Part I

Four Concepts

Chapter 1

Computation

B. Jack Copeland

The Birth of the Modern Computer

As everyone who can operate a personal computer knows, the way to make the machine perform some desired task is to open the appropriate program stored in the computer's memory. Life was not always so simple. The earliest large-scale electronic digital computers, the British Colossus (1943) and the American ENIAC (1945), did not store programs in memory (see Copeland 2001). To set up these computers for a fresh task, it was necessary to modify some of the machine's wiring, rerouting cables by hand and setting switches. The basic principle of the modern computer – the idea of controlling the machine's operations by means of a program of coded instructions stored in the computer's memory – was thought of by Alan Turing in 1935. His abstract "universal computing machine," soon known simply as the universal Turing machine (UTM), consists of a limitless memory, in which both data and instructions are stored, and a scanner that moves back and forth through the memory, symbol by symbol, reading what it finds and writing further symbols. By inserting different programs into the memory, the machine is made to carry out different computations.

Turing's idea of a universal stored-program computing machine was promulgated in the US by John von Neumann and in the UK by Max Newman, the two mathematicians who were by and large responsible for placing Turing's abstract universal machine into the hands of electronic engineers (Copeland 2001). By 1945, several groups in both countries had embarked on creating a universal Turing machine in hardware. The race to get the first electronic stored-program computer up and running was won by Manchester University where, in Newman's Computing Machine Laboratory, the "Manchester Baby" ran its first program on June 21, 1948. By 1951, electronic stored-program computers had begun to arrive in the marketplace. The first model to go on sale was the Ferranti Mark I, the production version of the Manchester computer (built by the Manchester firm Ferranti Ltd.). Nine of the Ferranti machines were sold, in Britain, Canada, Holland, and Italy, the first being installed at Manchester University in February 1951. In the US, the Computer Corporation sold its first UNIVAC later the same year. The LEO computer also made its debut in 1951; LEO was a commercial version of the prototype EDSAC machine, which at Cambridge University in 1949 had become the second stored-program electronic computer to function. In 1953 came the IBM 701, the company's first mass-produced stored-program electronic computer (strongly influenced by von Neumann's prototype IAS computer, which was working at

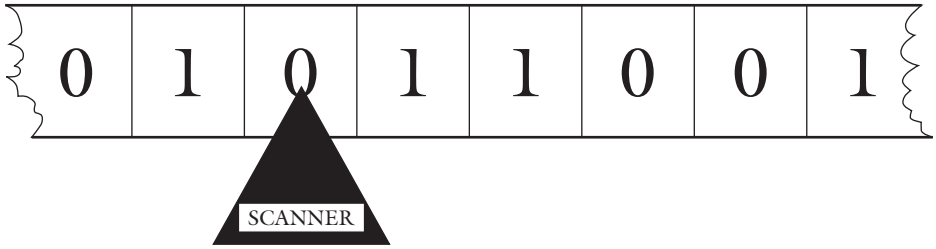

Figure 1.1: A Turing machine

Princeton University by the summer of 1951). A new era had begun.

Turing introduced his abstract Turing machines in a famous article entitled "On Computable Numbers, with an Application to the Entscheidungsproblem" (published in 1936). Turing referred to his abstract machines simply as "computing machines" – the American logician Alonzo Church dubbed them "Turing machines" (Church 1937: 43). "On Computable Numbers" pioneered the theory of computation and is regarded as the founding publication of the modern science of computing. In addition, Turing charted areas of mathematics lying beyond the reach of the UTM. He showed that not all precisely-stated mathematical problems can be solved by a Turing machine. One of them is the *Entscheidungsproblem* – "decision problem" – described below. This discovery wreaked havoc with received mathematical and philosophical opinion. Turing's work – together with contemporaneous work by Church (1936a, 1936b) – initiated the important branch of mathematical logic that investigates and codifies problems "too hard" to be solvable by Turing machine. In a single article, Turing ushered in both the modern computer and the mathematical study of the uncomputable.

What is a Turing Machine?

A Turing machine consists of a limitless memory and a scanner that moves back and forth through the memory, symbol by symbol, reading what it finds and writing further symbols. The memory consists of a tape divided into squares. Each square may be blank or may bear a single symbol, "0" or "1," for example, or some other symbol taken from a finite alphabet. The scanner is able to examine only one square of tape at a time (the "scanned square"). (See figure 1.1.) The tape is the machine's general-purpose storage medium, serving as the vehicle for input and output, and as a working memory for storing the results of intermediate steps of the computation. The tape may also contain a program of instructions. The input that is inscribed on the tape before the computation starts must consist of a finite number of symbols. However, the tape itself is of unbounded length – since Turing's aim was to show that there are tasks which these machines are unable to perform, even given unlimited working memory and unlimited time. (A Turing machine with a tape of fixed finite length is called a *finite state automaton*. The theory of finite state automata is not covered in this chapter. An introduction may be found in Sipser 1997.)

The Basic Operations of a Turing Machine

Each Turing machine has the same small repertoire of *basic* (or "atomic") operations. These are logically simple. The scanner contains mechanisms that enable it to *erase* the symbol on the scanned square, to *write* a symbol on the scanned square (first erasing any existing symbol), and to *shift position* one square to the left or right. Complexity of operation is achieved by chaining together large numbers of these simple

Table 1.1

State	*Scanned square*	*Operations*	*Next state*
a	blank	P[0], R	**b**
b	blank	R	**c**
c	blank	P[1], R	**d**
d	blank	R	**a**

basic actions. The scanner will *halt* if instructed to do so, i.e. will cease work, coming to rest on some particular square, for example the square containing the output (or if the output consists of a string of several digits, then on the square containing the left-most digit of the output, say).

In addition to the operations just mentioned, *erase*, *write*, *shift*, and *halt*, the scanner is able to *change state*. A device within the scanner is capable of adopting a number of different positions. This device may be conceptualized as consisting of a dial with a finite number of positions, labeled "a," "b," "c," etc. Each of these positions counts as a different state, and changing state amounts to shifting the dial's pointer from one labeled position to another. The device functions as a simple memory. As Turing said, by altering its state the "machine can effectively remember some of the symbols which it has 'seen' (scanned) previously" (1936: 231). For example, a dial with two positions can be used to keep a record of which binary digit, 0 or 1, is present on the square that the scanner has just vacated. If a square might also be blank, then a dial with three positions is required.

Commercially available computers are hard-wired to perform basic operations considerably more sophisticated than those of a Turing machine – add, multiply, decrement, store-at-address, branch, and so forth. The precise list of basic operations varies from manufacturer to manufacturer. It is a remarkable fact that none of these computers can out-compute the UTM. Despite the austere simplicity of Turing's machines, they are capable of computing anything that any computer on the market can compute. Indeed, because they are abstract machines, they are capable of computations that no "real" computer could perform.

Example of a Turing machine

The following simple example is from "On Computable Numbers" (Turing 1936: 233). The machine – call it **M** – starts work with a blank tape. The tape is endless. The problem is to set up the machine so that if the scanner is positioned over any square of the tape and the machine set in motion, it will print alternating binary digits on the tape, 0 1 0 1 0 1 . . . , working to the right from its starting place, leaving a blank square in between each digit. In order to do its work **M** makes use of four states labeled "**a**," "**b**," "**c**," and "**d**." **M** is in state **a** when it starts work. The operations that **M** is to perform can be set out by means of a table with four columns (see table 1.1). "R" abbreviates the instruction "shift right one square," "P[0]" abbreviates "print 0 on the scanned square," and likewise "P[1]." The top line of table 1.1 reads: if you are in state **a** and the square you are scanning is blank, then print 0 on the scanned square, shift right one square, and go into state **b**. A machine acting in accordance with this table of instructions – or program – toils endlessly on, printing the desired sequence of digits while leaving alternate squares blank.

Turing did not explain how it is to be brought about that the machine acts in accordance with the instructions. There was no need. Turing's machines are abstractions and it is not necessary to propose any specific mechanism for causing the machine to follow the instructions. However, for purposes of visualization, one might imagine the scanner to be accompanied by a bank of switches and plugs resembling an old-fashioned telephone switchboard. Arranging the plugs and setting the switches in a certain way causes the machine to act in accordance

with the instructions in table 1.1. Other ways of setting up the "switchboard" cause the machine to act in accordance with other tables of instructions.

The universal Turing machine

The UTM has a single, fixed table of instructions, which we may imagine to have been set into the machine by way of the switchboard-like arrangement just mentioned. Operating in accordance with this table of instructions, the UTM is able to carry out *any* task for which a Turing-machine instruction table can be written. The trick is to place an instruction table for carrying out the desired task onto the tape of the universal machine, the first line of the table occupying the first so many squares of the tape, the second line the next so many squares, and so on. The UTM reads the instructions and carries them out on its tape. This ingenious idea is fundamental to computer science. The universal Turing machine is in concept the stored-program digital computer.

Turing's greatest contributions to the development of the modern computer were:

- The idea of controlling the function of the computing machine by storing a program of (symbolically or numerically encoded) instructions in the machine's memory.
- His proof that, by this means, a *single* machine of *fixed structure* is able to carry out every computation that can be carried out by any Turing machine whatsoever.

Human Computation

When Turing wrote "On Computable Numbers," a computer was not a machine at all, but a human being – a mathematical assistant who calculated by rote, in accordance with some "effective method" supplied by an overseer prior to the calculation. A paper-and-pencil method is said to be effective, in the mathematical sense, if it (a) demands no insight or ingenuity from the human carrying it out, and (b) produces the correct answer in a finite number of steps. (An example of an effective method well-known among philosophers is the truth table test for tautologousness.) Many thousands of human computers were employed in business, government, and research establishments, doing some of the sorts of calculating work that nowadays is performed by electronic computers. Like filing clerks, computers might have little detailed knowledge of the end to which their work was directed.

The term "computing machine" was used to refer to calculating machines that mechanized elements of the human computer's work. These were in effect homunculi, calculating more quickly than an unassisted human computer, but doing nothing that could not in principle be done by a human clerk working effectively. Early computing machines were somewhat like today's nonprogrammable hand-calculators: they were not automatic, and each step – each addition, division, and so on – was initiated manually by the human operator. For a complex calculation, several dozen human computers might be required, each equipped with a desk-top computing machine. By the 1940s, however, the scale of some calculations required by physicists and engineers had become so great that the work could not easily be done in a reasonable time by even a roomful of human computers with desk-top computing machines. The need to develop high-speed, large-scale, automatic computing machinery was pressing.

In the late 1940s and early 1950s, with the advent of electronic computing machines, the phrase "computing machine" gave way gradually to "computer." During the brief period in which the old and new meanings of "computer" co-existed, the prefix "electronic" or "digital" would usually be used in order to distinguish machine from human. As Turing stated, the new electronic machines were "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" (Turing 1950: 1). Main-frames, laptops, pocket calculators, palm-pilots – all carry out work that a human rote-worker could do, if he or she

worked long enough, and had a plentiful enough supply of paper and pencils.

The Turing machine is an idealization of the human computer (Turing 1936: 231). Wittgenstein put this point in a striking way:

> Turing's "Machines." These machines are *humans* who calculate. (Wittgenstein 1980: §1096)

It was not, of course, some deficiency of imagination that led Turing to model his logical computing machines on what can be achieved by a human being working effectively. The purpose for which he introduced them demanded it. The Turing machine played a key role in his demonstration that there are mathematical tasks which *cannot* be carried out by means of an effective method.

The Church–Turing Thesis

The concept of an effective method is an informal one. Attempts such as the above to explain what counts as an effective method are not rigorous, since the requirement that the method demand neither insight nor ingenuity is left unexplicated. One of Turing's leading achievements – and this was a large first step in the development of the mathematical theory of computation – was to propose a rigorously defined expression with which the informal expression "by means of an effective method" might be replaced. The rigorously defined expression, of course, is "by means of a Turing machine." The importance of Turing's proposal is this: if the proposal is correct, then talk about the existence and non-existence of effective methods can be replaced throughout mathematics and logic by talk about the existence or non-existence of Turing machine programs. For instance, one can establish that there is no effective method at all for doing such-and-such a thing by proving that no Turing machine can do the thing in question.

Turing's proposal is encapsulated in the *Church–Turing thesis*, also known simply as *Turing's thesis*:

> The UTM is able to perform any calculation that any human computer can carry out.

An equivalent way of stating the thesis is:

> Any effective – or *mechanical* – method can be carried out by the UTM.

("Mechanical" is a term of art in mathematics and logic. It does not carry its everyday meaning, being in its technical sense simply a synonym for "effective.") Notice that the converse of the thesis – any problem-solving method that can be carried out by the UTM is effective – is obviously true, since a human being can, in principle, work through any Turing-machine program, obeying the instructions ("in principle" because we have to assume that the human does not go crazy with boredom, or die of old age, or use up every sheet of paper in the universe).

Church independently proposed a different way of replacing talk about effective methods with formally precise language (Church 1936a). Turing remarked that his own way of proceeding was "possibly more convincing" (1937: 153); Church acknowledged the point, saying that Turing's concept of computation by Turing machine "has the advantage of making the identification with effectiveness . . . evident immediately" (Church 1937: 43).

The name "Church–Turing thesis," now standard, seems to have been introduced by Kleene, with a flourish of bias in favor of his mentor Church (Kleene 1967: 232):

> Turing's and Church's theses are equivalent. We shall usually refer to them both as *Church's thesis*, or in connection with that one of its . . . versions which deals with "Turing machines" as *the Church–Turing thesis.*

Soon ample evidence amassed for the Church–Turing thesis. (A survey is given in chs. 12 and 13 of Kleene 1952.) Before long it was (as Turing put it) "agreed amongst logicians" that his proposal gives the "correct accurate rendering" of talk about effective methods (Turing 1948: 7). (Nevertheless, there have been occasional dissenting voices over the years; for example Kalmár 1959 and Péter 1959.)

Beyond the Universal Turing Machine

Computable and uncomputable numbers

Turing calls any number that can be written out by a Turing machine a *computable* number. That is, a number is computable, in Turing's sense, if and only if there is a Turing machine that calculates each digit of the number's decimal representation, in sequence. π, for example, is a computable number. A suitably programmed Turing machine will spend all eternity writing out the decimal representation of π digit by digit, 3.14159 . . .

Straight off, one might expect it to be the case that *every* number that *has* a decimal representation (that is to say, every real number) is computable. For what could prevent there being, for any particular number, a Turing machine that "churns out" that number's decimal representation digit by digit? However, Turing proved that not every real number is computable. In fact, computable numbers are relatively scarce among the real numbers. There are only *countably* many computable numbers, because there are only countably many different Turing-machine programs (instruction tables). (A collection of things is countable if and only if either the collection is finite or its members can be put into a one-to-one correspondence with the integers, 1, 2, 3, ...) As Georg Cantor proved in 1874, there are *uncountably* many real numbers – in other words, there are more real numbers than integers. There are literally not enough Turing-machine programs to go around in order for every real number to be computable.

The printing problem and the halting problem

Turing described a number of mathematical problems that cannot be solved by Turing machine. One is the *printing problem*. Some programs print "0" at some stage in their computations; all the remaining programs never print "0." The printing problem is the problem of deciding, given any arbitrarily selected program, into which of these two categories it falls. Turing showed that this problem cannot be solved by the UTM.

The *halting problem* (Davis 1958) is another example of a problem that cannot be solved by the UTM (although not one explicitly considered by Turing). This is the problem of determining, given any arbitrary Turing machine, whether or not the machine will eventually halt when started on a blank tape. The machine shown in table 1.1 is rather obviously one of those that never halts – but in other cases it is definitely not obvious from a machine's table whether or not it halts. And, of course, simply watching the machine run (or a simulation of the machine) is of no help at all, for what can be concluded if after a week or a year the machine has not halted? If the machine does eventually halt, a watching human – or Turing machine – will sooner or later find this out; but in the case of a machine that has not yet halted, there is no effective method for deciding whether or not it is going to halt.

The halting function

A *function* is a mapping from "arguments" (or inputs) to "values" (or outputs). For example, addition (+) is a function that maps pairs of numbers to single numbers: the value of the function + for the pair of arguments 5, 7 is the number 12. The squaring function maps single numbers to single numbers: e.g. the value of n^2 for the argument 3 is 9.

A function is said to be *computable by Turing machine* if some Turing machine will take in arguments of the function (or pairs of arguments, etc.) and, after carrying out some finite number of basic operations, produce the corresponding value – and, moreover, will do this *no matter which* argument of the function is presented. For example, addition over the integers is computable by Turing machine, since a Turing machine can be set up so that whenever two integers are inscribed on its tape (in binary notation, say), the machine will output their sum.

The *halting function* is as follows. Assume the Turing machines to be ordered in some way, so that we may speak of the first machine in the

ordering, the second, and so on. (There are various standard ways of accomplishing this ordering, e.g. in terms of the number of symbols in each machine's instruction table.) The arguments of the halting function are simply 1, 2, 3, ... (Like the squaring function, the halting function takes single arguments.) The value of the halting function for any argument n is 1 if the n^{th} Turing machine in the ordering eventually halts when started on a blank tape, and is 0 if the n^{th} machine runs on forever (as would, for example, a Turing machine programmed to produce in succession the digits of the decimal representation of π).

The theorem that the UTM cannot solve the halting problem is often expressed in terms of the halting function.

Halting theorem: The halting function is not computable by Turing machine.

The Entscheidungsproblem

The *Entscheidungsproblem*, or decision problem, was Turing's principal quarry in "On Computable Numbers." The decision problem was brought to the fore of mathematics by the German mathematician David Hilbert (who in a lecture given in Paris in 1900 set the agenda for much of twentieth-century mathematics). Hilbert and his followers held that mathematicians should seek to express mathematics in the form of a complete, consistent, decidable formal system – a system expressing "the entire thought-content of mathematics in a uniform way" (Hilbert 1927: 475). The project of formulating mathematics in this way became known as the "Hilbert program."

A consistent system is one that contains no contradictions; a complete system one in which every true mathematical statement is provable. "Decidable" means that there is an effective method for telling, of each mathematical statement, whether or not the statement is provable in the system. A complete, consistent, decidable system would banish ignorance from mathematics. Given any mathematical statement, one would be able to tell whether the statement is true or false by deciding whether or not it is provable in the system. Hilbert famously declared in his Paris lecture: "in mathematics there is no *ignorabimus*" (there is no *we shall not know*) (Hilbert 1902: 445).

It is important that the system expressing the "whole thought content of mathematics" be consistent. An inconsistent system – a system containing contradictions – is worthless, since *any* statement whatsoever, true or false, can be derived from a contradiction by simple logical steps. So in an inconsistent system, absurdities such as $0 = 1$ and $6 \neq 6$ are provable. An inconsistent system would indeed contain all true mathematical statements – would be complete, in other words – but would in addition also contain all false mathematical statements.

If ignorance is to be banished absolutely, the system must be decidable. An undecidable system might on occasion leave us in ignorance. Only if the mathematical system were decidable could we be confident of always being able to tell whether or not any given statement is provable. Unfortunately for the Hilbert program, however, it became clear that most interesting mathematical systems are, if consistent, incomplete and undecidable.

In 1931 Gödel showed that Hilbert's ideal is impossible to satisfy, even in the case of simple arithmetic. He proved that the system called Peano arithmetic is, if consistent, incomplete. This is known as Gödel's *first incompleteness theorem*. (Gödel later generalized this result, pointing out that "due to A. M. Turing's work, a precise and unquestionably adequate definition of the general concept of formal system can now be given," with the consequence that incompleteness can "be proved rigorously for *every* consistent formal system containing a certain amount of finitary number theory" (Gödel 1965: 71).) Gödel had shown that no matter how hard mathematicians might try to construct the all-encompassing formal system envisaged by Hilbert, the product of their labors would, if consistent, inevitably be incomplete. As Hermann Weyl – one of Hilbert's greatest pupils – observed, this was nothing less than "a catastrophe" for the Hilbert program (Weyl 1944: 644).

Gödel's theorem does not mention decidability. This aspect was addressed by Turing and by Church. Each showed, working independently, that no consistent formal system of arithmetic is

decidable. They showed this by proving that not even the weaker, purely logical system presupposed by any formal system of arithmetic and called the *first-order predicate calculus* is decidable. Turing's way of proving that the first-order predicate calculus is undecidable involved the printing problem. He showed that if a Turing machine could tell, of any given statement, whether or not the statement is provable in the first-order predicate calculus, then a Turing machine could tell, of any given Turing machine, whether or not it ever prints "0." Since, as he had already established, no Turing machine can do the latter, it follows that no Turing machine can do the former. The final step of the argument is to apply Turing's thesis: if no Turing machine can perform the task in question, then there is no effective method for performing it. The Hilbertian dream lay in total ruin.

Poor news though Turing's and Church's result was for the Hilbert school, it was welcome news in other quarters, for a reason that Hilbert's illustrious pupil von Neumann had given in 1927 (von Neumann 1927: 12):

> If undecidability were to fail then mathematics, in today's sense, would cease to exist; its place would be taken by a completely mechanical rule, with the aid of which any man would be able to decide, of any given statement, whether the statement can be proven or not.

In a similar vein, the Cambridge mathematician G. H. Hardy said in a lecture in 1928 (Hardy 1929: 16):

> if there were . . . a mechanical set of rules for the solution of all mathematical problems . . . our activities as mathematicians would come to an end.

The next section is based on Copeland 1996.

Misunderstandings of the Church–Turing Thesis: The Limits of Machines

A myth has arisen concerning Turing's work, namely that he gave a treatment of the limits of mechanism, and established a fundamental result to the effect that the UTM can simulate the behavior of *any* machine. The myth has passed into the philosophy of mind, theoretical psychology, cognitive science, Artificial Intelligence, and Artificial Life, generally to pernicious effect. For example, the *Oxford Companion to the Mind* states: "Turing showed that his very simple machine . . . can specify the steps required for the solution of any problem that can be solved by instructions, explicitly stated rules, or procedures" (Gregory 1987: 784). Dennett maintains that "Turing had proven – and this is probably his greatest contribution – that his Universal Turing machine can compute any function that any computer, with any architecture, can compute" (1991: 215); also that every "task for which there is a clear recipe composed of simple steps can be performed by a very simple computer, a universal Turing machine, the universal recipe-follower" (1978: xviii). Paul and Patricia Churchland assert that Turing's "results entail something remarkable, namely that a standard digital computer, given only the right program, a large enough memory and sufficient time, can compute *any* rule-governed input–output function. That is, it can display any systematic pattern of responses to the environment whatsoever" (1990: 26). Even Turing's biographer, Hodges, has endorsed the myth:

> Alan had . . . discovered something almost . . . miraculous, the idea of a universal machine that could take over the work of *any* machine. (Hodges 1992: 109)

Turing did not show that his machines can solve any problem that can be solved "by instructions, explicitly stated rules, or procedures," and nor did he prove that the UTM "can compute any function that any computer, with any architecture, can compute" or perform any "task for which there is a clear recipe composed of simple steps." As previously explained, what he proved is that the UTM can carry out any task that any *Turing machine* can carry out. Each of the claims just quoted says considerably more than this.

If what the Churchlands assert were true, then the view that psychology must be capable of being expressed in standard computational terms

would be secure (as would a number of other controversial claims). But Turing had no result entailing that "a standard digital computer . . . can compute *any* rule-governed input–output function." What he did have was a result entailing the exact opposite. The theorem that no Turing machine can decide the predicate calculus entails that there are rule-governed input–output functions that no Turing machine is able to compute – for example, the function whose output is 1 whenever the input is a statement that is provable in the predicate calculus, and is 0 for all other inputs. There are certainly possible patterns of responses to the environment, perfectly systematic patterns, that no Turing machine can display. One is the pattern of responses just described. The halting function is a mathematical characterization of another such pattern.

Distant cousins of the Church–Turing thesis

As has already been emphasized, the Church–Turing thesis concerns the extent of effective methods. Putting this another way (and ignoring contingencies such as boredom, death, or insufficiency of paper), the thesis concerns what a *human being* can achieve when working by rote with paper and pencil. The thesis carries no implication concerning the extent of what *machines* are capable of achieving (even digital machines acting in accordance with "explicitly stated rules"). For among a machine's repertoire of basic operations, there may be those that no human working by rote with paper and pencil can perform.

Essentially, then, the Church–Turing thesis says that no human computer, or machine that mimics a human computer, can out-compute the UTM. However, a variety of other propositions, very different from this, are from time to time called the Church–Turing thesis (or Church's thesis), sometimes but not always with accompanying hedges such as "strong form" and "physical version." Some examples from the recent literature are given below. This loosening of established terminology is unfortunate, and can easily lead to misunderstandings. In what follows I use the expression "Church–Turing thesis properly so called" for the proposition that Turing and Church themselves endorsed.

> [C]onnectionist models . . . may possibly even challenge the strong construal of Church's Thesis as the claim that the class of well-defined computations is exhausted by those of Turing machines. (Smolensky 1988: 3)

> Church–Turing thesis: If there is a well defined procedure for manipulating symbols, then a Turing machine can be designed to do the procedure. (Henry 1993: 149)

> [I]t is difficult to see how any language that could actually be run on a physical computer could do more than Fortran can do. The idea that there is no such language is called Church's thesis. (Geroch & Hartle 1986: 539)

> The first aspect that we examine of Church's Thesis . . . [w]e can formulate, more precisely: The behaviour of any discrete physical system evolving according to local mechanical laws is recursive. (Odifreddi 1989: 107)

> I can now state the physical version of the Church–Turing principle: "Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means." This formulation is both better defined and more physical than Turing's own way of expressing it. (Deutsch 1985: 99)

> That there exists a most general formulation of machine and that it leads to a unique set of input–output functions has come to be called *Church's thesis.* (Newell 1980: 150)

The maximality thesis

It is important to distinguish between the Church–Turing thesis properly so called and what I call the "maximality thesis" (Copeland 2000). (Among the few writers to distinguish explicitly between Turing's thesis and stronger propositions along the lines of the maximality thesis are Gandy 1980 and Sieg 1994.)

A machine *m* is said to be able to *generate* a certain function if *m* can be set up so that if *m* is

presented with any of the function's arguments, *m* will carry out some finite number of atomic processing steps at the end of which *m* produces the corresponding value of the function (*mutatis mutandis* in the case of functions that, like addition, demand more than one argument).

> **Maximality Thesis**: All functions that can be generated by machines (working on finite input in accordance with a finite program of instructions) are computable by Turing machine.

The maximality thesis ("thesis M") admits of two interpretations, according to whether the phrase "can be generated by machine" is taken in the this-worldly sense of "can be generated by a machine that conforms to the physical laws (if not to the resource constraints) of the actual world," or in a sense that abstracts from whether or not the envisaged machine could exist in the actual world. Under the latter interpretation, thesis M is false. It is straightforward to describe abstract machines that generate functions that cannot be generated by the UTM (see e.g. Abramson 1971, Copeland 2000, Copeland & Proudfoot 2000, Stewart 1991). Such machines are termed "hypercomputers" in Copeland and Proudfoot (1999a).

It is an open empirical question whether or not the this-worldly version of thesis M is true. Speculation that there may be physical processes – and so, potentially, machine-operations – whose behavior conforms to functions not computable by Turing machine stretches back over at least five decades. (Copeland & Sylvan 1999 is a survey; see also Copeland & Proudfoot 1999b.)

A source of potential misunderstanding about the limits of machines lies in the difference between the technical and everyday meanings of the word "mechanical." As previously remarked, in technical contexts "mechanical" and "effective" are often used interchangeably. (Gandy 1988 outlines the history of this usage of the word "mechanical.") For example:

> Turing proposed that a certain class of abstract machines could perform any "mechanical" computing procedure. (Mendelson 1964: 229)

Understood correctly, this remark attributes to Turing not a thesis concerning the limits of what can be achieved by machine but the Church–Turing thesis properly so called.

The technical usage of "mechanical" tends to obscure the possibility that there may be machines, or biological organs, that generate (or compute, in a broad sense) functions that cannot be computed by Turing machine. For the question "Can a machine execute a procedure that is not mechanical?" may appear self-answering, yet this is precisely what is asked if thesis M is questioned.

In the technical literature, the word "computable" is often tied by definition to effectiveness: a function is said to be computable if and only if there is an effective method for determining its values. The Church–Turing thesis then becomes:

> Every computable function can be computed by Turing machine.

Corollaries such as the following are sometimes stated:

> [C]ertain functions are uncomputable in an absolute sense: uncomputable even by [Turing machine], and, therefore, uncomputable by any past, present, or future real machine. (Boolos & Jeffrey 1980: 55)

When understood in the sense in which it is intended, this remark is perfectly true. However, to a casual reader of the technical literature, such statements may appear to say more than they in fact do.

Of course, the decision to tie the term "computable" and its cognates to the concept of effectiveness does not settle the truth-value of thesis M. Those who abide by this terminological decision will not describe a machine that falsifies thesis M as *computing* the function that it generates.

Putnam is one of the few writers on the philosophy of mind to question the proposition that Turing machines provide a maximally general formulation of the notion of machine:

> [M]aterialists are committed to the view that a human being is – at least metaphorically – a machine. It is understandable that the notion of a Turing machine might be seen as just a

> way of making this materialist idea precise. Understandable, but hardly well thought out. The problem is the following: a "machine" in the sense of a physical system obeying the laws of Newtonian physics need not be a Turing machine. (Putnam 1992: 4)

The Church–Turing fallacy

To commit what I call the *Church–Turing fallacy* (Copeland 2000, 1998) is to believe that the Church–Turing thesis, or some formal or semi-formal result established by Turing or Church, secures the following proposition:

> If the mind–brain is a machine, then the Turing-machine computable functions provide sufficient mathematical resources for a full account of human cognition.

Perhaps some who commit this fallacy are misled purely by the terminological practice already mentioned, whereby a thesis concerning which there is little real doubt, the Church–Turing thesis properly so called, and a nexus of different theses, some of unknown truth-value, are all referred to as Church's thesis or the Church–Turing thesis.

The Church–Turing fallacy has led to some remarkable claims in the foundations of psychology. For example, one frequently encounters the view that psychology *must* be capable of being expressed ultimately in terms of the Turing machine (e.g. Fodor 1981: 130; Boden 1988: 259). To anyone in the grip of the Church–Turing fallacy, conceptual space will seem to contain no room for mechanical models of the mind–brain that are not equivalent to a Turing machine. Yet it is certainly possible that psychology will find the need to employ models of human cognition that transcend Turing machines (see Chapter 10, COMPUTATIONALISM, CONNECTIONISM, AND THE PHILOSOPHY OF MIND).

The simulation fallacy

A closely related error, unfortunately also common in modern writing on computation and the brain, is to hold that Turing's results somehow entail that the brain, and indeed any biological or physical system whatever, can be *simulated* by a Turing machine. For example, the entry on Turing in *A Companion to the Philosophy of Mind* contains the following claims: "we can depend on there being a Turing machine that captures the functional relations of the brain," for so long as "these relations between input and output are functionally well-behaved enough to be describable by . . . mathematical relationships . . . we know that some specific version of a Turing machine will be able to mimic them" (Guttenplan 1994: 595). Even Dreyfus, in the course of *criticizing* the view that "man is a Turing machine," succumbs to the belief that it is a "fundamental truth that every form of 'information processing' (even those which *in practice* can only be carried out on an 'analogue computer') must *in principle* be simulable on a [Turing machine]" (1992: 195).

Searle writes in a similar fashion:

> If the question ["Is consciousness computable?"] asks "Is there some level of description at which conscious processes and their correlated brain processes can be simulated [by a Turing machine]?" the answer is trivially yes. Anything that can be described as a precise series of steps can be simulated [by a Turing machine]. (Searle 1997: 87)

> Can the operations of the brain be simulated on a digital computer? . . . The answer seems to me . . . demonstrably "Yes" . . . That is, naturally interpreted, the question means: Is there some description of the brain such that under that description you could do a computational simulation of the operations of the brain. But given Church's thesis that anything that can be given a precise enough characterization as a set of steps can be simulated on a digital computer, it follows trivially that the question has an affirmative answer. (Searle 1992: 200)

Church's thesis properly so called does *not* say that anything that can be described as a precise series of of steps can be simulated by Turing machine.

Similarly, Johnson-Laird and the Churchlands argue:

> If you assume that [consciousness] is scientifically explicable . . . [and] [g]ranted that the [Church–Turing] thesis is correct, then the final dichotomy rests on Craik's functionalism. If you believe [functionalism] to be false . . . then presumably you hold that consciousness could be modelled in a computer program in the same way that, say, the weather can be modelled . . . If you accept functionalism, however, then you should believe that consciousness is a computational process. (Johnson-Laird 1987: 252)

> Church's Thesis says that whatever is computable is Turing computable. Assuming, with some safety, that what the mind-brain does is computable, then it can in principle be simulated by a computer. (Churchland & Churchland 1983: 6)

As previously mentioned, the Churchlands believe, incorrectly, that Turing's "results entail . . . that a standard digital computer, given only the right program, a large enough memory and sufficient time, can . . . display any systematic pattern of responses to the environment whatsoever" (1990: 26). This no doubt explains why they think they can assume "with some safety" that what the mind–brain does is computable, for on their understanding of matters, this is to assume only that the mind–brain is characterized by a "rule-governed" (1990: 26) input–output function.

The Church–Turing thesis properly so called does not entail that the brain (or the mind, or consciousness) can be simulated by a Turing machine, not even in conjunction with the belief that the brain (or mind, etc.) is scientifically explicable, or exhibits a systematic pattern of responses to the environment, or is "rule-governed" (etc.). Each of the authors quoted seems to be assuming the truth of a close relative of thesis M, which I call "thesis S" (Copeland 2000).

> **Thesis S**: Any process that can be given a mathematical description (or a "precise enough characterization as a set of steps," or that is scientifically describable or scientifically explicable) can be simulated by a Turing machine.

As with thesis M, thesis S is trivially false if it is taken to concern all conceivable processes, and its truth-value is unknown if it is taken to concern only processes that conform to the physics of the real world. For all we presently know, a completed neuroscience may present the mind–brain as a machine that – when abstracted out from sources of inessential boundedness, such as mortality – generates functions that no Turing machine can generate.

The equivalence fallacy

Paramount among the evidence for the Church–Turing thesis properly so called is the fact that all attempts to give an exact analysis of the intuitive notion of an effective method have turned out to be *equivalent*, in the sense that each analysis has been proved to pick out the same class of functions, namely those that are computable by Turing machine. (For example, there have been analyses in terms of lambda-definability, recursivenes, register machines, Post's canonical and normal systems, combinatory definability, Markov algorithms, and Gödel's notion of reckonability.) Because of the diversity of these various analyses, their equivalence is generally considered very strong evidence for the Church–Turing thesis (although for a skeptical point of view see Kreisel 1965: 144).

However, the equivalence of these diverse analyses is sometimes taken to be evidence also for stronger theses like M and S. This is nothing more than a confusion – the *equivalence fallacy* (Copeland 2000). The analyses under discussion are of the notion of an effective method, not of the notion of a machine-generable function; the equivalence of the analyses bears only on the issue of the extent of the former notion and indicates nothing concerning the extent of the latter.

Artificial intelligence and the equivalence fallacy

Newell, discussing the possibility of artificial intelligence, argues that (what he calls) a "physical symbol system" can be organized to exhibit

general intelligence. A "physical symbol system" is a universal Turing machine, or any equivalent system, situated in the physical – as opposed to the conceptual – world. (The tape of the machine is accordingly finite; Newell specifies that the storage capacity of the tape [or equivalent] be unlimited in the practical sense of finite yet not small enough to "force concern.")

> A [physical symbol] system always contains the potential for being any other system if so instructed. Thus, a [physical symbol] system can become a generally intelligent system. (Newell 1980: 170)

Is the premise of this pro-AI argument true? A physical symbol system, being a *universal* Turing machine situated in the real world, can, if suitably instructed, simulate (or, metaphorically, become) any other physical symbol system (*modulo* some fine print concerning storage capacity). If this is what the premise means, then it is true. However, if taken literally, the premise is false, since as previously remarked, systems can be specified which no Turing machine – and so no physical symbol system – can simulate. However, if the premise is interpreted in the former manner, so that it is true, the conclusion fails to follow from the premise. Only to one who believes, as Newell does, that "the notion of machine or determinate physical mechanism" is "formalized" by the notion of a Turing machine (ibid.) will the argument appear deductively valid.

Newell's defense of his view that the universal Turing machine exhausts the possibilities of mechanism involves an example of the equivalence fallacy:

> [An] important chapter in the theory of computing . . . has shown that all attempts to . . . formulate . . . general notions of mechanism . . . lead to classes of machines that are equivalent in that they encompass in toto exactly the same set of input–output functions. In effect, there is a single large frog pond of functions no matter what species of frogs (types of machines) is used. . . . A large zoo of different formulations of maximal classes of machines is known by now – Turing machines, recursive functions, Post canonical systems, Markov algorithms . . . (Newell 1980: 150)

Newell's *a priori* argument for the claim that a physical symbol system can become generally intelligent founders in confusion.

Conclusion

Since there are problems that cannot be solved by Turing machine, there are – given the Church–Turing thesis – limits to what can be accomplished by any form of machine that works in accordance with effective methods. However, not all *possible* machines share those limits. It is an open empirical question whether there are actual deterministic physical processes that, in the long run, elude simulation by Turing machine; and, if so, whether any such processes could usefully be harnessed in some form of calculating machine. It is, furthermore, an open empirical question whether any such processes are involved in the working of the human brain.

References

Abramson, F. G. 1971. "Effective computation over the real numbers." *Twelfth Annual Symposium on Switching and Automata Theory*. Northridge, CA: Institute of Electrical and Electronics Engineers.

Boden, M. A. 1988. *Computer Models of Mind*. Cambridge: Cambridge University Press.

Boolos, G. S. and Jeffrey, R. C. 1980. *Computability and Logic*, 2nd ed. Cambridge: Cambridge University Press.

Church, A. 1936a. "An unsolvable problem of elementary number theory." *American Journal of Mathematics* 58: 345–63.

——. 1936b. "A note on the Entscheidungsproblem." *Journal of Symbolic Logic* 1: 40–1.

——. 1937. Review of Turing 1936. *Journal of Symbolic Logic* 2: 42–3.

Churchland, P. M. and Churchland, P. S. 1983. "Stalking the wild epistemic engine." *Nous* 17: 5–18.

—— and ——. 1990. "Could a machine think?" *Scientific American* 262 (Jan.): 26–31.

Copeland, B. J. 1996. "The Church–Turing Thesis." In E. Zalta, ed., *The Stanford Encyclopaedia of Philosophy*, <http://plato.stanford.edu>.

——. 1998. "Turing's O-machines, Penrose, Searle, and the Brain." *Analysis* 58: 128–38.
——. 2000. "Narrow versus wide mechanism, including a re-examination of Turing's views on the mind–machine issue." *Journal of Philosophy* 97: 5–32. Repr. in M. Scheutz, ed., *Computationalism: New Directions*. Cambridge, MA: MIT Press, 2002.
——. 2001. "Colossus and the dawning of the computer age." In M. Smith and R. Erskine, eds., *Action This Day*. London: Bantam.
——. and Proudfoot, D. 1999a. "Alan Turing's forgotten ideas in computer science." *Scientific American* 280 (April): 76–81.
—— and ——. 1999b. "The legacy of Alan Turing." *Mind* 108: 187–95.
—— and ——. 2000. "What Turing did after he invented the universal Turing machine." *Journal of Logic, Language, and Information* 9: 491–509.
—— and Sylvan, R. 1999. "Beyond the universal Turing machine." *Australasian Journal of Philosophy* 77: 46–66.
Davis, M. 1958. *Computability and Unsolvability*. New York: McGraw-Hill.
Dennett, D. C. 1978. *Brainstorms: Philosophical Essays on Mind and Psychology*. Brighton: Harvester.
——. 1991. *Consciousness Explained*. Boston: Little, Brown.
Deutsch, D. 1985. "Quantum theory, the Church–Turing principle and the universal quantum computer." *Proceedings of the Royal Society*, Series A, 400: 97–117.
Dreyfus, H. L. 1992. *What Computers Still Can't Do: A Critique of Artificial Reason*. Cambridge, MA: MIT Press.
Fodor, J. A. 1981. "The mind–body problem." *Scientific American* 244 (Jan.): 124–32.
Gandy, R. 1980. "Church's thesis and principles for mechanisms." In J. Barwise, H. Keisler, and K. Kunen, eds., *The Kleene Symposium*. Amsterdam: North-Holland.
——. 1988. "The confluence of ideas in 1936." In R. Herken, ed., *The Universal Turing Machine: A Half-century Survey*. Oxford: Oxford University Press.
Geroch, R. and Hartle, J. B. 1986. "Computability and physical theories." *Foundations of Physics* 16: 533–50.
Gödel, K. 1965. "Postscriptum." In M. Davis, ed., *The Undecidable*. New York: Raven, pp. 71–3.
Gregory, R. L. 1987. *The Oxford Companion to the Mind*. Oxford: Oxford University Press.
Guttenplan, S. 1994. *A Companion to the Philosophy of Mind*. Oxford: Blackwell.
Hardy, G. H. 1929. "Mathematical proof." *Mind* 38: 1–25.
Henry, G. C. 1993. *The Mechanism and Freedom of Logic*. Lanham, MD: University Press of America.
Hilbert, D. 1902. "Mathematical problems: lecture delivered before the International Congress of Mathematicians at Paris in 1900." *Bulletin of the American Mathematical Society* 8: 437–79.
——. 1927. "Die Grundlagen der Mathematik" [The Foundations of Mathematics]. English trans. in J. van Heijenoort, ed., *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Cambridge, MA: Harvard University Press, 1967.
Hodges, A. 1992. *Alan Turing: The Enigma*. London: Vintage.
Johnson-Laird, P. 1987. "How could consciousness arise from the computations of the brain?" In C. Blakemore and S. Greenfield, eds., *Mindwaves*. Oxford: Blackwell.
Kalmár, L. 1959. "An argument against the plausibility of Church's thesis." In A. Heyting, ed., *Constructivity in Mathematics*. Amsterdam: North-Holland.
Kleene, S. C. 1952. *Introduction to Metamathematics*. Amsterdam: North-Holland.
——. 1967. *Mathematical Logic*. New York: Wiley.
Kreisel, G. 1965. "Mathematical logic." In T. L. Saaty, ed., *Lectures on Modern Mathematics*, vol. 3. New York: Wiley.
Langton, C. R. 1989. "Artificial life." In Langton, ed., *Artificial Life*. Redwood City: Addison-Wesley.
Mendelson, E. 1964. *Introduction to Mathematical Logic*. New York: Van Nostrand.
Newell, A. 1980. "Physical symbol systems." *Cognitive Science* 4: 135–83.
Odifreddi, P. 1989. *Classical Recursion Theory*. Amsterdam: North-Holland.
Péter, R. 1959. "Rekursivität und Konstruktivität." In A. Heyting, ed., *Constructivity in Mathematics*. Amsterdam: North-Holland.
Putnam, H. 1992. *Renewing Philosophy*. Cambridge, MA: Harvard University Press.
Searle, J. 1992. *The Rediscovery of the Mind*. Cambridge, MA: MIT Press.
——. 1997. *The Mystery of Consciousness*. New York: New York Review of Books.
Sieg, W. 1994. "Mechanical procedures and mathematical experience." In A. George, ed.,

Mathematics and Mind. Oxford: Oxford University Press.

Sipser, M. 1997. *Introduction to the Theory of Computation*. Boston: PWS Publishing.

Smolensky, P. 1988. "On the proper treatment of connectionism." *Behavioral and Brain Sciences* 11: 1–23.

Stewart, I. 1991. "Deciding the undecidable." *Nature* 352: 664–5.

Turing, A. M. 1936. "On computable numbers, with an application to the Entscheidungsproblem." *Proceedings of the London Mathematical Society*, series 2, 42 (1936–7): 230–65.

——. 1937. "Computability and λ-definability." *Journal of Symbolic Logic* 2: 156–63.

——. 1948. "Intelligent machinery." National Physical Laboratory Report. In B. Meltzer and D. Michie, eds., *Machine Intelligence 5*. Edinburgh: Edinburgh University Press, 1969. [A digital facsimile is available in The Turing Archive for the History of Computing, <http://www.AlanTuring.net/intelligent_machinery>.]

——. 1950. "Programmers' handbook for Manchester electronic computer." University of Manchester Computing Laboratory. [A digital facsimile is available in The Turing Archive for the History of Computing, <http://www.AlanTuring. net/programmers_handbook>.]

von Neumann, J. 1927. "Zur Hilbertschen Beweistheorie" [On Hilbert's proof theory], *Mathematische Zeitschrift* 26: 1–46.

Weyl, H. 1944. "David Hilbert and his Mathematical Work," *Bulletin of the American Mathematical Society* 50: 612–54.

Wittgenstein, L. 1980. *Remarks on the Philosophy of Psychology*, vol. 1. Oxford: Blackwell.