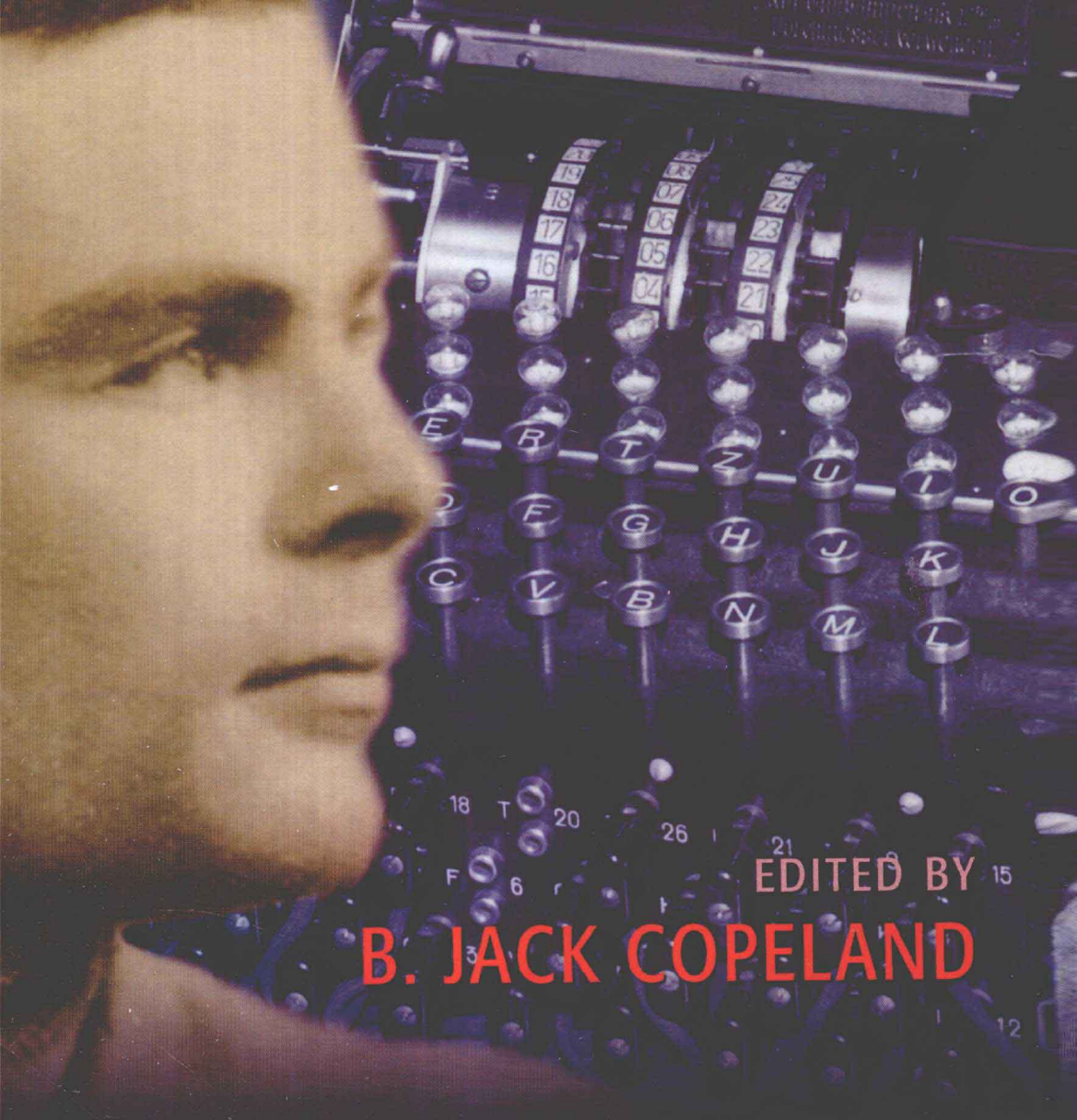


EDITED BY

B. JACK COPELAND

THE ESSENTIAL TURING

The ideas that gave birth to the computer age

The Essential Turing

Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life *plus* The Secrets of Enigma

Edited by **B. Jack Copeland**

CLARENDON PRESS · OXFORD

OXFORD
UNIVERSITY PRESS

Great Clarendon Street, Oxford OX2 6DP

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide in

Oxford New York

Auckland Cape Town Dar es Salaam Hong Kong Karachi
Kuala Lumpur Madrid Melbourne Mexico City Nairobi
New Delhi Shanghai Taipei Toronto

With offices in

Argentina Austria Brazil Chile Czech Republic France Greece
Guatemala Hungary Italy Japan South Korea Poland Portugal
Singapore Switzerland Thailand Turkey Ukraine Vietnam

Published in the United States
by Oxford University Press Inc., New York

First published 2004

British Library Cataloguing in Publication Data
Data available

Library of Congress Cataloging in Publication Data
Data available

ISBN 0–19–825079–7
ISBN 0–19–825080–0 (pbk.) ; 978-0-19-825080-7 (pbk.)

Typeset by Kolam Information Services Pvt. Ltd, Pondicherry, India
Printed in Great Britain
on acid-free paper by Biddles Ltd., King's Lynn, Norfolk

Acknowledgements

Work on this book began in 2000 at the Dibner Institute for the History of Science and Technology, Massachusetts Institute of Technology, and was completed at the University of Canterbury, New Zealand. I am grateful to both these institutions for aid, and to the following for scholarly assistance: John Andreae, Friedrich Bauer, Frank Carter, Alonzo Church Jnr, David Clayden, Bob Doran, Ralph Erskine, Harry Fensom, Jack Good, John Harper, Geoff Hayes, Peter Hilton, Harry Huskey, Eric Jacobson, Elizabeth Mahon, Philip Marks, Elisabeth Norcliffe, Rolf Noskwith, Gualtiero Piccinini, Andrés Sicard, Wilfried Sieg, Frode Weierud, Maurice Wilkes, Mike Woodger, and especially Diane Proudfoot. This book would not have existed without the support of Turing's literary executor, P. N. Furbank, and that of Peter Momtchiloff at Oxford University Press.

B.J.C.

Contents

Alan Turing 1912–1954

Jack Copeland

Alan Mathison Turing was born on 23 June 1912 in London[1]; he died on 7 June 1954 at his home in Wilmslow, Cheshire. Turing contributed to logic, mathematics, biology, philosophy, cryptanalysis, and formatively to the areas later known as computer science, cognitive science, Artificial Intelligence, and Artificial Life.

Educated at Sherborne School in Dorset, Turing went up to King's College, Cambridge, in October 1931 to read Mathematics. He graduated in 1934, and in March 1935 was elected a Fellow of King's, at the age of only 22. In 1936 he published his most important theoretical work, 'On Computable Numbers, with an Application to the Entscheidungsproblem [Decision Problem]' (Chapter 1, with corrections in Chapter 2). This article described the abstract digital computing machine—now referred to simply as the universal Turing machine—on which the modern computer is based. Turing's fundamental idea of a universal stored-programme computing machine was promoted in the United States by John von Neumann and in England by Max Newman. By the end of 1945 several groups, including Turing's own in London, were devising plans for an electronic stored-programme universal digital computer—a Turing machine in hardware.

In 1936 Turing left Cambridge for the United States in order to continue his research at Princeton University. There in 1938 he completed a Ph.D. entitled 'Systems of Logic Based on Ordinals', subsequently published under the same title (Chapter 3, with further exposition in Chapter 4). Now a classic, this work addresses the implications of Gödel's famous incompleteness result. Turing gave a new analysis of mathematical reasoning, and continued the study, begun in 'On Computable Numbers', of uncomputable problems—problems that are 'too hard' to be solved by a computing machine (even one with unlimited time and memory).

Turing returned to his Fellowship at King's in the summer of 1938. At the outbreak of war with Germany in September 1939 he moved to Bletchley Park, the wartime headquarters of the Government Code and Cypher School (GC & CS). Turing's brilliant work at Bletchley Park had far-reaching consequences.

[1] At 2 Warrington Crescent, London W9, where now there is a commemorative plaque.

'I won't say that what Turing did made us win the war, but I daresay we might have lost it without him', said another leading Bletchley cryptanalyst.[2] Turing broke Naval Enigma—a decisive factor in the Battle of the Atlantic—and was the principal designer of the 'bombe', a high-speed codebreaking machine. The ingenious bombes produced a flood of high-grade intelligence from Enigma. It is estimated that the work done by Turing and his colleagues at GC & CS shortened the war in Europe by at least two years.[3] Turing's contribution to the Allied victory was a state secret and the only official recognition he received, the Order of the British Empire, was in the circumstances derisory. The full story of Turing's involvement with Enigma is told for the first time in this volume, the material that forms Chapters 5, 6, and 8 having been classified until recently.

In 1945, the war over, Turing was recruited to the National Physical Laboratory (NPL) in London, his brief to design and develop an electronic digital computer—a concrete form of the universal Turing machine. His design (for the Automatic Computing Engine or ACE) was more advanced than anything else then under consideration on either side of the Atlantic. While waiting for the engineers to build the ACE, Turing and his group pioneered the science of computer programming, writing a library of sophisticated mathematical programmes for the planned machine.

Turing founded the field now called 'Artificial Intelligence' (AI) and was a leading early exponent of the theory that the human brain is in effect a digital computer. In February 1947 he delivered the earliest known public lecture to mention computer intelligence ('Lecture on the Automatic Computing Engine' (Chapter 9)). His technical report 'Intelligent Machinery' (Chapter 10), written for the NPL in 1948, was effectively the first manifesto of AI. Two years later, in his now famous article 'Computing Machinery and Intelligence' (Chapter 11), Turing proposed (what subsequently came to be called) the Turing test as a criterion for whether machines can think. *The Essential Turing* collects together for the first time the series of five papers that Turing devoted exclusively to Artificial Intelligence (Chapters 10, 11, 12, 13, 16). Also included is a discussion of AI by Turing, Newman, and others (Chapter 14).

In the end, the NPL's engineers lost the race to build the world's first working electronic stored-programme digital computer—an honour that went to the Computing Machine Laboratory at the University of Manchester in June 1948. The concept of the universal Turing machine was a fundamental influence on the Manchester computer project, via Newman, the project's instigator. Later in

[2] Jack Good in an interview with Pamela McCorduck, on p. 53 of her *Machines Who Think* (New York: W. H. Freeman, 1979).

[3] This estimate is given by Sir Harry Hinsley, official historian of the British Secret Service, writing on p. 12 of his and Alan Stripp's edited volume *Codebreakers: The Inside Story of Bletchley Park* (Oxford: Oxford University Press, 1993).

1948, at Newman's invitation, Turing took up the deputy directorship of the Computing Machine Laboratory (there was no Director). Turing spent the rest of his short career at Manchester University. He was elected a Fellow of the Royal Society of London in March 1951 (a high honour) and in May 1953 was appointed to a specially created Readership in the Theory of Computing at Manchester.

It was at Manchester, in March 1952, that he was prosecuted for homosexual activity, then a crime in Britain, and sentenced to a period of twelve months' hormone 'therapy'—the shabbiest of treatment from the country he had helped save, but which he seems to have borne with amused fortitude.

Towards the end of his life Turing pioneered the area now known as Artificial Life. His 1952 article 'The Chemical Basis of Morphogenesis' (Chapter 15) describes some of his research on the development of pattern and form in living organisms. This research dominated his final years, but he nevertheless found time to publish in 1953 his classic article on computer chess (Chapter 16) and in 1954 'Solvable and Unsolvable Problems' (Chapter 17), which harks back to 'On Computable Numbers'. From 1951 he used the Computing Machine Laboratory's Ferranti Mark I (the first commercially produced electronic stored-programme computer) to model aspects of biological growth, and in the midst of this groundbreaking work he died.

Turing's was a far-sighted genius and much of the material in this book is of even greater relevance today than in his lifetime. His research had remarkable breadth and the chapters range over a diverse collection of topics—mathematical logic and the foundations of mathematics, computer design, mechanical methods in mathematics, cryptanalysis and chess, the nature of intelligence and mind, and the mechanisms of biological growth. The chapters are united by the overarching theme of Turing's work, his enquiry into (as Newman put it) 'the extent and the limitations of mechanistic explanations'.[4]

Biographies of Turing

Gottfried, T., *Alan Turing: The Architect of the Computer Age* (Danbury, Conn.: Franklin Watts, 1996).

Hodges, A., *Alan Turing: The Enigma* (London: Burnett, 1983).

Newman, M. H. A., 'Alan Mathison Turing, 1912–1954', *Biographical Memoirs of Fellows of the Royal Society*, 1 (1955), 253–63.

Turing, S., *Alan M. Turing* (Cambridge: W. Heffer, 1959).

[4] M. H. A. Newman, 'Alan Mathison Turing, 1912–1954', *Biographical Memoirs of Fellows of the Royal Society*, 1 (1955), 253–63 (256).

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 Entscheidungsproblem' 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.

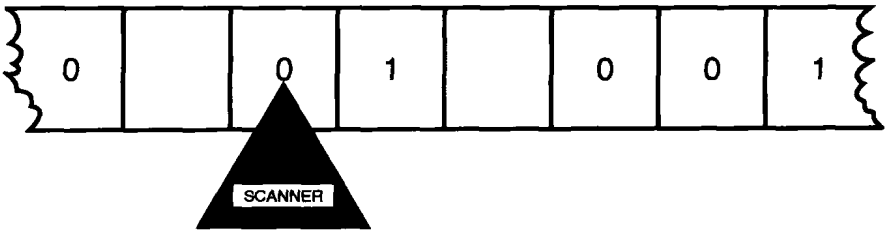
In addition to the operations just mentioned, the scanner is able to alter what Turing calls its '*m*-configuration'. In modern Turing-machine jargon it is usual to

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.)

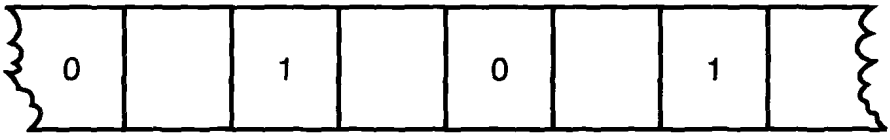
The operations just described—erase, print, move, and change state—are the *basic* (or *atomic*) operations of the Turing machine. Complexity of operation is achieved by chaining together large numbers of these simple basic actions. Commercially available computers are hard-wired to perform basic operations 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-0Rb** (using '-' to represent a blank). The third line is: **c-1Rd**.

The second line of the table, which does not require the contents of the scanned square (a blank) to be changed, is written: **b--Rc**. 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: **dxxLc**.

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*1Nc**.

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***-Rn.

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-0Rb; **b**--Rc; **c**-1Rd; **d**--Ra;

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:

aDDCRb; **b**DDRc; **c**DDCCR**d**; **d**DDRa;

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:

3133253117311335311173111332253111173111133531 7.[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 '3133253117311335311173111332253111173111133531 7', but is the number denoted by this string of numerals.