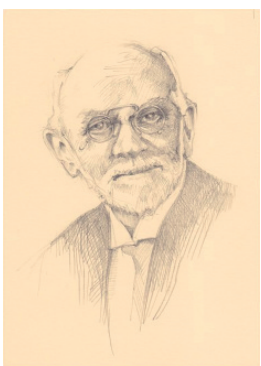
6 Mr. Turing's amazing machines

> Electronic computers are intended to carry out any definite rule of thumb process which could have been done by a human operator working in a disciplined but unintelligent manner.
>
> Alan Turing[1]

WARNING: This chapter is more mathematical in character than the rest of the book. It may therefore be hard going for some readers, and they are strongly advised either to skip or skim through this chapter and proceed to the next chapter. This chapter describes the theoretical basis for much of formal computer science.

Hilbert's challenge

Are there limits to what we can, in principle, compute? If we build a big enough computer, surely it can compute anything we want it to? Or are there some questions that computers can never answer, no matter how big and powerful a computer we build. These fundamental questions for computer science were being addressed long before computers were built!

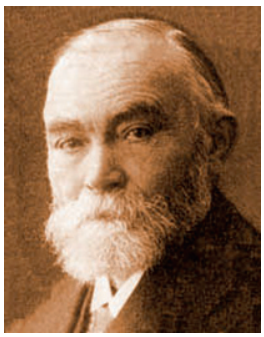


B.6.1. David Hilbert (1862–1943) believed that in mathematics all problems could be solved, provided we work hard enough. The writing on his tombstone reads "Wir müssen wissen, wir werden wissen" – or in English "We must know, we will know."

In the early part of the twentieth century, mathematicians were struggling to come to terms with many new concepts including the theory of infinite numbers and the puzzling paradoxes of set theory. The great German mathematician David Hilbert (B.6.1) had put forward this challenge to the mathematics community: put mathematics on a consistent logical foundation. It is now difficult to imagine, but in the early twentieth century, mathematics was in as great a turmoil as physics was at that time. In physics, the new theories of relativity and quantum mechanics were overturning all our classical assumptions about nature. What was happening in mathematics that could be comparable to these revolutions in physics?

In the late nineteenth century, mathematics was becoming liberated from its traditional role in just having application to counting and measurement. In high school algebra, letters started to be used as symbols for numerical quantities. By the twentieth century, a more abstract view had emerged in which an "obvious" numerical rule of algebra such as $x + y = y + x$, was taken to be just a rule about how symbols could be moved around, and not necessarily to be interpreted in terms of numbers. Hilbert was one of the leaders of the formalistic approach to mathematics, which transformed mathematics into a

more abstract formulation that allowed the exploration of new kinds of algebra with different rules and symbols. This led to the development of new fields of research such as "group theory" and "Hilbert spaces," both of which ultimately found application in physics. However, attempts to show that the whole edifice of this more formal approach to mathematics was consistent and free from contradictions had run into trouble. The key challenge was to show that manipulating the symbols according to the agreed rules always made sense and did not lead to contradictions such as being able to prove that 2 + 2 = 5.



B.6.2. Gottlob Frege (1848–1925) was working on the axiomatization of mathematics trying to derive a logical system that is complete and has no contradictions. His pioneering work had a significant impact on later discoveries in mathematics.

The German logician Gottlob Frege (B.6.2) and the Welsh mathematician Bertrand Russell (B.6.3) had independently approached the problem of proving the consistency of mathematics using the ideas of set theory. A set is just a collection of objects characterized by a particular property. For example, we can define a set that contains all the men in a town. We can also define a subset of this set that comprises all the men in the town with red hair, and so on. However, in 1901, Russell noticed that logical contradictions arose when he tried to use "sets of all sets" in his arguments. He explained his paradox with the example of a town that had only one male barber and where all the men were clean shaven (Fig. 6.1). The paradox can then be stated as: *The barber in this town shaves only men who do not shave themselves.* In this case, we have the set of all men in the town, and within this set there are two subsets – men who shave themselves and men who are shaved by the barber. The paradox is into which subset do we put the barber? Since these sets were taken to be abstract entities there was no way of resolving the contradiction by asking what the symbols really meant. The whole idea of Frege and Russell's program was "to derive arithmetic from the most primitive logical ideas in an automatic, watertight, depersonalized way."[2] Russell wrote a letter about this paradox to Frege in 1902 and Frege's reply gives some idea of the consternation that Russell's letter caused:

B.6.3. Bertrand Russell (1872–1970) mathematician and philosopher. With Alfred Whitehead in 1910 he published *Principia Mathematica* with more than a thousand pages in which they tried to put mathematics on solid foundations. They soon learned that mathematics is not "perfect" because there are many paradoxes that mathematics cannot answer.

> Your discovery of the contradiction has surprised me beyond words and, I should almost like to say, left me thunderstruck, because it has rocked the ground on which I meant to build arithmetic. Your discovery is at any rate a very remarkable one and it may perhaps lead to a great advance in logic, undesirable as it may seem at first sight.[3]

By 1928, the focus had moved on from Frege and Russell's ambitious attempts to determine what mathematics really was. Instead, Hilbert was asking deep questions about the logical foundations of mathematics. In 1899 he had succeeded in finding a set of axioms – a small number of self-evident truths – from which he could prove all the theorems of Euclidean geometry without any need to relate his proofs to the geometry of the actual physical world. A year later, at a conference in Paris, Hilbert proposed a list of twenty-three important unsolved mathematical problems. The number two question on his list was "the compatibility of the arithmetical axioms." Of the many questions that could be asked about these axioms, he said that the most important question was:

> To prove that they are not contradictory, that is, that a definite number of logical steps based upon them can never lead to contradictory results.[4]

Fig. 6.1. Who is going to shave the barber? Illustration of barber's paradox.

Many years later, at the Bologna International Conference of Mathematicians in 1928, Hilbert made this question more precise. In Andrew Hodges's words, Hilbert's three questions were the following:

> First, was mathematics complete, in the technical sense that every statement (such as "every integer is the sum of four squares") could either be proved or disproved? Second, was mathematics consistent, in the sense that the statement "2+2 = 5" could not be arrived at by a sequence of valid steps of proof? And thirdly, was mathematics decidable? By this he meant, did there exist a definite method which could, in principle, be applied to any assertion, and which was guaranteed to produce a correct decision as to whether that assertion was true.[5]

This last question came to be known as the decision problem – more often known by its more intimidating German name as the *Entscheidungsproblem*. Hilbert clearly believed that the answer to all of these questions would be "yes" – but within a few years, Kurt Gödel (B.6.4) had dealt Hilbert's newly announced program a fatal blow.

Gödel was known for being extremely meticulous – in his secondary school he was famous for never making a single grammatical error. He went to the University of Vienna in 1924 and wrote his famous paper on the incompleteness of mathematics in 1931. In that paper Gödel proved a startling result. He showed that mathematics must be incomplete – in that there are statements that can neither be proved nor disproved starting from a given set of axioms. In order to prove this result, Gödel first showed how the rules of procedure of any formal mathematical system, and the use of its axioms, could be encoded as purely arithmetical operations. Having done this, Gödel was able to reduce things like the property of "being a proof" or of "being provable" to arithmetic statements. He could then construct arithmetical statements that referred to themselves, rather like Russell's use of "sets of all sets" arguments. In particular, Gödel was able to construct a mathematical statement that effectively said "This statement is unprovable." The statement cannot be proved *true*, for this would immediately be a contradiction. But similarly, it cannot be proved *false*, because this also leads to a contradiction. This is reminiscent of the famous liar paradox: when a man says "I am lying," is he telling the truth or not?

Gödel was also able to show that arithmetic could not be proved to be consistent with just the use of its own axioms. In one remarkable paper, Gödel had answered the first two questions of Hilbert's program in the negative! This left

B.6.4. Kurt Gödel (1906–78) with Albert Einstein in Princeton. John von Neumann had a very high respect for Gödel and at Princeton he helped him to get a permanent position. Allegedly he said "How can any of us be called professor when Gödel is not?"[B1] Despite their age difference, Einstein and Gödel were close friends. They used to walk together to the Institute for Advanced Studies every morning and toward the end of his life, Einstein remarked that "his own work no longer meant much, that he came to the Institute merely to have the privilege of walking home with Gödel."[B2]

just the last question – Hilbert's *Entscheidungsproblem* (the decision problem). Enter Alan Turing and his amazing machines.

Turing machines

At the age of nineteen, Turing (B.6.5) went to King's College, Cambridge in 1931 to study mathematics. He passed his final mathematics examinations at Cambridge in 1934 with a distinction and was awarded the title of "Wrangler," a title still used at Cambridge to denote the top mathematics students each year. In the spring of 1935, Turing attended a lecture course given by Max Newman on "The Foundations of Mathematics." Unusual for mathematicians at Cambridge at that time, Newman was an expert in the emerging field of topology and had also followed the progress of mathematical logic and set theory since the efforts of Frege and Russell. In particular, Newman had attended the 1928 international congress at which Hilbert had announced his challenge to

B.6.5. Computer-generated image of Alan Turing. Alan Mathison Turing (1912–54) was one of the founders of computer science. His name is mainly associated with Turing machines, universality, the Church-Turing thesis, and artificial intelligence and the Turing Test. After attending Sherbourne "public school" – in England this means a private school – Turing went to King's College, Cambridge in 1931 to study mathematics. He was twenty-four when he wrote his groundbreaking paper, "On Computable Numbers, with an Application to the Entscheidungsproblem." Turing was a good long-distance runner – his best time for the marathon was only eleven minutes slower than the winning time at the 1948 Olympics – and, for recreation, he liked to go running in the countryside around Cambridge. He said later that he conceived the idea of how to answer Hilbert's third question while lying in the meadow at Grantchester, a village near Cambridge, at a break in one of his runs. It is no exaggeration to say that this paper is one of the cornerstones of computer science.

During the war he worked on code breaking at Bletchley Park, for which he was honored as an Officer of the British Empire. After the war, Turing returned to his ideas of building a physical realization of his abstract machine. At the U.K. National Physical laboratory (NPL) in 1945 he designed the Automatic Computing Engine (ACE) which could have been the first stored-program computer. Because of bureaucratic delays for the ACE project, Turing became frustrated and left NPL. In 1949 he started to work at the computing laboratory in Manchester where he developed programs for the Manchester Mark I computer.

The following year Turing published the paper "Computing Machinery and Intelligence" in which he speculated about whether computers can think. In this paper he described what has become known as the Turing test. This is a purely operational definition of intelligence. A human interrogator poses questions to a closed room containing either a computer or a person. If the interrogator is unable to tell which is the computer and which is the person from the responses the computer is deemed to have Turing's test for intelligence.

In 1952 homosexuality was still illegal in Britain and Turing was charged with committing a homosexual act by Manchester police. As an alternative to prison Turing opted for hormone therapy which had some unpleasant side effects. Turing died in 1954 after eating an apple containing cyanide; an inquest ruled his death to be suicide. Details of his school days, his foundational research on computability, his work on the German Enigma machine at Bletchley Park, and the tensions caused by his homosexuality are contained in a wonderful biography by Andrew Hodges, *Alan Turing: The Enigma of Intelligence*. In September 2009, after an Internet campaign, the then British prime minister Gordon Brown issued an official public apology "for the appalling way he [Turing] was treated." Finally on the 24 December 2013, Turing was given a posthumous pardon by the Queen.

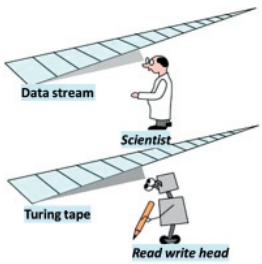


Fig. 6.2. Turing designed his machine to mimic the behavior of a human computer.

clarify the foundations of mathematics. Newman's 1935 lectures finished with an account of Gödel's theorem and made clear that the third of Hilbert's questions – the *Entscheidungsproblem* – remained unanswered. In his biography of Turing, Hodges highlights the question asked by Newman that started Turing on his journey to Turing machines in the following way:

> Was there a definite method, or as Newman put it, a *mechanical process* which could be applied to a mathematical statement, and which would come up with the answer as to whether it was provable?[6]

By using the phrase "mechanical process," Newman probably meant nothing more than an algorithm – a set of detailed instructions that leads to the solution of a problem. However, the phrase must have struck a chord with Turing. He decided to work on Hilbert's problem that summer but characteristically did not ask Newman for advice or even tell him about his intentions, nor did he read up on all the available research literature. This isolation from any of the accepted modes of thinking about the problem was undoubtedly one of the reasons that Turing followed such a strikingly unconventional approach to Hilbert's problem.

Turing equated the concept of "computability" with the ability of a very simple machine to perform a computation. His idea was to imagine a machine that worked like a human "computer" who just had to follow a set of rules (Fig. 6.2). Instead of a standard sheet of paper, Turing idealized his computer – who we shall take as female – as using a long strip of paper or "tape" on which to do her calculations. The tape is broken up into square "boxes," in each of which she can read or write a symbol. Using a tape with only one row of symbols to do the calculations – rather than a piece of paper that could accommodate multiple rows – would undoubtedly be a very tedious restriction for a real human computer but it is perfectly possible to arrange to perform the calculation in this way. Turing imagined that his human calculator would have a number of different "states of mind" that tell her how she should use the information that she reads in each box of the tape. Thus our female computer starts off in one specific state of mind and examines the content of one of the boxes on the tape. After reading the symbol in the box, she can overwrite the symbol in the box, change to a new state of mind, and move to consider the symbol in the next square – to the left or the right. It is her state of mind that tells her what to do with the symbol she has read – whether it should be used as part of the process of addition or of multiplication, for example. Turing envisaged that a human computer would need only a finite number of states of mind to complete any given calculation. Having broken down how a human would actually go about performing the calculation into these simple steps, Turing then proposed a very simple machine that could mimic all the actions of the human computer and so work through the algorithmic steps to complete the same calculation (Fig. 6.3).

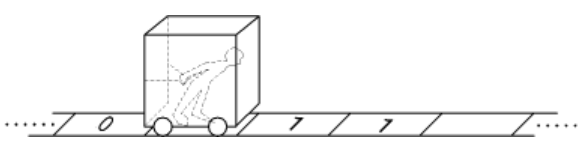


Fig. 6.3. This figure illustrates the concept of a Turing machine as "a human in a box." The box has no bottom so the human can read the symbol under the box.

To summarize, Turing machines were to be provided with paper in the form of a tape. The tape is marked off into boxes and each box can contain at most one symbol (Fig. 6.4). At each step of the algorithm, the head of this "super-typewriter" machine can move one space, to the adjacent box on the left or on the right. The paper tape is assumed to be unlimited in length so that although the machine has a finite number of symbols and states, it is allowed an unlimited space for its calculations. This is not to say that the amount of paper attached to such a machine actually is infinite. At any given stage in any calculation the length of tape will be finite but we have the option of adding more tape when we need to. Turing's machine is therefore able to read and write, move left or right along the tape, as specified by its set of states. The action of the machine is simple: it starts off in a certain state and looks at the contents of the first box. Depending on the state and the box contents, it will either erase the contents of the box and write something new, or leave the box as it is. Whatever it does, it next moves one box to the left or the right and changes to a new internal state. A simple example of a "Parity Counter" Turing Machine – which determines if the number of 1s or 0s in a binary string is even or odd – is described in detail at the end of this chapter.

Computable numbers and computability

With this simple machine, Turing was able to define what was meant by "computability." To illustrate this we shall look at the question of "computable numbers." We begin by defining what we mean by the term *real numbers*.

The *natural numbers* are the whole numbers (no fractions or decimal points) starting from zero:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 …

Natural numbers can be added or multiplied together to produce new natural numbers. If we want to allow for subtraction, however, we need to include negative numbers as well. We define the *integers* as:

… , –6, –5, –4, –3, –2, –1, 0, 1, 2, 3, 4, 5, 6, 7 …

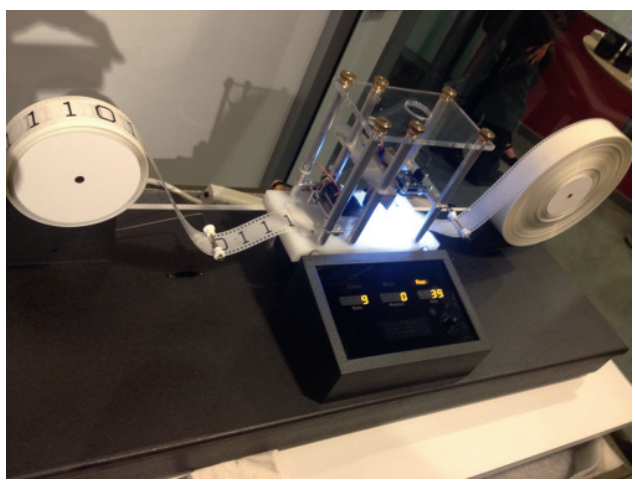


Fig. 6.4. The photograph shows a working model of a Turing machine.

If we also want to include division, the integers are still too limited. We need to include fractions or *rational numbers*:

$$0, 1, -1, 1/2, -1/2, 2, -2, 3/2, -3/2, 1/3, -1/3, \ldots$$

The rational numbers are nice and neat but they leave out important quantities such as π or $\sqrt{2}$. These are called *irrational numbers* because they cannot be expressed as integers or fractions of integers. Both π and $\sqrt{2}$ can only be expressed as infinite series – as a sum of an infinite number of terms. In practice, for a useful approximate value of π or $\sqrt{2}$, we need only to sum up a few terms of the series. For example, for π, we could use the so-called Gregory-Leibniz expansion:

$$\pi = 4(1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + \ldots)$$

and for $\sqrt{2}$, we could use the Taylor expansion

$$\sqrt{2} = 1 + 1/2 - 1/(2 \times 4) + (1\times3)/(2\times4\times6) - (1\times3\times5)/(2\times4\times6\times8) + \ldots$$

There are many other methods for calculating π and for taking square roots. All these methods lead to the well-known decimal approximations for π and $\sqrt{2}$

$$\pi = 3.14159265 \ldots$$

and

$$\sqrt{2} = 1.41421356\ldots$$

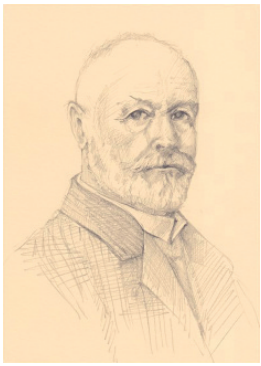
In the struggle to understand what could and could not be proved, the question arose of what numbers could be calculated. This led to the concept of an "effective procedure" – a set of rules telling you, step-by-step, what to do to complete a calculation. In other words, if there is an effective procedure for some computational problem it means that there is an algorithm that can be executed to solve the problem. These methods for calculating π and $\sqrt{2}$ are examples of effective procedures. They may not be the most efficient way to calculate π or $\sqrt{2}$ but these algorithms will work and will produce an answer.

The number system that includes irrational numbers like these is the system of *real numbers.* In everyday life, we use approximations to real numbers and do our calculations accurate to a specific number of decimal places.

How many real numbers are there? Georg Cantor (B.6.6), who developed the theory of infinite numbers in the late 1800s, showed that the number of integers is the same as the number of natural numbers. He did this by setting up a one-to-one correspondence as follows:

Integers	0	–1	1	–2	2	–3	3	–4	…
Natural numbers	0	1	2	3	4	5	6	7	…

Although it may seem that there are more integers than natural numbers, Cantor showed that the integers could in principle be counted off against the natural numbers in this way. Although both were infinite, the existence of such



B.6.6. The name of Georg Cantor (1845–1918) is associated with set theory and with tackling the problem of infinity in a mathematically rigorous way. Cantor came to the conclusion that the infinite set of real numbers is larger than the infinite set of natural numbers. Furthermore, he was able to show that there is an infinite number of infinities. Cantor's ideas met with considerable resistance from fellow mathematicians. The great German mathematician, David Hilbert, was an exception and was early to recognize the significance of Cantor's work. Hilbert later said: "No one shall expel us from the Paradise that Cantor has created."[B3]

a one-to-one correspondence in Cantor's theory of infinities establishes that in a technical sense the number of objects in the top row is the *same* as the number of objects in the bottom row. Thus the number of integers is the same as the number of natural numbers, although both are infinite. Sets that can be put into one-to-one correspondence with the natural numbers are said to be "countable." Similarly, we can arrange for all the rational numbers to be in a one-to-one correspondence with the natural numbers so they, too, are countably infinite. But what about the real numbers? Here the situation is very different and Cantor proved this using his "diagonal slash" method that was later used by both Gödel and Turing. Using this technique Cantor was able to show that the number of real numbers must be greater than the number of natural numbers and is therefore not countable.

Let us see how the technique works. We begin by assuming the opposite – namely that we can pair off the real numbers with the natural numbers in some way. We make a list of all the real numbers we can think of and associate each decimal number with a natural number as follows:

Natural	Real
0	0.124 …
1	0.015 …
2	0.53692 …
3	0.8003444 …
4	0.3341005011 …
5	0.3425678 …

The exact assignment of real numbers to the natural numbers is arbitrary: all we need to do is to assign one real number per natural number so that all the real numbers are accounted for. But this cannot be so! To see why, Cantor showed how to find another real number that cannot be already on our list. In the preceding list, we underline the first digit of the first number, the second digit of the second, the third of the third, and so on. This gives us the sequence:

1, 1, 6, 3, 0, 7, …

The diagonal slash procedure is to construct a new real number from this sequence that differs from the digits of this number in each corresponding place. We make this new number by ensuring that the nth digit of this new number differs from the nth digit in this sequence. For example we can define a new real number by adding one to each of the underlined digits (with the rule that 9 + 1 = 0) to get:

0.227418 …

What have we achieved? By construction, this number differs from the first number in the first decimal place, from the second number in the second place, from the third number in the third place, and so on. By construction this number is different from any of the real numbers on our original list. Hence we have found a real number that cannot be on our list. This contradiction establishes the fact that there cannot be a one-to-one correspondence

between the real numbers and the natural numbers and that the real numbers are not countably infinite.

How does this argument relate to Turing machines and computable numbers? We can obviously construct a machine to calculate the decimal expansion of π by using one of the many algorithms for determining π. This just requires a set of rules for adding, multiplying, and so on. However, because π is an infinite decimal, the work of the machine would never end and the machine would need an unlimited amount of working space on its tape. A legitimate Turing machine must halt, so we need to set up the machine to calculate each successive decimal place of π as a separate calculation. Each number in the decimal expansion would then use only a finite amount of tape and take some finite time for the machine to compute. So a Turing machine for producing the decimal expansion of π to any number of decimal digits does exist in this sense, although it would be a little complicated to set up. And we can obviously do the same for a real number like the square root of two. The real numbers that can be generated in this way Turing called "computable numbers."

In his paper Turing showed that the number of his machines was countable. To see this, we specify any given Turing machine by the "quintuple" description of the machine as we see in our detailed discussion of the Parity Counter Turing Machine at the end of this chapter. The quintuple description is just an explicit labeling of the actions of the Turing machine in terms of five items: the initial state and the symbol that is read plus the new state, the symbol written, and the motion of the head to the left or right. The machine is specified by a set of quintuples describing exactly what happens for each initial state and symbol read. This set of quintuples may be written out as a binary string. The resulting binary number can now be used to uniquely label the machine by a one-to-one correspondence with the set of natural numbers. In principle then we can now make a list with a natural number specifying each Turing machine and the corresponding number the machine computes. The resulting infinite list now includes every number that is computable! Turing now made use of Cantor's diagonal slash method to add one to each of the computable numbers on the diagonal as we did in the preceding text to generate a new real number. In this way Turing showed that there are real numbers that are noncomputable. The details of Turing's proof are a bit more complicated, but this is the basic argument that convinced Turing that the answer to Hilbert's third question was "no."

Universality and the Church-Turing thesis

Before we return to the *Entscheidungsproblem*, we need to look at another marvelously original idea in Turing's paper – the Universal Turing Machine. This is a Turing machine that can do anything that any specific, special-purpose Turing machine can do, albeit more slowly and less efficiently. Suppose we have a specific Turing machine **T** that acts on a tape **t** to produce its result. What Turing showed was that it was possible to construct another Turing machine **U** that, if we give it as input the specification of **T** and the tape **t**, will output the result that machine **T** would have produced acting on tape **t**. The behavior of

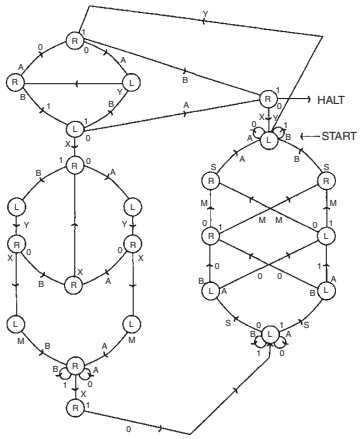


Fig. 6.5. A representation of a Universal Turing Machine due to Marvin Minsky.

Fig. 6.6. The "Knights of the Lambda Calculus," the unofficial badge of LISP programmers at MIT.

U is simple to describe but complicated to write down in detail. The Universal Turing Machine **U** must imitate **T** step-by-step, keeping a record of the state of **T**'s tape at each stage. By examining its simulated input tape **t**, the machine can see what **T** would read at any given stage. Then, by looking at the description it has of **T**, **U** can find out what **T** is supposed to do next. This is essentially just what we would do when using a list of quintuples and a tape to figure out what a Turing machine does. Turing's universal machine **U** is just a slower version of us!

Turing went into great detail to prove the existence of a Universal Turing Machine and, as you can see in Figure 6.5, the resulting state transition diagram for **U** is much more complicated than that for our simple Parity Counter (Fig. 6.10) that we describe in detail later. This example, from MIT computer scientist Marvin Minsky, makes use of eight symbols and twenty-three states. Most digital computers built today are effectively universal computers. With the right program, enough time, and enough memory, any universal computer can simulate any other computer.

If we ignore the slowness and inefficiency of using a Turing machine, we can ask this fundamental question: what problems can be solved by such a machine? The answer is very surprising. Everything that is algorithmically computable is computable by a Turing machine. Why should we believe this? At around the same time as Turing was devising his ingenious machines, Alonzo Church (B.6.7), a U.S. mathematician based in Princeton, New Jersey, had defined a formalism of logic and propositions that he called *lambda calculus* (Fig. 6.6). Church argued that any "effectively calculable" problem corresponded to a lambda calculus expression. He had also been able to use his formalism to show that the problem of deciding whether one string of symbols could be converted into another string was unsolvable, in the sense that there was no lambda expression that could do this. In this way Church had been able to show that Hilbert's third question, the *Entscheidungsproblem*, was also unsolvable. Although coming at the problem from very different perspectives, Turing and Church had both proved the problem's insolvability.

The statement – that anything effectively computable is computable by a Turing machine – is known as the *Church-Turing thesis*. It is called a thesis rather than a theorem because it involves the informal concept of effective computability. The thesis equates the mathematically precise statement, "computable by a Turing machine," with the informal, intuitive idea of a problem being solvable by some algorithm on any machine whatsoever. It applies to all computable problems written in any programming language on any computer!

The Church-Turing thesis is only a thesis but the majority of computer scientists accept its validity because many people besides Church and Turing have arrived at an equivalent result. At about the same time as Church and Turing's work, mathematicians Stephen Kleene and Emil Post devised alternative formalisms that led to similar notions of computability. Many others have looked at variants of the simple Turing machine such as machines with multiple tapes or machines with two-dimensional tapes. None of these machines can solve problems that cannot be solved by the basic Turing machine.

B.6.7. Alonzo Church (1903–95) was very supportive of Turing's ideas and he was first to use the term *Turing machine*. This, along with the Church-Turing thesis on computability, is now one of the cornerstones of computer science.

The halting problem and the *Entscheidungsproblem*

Using Turing's universal machine, it is possible to prove that there is no way to tell, in general, whether the execution of a given program will terminate on any given input. If we have a program to calculate the square of x we can be confident that, after we input x into the machine, we will be able to read x^2 on the tape when the machine halts. Termination is not always so obvious. Suppose we have a program that takes a number as input and either divides it by two, if the number is even, or triples it and adds one, if it is odd. The program then takes this new number as input and repeats the process. When the output is one, the program halts. Can we be sure that this will happen? This is known as the $3x + 1$ problem – what computer scientist David Harel calls the "simplest-to-describe open problem in mathematics."[7] If we try this with the starting value of x = 7 we get the sequence 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 and the sequence terminates. With other values of x we find that the program sometimes terminates but for some values of x it just keeps on generating numbers with no repeating pattern until we decide we have seen enough. This is one specific instance of the "halting problem" – a program for which we cannot determine whether it terminates or not. More generally, the halting problem is concerned with the termination of any program for any input.

How can this result be proved? If we have a Turing machine **T** that calculates some function **F**, can we find a computable function that predicts whether or not the machine **T** will halt or not? If there is such a function, we know that it too must be describable by another Turing machine. This concept, of Turing machines telling us about other Turing machines, is a very powerful tool. This device can be used to prove that the halting problem is noncomputable. The trick is to assume the existence of a machine that can predict whether a program halts and then show that this leads to a contradiction, meaning that the original assumption that such a machine exists is incorrect.

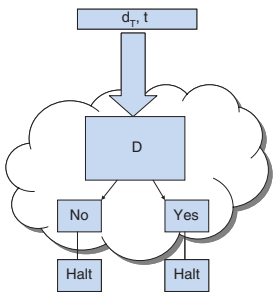


Fig. 6.7a. A hypothetical Turing machine for the halting problem.

We begin our sketch of a proof by supposing we have a machine **D** that takes as input a tape that contains a description $\mathbf{d_T}$ of the machine **T** – these are just the quintuples that define **T** – as well as **T**'s input tape **t**. Machine **D** is required to tell us whether **T** will halt or not and then come to a halt (Fig. 6.7.a). We now introduce another machine **Z** which takes the machine description $\mathbf{d_T}$ and uses this as the input tape for the machine **D**. This machine **Z** reacts to the output from **D** in the following way:

If **T** halts (**D** says "yes"), then **Z** does not halt.
If **T** does not halt (**D** says "no"), then **Z** halts.

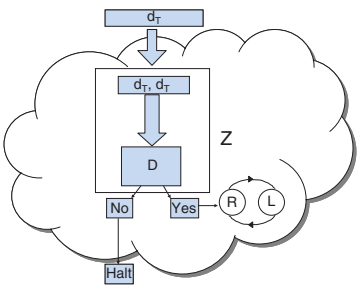


Fig. 6.7b. A paradoxical Turing machine to demonstrate the halting problem.

We can arrange this to happen by introducing two new states in the "yes" branch. The machine now oscillates between them indefinitely and this prevents our **Z** machine from halting if **D** halts (says "yes") as in Figure 6.7b. Now we arrive at the crux of the argument. We get **Z** to operate on itself by taking as input the quintuples $\mathbf{d_Z}$ that define the **Z** machine and substitute **Z** for **T** in the above argument. We find:

Z applied to $\mathbf{d_Z}$ halts if and only if **Z** applied to $\mathbf{d_Z}$ does not halt.

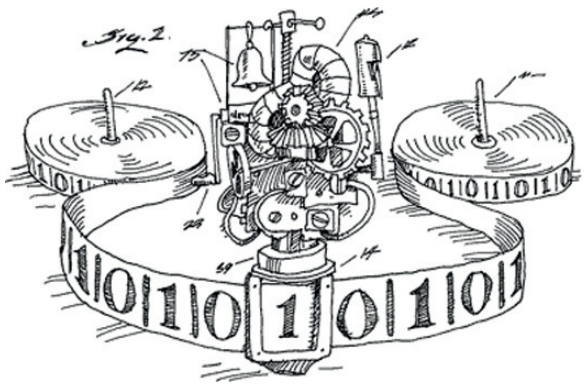
The origin of this contradiction can be traced back to our assumption that machine **D** exists. Therefore no such machine can exist and this shows that the halting problem is not decidable. Using such techniques, Turing was able to demonstrate an unsolvable problem: in so doing, he had shown that Hilbert's *Entscheidungsproblem* has no solution.

The halting problem has a number of important implications. In writing programs we would naturally like to be able to check whether our program actually does do what it is supposed to do. This turns out to be a decision problem. We need to input a description of the algorithmic problem and the text of our program implementing an algorithm that we think solves the problem. We want a "yes" if, for all of the legal inputs for the problem, our algorithm will terminate and give the correct solution; and we want a "no" if there is any input for which our program fails to terminate or gives the wrong result. Now we know about the halting problem we can see immediately that such an automatic verifier is not possible. However, although we cannot guarantee that our program will halt for all inputs, it is still possible to produce formal verification tools that can deliver useful results most of the time!

In another application, Fred Cohen, in 1986 in his doctoral thesis for the University of Southern California, showed that the problem of detecting the presence of a computer virus was an instance of the halting problem. Unfortunately this means that the general problem of identifying a virus cannot be solved. We will have more to say about computer viruses in a later chapter.

Key concepts

- Hilbert's *Entscheidungsproblem* (decision problem)
- Turing machines
- Natural, integer, and rational numbers
- Irrational numbers and effective procedures
- Computable and noncomputable numbers
- Universal Turing Machines
- Church-Turing thesis
- The halting problem



More on Turing machines

Turing's super-typewriter

According to Andrew Hodges, as a boy, Turing dreamed of inventing ways of improving typewriters, and this might have provided him with the starting point for his later ideas about computation. In any case, the way that typewriters manipulate symbols serves as a good introduction to Turing machines. A typewriter is mechanical in that its response to any action of the operator is built in. However, the specific response will depend on whether it was set to type lowercase or uppercase letters; this is the configuration – or state – of the machine. Turing machines generalize this idea to include a larger but still finite number of possible states. A typewriter keyboard contains only a finite number of symbols – the letters of the alphabet and the numbers 0 to 9, plus a few special symbols. Similarly, Turing assumed that his machine was only allowed a finite number of possible operations. Together with the description of the allowed states, this allowed him to write a complete description of the behavior of his machine. The other relevant feature of the typewriter is that the typing point – the point where the typewriter's "head" strikes the paper – can move relative to the page. Turing incorporated this feature – albeit with symbols written on a tape rather than on a page of paper – into his idea for a primitive computing machine.

The typewriter analogy is limited in that a typewriter can only write symbols on a page when they are selected by a human operator, who also decides when to change configuration and where to type the symbol on the page. Turing wanted a much more general kind of machine to manipulate symbols. In addition to writing, Turing wanted his machine to be able to "scan" (that is, read) a symbol on the tape as well as to write or erase a symbol. Such a "super-typewriter" would retain the property of a typewriter in having a finite number of states and an exactly determined behavior for each operation. In addition, unlike in our typewriter analogy, instead of human-operated machines, Turing was interested in investigating what he called *automatic* machines, for which no human intervention would be necessary.

Turing machines in detail

Let us look in more detail at how we can define a Turing machine to do a particular job. We shall label the various possible states of the machine by the symbol Q, and any particular state "i" as the state Q_i. Similarly, we will label the entries on the tape by the symbol S and a particular symbol "i" as the symbol S_i. When we start, only a finite part of the tape has any writing on it: either side of this region, the tape is blank. We start the machine to the left of the writing on the tape at time T. It then proceeds to march along, step-by-step, in uniform time steps, as if following the ticks of a clock. What the state of the machine and the tape is at step T + 1 will then be determined by three functions, each of which will depend on the initial state Q_i at step T and the symbol S_i the head has just read. These three functions define what its new state, Q_j will be; what symbol, S_j, it has written on the tape in the original box; and what was the direction, D, of its subsequent motion after writing the new symbol. In mathematical notation, we can write this behavior in terms of three functions – F, G, and D, each depending on the initial Q_i and S_i:

$$Q_j = F(Q_i, S_i)$$

$$S_j = G(Q_i, S_i)$$

$$D_j = D(Q_i, S_i)$$

The Turing machine is fully defined by these three functions, which can be written out as a table of *quintuples*. This is just a fancy name for the set of these two variables and three functions we have defined: Q_i and S_i at time T and Q_j, S_j, and D_j at time T + 1. All we have to do now is write some data on the tape and start the machine at the right position. The machine will then calculate away and print out the result of its calculation somewhere on the tape for us to look at when the machine has finished. Note that we have to explicitly instruct the machine when to halt. This sounds pretty trivial but we will see later that the issue of whether a machine will halt or not leads to a profound issue in the theory of computation.

Let's try to construct a very simple Turing machine that measures the *parity* of any string of 0s and 1s. The parity of a string is defined to be whether the number of 1s is even or odd. We are given the string 1101101, and begin by writing this binary string as input data on the tape as shown in Figure 6.8, with one symbol in each box. The reading head of the machine starts at the far left of the string, on the first digit. The end of the string is designated by the letter E. On either side of the string there are only zeros on the tape.

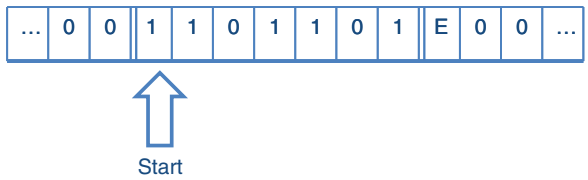


Fig. 6.8. Input tape for the Parity Counter Turing Machine.

Before it has read any symbols, the machine starts off in state Q_0, corresponding to even parity. If the machine encounters a 0, it stays in the state Q_0 – because the parity has not changed – and then moves one space to the right. If the symbol it reads is a 1, the machine erases this, replaces it with a 0, moves one space to the right and changes to the state Q_1. This is the state for odd parity. Continuing, if the machine now hits a 0, it stays in the state Q_1 and moves another space to the right. If it hits a 1, it erases it, prints a 0, and moves another cell to the right, changing the state back to Q_0. The machine continues working along the string in this way, changing state whenever it encounters a 1, and leaving a string of 0s behind. If the machine is in the state Q_0 after it has read the last symbol, the string has even parity; if it is in the state Q_1, the parity is odd.

How does the machine tell us the parity and the result of its calculation? We need to include a rule that tells the machine what to do when it meets the end symbol E. If it is in state Q_0 and reads E, it erases the E and writes a 0 meaning that the string had even parity. If it is the state Q_1, it replaces E by a 1 meaning the string had odd parity. In both cases the machines enter a new state Q_H, meaning "halt." It does not need to move to the right or the left: the answer can be observed by looking at the box on the tape where the machine halted (Fig. 6.9).

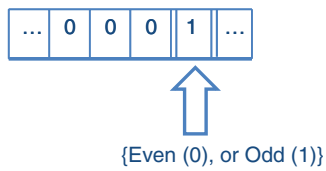


Fig. 6.9. Output tape from the Parity Counter Turing Machine.

To understand what happens, we have painstakingly described the operation of this Parity Counter Turing Machine in words. In practice, it is more economical to summarize the machine's behavior as a table of quintuples. We can summarize this table of quintuples in a diagram (Fig. 6.10). Here we have indicated the states Q_0 and Q_1 – Even and Odd – by the circles and the direction of motion after reading the box, by R, for right which we also write in the circle. The directed arcs, starting with either a 0 or 1 on them, indicate what happens to

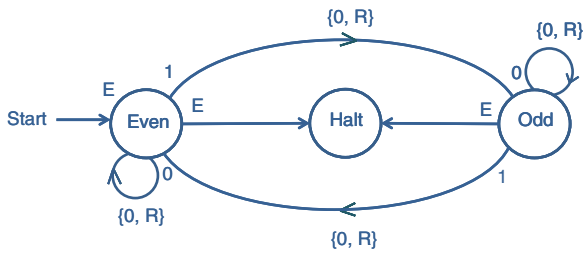


Fig. 6.10. Diagram of a Parity Counter Turing Machine.

the machine if this is the symbol that is read. The symbol on the arc indicates what the head overwrites in the box. Thus reading a 0 from the state Q_0 = Even just takes us to the same state: reading a 1, takes us to the other circle corresponding to Q_1 = Odd. We have also indicated the start and the halt conditions in the diagram.

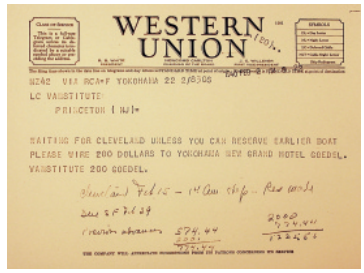
Similar diagrams are used to describe the behavior of "Finite State Machines" or FSMs that are often used to summarize the behavior of devices whose actions depend not just on the current input but also on the previous inputs. The FSM has enough memory to store a summary of a finite number of past inputs. A combination lock is an example of an FSM. The lock cannot remember all the numbers dialed into the lock but it remembers enough to know whether the would-be user has entered the correct small sequence of numbers to open the lock. A Turing machine is just an FSM with an infinitely long tape that serves the same function as memory in a computer.

We can now go on to construct Turing machines for adding, multiplying, copying, and so on. To build up more complex machines it is convenient to reuse these simpler machines as components of the complex machine, rather like subroutines in a software program. This greatly simplifies the construction of such machines.

Gödel, von Neumann, Turing, and Church

Gödel and the U.S. Constitution

After Austria's annexation by Germany in 1938, Gödel lost his position at the University of Vienna and was found fit for conscription into the German army. With the outbreak of World War II in 1939, Gödel and his wife set out for Princeton in the United States. Because of the dangers of a North Atlantic crossing, they traveled via the trans-Siberian railway and then by ship across the Pacific. He ran out of money in Japan and had to telegraph to Princeton for a loan (Fig. 6.11).


WESTERN UNION

NL42 VIA RCA-F YOKOHAMA 22 2/6305

LC VAINSTITUTE

PRINCETON NJ

WAITING FOR CLEVELAND UNLESS YOU CAN RESERVE EARLIER BOAT PLEASE WIRE 200 DOLLARS TO YOKOHAMA NEW GRAND HOTEL GOEDEL. VAINSTITUTE 200 GOEDEL.

Fig. 6.11. Telegram from Gödel in Yokohama, Japan, to the Institute for Advanced Study in Princeton, requesting $200 for emergency travel expenses.

After the war, Gödel wanted to become an American citizen, and he asked Albert Einstein and economist Oskar Morgenstern to be his witnesses. Of course, Gödel took his preparation for the citizenship hearing very seriously and studied the history of North America, and of Princeton, as well as the U.S. Constitution. At this point, Morgenstern recounts:

> [Gödel] rather excitedly told me that in looking at the Constitution, to his distress, he had found some inner contradictions and that he could show how in a perfectly legal manner it would be possible for somebody to become a dictator and set up a Fascist regime, never intended by those who drew up the Constitution.[8]

Einstein and Morgenstern went with Gödel to the citizenship ceremony, and the three of them sat down before the examiner. The examiner first asked Einstein and Morgenstern whether they thought Gödel would make a good citizen, to which they assured him that this would be the case. The examiner then turned to Gödel.

Examiner: Now Mr Gödel, where do you come from?
Gödel: Where do I come from? Austria.
Examiner: What kind of government did you have in Austria?
Gödel: It was a republic, but the constitution was such that it finally was changed into a dictatorship.
Examiner: Oh! This is very bad. This could not happen in this country.
Gödel: Oh yes [it can], I can prove it![9]

Fortunately, the examiner was a wise man and refrained from following up on Gödel's new inconsistency proof of the U.S. Constitution!

Turing and the conceptual foundation of computers

Although John von Neumann did not refer explicitly to Turing's paper on computability and Turing machines when he wrote the famous "First Draft of a Report on the EDVAC," he was well aware of the importance of Turing's work and even offered him a post as his research assistant at Princeton. The mathematician Stanislaw Ulam, who later worked at Los Alamos on the Manhattan Project, recalled that "von Neumann mentioned to [him] Turing's name several times in 1939 … concerning mechanical ways to develop formal mathematical systems."[10] Similarly, another physicist who worked at Los Alamos, Stanley Frankel, remembers von Neumann's enthusiasm for Turing's work in 1943 or 1944:

> Von Neumann introduced me to that paper and at his urging I studied it with care. Many people have acclaimed von Neumann as the "father of the computer" … but I am sure that he would never have made that mistake

> himself. He might well be called the midwife, perhaps, but he firmly emphasized to me, and to others I am sure, that the fundamental conception is owing to Turing – insofar as not anticipated by Babbage, Lovelace, and others. In my view von Neumann's essential role was in making the world aware of these fundamental concepts introduced by Turing and of the development work carried out in the Moore school and elsewhere.[11]

And in 1946, von Neumann wrote to his friend Norbert Wiener of "the great positive contribution of Turing … one, definite mechanism can be 'universal.'"[12]

Some of the early pioneers of computers did not recognize that they were, in essence, building a variant of a universal Turing Machine. In 1956 Howard Aiken said:

> [If] it should turn out that the basic logics of a machine designed for the numerical solution of differential equations coincide with the logics of a machine intended to make bills for a department store, I would regard this as the most amazing coincidence I have ever encountered.[13]

Alan Turing himself, in contrast with Howard Aiken's remarks, said in 1950:

> This special property of digital computers, that they can mimic any discrete state machine, is described by saying that they are universal machines. The existence of machines with this property has the important consequence that, considerations of speed apart, it is unnecessary to design various new machines to do various computing processes. They can all be done with one digital computer, suitably programmed for each case. It will be seen that as a consequence of this all digital computers are in a sense equivalent.[14]

In 1945 Turing produced a report for the construction of his ACE Automatic Computing Engine. Compared to von Neumann's EDVAC Report, which "is a draft and is unfinished," the ACE Report "is a complete description of a computer, right down to the logical circuit diagrams." In contrast to the computer designs based on the EDVAC ideas – which were focused on delivering fast numerical calculations – Turing's design recognized the full power of digital computers as all-purpose machines, capable of manipulating symbols and playing chess as well as performing numerical operations. To characterize these designs as merely embodying the "stored-program concept" is to underappreciate the breadth of Turing's vision – of which von Neumann was well aware.

Turing and Church

In April 1936, Turing had just delivered his paper to Max Newman, much to Newman's surprise. Newman read the paper and realized the significance of Turing's work. He encouraged Turing to publish the paper in the *Proceedings of the London Mathematical Society*. As Turing was tidying up his paper for publication, in mid-May Newman received a copy of Church's paper. Since the subject matter of the two papers had much overlap and Church had priority in terms of publication, there was some doubt as to whether Turing's paper could be published. Newman wrote to the editor of the journal:

> I think you know the history of Turing's paper on Computable numbers. Just as it was reaching its final state an offprint arrived, from Alonzo Church of Princeton, of a paper anticipating Turing's results to a large extent. I hope it will nevertheless be possible to publish the paper. The methods are to a large extent different, and the result is so important that different treatments of it should be of interest.[15]

Fortunately, the editor agreed, and Turing's paper with his machines was published in the *Proceedings*. Newman also wrote to Church in Princeton:

Dear Professor Church, 31 May 1936

An offprint which you kindly sent me recently of your paper in which you define 'calculable numbers', and show that the Entscheidungsproblem for Hilbert logic is insoluble, had a rather painful interest for a young man, A. M. Turing, here, who was just about to send in for publication a paper in which he had used a definition of 'Computable numbers' for the same purpose. His treatment – which consists in describing a machine which will grind out any computable sequence – is rather different from yours, but seems to be of great merit, and I think it is of great importance that he should come and work with you next year if that is at all possible. He is sending you the typescript of his paper for your criticisms.... I should mention that Turing's work is entirely independent: he has been working without any supervision or criticism from anyone. This makes it all the more important that he should come into contact as soon as possible with the leading workers on this line, so that he should not develop into a confirmed solitary.[16]

Turing read Church's paper in the summer and added an appendix to his paper that demonstrated that his definition of 'computable' – meaning anything that could be computed by a Turing machine – was equivalent to what Church had called "effectively calculable" – meaning anything that could be described by a formula using the lambda calculus. When Turing's paper was published in January 1937, Church generously reviewed it very positively in the well-known *Journal of Symbolic Logic*. He also used the description "Turing machine" in print for the first time, writing that "a human calculator, provided with pencil and paper and explicit instructions, can be regarded as a type of Turing machine."[17]