Computable Numbers: A Guide

Jack Copeland

'On Computable Numbers, with an Application to the Entscheidungsproblem' appeared in the *Proceedings of the London Mathematical Society* in 1936.[1] This,

[1] *Proceedings of the London Mathematical Society*, 42 (1936–7), 230–65. The publication date of 'On Computable Numbers' is sometimes cited, incorrectly, as 1937. The article was published in two parts, both parts appearing in 1936. The break between the two parts occurred, rather inelegantly, in the middle of Section 5, at the bottom of p. 240 (p. 67 in the present volume). Pages 230–40 appeared in part 3 of volume 42, issued on 30 Nov. 1936, and the remainder of the article appeared in part 4, issued on 23 Dec. 1936. This information is given on the title pages of parts 3 and 4 of volume 42, which show the contents of each part and their dates of issue. (I am grateful to Robert Soare for sending me these pages. See R. I. Soare, 'Computability and Recursion', *Bulletin of Symbolic Logic*, 2 (1996), 284–321.)

The article was published bearing the information 'Received 28 May, 1936.—Read 12 November, 1936.' However, Turing was in the United States on 12 November, having left England in September 1936 for what was to be a stay of almost two years (see the introductions to Chapters 3 and 4). Although papers were read at the meetings of the London Mathematical Society, many of those published in the *Proceedings* were 'taken as read', the author not necessarily being present at the meeting in question. Mysteriously, the minutes of the meeting held on 18 June 1936 list 'On Computable Numbers, with an Application to the Entscheidungs-problem' as one of 22 papers taken as read at that meeting. The minutes of an Annual General Meeting held

Turing's second publication,[2] contains his most significant work. Here he pioneered the theory of computation, introducing the famous abstract computing machines soon dubbed 'Turing machines' by the American logician Alonzo Church.[3] 'On Computable Numbers' is regarded as the founding publication of the modern science of computing. It contributed vital ideas to the development, in the 1940s, of the electronic stored-programme digital computer. 'On Computable Numbers' is the birthplace of the fundamental principle of the modern computer, the idea of controlling the machine's operations by means of a programme of coded instructions stored in the computer's memory.

In addition Turing charted areas of mathematics lying beyond the scope of the Turing machine. He proved that not all precisely stated mathematical problems can be solved by computing machines. One such is the *Entscheidungsproblem* or 'decision problem'. This work—together with contemporaneous work by Church[4] —initiated the important branch of mathematical logic that investigates and codifies problems 'too hard' to be solvable by Turing machine.

In this one article, Turing ushered in both the modern computer and the mathematical study of the uncomputable.

Part I The Computer

1. Turing Machines

A Turing machine consists of a scanner and a limitless memory-tape that moves back and forth past the scanner. The tape is 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').

The scanner contains mechanisms that enable it to *erase* the symbol on the scanned square, to *print* a symbol on the scanned square, and to *move* the tape to the left or right, one square at a time.

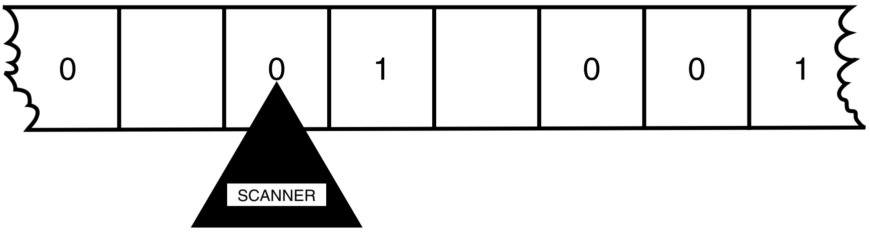
In addition to the operations just mentioned, the scanner is able to alter what Turing calls its '*m*-configuration'. In modern Turing-machine jargon it is usual to

on 12 Nov. 1936 contain no reference to the paper. (I am grateful to Janet Foster, Archives Consultant to the London Mathematical Society, for information.)

[2] The first was 'Equivalence of Left and Right Almost Periodicity', *Journal of the London Mathematical Society*, 10 (1935), 284–5.

[3] Church introduced the term 'Turing machine' in a review of Turing's paper in the *Journal of Symbolic Logic*, 2 (1937), 42–3.

[4] A. Church, 'An Unsolvable Problem of Elementary Number Theory', *American Journal of Mathematics*, 58 (1936), 345–63, and 'A Note on the Entscheidungsproblem', *Journal of Symbolic Logic*, 1 (1936), 40–1.



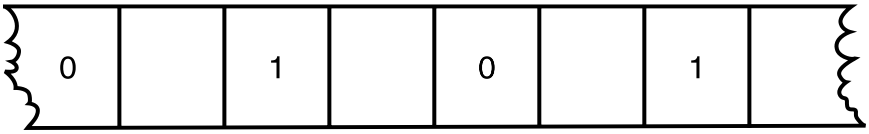
use the term 'state' in place of '*m*-configuration'. A device within the scanner is capable of adopting a number of different states (*m*-configurations), and the scanner is able to alter the state of this device whenever necessary. The device may be conceptualized as consisting of a dial with a (finite) number of positions, labelled 'a', 'b', 'c', etc. Each of these positions counts as an *m*-configuration or state, and changing the *m*-configuration or state amounts to shifting the dial's pointer from one labelled position to another. This device functions as a simple memory. As Turing says, 'by altering its *m*-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously' (p. 59). 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.)

The operations just described—erase, print, move, and change state—are the *basic* (or *atomic*) operations of the Turing machine. Complexity of operation is achieved by chaining together large numbers of these simple basic actions. 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, however, that 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, with unlimited memory, they are capable of computations that no actual computer could perform in practice.

Example of a Turing machine

The following simple example is from Section 3 of 'On Computable Numbers' (p. 61). The once-fashionable Gothic symbols that Turing used in setting out the example—and also elsewhere in 'On Computable Numbers'—are not employed in this guide. I also avoid typographical conventions used by Turing that seem likely to hinder understanding (for example, his special symbol 'ə', which he used to mark the beginning of the tape, is here replaced by '!').

The machine in Turing's example—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, the scanner will print alternating binary digits on the tape, 0 1 0 1 0 1 . . . , working to the right from its starting place, and leaving a blank square in between each digit:



In order to do its work, **M** makes use of four states or *m*-configurations. These are labelled '**a**', '**b**', '**c**', and '**d**'. (Turing employed less familiar characters.) **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 (Table 1). 'R' abbreviates the instruction 'reposition the scanner one square to the right'. This is achieved by moving the tape one square to the left. 'L' abbreviates 'reposition the scanner one square to the left', 'P[0]' abbreviates 'print 0 on the scanned square', and likewise 'P[1]'. Thus the top line of Table 1 reads: if you are in state **a** and the square you are scanning is blank, then print 0 on the scanned square, move the scanner one square to the right, and go into state **b**.

A machine acting in accordance with this table of instructions—or *programme*—toils endlessly on, printing the desired sequence of digits while leaving alternate squares blank.

Turing does not explain how it is to be brought about that the machine acts in accordance with the instructions. There is no need. Turing's machines are abstractions and it is not necessary to propose any specific mechanism for causing the machine to act in accordance with 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. Other ways of setting up the 'switchboard' cause the machine to act in accordance with other tables of instructions. In fact, the earliest electronic digital computers, the British Colossus (1943) and the American ENIAC (1945), were programmed in very much this way. Such machines are described as 'programme-controlled', in order to distinguish them from the modern 'stored-programme' computer.

Table 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**

As everyone who can operate a personal computer knows, the way to set up a stored-programme machine to perform some desired task is to open the appropriate programme of instructions stored in the computer's memory. The stored-programme concept originates with Turing's *universal* computing machine, described in detail in Section 4 of this guide. By inserting different programmes into the memory of the universal machine, the machine is made to carry out different computations. Turing's 1945 technical report 'Proposed Electronic Calculator' was the first relatively complete specification of an electronic stored-programme digital computer (see Chapter 9).

E-squares and F-squares

After describing **M** and a second example of a computing machine, involving the start-of-tape marker '!' (p. 62), Turing introduces a convention which he makes use of later in the article (p. 63). Since the tape is the machine's general-purpose storage medium—serving not only as the vehicle for data storage, input, and output, but also as 'scratchpad' for use during the computation—it is useful to divide up the tape in some way, so that the squares used as scratchpad are distinguished from those used for the various other functions just mentioned.

Turing's convention is that every alternate square of the tape serves as scratchpad. These he calls the 'E-squares', saying that the 'symbols on E-squares will be liable to erasure' (p. 63). The remaining squares he calls 'F-squares'. ('E' and 'F' perhaps stand for 'erasable' and 'fixed'.)

In the example just given, the 'F-squares' of **M**'s tape are the squares bearing the desired sequence of binary digits, 0 1 0 1 0 1... In between each pair of adjacent F-squares lies a blank E-square. The computation in this example is so simple that the E-squares are never used. More complex computations make much use of E-squares.

Turing mentions one important use of E-squares at this point (p. 63): any F-square can be 'marked' by writing some special symbol, e.g. '$*$', on the E-square immediately to its right. By this means, the scanner is able to find its way back to a particular string of binary digits—a particular item of data, say. The scanner locates the first digit of the string by finding the marker '$*$'.

Adjacent blank squares

Another useful convention, also introduced on p. 63, is to the effect that the tape must never contain a run of non-blank squares followed by two or more adjacent blank squares that are themselves followed by one or more non-blank squares. The value of this convention is that it gives the machine an easy way of finding the last non-blank square. As soon as the machine finds two adjacent blank squares, it knows that it has passed beyond the region of tape that has been written on and has entered the region of blank squares stretching away endlessly.

The start-of-tape marker

Turing usually considers tapes that are endless in one direction only. For purposes of visualization, these tapes may all be thought of as being endless to the right. By convention, each of the first two squares of the tape bears the symbol '!', mentioned previously. These 'signposts' are never erased. The scanner searches for the signposts when required to find the beginning of the tape.

2. Standard Descriptions and Description Numbers

In the final analysis, a computer programme is simply a (long) stream, or row, of characters. Combinations of characters encode the instructions. In Section 5 of 'On Computable Numbers' Turing explains how an instruction table is to be converted into a row of letters, which he calls a 'standard description'. He then explains how a standard description can be converted into a single number. He calls these 'description numbers'.

Each line of an instruction table can be re-expressed as a single 'word' of the form $\mathbf{q}_i S_j S_k M \mathbf{q}_l$. $\mathbf{q}_i$ is the state shown in the left-hand column of the table. S_j is the symbol on the scanned square (a blank is counted as a type of symbol). S_k is the symbol that is to be printed on the scanned square. M is the direction of movement (if any) of the scanner, left or right. $\mathbf{q}_l$ is the next state. For example, the first line of Table 1 can be written: **a**-0R**b** (using '-' to represent a blank). The third line is: **c**-1R**d**.

The second line of the table, which does not require the contents of the scanned square (a blank) to be changed, is written: **b**--R**c**. That is to say we imagine, for the purposes of this new notation, that the operations column of the instruction table contains the redundant instruction P[-]. This device is employed whenever an instruction calls for no change to the contents of the scanned square, as in the following example:

State	Scanned Square	Operations	Next State
d	x	L	**c**

It is imagined that the operations column contains the redundant instruction P[x], enabling the line to be expressed: **d**xxL**c**.

Sometimes a line may contain no instruction to move. For example:

State	Scanned Square	Operations	Next State
d	∗	P[1]	**c**

The absence of a move is indicated by including 'N' in the instruction-word: **d**∗1N**c**.

Sometimes a line may contain an instruction to erase the symbol on the scanned square. This is denoted by the presence of 'E' in the 'operations' column:

State	Scanned Square	Operations	Next State
m	∗	E, R	**n**

Turing notes that E is equivalent to P[-]. The corresponding instruction-word is therefore **m**∗-R**n**.

Any table of instructions can be rewritten in the form of a stream of instruction-words separated by semicolons.[5] Corresponding to Table 1 is the stream:

a-0R**b**; **b**--R**c**; **c**-1R**d**; **d**--R**a**;

This stream can be converted into a stream consisting uniformly of the letters A, C, D, L, R, and N (and the semicolon). Turing calls this a *standard description* of the machine in question. The process of conversion is done in such a way that the individual instructions can be retrieved from the standard description.

The standard description is obtained as follows. First, '-' is replaced by 'D', '0' by 'DC', and '1' by 'DCC'. (In general, if we envisage an ordering of all the printable symbols, the *n*th symbol in the ordering is replaced by a 'D' followed by *n* repetitions of 'C'.) This produces:

aDDCR**b**; **b**DDR**c**; **c**DDCCR**d**; **d**DDR**a**;

Next, the lower case state-symbols are replaced by letters. '**a**' is replaced by 'DA', '**b**' by 'DAA', '**c**' by 'DAAA', and so on. An obvious advantage of the new notation is that there is no limit to the number of states that can be named in this way.

The standard description corresponding to Table 1 is:

DADDCRDAA; DAADDRDAAA; DAAADDCCRDAAAA; DAAAADDRDA;

Notice that occurrences of 'D' serve to mark out the different segments or regions of each instruction-word. For example, to determine which symbol an instruction-word says to print, find the third 'D' to the right from the beginning of the word, and count the number of occurrences of 'C' between it and the next D to the right.

The standard description can be converted into a number, called a *description number*. Again, the process of conversion is carried out in such a way that the individual instructions can be retrieved from the description number. A standard description is converted into a description number by means of replacing each 'A' by '1', 'C' by '2', 'D' by '3', 'L' by '4', 'R' by '5', 'N' by '6', and ';' by 7. In the case of the above example this produces:

31332531173113353111731113322531111731111335317.[6]

[5] There is a subtle issue concerning the placement of the semicolons. See Davies's 'Corrections to Turing's Universal Computing Machine', Sections 3, 7, 10.

[6] Properly speaking, the description number is not the string '313325311731133531117311133225 31111731111335317', but is the number denoted by this string of numerals.

Occurrences of '7' mark out the individual instruction-words, and occurrences of '3' mark out the different regions of the instruction-words. For example: to find out which symbol the third instruction-word says to print, find the second '7' (starting from the left), then the third '3' to the right of that '7', and count the number of occurrences of '2' between that '3' and the next '3' to the right. To find out the exit state specified by the third instruction-word, find the last '3' in that word and count the number of occurrences of '1' between it and the next '7' to the right.

Notice that *different* standard descriptions can describe the behaviour of one and the same machine. For example, interchanging the first and second lines of Table 1 does not in any way affect the behaviour of the machine operating in accordance with the table, but a different standard description—and therefore a different description number—will ensue if the table is modified in this way.

This process of converting a table of instructions into a standard description or a description number is analogous to the process of *compiling* a computer programme into 'machine code'. Programmers generally prefer to work in so-called high-level languages, such as Pascal, Prolog, and C. Programmes written in a high-level language are, like Table 1, reasonably easy for a trained human being to follow. Before a programme can be executed, the instructions must be translated, or compiled, into the form required by the computer (machine code).

The importance of standard descriptions and description numbers is explained in what follows.

3. Subroutines

Subroutines are programmes that are used as components of other programmes. A subroutine may itself have subroutines as components. Programmers usually have access to a 'library' of commonly used subroutines—the programmer takes ready-made subroutines 'off the shelf' whenever necessary.

Turing's term for a subroutine was 'subsidiary table'. He emphasized the importance of subroutines in a lecture given in 1947 concerning the Automatic Computing Engine or ACE, the electronic stored-programme computer that he began designing in 1945 (see Chapter 9 and the introduction to Chapter 10):

Probably the most important idea involved in instruction tables is that of standard *subsidiary tables*. Certain processes are used repeatedly in all sorts of different connections, and we wish to use the same instructions . . . every time . . . We have only to think out how [a process] is to be done once, and forget then how it is done.[7]

In 'On Computable Numbers'—effectively the first programming manual of the computer age—Turing introduced a library of subroutines for Turing machines (in Sections 4 and 7), saying (p. 63):

[7] The quotation is from p. 389 below.

There are certain types of process used by nearly all machines, and these, in some machines, are used in many connections. These processes include copying down sequences of symbols, comparing sequences, erasing all symbols of a given form, etc.

Some examples of subroutines are:

cpe(A, B, x, y) (p. 66):

'**cpe**' may be read 'compare for equality'. This subroutine compares the string of symbols marked with an x to the string of symbols marked with a y. The subroutine places the machine in state B if the two strings are the same, and in state A if they are different. Note: throughout these examples, 'A' and 'B' are variables representing any states; 'x' and 'y' are variables representing any symbols.

f(A, B, x) (p. 63):

'**f**' stands for 'find'. This subroutine finds the leftmost occurrence of x. **f**(A, B, x) moves the scanner left until the start of the tape is encountered. Then the scanner is moved to the right, looking for the first x. As soon as an x is found, the subroutine places the machine in state A, leaving the scanner resting on the x. If no x is found anywhere on the portion of tape that has so far been written on, the subroutine places the machine in state B, leaving the scanner resting on a blank square to the right of the used portion of the tape.

e(A, B, x) (p. 64):

'**e**' stands for 'erase'. The subroutine **e**(A, B, x) contains the subroutine **f**(A, B, x). **e**(A, B, x) finds the leftmost occurrence of symbol x and erases it, placing the machine in state A and leaving the scanner resting on the square that has just been erased. If no x is found the subroutine places the machine in state B, leaving the scanner resting on a blank square to the right of the used portion of the tape.

The subroutine **f**(A, B, x)

It is a useful exercise to construct **f**(A, B, x) explicitly, i.e. in the form of a table of instructions. Suppose we wish the machine to enter the subroutine **f**(A, B, x) when placed in state **n**, say. Then the table of instructions is as shown in Table 2. (Remember that by the convention mentioned earlier, if ever the scanner encounters two adjacent blank squares, it has passed beyond the region of tape that has been written on and has entered the region of blank squares stretching away to the right.)

As Turing explains, **f**(A, B, x) is in effect built out of two further subroutines, which he writes $\mathbf{f}_1$(A, B, x) and $\mathbf{f}_2$(A, B, x). The three rows of Table 2 with an '**m**' in the first column form the subroutine $\mathbf{f}_1$(A, B, x), and the three rows with '**o**' in the first column form $\mathbf{f}_2$(A, B, x).

Skeleton tables

For ease of defining subroutines Turing introduces an abbreviated form of instruction table, in which one is allowed to write expressions referring to

Table 2

State	Scanned Square	Operations	Next State	Comments
n	does not contain !	L	**n**	Search for the first square.
n	!	L	**m**	Found right-hand member of the pair '!!'; move left to first square of tape; go into state **m**. (Notice that x might be '!'.)
m	x	none	A	Found x; go into state A; subroutine ends.
m	neither x nor blank	R	**m**	Keep moving right looking for x or a blank.
m	blank	R	**o**	Blank square encountered; go into state **o** and examine next square to the right.
o	x	none	A	Found x; go into state A; subroutine ends.
o	neither x nor blank	R	**m**	Found a blank followed by a non-blank square but no x; switch to state **m** and keep looking for x.
o	blank	R	B	Two adjacent blank squares encountered; go into state B; subroutine ends.

Table 3

$\mathfrak{f}(A, B, x)$	not !	L	$\mathfrak{f}(A, B, x)$
	!	L	$\mathfrak{f}_1(A, B, x)$
$\mathfrak{f}_1(A, B, x)$	x		A
	neither x nor blank	R	$\mathfrak{f}_1(A, B, x)$
	blank	R	$\mathfrak{f}_2(A, B, x)$
$\mathfrak{f}_2(A, B, x)$	x		A
	neither x nor blank	R	$\mathfrak{f}_1(A, B, x)$
	blank	R	B

subroutines in the first and fourth columns (the state columns). Turing calls these abbreviated tables 'skeleton tables' (p. 63). For example, the skeleton table corresponding to Table 2 is as in Table 3.

Turing's notation for subroutines is explained further in the appendix to this guide ('Subroutines and *m*-functions').

4. The Universal Computing Machine

In Section 7 of 'On Computable Numbers' Turing introduces his 'universal computing machine', now known simply as the universal Turing machine. The universal Turing machine is the stored-programme digital computer in abstract conceptual form.

The universal computing machine has a single, fixed table of instructions (which we may imagine to have been set into the machine, once and for all, by way of the switchboard-like arrangement mentioned earlier). Operating in accordance with this table of instructions, the universal machine is able to carry out *any* task for which an instruction table can be written. The trick is to put an instruction table—programme—for carrying out the desired task onto the tape of the universal machine.

The instructions are placed on the tape in the form of a standard description—i.e. in the form of a string of letters that encodes the instruction table. The universal machine reads the instructions and carries them out on its tape.

The universal Turing machine and the modern computer

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

- The idea of controlling the function of a computing machine by storing a programme of symbolically encoded instructions in the machine's memory.
- His demonstration (in Section 7 of 'On Computable Numbers') 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, i.e. is universal.

Turing's teacher and friend Max Newman has testified that Turing's interest in building a stored-programme computing machine dates from the time of 'On Computable Numbers'. In a tape-recorded interview Newman stated, 'Turing himself, right from the start, said it would be interesting to try and make such a machine'.[8] (It was Newman who, in a lecture on the foundations of mathematics and logic given in Cambridge in 1935, launched Turing on the research that led to the universal Turing machine; see the introduction to Chapter 4.[9]) In his obituary of Turing, Newman wrote:

The description that [Turing] gave of a 'universal' computing machine was entirely theoretical in purpose, but Turing's strong interest in all kinds of practical experiment

[8] Newman in interview with Christopher Evans ('The Pioneers of Computing: An Oral History of Computing', London, Science Museum).

[9] Ibid.

made him even then interested in the possibility of actually constructing a machine on these lines.[10]

Turing later described the connection between the universal computing machine and the stored-programme digital computer in the following way (Chapter 9, pp. 378 and 383):

> Some years ago I was researching on what might now be described as an investigation of the theoretical possibilities and limitations of digital computing machines. I considered a type of machine which had a central mechanism, and an infinite memory which was contained on an infinite tape . . . It can be shown that a single special machine of that type can be made to do the work of all . . . The special machine may be called the universal machine; it works in the following quite simple manner. When we have decided what machine we wish to imitate we punch a description of it on the tape of the universal machine. This description explains what the machine would do in every configuration in which it might find itself. The universal machine has only to keep looking at this description in order to find out what it should do at each stage. Thus the complexity of the machine to be imitated is concentrated in the tape and does not appear in the universal machine proper in any way . . . [D]igital computing machines such as the ACE . . . are in fact practical versions of the universal machine. There is a certain central pool of electronic equipment, and a large memory. When any particular problem has to be handled the appropriate instructions for the computing process involved are stored in the memory of the ACE and it is then 'set up' for carrying out that process.

Turing's idea of a universal stored-programme computing machine was promulgated in the USA by von Neumann and in the UK by Newman, the two mathematicians who, along with Turing himself, were by and large responsible for placing Turing's abstract universal machine into the hands of electronic engineers.

By 1946 several groups in both countries had embarked on creating a universal Turing machine in hardware. The race to get the first electronic stored-programme computer up and running was won by Manchester University where, in Newman's Computing Machine Laboratory, the 'Manchester Baby' ran its first programme on 21 June 1948. Soon after, Turing designed the input/output facilities and the programming system of an expanded machine known as the Manchester Mark I.[11] (There is more information about the Manchester computer in the introductions to Chapters 4, 9, and 10, and in 'Artificial Life'.) A small pilot version of Turing's Automatic Computing Engine first ran in 1950, at the National Physical Laboratory in London (see the introductions to Chapters 9 and 10).

[10] 'Dr. A. M. Turing', *The Times*, 16 June 1954, p. 10.

[11] F. C. Williams described some of Turing's contributions to the Manchester machine in a letter written in 1972 to Brian Randell (parts of which are quoted in B. Randell, 'On Alan Turing and the Origins of Digital Computers', in B. Meltzer and D. Michie (eds.), *Machine Intelligence 7* (Edinburgh: Edinburgh University Press, 1972)); see the introduction to Chapter 9 below. A digital facsimile of Turing's *Programmers' Handbook for Manchester Electronic Computer* (University of Manchester Computing Machine Laboratory, 1950) is in The Turing Archive for the History of Computing <www.AlanTuring.net/programmers_handbook>.

By 1951 electronic stored-programme computers had begun to arrive in the market place. The first model to go on sale was the Ferranti Mark I, the production version of the Manchester Mark I (built by the Manchester firm Ferranti Ltd.). Nine of the Ferranti machines were sold, in Britain, Canada, the Netherlands, and Italy, the first being installed at Manchester University in February 1951.[12] In the United States the first UNIVAC (built by the Eckert-Mauchly Computer Corporation) was installed 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-programme electronic computer to function.[13] 1953 saw the IBM 701, the company's first mass-produced stored-programme electronic computer. A new era had begun.

How the universal machine works

The details of Turing's universal machine, given on pp. 69–72, are moderately complicated. However, the basic principles of the universal machine are, as Turing says, simple.

Let us consider the Turing machine **M** whose instructions are set out in Table 1. (Recall that **M**'s scanner is positioned initially over any square of **M**'s endless tape, the tape being completely blank.) If a standard description of **M** is placed on the universal machine's tape, the universal machine will *simulate* or *mimic* the actions of **M**, and will produce, on specially marked squares of its tape, the output sequence that **M** produces, namely:

$$0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \ldots$$

The universal machine does this by reading the instructions that the standard description contains and carrying them out on its own tape.

In order to start work, the universal machine requires on its tape not only the standard description but also a record of **M**'s intial state (**a**) and the symbol that **M** is initially scanning (a blank). The universal machine's own tape is initially blank except for this record and **M**'s standard description (and some ancillary punctuation symbols mentioned below). As the simulation of **M** progresses, the universal machine prints a record on its tape of:

- the symbols that **M** prints
- the position of **M**'s scanner at each step of the computation
- the symbol 'in' the scanner
- **M**'s state at each step of the computation.

[12] S. Lavington, 'Computer Development at Manchester University', in N. Metropolis, J. Howlett, and G. C. Rota (eds.), *A History of Computing in the Twentieth Century* (New York: Academic Press, 1980).

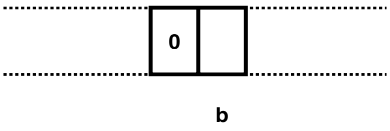
[13] See M. V. Wilkes, *Memoirs of a Computer Pioneer* (Cambridge, Mass.: MIT Press, 1985).

When the universal machine is started up, it reads from its tape **M**'s initial state and initial symbol, and then searches through **M**'s standard description for the instruction beginning: 'when in state **a** and scanning a blank...' The relevant instruction from Table 1 is:

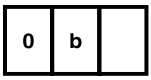
a	blank	P[0], R	**b**

The universal machine accordingly prints '0'. It then creates a record on its tape of **M**'s new state, **b**, and the new position of **M**'s scanner (i.e. immediately to the right of the '0' that has just been printed on **M**'s otherwise blank tape). Next, the universal machine searches through the standard description for the instruction beginning 'when in state **b** and scanning a blank...'. And so on.

How does the universal machine do its record-keeping? After **M** executes its first instruction, the relevant portion of **M**'s tape would look like this—using '**b**' both to record **M**'s state and to indicate the position of the scanner. All the other squares of **M**'s tape to the left and right are blank.



The universal machine keeps a record of this state of affairs by employing three squares of tape (pp. 62, 68):

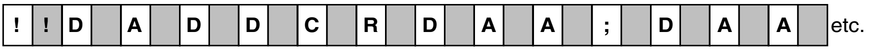


The symbol '**b**' has the double function of recording **M**'s state and indicating the position of **M**'s scanner. The square immediately to the right of the state-symbol displays the symbol 'in' **M**'s scanner (a blank).

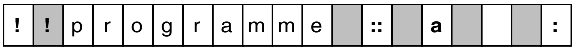
What does the universal machine's tape look like before the computation starts? The standard description corresponding to Table 1 is:

DADDCRDAA; DAADDRDAAA; DAAADDCCRDAAAA; DAAAADDRDA;

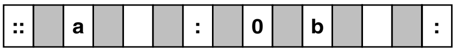
The operator places this programme on the universal machine's tape, writing only on F-squares and beginning on the second F-square of the tape. The first F-square and the first E-square are marked with the start-of-tape symbol '!'. The E-squares (shaded in the diagram) remain blank (except for the first).



On the F-square following the final semicolon of the programme, the operator writes the end-of-programme symbol '::'. On the next F-square to the right of this symbol, the operator places a record of **M**'s initial state, **a**, and leaves the following F-square blank in order to indicate that **M** is initially scanning a blank. The next F-square to the right is then marked with the punctuation symbol '**:**'. This completes the setting-up of the tape:



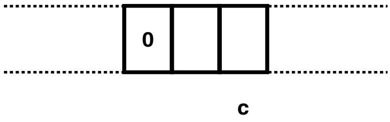
What does the universal machine's tape look like as the computation progresses? In response to the first instruction in the standard description, the universal machine creates the record '**0b-:**' (in describing the tape, '-' will be used to represent a blank) on the next four F-squares to the right of the first '**:**'. Depicting only the portion of tape to the right of the end-of-programme marker '**::**' (and ignoring any symbols which the universal machine may have written on the E-squares in the course of dealing with the first instruction), the tape now looks like this:



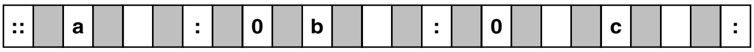
Next the universal machine searches for the instruction beginning 'when in state **b** and scanning a blank . . .'. The relevant instruction from Table 1 is

b blank R **c**

This instruction would put **M** into the condition:



So the universal machine creates the record '**0-c-:**' on its tape:

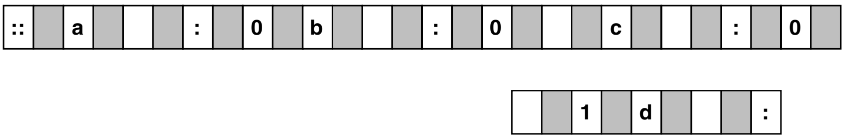


Each pair of punctuation marks frames a representation (on the F-squares) of **M**'s tape extending from the square that was in the scanner at start-up to the furthest square to the right to have been scanned at that stage of the computation.

The next instruction is:

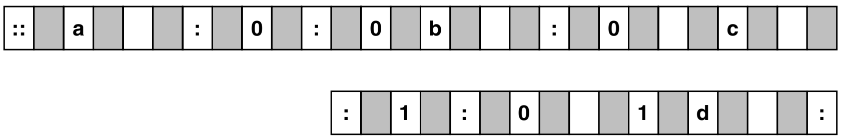
c blank P[1], R **d**

This causes the universal machine to create the record '**0-1d-:**'. (The diagram represents a single continuous strip of tape.)



And so on. Record by record, the outputs produced by the instructions in Table 1 appear on the universal machine's tape.

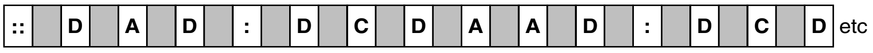
Turing also introduces a variation on this method of record-keeping, whereby the universal machine additionally prints on the tape a second record of the binary digits printed by **M**. The universal machine does this by printing *in front of* each record shown in the above diagram a record of any digit newly printed by **M** (plus an extra colon):



These single digits bookended by colons form a representation of what has been printed by **M** on the F-squares of its tape.

Notice that the record-keeping scheme employed so far requires the universal machine to be able to print each type of symbol that the machine being simulated is able to print. In the case of **M** this requirement is modest, since **M** prints only '0', '1', and the blank. However, if the universal machine is to be able to simulate each of the infinitely many Turing machines, then this record-keeping scheme requires that the universal machine have the capacity to print an endless variety of types of discrete symbol. This can be avoided by allowing the universal machine to keep its record of **M**'s tape in the same notation that is used in forming standard descriptions, namely with 'D' replacing the blank, 'DC' replacing '0', 'DCC' replacing '1', 'DA' replacing '**a**', 'DAA' replacing '**b**', and so on.

The universal machine's tape then looks like this (to the right of the end-of-programme symbol '**::**' and not including the second record of digits printed by **M**):



In this elegant notation of Turing's, 'D' serves to indicate the start of each new term on the universal machine's tape. The letters 'A' and 'C' serve to distinguish terms representing **M**'s states from terms representing symbols on **M**'s tape.

The E-squares and the instruction table

The universal machine uses the E-squares of its tape to mark up each instruction in the standard description. This facilitates the copying that the universal machine must do in order to produce its records of **M**'s activity. For example, the machine temporarily marks the portion of the current instruction specifying **M**'s next state with '*y*' and subsequently the material marked '*y*' is copied to the appropriate place in the record that is being created. The universal machine's records of **M**'s tape are also temporarily marked in various ways.

In Section 7 Turing introduces various subroutines for placing and erasing markers on the E-squares. He sets out the table of instructions for the universal machine in terms of these subroutines. The table contains the detailed instructions for carrying out the record-keeping described above.

In Section 2.4 of Chapter 2 Turing's sometime colleague Donald Davies gives an introduction to these subroutines and to Turing's detailed table of instructions for the universal machine (and additionally corrects some errors in Turing's own formulation).

5. Turing, von Neumann, and the Computer

In the years immediately following the Second World War, the Hungarian-American logician and mathematician John von Neumann—one of the most important and influential figures of twentieth-century mathematics—made the concept of the stored-programme digital computer widely known, through his writings and his charismatic public addresses. In the secondary literature, von Neumann is often said to have himself invented the stored-programme computer. This is an unfortunate myth.

From 1933 von Neumann was on the faculty of the prestigious Institute for Advanced Study at Princeton University. He and Turing became well acquainted while Turing was studying at Princeton from 1936 to 1938 (see the introduction to Chapter 3). In 1938 von Neumann offered Turing a position as his assistant, which Turing declined. (Turing wrote to his mother on 17 May 1938: 'I had an offer of a job here as von Neumann's assistant at $1500 a year but decided not to take it.'[14] His father had advised him to find a job in America,[15] but on 12 April of the same year Turing had written: 'I have just been to see the Dean [Luther Eisenhart] and ask him about possible jobs over here; mostly for Daddy's information, as I think it unlikely I shall take one unless you are actually at war before July. He didn't know of one at present, but said he would bear it all in mind.')

It was during Turing's time at Princeton that von Neumann became familiar with the ideas in 'On Computable Numbers'. He was to become intrigued with Turing's concept of a universal computing machine.[16] It is clear that von

[14] Turing's letters to his mother are among the Turing Papers in the Modern Archive Centre, King's College Library, Cambridge (catalogue reference K 1).

[15] S. Turing, *Alan M. Turing* (Cambridge: Heffer, 1959), 55.

[16] 'I know that von Neumann was influenced by Turing . . . during his Princeton stay before the war,' said von Neumann's friend and colleague Stanislaw Ulam (in an interview with Christopher Evans in 1976; 'The Pioneers of Computing: An Oral History of Computing', Science Museum, London). When Ulam and von Neumann were touring in Europe during the summer of 1938, von Neumann devised a mathematical game involving Turing-machine-like descriptions of numbers (Ulam reported by W. Aspray on pp. 178, 313 of his *John von Neumann and the Origins of Modern Computing* (Cambridge, Mass.: MIT Press, 1990)). The word

Neumann held Turing's work in the highest regard.[17] One measure of his esteem is that the only names to receive mention in his pioneering volume *The Computer and the Brain* are those of Turing and the renowned originator of information theory, Claude Shannon.[18]

The Los Alamos physicist Stanley Frankel—responsible with von Neumann and others for mechanizing the large-scale calculations involved in the design of the atomic and hydrogen bombs—has recorded von Neumann's view of the importance of 'On Computable Numbers':

I know that in or about 1943 or '44 von Neumann was well aware of the fundamental importance of Turing's paper of 1936 'On computable numbers . . .', which describes in principle the 'Universal Computer' of which every modern computer (perhaps not ENIAC as first completed but certainly all later ones) is a realization. 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' (in a modern sense of the term) 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.[19]

In 1944 von Neumann joined the ENIAC group, led by Presper Eckert and John Mauchly at the Moore School of Electrical Engineering (part of the University of Pennsylvania).[20] At this time von Neumann was involved in the Manhattan Project at Los Alamos, where roomfuls of clerks armed with desk calculating machines were struggling to carry out the massive calculations required by the physicists. Hearing about the Moore School's planned computer during a chance encounter on a railway station (with Herman Goldstine), von Neumann immediately saw to it that he was appointed as consultant to the project.[21] ENIAC—under construction since 1943—was, as previously mentioned, a programme-controlled (i.e. not stored-programme) computer: programming consisted of

'intrigued' is used in this connection by von Neumann's colleague Herman Goldstine on p. 275 of his *The Computer from Pascal to von Neumann* (Princeton: Princeton University Press, 1972).)

[17] Turing's universal machine was crucial to von Neumann's construction of a self-reproducing automaton; see the chapter 'Artificial Life', below.

[18] J. von Neumann, *The Computer and the Brain* (New Haven: Yale University Press, 1958).

[19] Letter from Frankel to Brain Randell, 1972 (first published in B. Randell, 'On Alan Turing and the Origins of Digital Computers', in Meltzer and Michie (eds.), *Machine Intelligence 7*. I am grateful to Randell for giving me a copy of this letter.

[20] John Mauchly recalled that 7 September 1944 'was the first day that von Neumann had security clearance to see the ENIAC and talk with Eckert and me' (J. Mauchly, 'Amending the ENIAC Story', *Datamation*, 25/11 (1979), 217–20 (217)). Goldstine (*The Computer from Pascal to von Neumann*, 185) suggests that the date of von Neumann's first visit may have been a month earlier: 'I probably took von Neumann for a first visit to the ENIAC on or about 7 August'.

[21] Goldstine, *The Computer from Pascal to von Neumann*, 182.

rerouting cables and setting switches. Moreover, the ENIAC was designed with only one very specific type of task in mind, the calculation of trajectories of artillery shells. Von Neumann brought his knowledge of 'On Computable Numbers' to the practical arena of the Moore School. Thanks to Turing's abstract logical work, von Neumann knew that by making use of coded instructions stored in memory, a single machine of fixed structure could in principle carry out *any* task for which an instruction table can be written.

Von Neumann gave his engineers 'On Computable Numbers' to read when, in 1946, he established his own project to build a stored-programme computer at the Institute for Advanced Study.[22] Julian Bigelow, von Neumann's chief engineer, recollected:

The person who really . . . pushed the whole field ahead was von Neumann, because he understood logically what [the stored-programme concept] meant in a deeper way than anybody else . . . The reason he understood it is because, among other things, he understood a good deal of the mathematical logic which was implied by the idea, due to the work of A. M. Turing . . . in 1936–1937. . . . Turing's [universal] machine does not sound much like a modern computer today, but nevertheless it was. It was the germinal idea . . . So . . . [von Neumann] saw . . . that [ENIAC] was just the first step, and that great improvement would come.[23]

Von Neumann repeatedly emphasized the fundamental importance of 'On Computable Numbers' in lectures and in correspondence. In 1946 von Neumann wrote to the mathematician Norbert Wiener of 'the great positive contribution of Turing', Turing's mathematical demonstration that 'one, definite mechanism can be "universal" '.[24] In 1948, in a lecture entitled 'The General and Logical Theory of Automata', von Neumann said:

The English logician, Turing, about twelve years ago attacked the following problem. He wanted to give a general definition of what is meant by a computing automaton . . . Turing carried out a careful analysis of what mathematical processes can be effected by automata of this type . . . He . . . also introduce[d] and analyse[d] the concept of a 'universal automaton' . . . An automaton is 'universal' if any sequence that can be produced by any automaton at all can also be solved by this particular automaton. It will, of course, require in general a different instruction for this purpose. *The Main Result of the Turing Theory.* We might expect a priori that this is impossible. How can there be an automaton which is

[22] Letter from Julian Bigelow to Copeland (12 Apr. 2002). See also Aspray, *John von Neumann*, 178.

[23] Bigelow in a tape-recorded interview made in 1971 by the Smithsonian Institution and released in 2002. I am grateful to Bigelow for sending me a transcript of excerpts from the interview.

[24] The letter, dated 29 Nov. 1946, is in the von Neumann Archive at the Library of Congress, Washington, DC. In the letter von Neumann also remarked that Turing had 'demonstrated in absolute . . . generality that anything and everything Brouwerian can be done by an appropriate mechanism' (a Turing machine). He made a related remark in a lecture: 'It has been pointed out by A. M. Turing [in "On Computable Numbers"] . . . that effectively constructive logics, that is, intuitionistic logics, can be best studied in terms of automata' ('Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components', in vol. v of von Neumann's *Collected Works*, ed. A. H. Taub (Oxford: Pergamon Press, 1963), 329).

at least as effective as any conceivable automaton, including, for example, one of twice its size and complexity? Turing, nevertheless, proved that this is possible.[25]

The following year, in a lecture delivered at the University of Illinois entitled 'Rigorous Theories of Control and Information', von Neumann said:

The importance of Turing's research is just this: that if you construct an automaton right, then any additional requirements about the automaton can be handled by sufficiently elaborate instructions. This is only true if [the automaton] is sufficiently complicated, if it has reached a certain minimal level of complexity. In other words . . . there is a very definite finite point where an automaton of this complexity can, when given suitable instructions, do anything that can be done by automata at all.[26]

Von Neumann placed Turing's abstract 'universal automaton' into the hands of American engineers. Yet many books on the history of computing in the United States make no mention of Turing. No doubt this is in part explained by the absence of any explicit reference to Turing's work in the series of technical reports in which von Neumann, with various co-authors, set out a logical design for an electronic stored-programme digital computer.[27] Nevertheless there is evidence in these documents of von Neumann's knowledge of 'On Computable Numbers'. For example, in the report entitled 'Preliminary Discussion of the Logical Design of an Electronic Computing Instrument' (1946), von Neumann and his co-authors, Burks and Goldstine—both former members of the ENIAC group, who had joined von Neumann at the Institute for Advanced Study—wrote the following:

3.0. *First Remarks on the Control and Code*: It is easy to see by formal-logical methods, that there exist codes that are in abstracto adequate to control and cause the execution of any sequence of operations which are individually available in the machine and which are, in their entirety, conceivable by the problem planner. The really decisive considerations from the present point of view, in selecting a code, are more of a practical nature: Simplicity of the equipment demanded by the code, and the clarity of its application to the actually important problems together with the speed of its handling of those problems.[28]

Burks has confirmed that the first sentence of this passage is a reference to Turing's universal computing machine.[29]

[25] The text of 'The General and Logical Theory of Automata' is in vol. v of von Neumann, *Collected Works*; see pp. 313–14.

[26] The text of 'Rigorous Theories of Control and Information' is printed in J. von Neumann, *Theory of Self-Reproducing Automata*, ed. A. W. Burks (Urbana: University of Illinois Press, 1966); see p. 50.

[27] The first papers in the series were the 'First Draft of a Report on the EDVAC' (1945, von Neumann; see n. 31), and 'Preliminary Discussion of the Logical Design of an Electronic Computing Instrument' (1946, Burks, Goldstine, von Neumann; see n. 28).

[28] A. W. Burks, H. H. Goldstine, and J. von Neumann, 'Preliminary Discussion of the Logical Design of an Electronic Computing Instrument', 28 June 1946, Institute for Advanced Study, Princeton University, Section 3.1 (p. 37); reprinted in vol. v of von Neumann, *Collected Works*.

[29] Letter from Burks to Copeland (22 Apr. 1998). See also Goldstine, *The Computer from Pascal to von Neumann*, 258.

The situation in 1945–1946

The passage just quoted is an excellent summary of the situation at that time. In 'On Computable Numbers' Turing had shown *in abstracto* that, by means of instructions expressed in the programming code of standard descriptions, a single machine of fixed structure is able to carry out any task that a 'problem planner' is able to analyse into effective steps. By 1945, considerations *in abstracto* had given way to the practical problem of devising an equivalent programming code that could be implemented efficiently by means of thermionic valves (vacuum tubes).

A machine-level programming code in effect specifies the basic operations that are available in the machine. In the case of Turing's universal machine these are move left one square, scan one symbol, write one symbol, and so on. These operations are altogether too laborious to form the basis of efficient electronic computation. A *practical* programming code should not only be universal, in the sense of being adequate in principle for the programming of any task that can be carried out by a Turing machine, but must in addition:

- employ basic operations that can be realized simply, reliably, and efficiently by electronic means;
- enable the 'actually important problems' to be solved on the machine as rapidly as the electronic hardware permits;
- be as easy as possible for the human 'problem planner' to work with.

The challenge of designing a practical code, and the underlying mechanism required for its implementation, was tackled in different ways by Turing and the several American groups.

Events at the Moore School

The 'Preliminary Discussion of the Logical Design of an Electronic Computing Instrument' was not intended for formal publication and no attempt was made to indicate those places where reference was being made to the work of others. (Von Neumann's biographer Norman Macrae remarked: 'Johnny borrowed (we must not say plagiarized) anything from anybody.'[30] The situation was the same in the case of von Neumann's 1945 paper 'First Draft of a Report on the EDVAC'.[31] This described the Moore School group's proposed stored-programme computer, the EDVAC. The 'First Draft' was distributed (by Goldstine and a Moore School administrator) before references had been added—and indeed without consideration of whether the names of Eckert and Mauchly

[30] N. Macrae, *John von Neumann* (New York: Pantheon Books, 1992), 23.

[31] J. von Neumann, 'First Draft of a Report on the EDVAC', Moore School of Electrical Engineering, University of Pennsylvania, 1945; reprinted in full in N. Stern, *From ENIAC to UNIVAC: An Appraisal of the Eckert-Mauchly Computers* (Bedford, Mass.: Digital Press, 1981).

should appear alongside von Neumann's as co-authors.[32] Eckert and Mauchly were outraged, knowing that von Neumann would be given credit for everything in the report—their ideas as well as his own. There was a storm of controversy and von Neumann left the Moore School group to establish his own computer project at Princeton. Harry Huskey, a member of the Moore School group from the spring of 1944, emphasizes that the 'First Draft' should have contained acknowledgement of the considerable extent to which the design of the proposed EDVAC was the work of other members of the group, especially Eckert.[33]

In 1944, before von Neumann came to the Moore School, Eckert and Mauchly had rediscovered the idea of using a single memory for data and programme.[34] (They were far, however, from rediscovering Turing's concept of a universal machine.) Even before the ENIAC was completed, Eckert and Mauchly were thinking about a successor machine, the EDVAC, in which the ENIAC's most glaring deficiencies would be remedied. Paramount among these, of course, was the crude wire'n'plugs method of setting up the machine for each new task. Yet if pluggable connections were not to be used, how was the machine to be controlled without a sacrifice in speed? If the computation were controlled by means of existing, relatively slow, technology—e.g. an electro-mechanical punched-card reader feeding instructions to the machine—then the high-speed electronic hardware would spend much of its time idle, awaiting the next instruction. Eckert explained to Huskey his idea of using a mercury 'delay line':

> Eckert described a mercury delay line to me, a five foot pipe filled with mercury which could be used to store a train of acoustic pulses . . . [O]ne recirculating mercury line would store more than 30 [32 bit binary] numbers . . . My first question to Eckert: thinking about the pluggable connections to control the ENIAC, 'How do you control the operations?' 'Instructions are stored in the mercury lines just like numbers,' he said. Of course! Once he said it, it was so obvious, and the *only* way that instructions could come available at rates comparable to the data rates. That was the *stored program computer.*[35]

[32] See N. Stern, 'John von Neumann's Influence on Electronic Digital Computing, 1944–1946', *Annals of the History of Computing*, 2 (1980), 349–62.

[33] Huskey in interview with Copeland (Feb. 1998). (Huskey was offered the directorship of the EDVAC project in 1946 but other commitments prevented him from accepting.)

[34] Mauchly, 'Amending the ENIAC Story'; J. P. Eckert, 'The ENIAC', in Metropolis, Howlett, and Rota, *A History of Computing in the Twentieth Century*; letter from Burks to Copeland (16 Aug. 2003): 'before von Neumann came' to the Moore School, Eckert and Mauchly were 'saying that they would build a mercury memory large enough to store the program for a problem as well as the arithmetic data'. Burks points out that von Neumann was however the first of the Moore School group to note the possibility, implict in the stored-programme concept, of allowing the computer to modify the addresses of selected instructions in a programme while it runs (A. W. Burks, 'From ENIAC to the Stored-Program Computer: Two Revolutions in Computers', in Metropolis, Howlett, and Rota, *A History of Computing in the Twentieth Century*, 340–1). Turing employed a more general form of the idea of instruction modification in his 1945 technical report 'Proposed Electronic Calculator' (in order to carry out conditional branching), and the idea of instruction modification lay at the foundation of his theory of machine learning (see Chapter 9).

[35] H. D. Huskey, 'The Early Days', *Annals of the History of Computing*, 13 (1991), 290–306 (292–3). The date of the conversation was 'perhaps the spring of 1945' (letter from Huskey to Copeland (5 Aug. 2003)).

Following his first visit to the ENIAC in 1944, von Neumann went regularly to the Moore School for meetings with Eckert, Mauchly, Burks, Goldstine, and others.[36] Goldstine reports that 'these meetings were scenes of greatest intellectual activity' and that 'Eckert was delighted that von Neumann was so keenly interested' in the idea of the high-speed delay line memory. It was, says Goldstine, 'fortunate that just as this idea emerged von Neumann should have appeared on the scene'.[37]

Eckert had produced the means to make the abstract universal computing machine of 'On Computable Numbers' concrete! Von Neumann threw himself at the key problem of devising a practical code. In 1945, Eckert and Mauchly reported that von Neumann 'has contributed to many discussions on the logical controls of the EDVAC, has prepared certain instruction codes, and has tested these proposed systems by writing out the coded instructions for specific problems'.[38] Burks summarized matters:

> Pres [Eckert] and John [Mauchly] invented the circulating mercury delay line store, with enough capacity to store program information as well as data. Von Neumann created the first modern order code and worked out the logical design of an electronic computer to execute it.[39]

Von Neumann's embryonic programming code appeared in May 1945 in the 'First Draft of a Report on the EDVAC'.

So it was that von Neumann became the first to outline a 'practical version of the universal machine' (the quoted phrase is Turing's; see p. 16). The 'First Draft' contained little engineering detail, however, in particular concerning electronics. Turing's own practical version of the universal machine followed later the same year. His 'Proposed Electronic Calculator' set out a detailed programming code—very different from von Neumann's—together with a detailed design for the underlying hardware of the machine (see Chapter 9).

6. Turing and Babbage

Charles Babbage, Lucasian Professor of Mathematics at the University of Cambridge from 1828 to 1839, was one of the first to appreciate the enormous potential of computing machinery. In about 1820, Babbage proposed an

[36] Goldstine, *The Computer from Pascal to von Neumann*, 186.

[37] Ibid.

[38] J. P. Eckert and J. W. Mauchly, 'Automatic High Speed Computing: A Progress Report on the EDVAC', Moore School of Electrical Engineering, University of Pennsylvania (Sept. 1945), Section 1; this section of the report is reproduced on pp. 184–6 of L. R. Johnson, *System Structure in Data, Programs, and Computers* (Englewood Cliffs, NJ: Prentice-Hall, 1970).

[39] Burks, 'From ENIAC to the Stored-Program Computer: Two Revolutions in Computers', 312.

'Engine' for the automatic production of mathematical tables (such as logarithm tables, tide tables, and astronomical tables).[40] He called it the 'Difference Engine'. This was the age of the steam engine, and Babbage's Engine was to consist of more accurately machined forms of components found in railway locomotives and the like—brass gear wheels, rods, ratchets, pinions, and so forth.

Decimal numbers were represented by the positions of ten-toothed metal wheels mounted in columns. Babbage exhibited a small working model of the Engine in 1822. He never built the full-scale machine that he had designed, but did complete several parts of it. The largest of these—roughly 10 per cent of the planned machine—is on display in the London Science Museum. Babbage used it to calculate various mathematical tables. In 1990 his 'Difference Engine No. 2' was finally built from the original design and this is also on display at the London Science Museum—a glorious machine of gleaming brass.

In 1843 the Swedes Georg and Edvard Scheutz (father and son) built a simplified version of the Difference Engine. After making a prototype they built two commercial models. One was sold to an observatory in Albany, New York, and the other to the Registrar-General's office in London, where it calculated and printed actuarial tables.

Babbage also proposed the 'Analytical Engine', considerably more ambitious than the Difference Engine.[41] Had it been completed, the Analytical Engine would have been an all-purpose mechanical digital computer. A large model of the Analytical Engine was under construction at the time of Babbage's death in 1871, but a full-scale version was never built.

The Analytical Engine was to have a memory, or 'store' as Babbage called it, and a central processing unit, or 'mill'. The behaviour of the Analytical Engine would have been controlled by a programme of instructions contained on punched cards, connected together by ribbons (an idea Babbage adopted from the Jacquard weaving loom). The Analytical Engine would have been able to select from alternative actions on the basis of outcomes of previous actions—a facility now called 'conditional branching'.

Babbage's long-time collaborator was Ada, Countess of Lovelace (daughter of the poet Byron), after whom the modern programming language ADA is named. Her vision of the potential of computing machines was in some respects perhaps more far-reaching even than Babbage's own. Lovelace envisaged computing that

[40] C. Babbage, *Passages from the Life of a Philosopher*, vol. xi of *The Works of Charles Babbage*, ed. M. Campbell-Kelly (London: William Pickering, 1989); see also B. Randell (ed.), *The Origins of Digital Computers: Selected Papers* (Berlin: Springer-Verlag, 3rd edn. 1982), ch. 1.

[41] See Babbage, *Passages from the Life of a Philosopher*; A. A. Lovelace and L. F. Menabrea, 'Sketch of the Analytical Engine Invented by Charles Babbage, Esq.' (1843), in B. V. Bowden (ed.), *Faster than Thought* (London: Pitman, 1953); Randell, *The Origins of Digital Computers: Selected Papers*, ch. 2; A. Bromley, 'Charles Babbage's Analytical Engine, 1838', *Annals of the History of Computing*, 4 (1982), 196–217.

went beyond pure number-crunching, suggesting that the Analytical Engine might compose elaborate pieces of music.[42]

Babbage's idea of a general-purpose calculating engine was well known to some of the modern pioneers of automatic calculation. In 1936 Vannevar Bush, inventor of the Differential Analyser (an analogue computer), spoke in a lecture of the possibility of machinery that 'would be a close approach to Babbage's large conception'.[43] The following year Howard Aiken, who was soon to build the digital—but not stored-programme and not electronic—Harvard Automatic Sequence Controlled Calculator, wrote:

> Hollerith . . . returned to the punched card first employed in calculating machinery by Babbage and with it laid the groundwork for the development of . . . machines as manufactured by the International Business Machines Company, until today many of the things Babbage wished to accomplish are being done daily in the accounting offices of industrial enterprises all over the world.[44]

Babbage's ideas were remembered in Britain also, and his proposed computing machinery was on occasion a topic of lively mealtime discussion at Bletchley Park, the wartime headquarters of the Government Code and Cypher School and birthplace of the electronic digital computer (see 'Enigma' and the introductions to Chapters 4 and 9).[45]

It is not known when Turing first learned of Babbage's ideas.[46] There is certainly no trace of Babbage's influence to be found in 'On Computable Numbers'. Much later, Turing generously wrote (Chapter 11, p. 446):

> The idea of a digital computer is an old one. Charles Babbage . . . planned such a machine, called the Analytical Engine, but it was never completed. Although Babbage had all the essential ideas, his machine was not at that time such a very attractive prospect.

Babbage had emphasized the generality of the Analytical Engine, claiming that 'the conditions which enable a finite machine to make calculations of unlimited extent are fulfilled in the Analytical Engine'.[47] Turing states (Chapter 11, p. 455) that the Analytical Engine was *universal*—a judgement possible only from the vantage point of 'On Computable Numbers'. The Analytical Engine was not, however, a stored-programme computer. The programme resided externally on

[42] Lovelace and Menabrea, 'Sketch of the Analytical Engine', 365.

[43] V. Bush, 'Instrumental Analysis', *Bulletin of the American Mathematical Society*, 42 (1936), 649–69 (654) (the text of Bush's 1936 Josiah Willard Gibbs Lecture).

[44] H. Aiken, 'Proposed Automatic Calculating Machine' (1937), in Randell, *The Origins of Digital Computers: Selected Papers*, 196.

[45] Thomas H. Flowers in interview with Copeland (July 1996).

[46] Dennis Babbage, chief cryptanalyst in Hut 6, the section at Bletchley Park responsible for Army, Airforce, and Railway Enigma, is sometimes said to have been a descendant of Charles Babbage. This was not in fact so. (Dennis Babbage in interview with Ralph Erskine.)

[47] Babbage, *Passages from the Life of a Philosopher*, 97.

punched cards, and as each card entered the Engine, the instruction marked on that card would be obeyed.

Someone might wonder what difference there is between the Analytical Engine and the universal Turing machine in that respect. After all, Babbage's cards strung together with ribbon would in effect form a tape upon which the programme is marked. The difference is that in the universal Turing machine, but not the Analytical Engine, there is no fundamental distinction between programme and data. It is the absence of such a distinction that marks off a stored-programme computer from a programme-controlled computer. As Gandy put the point, Turing's 'universal machine is a stored-program machine [in that], unlike Babbage's all-purpose machine, the mechanisms used in reading a program are *of the same kind* as those used in executing it'.[48]

7. Origins of the Term 'Computer Programme'

As previously mentioned, Turing's tables of instructions for Turing machines are examples of what are now called computer programmes. When he turned to designing an electronic computer in 1945 (the ACE), Turing continued to use his term 'instruction table' where a modern writer would use 'programme' or 'program'.[49] Later material finds Turing referring to the actual process of writing instruction tables for the electronic computer as 'programming' but still using 'instruction table' to refer to the programme itself (see Chapter 9, pp. 388, 390–91).[50]

In an essay published in 1950 Turing explained the emerging terminology to the layman (Chapter 11, p. 445): 'Constructing instruction tables is usually described as "programming". To "programme a machine to carry out the operation A" means to put the appropriate instruction table into the machine so that it will do A.'

Turing seems to have inherited the term 'programming' from the milieu of punched-card plug-board calculators. (These calculators were electro-mechanical, not electronic. Electro-mechanical equipment was based on the *relay*—a small electrically driven mechanical switch. Relays operated much more slowly than the thermionic valves (vacuum tubes) on which the first electronic computers were based; valves owe their speed to the fact that they

[48] R. Gandy, 'The Confluence of Ideas in 1936', in R. Herken (ed.), *The Universal Turing Machine: A Half-Century Survey* (Oxford: Oxford University Press, 1998), 90. Emphasis added.

[49] 'Program' is the original English spelling, in conformity with 'anagram', 'diagram', etc. The spelling 'programme' was introduced into Britain from France in approximately 1800 (*Oxford English Dictionary*). The earlier spelling persisted in the United States. Turing's spelling is followed in this volume (except in quotations from other authors and in the section by Davies).

[50] See also 'The Turing-Wilkinson Lecture Series on the Automatic Computing Engine' (ed. Copeland), in K. Furukawa, D. Michie, and S. Muggleton (eds.), *Machine Intelligence 15* (Oxford: Oxford University Press, 1999).

have no moving parts save a beam of electrons—hence the term 'electronic'.) Plug-board calculators were set up to perform a desired sequence of arithmetical operations by means of plugging wires into appropriate sockets in a board resembling a telephone switchboard. Data was fed into the calculator from punched cards, and a card-punching device or printer recorded the results of the calculation. An early example of a punched-card machine was constructed in the USA by Herman Hollerith for use in processing statistical data gathered in the 1890 census. By the mid-twentieth century most of the world's computing was being done by punched-card calculators. Gradually the technology was displaced by the electronic computer.

When Turing joined the National Physical Laboratory in 1945 there was a large room filled with punched-card calulating equipment. David Clayden, one of the engineers who built the ACE, describes the punched-card equipment and the terminology in use at that time:

> When I started at NPL in 1947 there was a well established punched card department, mainly Hollerith. The workhorse of punched card equipment is the 'Reproducer', which has a broadside card reader and a broadside card punch. By taking a stack of cards from the punch and putting them into the reader, it is possible to do iterative calculations. All functions are controlled by a plugboard on which there are two sets of 12 × 80 sockets, one for the reader and one for the punch. In addition there is a relay store [i.e. memory]. The plugboard can be connected in many ways (using short plugleads) in order to perform many functions, including addition, subtraction, and multiplication. The plugboards are removable. NPL had a stack of them and called them 'programme' boards.[51]

Turing's own preference for 'instruction table' over the noun 'programme' was not shared by all his colleagues at the NPL. Mike Woodger, Turing's assistant from 1946, says: ' "Programme" of course was an ordinary English word meaning a planned sequence of events. We adopted it naturally for any instruction table that would give rise to a desired sequence of events.'[52] The noun 'programme' was in use in its modern sense from the earliest days of the ACE project. A report (probably written by Turing's immediate superior, Womersley) describing work done by Turing and his assistants during 1946 stated: 'It is intended to prepare the instructions to the machine [the ACE] on Hollerith cards, and it is proposed to maintain a library of these cards with programmes for standard operations.'[53] By the early 1950s specially printed ruled sheets used at the NPL for writing out programmes bore the printed heading 'ACE Pilot Model Programme'.[54]

[51] Letter from Clayden to Copeland (3 Oct. 2000).

[52] Letter from Woodger to Copeland (6 Oct. 2000).

[53] 'Draft Report of the Executive Committee for the Year 1946', National Physical Laboratory, paper E.910, section Ma. 1, anon., but probably by Womersley (NPL Library; a digital facsimile is in The Turing Archive for the History of Computing <www.AlanTuring.net/annual_report_1946>).

[54] J. G. Hayes, 'The Place of Pilot Programming', MS, 2000.

A document written by Woodger in 1947 used the single 'm' spelling: 'A Program for Version H'.[55] Woodger recalls: 'We used both spellings carelessly for some years until Goodwin (Superintendent of Mathematics Division from 1951) laid down the rule that the "American" spelling should be used.'[56] It is possible that the single 'm' spelling first came to the NPL via the American engineer Huskey, who spent 1947 with the ACE group. Huskey was responsible for 'Version H', a scaled-down form of Turing's design for the ACE (see Chapter 10).

Like Turing, Eckert and Mauchly, the chief architects of ENIAC, probably inherited the terms 'programming' and 'program' from the plug-board calculator. In 1942, while setting out the idea of a high-speed electronic calculator, Mauchly used the term 'programming device' (which he sometimes shortened to 'program device') to refer to a mechanism whose function was to determine how and when the various component units of a calculator shall perform.[57] In the summer of 1946 the Moore School organized a series of influential lectures entitled 'Theory and Techniques for Design of Electronic Digital Computers'. In the course of these, Eckert used the term 'programming' in a similar sense when describing the new idea of storing instructions in high-speed memory: 'We . . . feed those pieces of information which relate to programming from the memory.'[58] Also in 1946, Burks, Goldstine, and von Neumann (all ex-members of the Moore School group) were using the verb-form 'program the machine', and were speaking of 'program orders' being stored in memory.[59] The modern nominalized form appears not to have been adopted in the USA until a little later. Huskey says, 'I am pretty certain that no one had written a "program" by the time I left Philadelphia in June 1946.'[60]

Part II Computability and Uncomputability

8. Circular and Circle-Free Machines

Turing calls the binary digits '0' and '1' symbols 'of the first kind'. Any symbols that a computing machine is able to print apart from the binary digits—such as

[55] M. Woodger, 'A Program for Version H', handwritten MS, 1947 (in the Woodger Papers, National Museum of Science and Industry, Kensington, London (catalogue reference N30/37)).

[56] Letter from Woodger to Copeland (6 Oct. 2000).

[57] J. W. Mauchly, 'The Use of High Speed Vacuum Tube Devices for Calculating' (1942), in Randell, *The Origins of Digital Computers: Selected Papers.*

[58] J. P. Eckert, 'A Preview of a Digital Computing Machine' (15 July 1946), in M. Campbell-Kelly and M. R. Williams (eds.), *The Moore School Lectures* (Cambridge, Mass.: MIT Press, 1985), 114.

[59] Sections 1.2, 5.3 of Burks, Goldstine, and von Neumann, 'Preliminary Discussion of the Logical Design of an Electronic Computing Instrument' (von Neumann, *Collected Works*, vol. v, 15, 43).

[60] Letter from Huskey to Copeland (3 Feb. 2002).