11} \times 17^5 \times 19^7 \times 23^5 \times 29^9.\]

Using this kind of numbering scheme, Gödel was able to attach a unique number to each and every statement and sequence of statements about arithmetic that could be expressed in the logical language of *Principia Mathematica*. So with this scheme, every possible proposition about the natural numbers can itself be expressed as a number, thereby opening up the possibility of using arithmetic to examine its own truths.

Deep insight and profound results necessarily involve seeing the connection linking several ideas at once. In the proof of Gödel's Theorem there are two crucial notions that Gödel had to juggle simultaneously, Gödel numbering being the first. Now for the second Big Idea, the substitution of the notion of a mathematical proof for the everyday concept of truth, together with the translation of a verbally stated logical paradox into an arithmetic statement.

Logical paradoxes of the sort that worried Hilbert are all based on the notion of self-reference, that is, they all involve statements that refer to themselves. The granddaddy of all such conundrums is the so-called Liar Paradox, one version of which is

This sentence is false.
Now why is this a paradox? Well, in everyday parlance for something to be "false" means that it does not correspond to reality. The sentence says that it is false. If that assertion does not correspond to reality, then the sentence must be true. On the other hand, if the sentence is true, this means that what it says does correspond to reality. But this true sentence says that it is false. Therefore, the sentence must indeed be false. So whether you assume that the sentence is true or that it's false, you are forced into concluding the opposite! Thus, the Paradox of the Liar.

What Gödel wanted to do was find a way to express such paradoxical self-referential statements within the framework of arithmetic. He needed such a statement in order to display an exception to Hilbert's thesis that all true assertions should be provable in a formal system. However, a statement like the Liar Paradox involves the notion of truth, something that logician Alfred Tarski had shown earlier could not be captured within the confines of a formal system. Enter Gödel's Big Idea #2.

Instead of dealing with the eternally slippery notion of truth, Gödel had the insight to think of replacing "truth" by something that is formalizable: the notion of provability. Thus, he modified the Liar Paradox above into the Gödel sentence

This statement is not provable.

This sentence, of course, is a self-referential claim about a particular "statement," the statement mentioned in the sentence. However, by his numbering scheme Gödel was able to code this assertion by a corresponding self-referential, metamathematical statement expressed in the language of arithmetic itself. Let's follow through the logical consequences of this mirroring.

If it turns out that the statement referred to is provable, then by Gödel's equating of truth with proof the statement must be true. Therefore, what it says must be true. But what it says is that it is not provable. Consequently, the statement and its negation are both provable, implying an inconsistency in our logical scheme of proof. On the other hand, if the statement referred to is not provable, then what it asserts is indeed the case, that is, the statement is true, but unprovable. Thus, there is a true statement that is not provable, implying that the formal system we are using for proving statements is incomplete.

Remember what Gödel showed was how to translate this verbal self-referential statement into an equivalent statement within the formal system accepted by mathematicians for proving statements of arithmetic. This means that the logical conclusions we have just drawn about inconsistency and incompleteness apply to the entire mathematical apparatus of numbers. Thus, if the formal system used for arithmetic is consistent, then it must necessarily be incomplete.

Gödel was able to show that for any consistent formal system powerful enough to allow us to express all statements of ordinary arithmetic, such a Gödel sentence must exist; consequently, the formalization must be incomplete. The bottom line then turns out to be that in every consistent formal system powerful enough to express all relationships among the whole numbers, there exists a statement that cannot be proved using the rules of the system. Nevertheless, that statement represents a true assertion about numbers, one that we can see is true by jumping outside of the system. Almost as an aside, Gödel also showed how to construct an arithmetical statement $A$, which translates into the metamathematical claim "arithmetic is consistent." He then demonstrated that the statement $A$ is not provable, implying that the consistency of arithmetic cannot be established by using any formal system representing arithmetic itself. Putting all these different notions together, we finally arrive at the following theorem.

**Gödel's Theorem—Formal Logic Version**

For every consistent formalization of arithmetic, there exist arithmetic truths that are not provable within that formal system. □

Since the steps leading up to Gödel's startling conclusions are both logically tricky and intricately intertwined, let me summarize the principal landmarks along the road in Table 4.5.

The proverbial perceptive reader will by now have noticed the striking similarity between the results of Turing and Gödel. But for those who haven't, let me spell out this parallel more explicitly. Here are restatements of both results that capture the distilled essence of the two theorems:
**Table 4.5** The main steps in Gödel’s proof.

<table>
<thead>
<tr>
<th><strong>Gödel Numbering:</strong></th>
<th>Development of a coding scheme to translate every logical formula and proof sequence in <em>Principia Mathematica</em> into a “mirror image” statement about the natural numbers.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Liar Paradox:</strong></td>
<td>Replace the notion of “truth” with that of “provability,” thereby translating the Liar Paradox into the assertion “This statement is unprovable.”</td>
</tr>
<tr>
<td><strong>Gödel Sentence:</strong></td>
<td>Show that the sentence “This statement is unprovable” has an arithmetical counterpart, its Gödel sentence ( G ), in every conceivable formalization of arithmetic.</td>
</tr>
<tr>
<td><strong>Incompleteness:</strong></td>
<td>Prove that the Gödel sentence ( G ) must be true, but unprovable, if the formal system is consistent.</td>
</tr>
<tr>
<td><strong>No Escape Clause:</strong></td>
<td>Prove that even if additional axioms are added to form a new system in which ( G ) is then provable, the new system with the additional axioms will have its own unprovable Gödel sentence.</td>
</tr>
<tr>
<td><strong>Consistency:</strong></td>
<td>Construct an arithmetical statement ( A ) asserting that “arithmetic is consistent.” Prove that this arithmetical statement is not provable, thus showing that arithmetic as a formal system is too weak to prove its own consistency.</td>
</tr>
</tbody>
</table>

---

**GÖDEL’S THEOREM** For any consistent formal system \( \mathcal{F} \) purporting to settle, that is, prove or disprove, all statements of arithmetic, there exists an arithmetical proposition that can be neither proved nor disproved in this system. Therefore, the formal system \( \mathcal{F} \) is incomplete. ■

**THE HALTING THEOREM** For any Turing machine program \( \mathcal{H} \) purporting to settle the halting or nonhalting of all Turing machine programs, there exists a program \( \mathcal{P} \) and input data \( I \) such that the program \( \mathcal{H} \) cannot determine whether or not \( \mathcal{P} \) will halt when processing the data \( I \). ■

When placed side by side in this fashion, it becomes fairly evident, I think, that the Halting Theorem is simply Gödel’s Theorem expressed in terms of computing machines and programs instead of in the language of logical deductive systems.

Turing’s solution of the Halting Problem and the equivalence of the Halting Problem to Hilbert’s Decision Problem, together with the faithful correspondence between Turing machines and formal systems, allows us to conclude that there cannot exist a Turing machine program that will print out all the true statements of arithmetic.

Gödel’s results show that there are statements about numbers that we can see to be true—yet which cannot be proved by following a chain of logical reasoning. Put another way, no single set of rules will ever “fence in” all possible true statements about numbers; truth is strictly bigger than proof. Some philosophers have taken this to mean that the power of the human mind somehow transcends the power of deductive reasoning. It’s but a small step from here to conclude that we will never create a computing machine with powers equal to the human mind, since computing machines are completely equivalent in the truths they can generate to those that can be obtained by following the rules of a formal logical system. Let’s take a harder look at this Gödelian argument against machines having mental states and thinking just like you and me.

---

**Manufactured Minds**

In 1950, Alan Turing published the paper “Computing Machinery and Intelligence,” which sparked off a debate that rages to this day over the question: Can a machine think? In addition to its pivotal role in drawing attention to the matter of machine intelligence, Turing’s paper was notable for its introduction of an operational test for deciding whether or not a machine really was thinking—human style. This criterion, now termed the *Turing Test*, is unabashedly behavioristic in nature, involving the machine’s fooling a human interrogator into thinking it is actually a human solely on the basis of what the machine does, ignoring completely the machine’s material structure and, in particular, how it does what it does. Turing’s rationale for proposing what he called the “Imitation Game” was that the only way we have for deciding whether or not other humans are thinking is to observe their behavior. If this criterion is good enough to decide if humans are thinking, then fairness to machines dictates that it’s good enough for them, too.
On November 8, 1991, the Boston Computer Museum held the world's first hands-on Turing Test, in which eight programs conversed with human inquisitors on a restricted range of topics that included women's clothing, romantic relationships, and Burgundy wines. At the day's end, the judges awarded first prize to a program called PC Therapist III, which was designed to engage its questioner in a whimsical conversation about nothing in particular. To illustrate the kind of dialogue such programs generate, here is a snippet of interchange between another contestant, the program Good Mood, which is designed to serve as an electronic psychoanalyst, and one of the interrogators. Good Mood begins by saying to the interrogator: "Perhaps you're not getting enough affection from your partner in the relationship." The interrogator replies, "What are the key elements that are important in relationships in order to prevent conflict or problems?" "I think you don't think I think," responded the machine.

This kind of interchange did little to fool the judges, most of whom said they were able to spot the mistakes rooted in a lack of everyday common sense that immediately singled out the computer programs from the responses displayed on the terminals controlled by the humans. Nevertheless, the overall conclusion from this historic experiment was that perhaps the Turing Test isn't as difficult as many people originally thought, since even the primitive programs in this contest managed to fool some of the judges most of the time. Of course, we should keep in mind that this wasn't a true Turing Test, since the domains of discourse were severely restricted. But it was still a pretty good initial approximation.

The Turing Test represents a third-party perspective on human intelligence, involving one standing outside the system and discerning humanlike intelligence in a machine by observing only the machine's behavioral output. The test says nothing about the internal constitution of the machine, how its program is structured, the architecture of the processing unit, or its material composition. In Turing's view of intelligence, only externally observed behavior counts. And if you behave like us, then you are a thinking machine.

In 1989, theoretical physicist Roger Penrose published The Emperor's New Mind, a book whose central argument is that the human mind is capable of transcending rational thought, hence can never be duplicated in a machine. Before going on, let me note that we are using the term rational thought in the strong sense of following rules or an algorithm to arrive at a result by a process of logical deductive inference. There is no connection here with the everyday, economic interpretation of rationality as relating to self-interest or prudent action. So what Penrose argues is that at least some human thought processes do not involve following any kind of rule. He justified this claim by a wildly speculative appeal to quantum processes in the human brain as the basis of consciousness and intelligence. A key ingredient in Penrose's argument is Gödel's result showing that there are true statements of arithmetic that the human mind can know but that cannot be the end result of following a fixed set of rules (that is, a computer program). Here Penrose is reviving a line of attack originally put forward by Oxford philosopher John Lucas in 1961. To understand a little bit better the relationship between Gödel's Theorem and the possible limits to machines ever displaying humanlike intelligence, let's take a little harder look at the chain of reasoning underlying the claims of both Lucas and Penrose.

Gödel's theorems show that any reasonably rich formal system is incomplete and that the consistency of such a system cannot be proved within the system itself. Furthermore, in Turing's work we saw that formal systems and machines are equivalent in what they can do by way of producing logically valid conclusions from given assumptions. Ergo, computers are subject to the same limitations that Gödel imposed on any formal system. Lucas and Penrose then use this fact to jump to the conclusion that machines are inherently limited in what they can do and, in particular, there are statements that the mind knows to be true but that the machine cannot prove. Interestingly enough, Turing anticipated this kind of objection to machine intelligence in his classic 1950 paper on thinking machines, in which he replied that people may well be subject to similar limitations. But John Lucas wasn't convinced by Turing's response, and in his 1961 paper titled "Minds, Machines, and Gödel" he attempted to strengthen the Gödelian argument against the view that the mind is a machine.

The heart of both Lucas's and Penrose's arguments takes the following course. By standing outside the incomplete, consistent formal system, Gödel's results imply that humans can know that there exists some true, but unprovable, statement. But the machine cannot prove this fact; hence, a human can beat every machine since such a true, but unprovable, statement exists for every machine. Furthermore, if the human mind were nothing more than a formal system, by Gödel's second
FIVE GOLDEN RULES

Theorem the mind could not prove its own consistency. But humans do assert their own consistency. Consequently, the mind must be more than a machine.

As with virtually all philosophical debates, the arguments against Lucas hinge upon the precise meaning he gives to terms like machine, consistency, and mind, as well as the hidden assumptions underpinning his conclusions. For example, Paul Benacerraf points out that Lucas has too limited a view of machines, since any machine that could reprogram itself would be exempt from the Gödel argument. Furthermore, it is also noted that Lucas assumes that mind is consistent. In fact, this is far from obvious, as the following paradox constructed by C. H. Whitley shows.

Consider the sentence “Lucas cannot consistently assert this sentence.” Lucas cannot assert the truth of this sentence even though he can clearly see that it’s true. Why? Because if Lucas could assert it, then that fact would undermine his assumed consistency. Thus, either there is something that Lucas can see to be true but can’t assert, or he is inconsistent. Whitley concludes that Lucas holds too high a regard for humans since even if there is an unprovable statement that a specific machine cannot assert, humans can’t always do it either.

Other arguments countering Lucas claim that he errs in his application of Gödel’s results. For instance, Gödel’s Theorem proves only that a machine \( M \) cannot prove the Gödel sentence of \( M \) from its axioms and according to its rules of inference. But the human mind cannot prove the Gödel sentence either, at least not by using the axioms and rules of inference available to the machine. Furthermore, Lucas doesn’t show that he can find a flaw in any machine, but only in any machine that the mechanist can construct. So in concluding this discussion of minds, machines, and artificial intelligence, it’s of some interest to listen to what Gödel himself had to say about the matter.

Unfortunately, Gödel was rather reclusive and secretive, especially in his later years, and his only published statement on the topic comes from a lecture delivered to the American Mathematical Society in 1951:

The human mind is incapable of formulating (or mechanizing) all its mathematical intuitions, i.e., if it has succeeded in formulating some of them, this very fact yields new intuitive knowledge, e.g., the consistency of this formalism. This fact may be called the “incompleteness” of mathematics. On the other hand, one, on the basis of what has been proved

THE HALTING THEOREM

so far, it remains possible that there may exist (and even be empirically discoverable) a theorem-proving machine which in fact is equivalent to mathematical intuition, but cannot be proved to be so, nor even be proved to yield only correct theorems of finitary number theory.

Thus, Gödel leaves open the possibility of the existence of a theorem-proving machine, and even concedes that it may be possible to discover such a machine by empirical investigation. Thus, he says that there could exist a machine whose abilities equaled human mathematical intuition, but whose program we could never understand. Nonetheless, we would be able to set up conditions leading to the existence of such a machine, for example, by evolution. So, machines too complex to design could nevertheless exist.

With this statement about programs too complex for the human mind to design or understand, Gödel calls for us to face squarely the issue of how to characterize and measure the complexity of a computer program. This investigation has led to new and deep insights into Gödel’s original results about the limitations of mechanical reasoning. These new results, in turn, suggest a variety of questions about the limits of science itself as a way of getting at the “scheme of things” in a real-world sense. So let’s turn our attention to a consideration of how “complex” things can get before human reasoning comes up against its limits.

Omega Is the End

Some years ago, Scientific American columnist Martin Gardner introduced a distinction between “dull” and “interesting” numbers. Interesting numbers, according to Gardner, are those that have some peculiar pattern or property that separates them from all other numbers. Dull numbers, on the other hand, are all those numbers that are not interesting. Gardner then went on to show the paradoxical nature of this dichotomy, by showing that dull numbers cannot possibly exist. His argument was to first list the integers in order, letting \( D \) stand for the first dull number on the list. But the very fact that \( D \) is the first dull number makes it interesting! Therefore, there can be no dull numbers.

One way to break out of this kind of paradoxical circle is to define a number to be interesting if it can be computed by a program of shorter length (that is, fewer bits) than the number itself. Such a short
program would then encapsulate some special feature of the number through which the number could be distinguished from the general run of numbers. By way of contrast, dull numbers would be those that are algorithmically incompressible in the sense that they contain no pattern that could be exploited to reduce the size of their minimal program. It's reasonable to term such "patternless" numbers random. Clearly, most numbers are random in this sense, since for any n there are more than twice as many numbers of length no greater than n bits as there are numbers no longer than n - 1 bits to serve as shorter descriptions. This is because there are exactly \(1 + 2 + 4 + \cdots + 2^n\) numbers less than or equal to n bits in length. So the ratio of numbers no more than n bits in length to those no longer than n - 1 bits is

\[
\frac{1 + 2 + 4 + \cdots + 2^n}{1 + 2 + 4 + \cdots + 2^{n-1}} = 2 + \frac{1}{2^n - 1} > 2.
\]

The idea of using the length of the shortest computer program to characterize the complexity of a number was introduced independently by Gregory Chaitin and Andrei Kolmogorov in the 1960s. Chaitin later used the idea to prove the following version of Gödel’s Theorem:

**Gödel’s Theorem—Complexity Version** Although almost all numbers are random, there is no formal axiomatic system that will allow us to prove this fact.

We can express this remarkable result more explicitly as follows. Suppose we have a formal system whose axioms and rules of inference require n bits to describe, that is, the Turing machine program for this system is n bits in length. Then this system cannot prove the randomness of any number longer than n bits. Why? Well, suppose there was a proof in this system establishing the randomness of a number that is substantially longer than n bits. Then we would have an n-bit program that could print out this random number. But, by definition, the randomness of the number means that there cannot exist a program shorter than the number in question that can produce that number. Thus, we have a contradiction, showing that such a program cannot exist after all. What this adds up to is a mathematical proof of the rather com-

monsense fact that you can't get more information out of a system than you put into it. As Georgia Tech physicist Joseph Ford once put it, "A 10-pound theory can no more generate a 20-pound theorem than a 100-pound pregnant woman can birth a 200-pound child." (Here, of course, Ford is referring to the weight of the woman while carrying her unborn child.)

**The Halting Probability**

Earlier, we gave examples of uncomputable quantities in terms of the Busy Beaver Function and the Turing Machine Game. Chaitin has exploited the notion of algorithmic complexity in order to produce an even more dramatic example of such a number, something that he calls \(\Omega\) ("omega"). It is closely related to the Halting Problem discussed above. Chaitin defines \(\Omega\) to be the probability that a randomly generated program for a universal Turing machine will halt. Here when we say that the program is generated "randomly," it means that whenever the computer asks for another bit of input, we simply toss a fair coin and give the computer a 1, say, if the coin comes up heads and a 0 if the coin reads tails. Since Table 4.1 has shown us that every Turing machine program can be coded as a binary string, this procedure makes perfectly good sense. Now suppose that we generate a large number of such random programs, each statistically independent of the others, and run each of them for, say, 1 million steps. The ratio of those programs that stop before 1 million steps to those that don't then constitutes a lower bound on the number \(\Omega\). If one lets the number of steps increase from 1 million to infinity, this ratio will then converge exactly to \(\Omega\).

The quantity \(\Omega\) is a perfectly well-defined number between 0 and 1, once the specific language for the UTM has been specified. For the types of programming languages we are familiar with, such as Fortran, C, or Pascal, \(\Omega\) is likely to be close to 1, since a program generated at random in one of these languages is far likelier to crash immediately due to a syntax error than to go into an endless loop. Nevertheless, it can be shown that after the first few digits \(\Omega\) would look very random indeed. What makes \(\Omega\) interesting is that the theory of computation goes is that it encodes the Halting Problem is a very compact form. For example, knowing the first n bits of \(\Omega\) enables us to solve the halting problem for any program up to n bits in length. Here's how.
Suppose we want to solve the Halting Problem for a particular randomly generated \( n \)-bit program \( P \). The program \( P \) corresponds to a specific sequence of \( n \) coin tosses, having probability \( 2^{-n} \). If \( P \) halts, this much probability is then contributed to the total halting probability \( \Omega \). Now let \( \Omega_n \) be the known first \( n \) bits of \( \Omega \). This means that \( \Omega \) is larger than \( \Omega_n \) and smaller than \( \Omega_n + 2^{-n} \). Thus, in order to decide whether or not the program \( P \) halts, we begin a systematic search for all programs that halt—regardless of their length. We do this by first running one program and then another for longer and longer times until we have seen enough halting programs to account for more than a fraction \( \Omega_n \) of the total probability. Figure 4.5 illustrates the general idea. Then either \( P \) is among the programs that have halted so far or it will never halt. This is because its halting would drive up the total probability beyond its known upper bound of \( \Omega_n + 2^{-n} \).

![Diagram](image)

**Figure 4.5** Using \( \Omega \) to solve the Halting Problem.

Not only can knowledge of \( \Omega \) be used to solve the Halting Problem, but it can also be used to settle many of the famous unproven conjectures in mathematics. For example, the famous Riemann Hypothesis, asserting that all the zeros of the Riemann zeta function lie on the line \( \text{Re } z = \frac{1}{2} \) in the complex plane, is equivalent to the assertion that some program, which searches systematically for zeros off the magical line, will never halt. Conjectures of this sort are usually simple enough that they can be encoded in the halting of small programs, generally not longer than a few thousand bits. Consequently, we could resolve all these conjectures if only we knew the first few thousand bits of \( \Omega \).

This whole line of argument can be extended to classes of statements of the form: “Proposition \( P \) is provable in formal system \( \mathcal{F} \).” If we suppose that proposition \( P \) and system \( \mathcal{F} \) together require \( n \) bits to describe, then there is a certain program of about \( n \) bits in length that will halt if and only if \( P \) is provable in \( \mathcal{F} \). Thus, for any proposition \( P \) and system \( \mathcal{F} \) simple enough to be comprehensible by the human mind, the first few thousand bits of \( \Omega \) are sufficient to decide whether (1) \( P \) is provable in \( \mathcal{F} \), (2) not-\( P \) is provable in \( \mathcal{F} \), or (3) \( P \) is independent of the axioms of \( \mathcal{F} \).

At this point in our narrative, it will probably strike the reader as odd—or even paradoxical—that \( \Omega \) can contain so much information about the Halting Problem and yet be computationally indistinguishable from the type of random sequence one gets by tossing a coin. The explanation for this oddity is as straightforward as it is remarkable. The fact is that \( \Omega \) is an absolutely informative message, in the sense that all redundancy has been squeezed out of it. So it just appears random because it hasn’t got a single bit of “fat” left in it. This means that \( \Omega \) is pure information.

The fact that \( \Omega \) contains the answer to almost any question that one can pose has moved IBM physicist Charles Bennett to call \( \Omega \) a “cabalistic” number. It can be known of, but not known, through human reason. As Bennett remarks,

> It embodies an enormous amount of wisdom in a very small space, inasmuch as its first few thousand digits ... contain the answers to more mathematical questions than could be written down in the entire universe, including all finitely-refutable conjectures. Its wisdom is useless precisely *because* it is universal; the only known way of extracting the
solution to one halting problem from $\Omega$ ... is by embarking on a vast computation that would at the same time yield solutions to all other equally simply-stated halting problems .... Ironically, although $\Omega$ cannot be computed, it might accidentally be generated by a random process, for example, a series of coin tosses, or an avalanche that left its digits spelled out in the pattern of boulders on a mountainside. The initial few digits of $\Omega$ are thus probably already recorded somewhere in the universe. Unfortunately, no mortal discoverer of this treasure could verify its authenticity or make practical use of it.

On this bittersweet note, let’s now turn our attention from cabalistic quantities that can be known about but never computed to quantities that can be computed, in principle, but that will probably never be known. Thus far, we have focused on the distinction between quantities that are computable—given an infinite amount of time and storage—and those quantities like the Busy Beaver function or $\Omega$ that are logically uncomputable even with unlimited computational resources at our disposal. This distinction is of great theoretical interest; however, from a practical point of view it may well turn out that there are quantities that are theoretically computable by the Turing machine criterion, but that would take an length of time greater than the age of the universe for even the fastest of supercomputers to produce. We devote the remainder of the chapter to an account of this question of computational intractability.

**Tough Times**

A famous problem of recreational mathematics is the so-called Tower of Hanoi. In this problem, there are three pegs $A$, $B$, and $C$, with $N$ rings of decreasing radii piled on the first peg $A$. The other two pegs are initially empty. The task is to transfer the rings from $A$ to $B$, perhaps using peg $C$ in the process. The rules stipulate that the rings are to be moved one at a time, and that a ring can never be placed upon one smaller than itself. Figure 4.6 shows the problem for the case of $N = 3$ rings.

In this case of three rings, it’s not too hard to see that the sequence of seven moves

\[
A \rightarrow B, \quad A \rightarrow C, \quad B \rightarrow C, \quad A \rightarrow B, \\
C \rightarrow A, \quad C \rightarrow B, \quad A \rightarrow B
\]

achieves the desired transfer of rings. And, in fact, it can be shown that there is a general algorithm, that is, a program, solving the game for any number of rings $n$. This program shows that the minimal number of transfers required is $2^n - 1$. Amazingly, the original version of this puzzle, dating back to ancient Tibet, involves $n = 64$ rings. So it’s not hard to see why the Tibetan monks who originated the game claim that the world will end when all 64 rings are correctly piled on peg $B$. To carry out the required $2^{64} - 1$ steps, even performing one ring transfer every 10 seconds, would take well over 5 trillion years! Thus, the number of steps needed for the solution of the Tower of Hanoi problem grows exponentially with the number of rings $n$. This is an example of a “hard” computational problem—one in which the number of computational steps needed to obtain a solution increases exponentially with the “size” of the problem.

By way of contrast, a computationally “easy” problem is the sorting of a deck of playing cards into the four suits in ascending order. First go through the deck until you find the ace of spades. Set it aside and then go through the remaining cards until you find the two of spades, which you also set aside. As one continues in this fashion, the deck is fully sorted. The worst that can happen with this sorting scheme is that the ace of spades is the last card in the unsorted deck, the two of spades is the next-to-last card, and so on. So starting with $n$ cards, you would have to examine at most $n^2$ cards. Thus, the number of steps needed to completely sort the deck is a quadratic function of the size of the problem, that is, the number of cards in the deck.

In the mid-1960s, J. Edmonds and A. Cobham introduced the idea of classifying the computational difficulty of a problem according to whether or not there exists an algorithm for solving the problem that requires at most a polynomial number of steps in the size of the problem. “Easy” problems can be solved in polynomial time; “hard” problems,