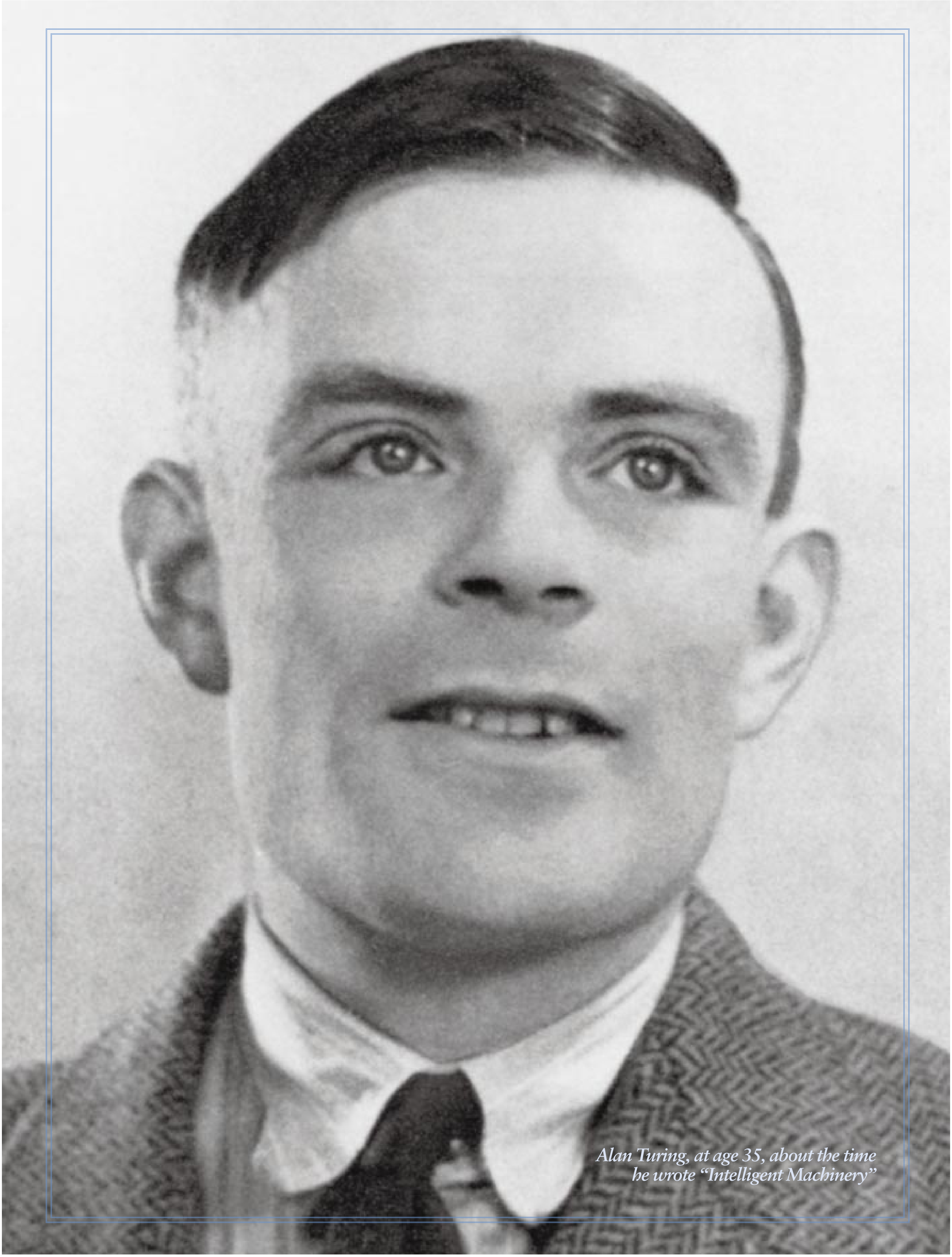


Alan Turing, at age 35, about the time he wrote "Intelligent Machinery"

Copyright 1998 Scientific American, Inc.

Alan Turing's Forgotten Ideas in Computer Science

Well known for the machine, test and thesis that bear his name, the British genius also anticipated neural-network computers and "hypercomputation"

by B. Jack Copeland and Diane Proudfoot

W. HEFFER AND SONS LTD., CAMBRIDGE, ENGLAND

Alan Mathison Turing conceived of the modern computer in 1935. Today all digital computers are, in essence, "Turing machines." The British mathematician also pioneered the field of artificial intelligence, or AI, proposing the famous and widely debated Turing test as a way of determining whether a suitably programmed computer can think. During World War II, Turing was instrumental in breaking the German Enigma code in part of a top-secret British operation that historians say shortened the war in Europe by two years. When he died at the age of 41, Turing was doing the earliest work on what would now be called artificial life, simulating the chemistry of biological growth.

Throughout his remarkable career, Turing had no great interest in publicizing his ideas. Consequently, important aspects of his work have been neglected or forgotten over the years. In particular, few people—even those knowledgeable about computer science—are familiar with Turing's fascinating anticipation of connectionism, or neuronlike computing. Also neglected are his groundbreaking theoretical concepts in the exciting area of "hypercomputation." According to some experts, hypercomputers might one day solve problems heretofore deemed intractable.

The Turing Connection

Digital computers are superb number crunchers. Ask them to predict a rocket's trajectory or calculate the financial figures for a large multinational corporation, and they can churn out the answers in seconds. But seemingly simple actions that people routinely perform, such as recognizing a face or reading handwriting, have been devilishly tricky to program. Perhaps the networks of neurons that make up the brain have a natural facility for such tasks that standard computers lack. Scientists have thus been investigating computers modeled more closely on the human brain.

Connectionism is the emerging science of computing with networks of artificial neurons. Currently researchers usually simulate the neurons and their interconnections within an ordinary digital computer (just as engineers create virtual models of aircraft wings and skyscrapers). A training algorithm that runs on the computer adjusts the connections between the neurons, honing the network into a special-purpose machine dedicated to some particular function, such as forecasting international currency markets.

Modern connectionists look back to Frank Rosenblatt, who published the first of many papers on the topic in 1957, as the founder of their approach. Few realize that Turing had already investigated connectionist networks as early as 1948, in a little-known paper entitled "Intelligent Machinery."

Written while Turing was working for the National Physical Laboratory in London, the manuscript did not meet with his employer's approval. Sir Charles Darwin, the rather headmasterly director of the laboratory and grandson of the great English naturalist, dismissed it as a "schoolboy essay." In reality, this farsighted paper was the first manifesto of the field of artificial intelli-

Copyright 1999 Scientific American, Inc.

gence. In the work—which remained unpublished until 1968, 14 years after Turing's death—the British mathematician not only set out the fundamentals of connectionism but also brilliantly introduced many of the concepts that were later to become central to AI, in some cases after reinvention by others.

In the paper, Turing invented a kind of neural network that he called a "B-type unorganized machine," which consists of artificial neurons and devices that modify the connections between them. B-type machines may contain any number of neurons connected in any pattern but are always subject to the restriction that each neuron-to-neuron connection must pass through a modifier device.

All connection modifiers have two training fibers. Applying a pulse to one of them sets the modifier to "pass mode," in which an input—either 0 or 1—passes through unchanged and becomes the output. A pulse on the other fiber places the modifier in "interrupt mode," in which the output is always 1, no matter what the input is. In this state the modifier destroys all information attempting to pass along the connection to which it is attached.

Once set, a modifier will maintain its function (either "pass" or "interrupt") unless it receives a pulse on the other training fiber. The presence of these ingenious connection modifiers enables the training of a B-type unorganized machine by means of what Turing called "appropriate interference, mimicking education." Actually, Turing theorized that "the cortex of an infant is an unorganized machine, which can be organized by suitable interfering training."

Each of Turing's model neurons has two input fibers, and the output of a neuron is a simple logical function of its two inputs. Every neuron in the network executes the same logical operation of "not and" (or NAND): the output is 1 if either of the inputs is 0. If both inputs are 1, then the output is 0.

Turing selected NAND because every other logical (or Boolean) operation can be accomplished by groups of NAND neurons. Furthermore, he showed that even the connection modifiers themselves can be built out of NAND neurons. Thus, Turing specified a network made up of nothing more than NAND neurons and their connecting fibers—about the simplest possible model of the cortex.

In 1958 Rosenblatt defined the theoretical basis of connectionism in one succinct statement: "Stored information takes the form of new connections, or transmission channels in the nervous system (or the creation of conditions which are functionally equivalent to new connections)." Because the destruction of existing connections can be functionally equivalent to the creation of new ones, researchers can build a network for accomplishing a specific task by taking one with an excess of connections and selectively destroying some of them. Both actions—destruction and creation—are employed in the training of Turing's B-types.

At the outset, B-types contain random interneural connections whose modifiers have been set by chance to either pass or interrupt. During training, unwanted connections are destroyed by switching their attached modifiers to interrupt mode. Conversely, changing a modifier from interrupt to pass in effect creates a connection. This selective culling and enlivening of connections hones the initially random network into one organized for a given job.

Turing wished to investigate other kinds of unorganized machines, and he longed to simulate a neural network and its training regimen using an ordinary digital computer. He would, he said, "allow the whole system to run for an appreciable period, and then break in as a kind of 'inspector of schools' and see what progress had been made." But his own work on neural networks was carried out shortly before the first general-purpose electronic computers became available. (It was not until 1954, the year of Turing's death, that Belmont G. Farley and Wesley A. Clark succeeded at the Massachusetts Institute of Technology in running the first computer simulation of a small neural network.)

Paper and pencil were enough, though, for Turing to show that a sufficiently large B-type neural network can be configured (via its connection modifiers) in such a way that it becomes a general-purpose computer. This discovery illuminates one of the most fundamental problems concerning human cognition.

From a top-down perspective, cognition includes complex sequential processes, often involving language or other forms of symbolic representation, as in mathematical calculation. Yet from a bottom-up view, cognition is nothing but the simple firings of neurons. Cognitive scientists face the problem of how to reconcile these very different perspectives.

Turing's discovery offers a possible solution: the cortex, by virtue of being a neural network acting as a general-purpose computer, is able to carry out the sequential, symbol-rich processing discerned in the view from the top. In 1948 this hypothesis was well ahead of its time, and today it remains among the best guesses concerning one of cognitive science's hardest problems.

Few realize that Turing had already investigated connectionist networks as early as 1948.

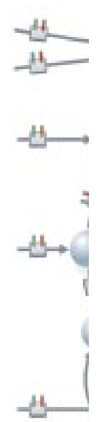
Computing the Uncomputable

In 1935 Turing thought up the abstract device that has since become known as the "universal Turing machine." It consists of a limitless memory

Turing's Anticipation of Connectionism

In a paper that went unpublished until 14 years after his death (*top*), Alan Turing described a network of artificial neurons connected in a random manner. In this "B-type unorganized machine" (*bottom left*), each connection passes through a modifier that is set either to allow data to pass unchanged (*green fiber*) or to destroy the transmitted information (*red fiber*). Switching the modifiers from one mode to the other enables the network to be trained. Note that each neuron has two inputs (*bottom left, inset*) and executes the simple logical operation of "not and," or NAND: if both inputs are 1, then the output is 0; otherwise the output is 1.

In Turing's network the neurons interconnect freely. In contrast, modern networks (*bottom center*) restrict the flow of information from layer to layer of neurons. Connectionists aim to simulate the neural networks of the brain (*bottom right*).



Copyright 1999 Scientific American, Inc.

that stores both program and data and a scanner that moves back and forth through the memory, symbol by symbol, reading the information and writing additional symbols. Each of the machine's basic actions is very simple—such as "identify the symbol on which the scanner is positioned," "write '1'" and "move one position to the left." Complexity is achieved by chaining together large numbers of these basic actions. Despite its simplicity, a universal Turing machine can execute any task that can be done by the most powerful of today's computers. In fact, all modern digital computers are in essence universal Turing machines [see "Turing Machines," by John E. Hopcroft; Scientific American, May 1984].

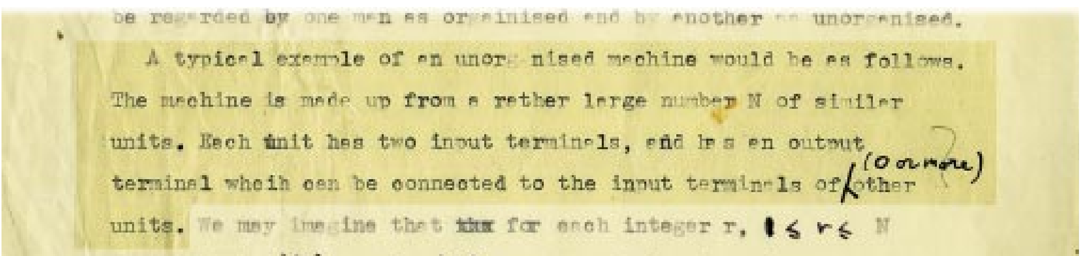
Turing's aim in 1935 was to devise a machine—one as simple as possible—capable of any calculation that a human mathematician working in accordance with some algorithmic method could perform, given unlimited time, energy, paper and pencils, and perfect concentration. Calling a machine "universal" merely signifies that it is capable of all such calculations. As Turing himself wrote, "Electronic computers are intended to carry out any definite rule-of-thumb process which could have been done by a human operator working in a disciplined but unintelligent manner."

Such powerful computing devices notwithstanding, an intriguing question arises: Can machines be devised that are capable of accomplishing even more? The answer is that these "hypermachines" can be described on paper, but no one as yet knows whether it will be possible to build one. The field of hypercomputation is currently attracting a growing number of scientists. Some speculate that the human brain itself—the most complex information processor known—is actually a naturally occurring example of a hypercomputer.

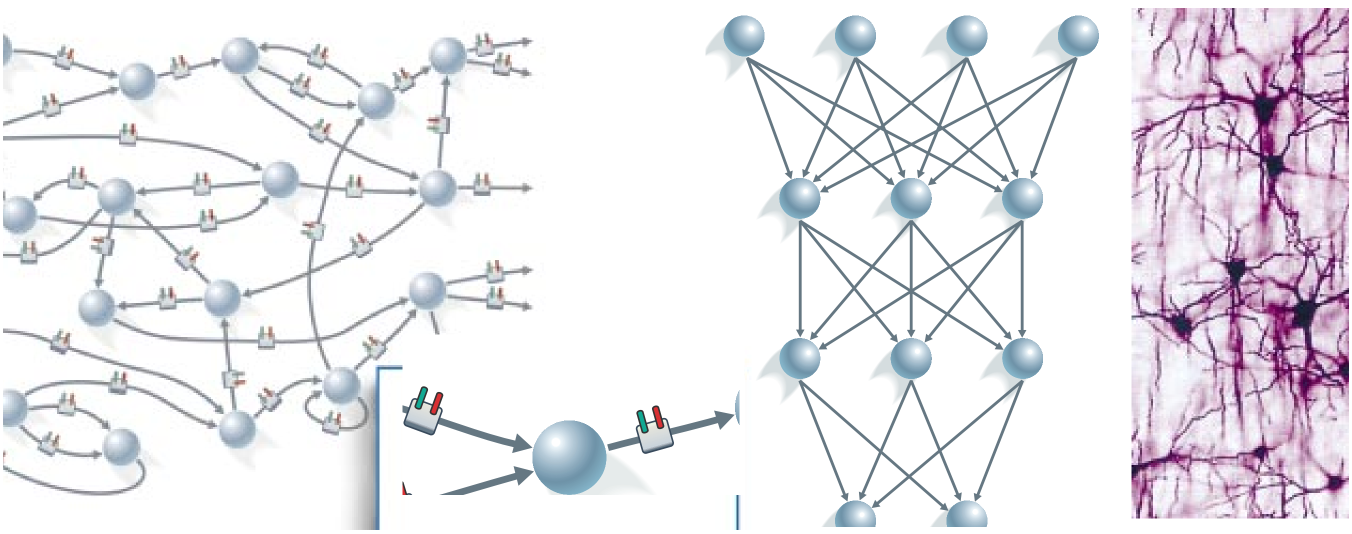
Before the recent surge of interest in hypercomputation, any information-processing job that was known to be too difficult for universal Turing machines was written off as "uncomputable." In this sense, a hypermachine computes the uncomputable.

Examples of such tasks can be found in even the most straightforward areas of mathematics. For instance, given arithmetical statements picked at random, a universal Turing machine may not always be able to tell which are theorems (such as "7 + 5 = 12") and which are nontheorems (such as "every number is the sum of two even numbers"). Another type of uncomputable problem comes from geometry. A set of tiles—variously sized squares with different colored edges—"tiles the plane" if the Euclidean plane can be covered by copies of the tiles with no gaps or overlaps and with adjacent edges always the same color. Logicians William Hanf and Dale Myers of the University of Hawaii have discovered a tile set that tiles the plane only in patterns too complicated for a universal Turing machine to calculate. In the field of computer science, a universal Turing machine cannot always predict whether a given program will terminate or continue running forever. This is sometimes expressed by saying that no general-purpose programming language (Pascal, BASIC, Prolog, C and so on) can have a foolproof crash debugger: a tool that detects all bugs that could lead to crashes, including errors that result in infinite processing loops.

Turing himself was the first to investigate the idea of machines that can perform mathematical tasks too difficult


be regarded by one man as organised and by another as unorganised.

A typical example of an unorganised machine would be as follows. The machine is made up from a rather large number N of similar units. Each unit has two input terminals, and has an output terminal which can be connected to the input terminals of (0 or more) other units. We may imagine that for each integer r, 1 ≤ r ≤ N



TOM MOORE (*illustrations*); KING'S COLLEGE MODERN ARCHIVES, CAMBRIDGE UNIVERSITY LIBRARY (*top*); PETER ARNOLD, INC. (*bottom right*)

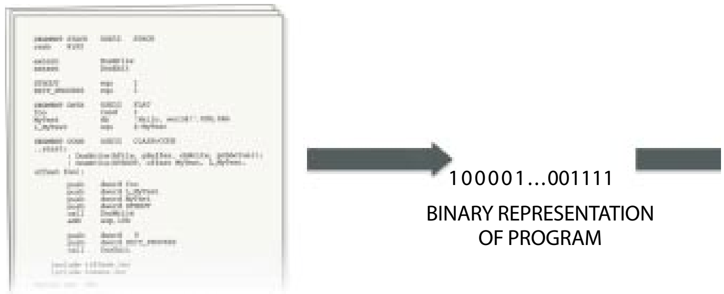
Copyright 1999 Scientific American, Inc.

Using an Oracle to Compute the Uncomputable

Alan Turing proved that his universal machine—and by extension, even today's most powerful computers—could never solve certain problems. For instance, a universal Turing machine cannot always determine whether a given software program will terminate or continue running forever. In some cases, the best the universal machine can do is execute the program and wait—maybe eternally—for it to finish. But in his doctoral thesis (*below*), Turing did imagine that a machine equipped with a special "oracle" could perform this and other "uncomputable" tasks. Here is one example of how, in principle, an oracle might work.

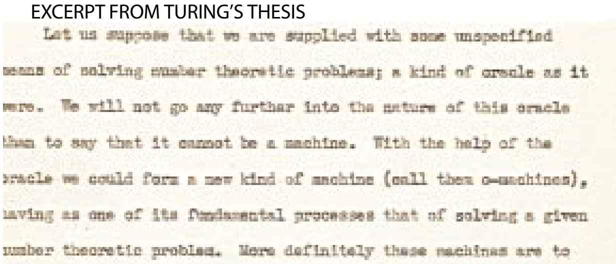
Consider a hypothetical machine for solving the formidable "terminating program" problem (*above*). A computer program can be represented as a finite string of 1s and 0s. This sequence of digits can also be thought of as the binary representation of an integer, just as 1011011 is the equivalent of 91. The oracle's job can then be restated as, "Given an integer that represents a program (for any computer that can be simulated by a universal Turing machine), output a '1' if the program will terminate or a '0' otherwise."

The oracle consists of a perfect measuring device and a store, or memory, that contains a precise value—call it τ for Turing—of some physical quantity. (The memory might, for example, resemble a capacitor storing an exact amount of



COMPUTER PROGRAM

EXCERPT FROM TURING'S THESIS


Let us suppose that we are supplied with some unspecified means of solving number theoretic problems; a kind of oracle as it were. We will not go any further into the nature of this oracle than to say that it cannot be a machine. With the help of the oracle we could form a new kind of machine (call them o-machines), having as one of its fundamental processes that of solving a given number theoretic problem. More definitely these machines are to

PRINCETON ARCHIVES

for universal Turing machines. In his 1938 doctoral thesis at Princeton University, he described "a new kind of machine," the "O-machine."

An O-machine is the result of augmenting a universal Turing machine with a black box, or "oracle," that is a mechanism for carrying out uncomputable tasks. In other respects, O-machines are similar to ordinary computers. A digitally encoded program is fed in, and the machine produces digital output from the input using a step-by-step procedure of repeated applications of the machine's basic operations, one of which is to pass data to the oracle and register its response.

Even among experts, Turing's pioneering theoretical concept of a hypermachine has largely been forgotten.

Turing gave no indication of how an oracle might work. (Neither did he explain in his earlier research how the basic actions of a universal Turing machine—for example, "identify the symbol in the scanner"—might take place.) But notional mechanisms that fulfill the specifications of an O-machine's black box are not difficult to imagine [*see box above*]. In principle, even a suitable B-type network can compute the uncomputable, provided the activity of the neurons is desynchronized. (When a central clock keeps the neurons in step with one another, the functioning of the network can be exactly simulated by a universal Turing machine.)

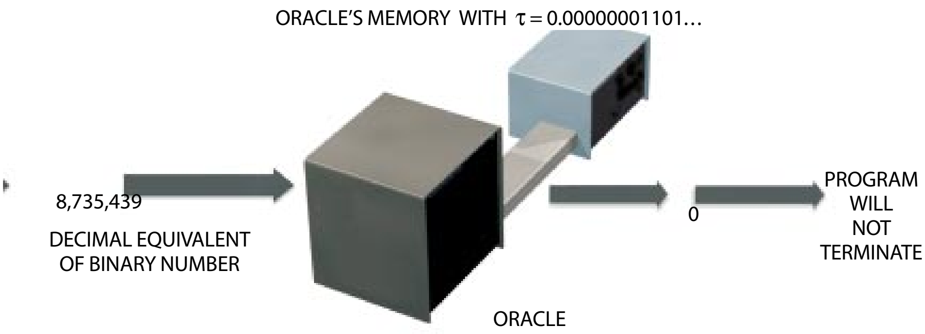
In the exotic mathematical theory of hypercomputation, tasks such as that of distinguishing theorems from nontheorems in arithmetic are no longer uncomputable. Even a debugger that can tell whether any program written in C, for example, will enter an infinite loop is theoretically possible.

If hypercomputers can be built—and that is a big if—the potential for cracking logical and mathematical problems hitherto deemed intractable will be enormous. Indeed, computer science may be approaching one of its most significant advances since researchers wired together the first electronic embodiment of a universal Turing machine decades ago. On the other hand, work on hypercomputers may simply fizzle out for want of some way of realizing an oracle.

The search for suitable physical, chemical or biological phenomena is getting under way. Perhaps the answer will be complex molecules or other structures that link together in patterns as complicated as those discovered by Hanf and Myers. Or, as suggested by Jon Doyle of M.I.T., there may be naturally occurring equilibrating systems with discrete spectra that can be seen as carrying out, in principle, an uncomputable task, producing appropriate output (1 or 0, for example) after being bombarded with input.

Outside the confines of mathematical logic, Turing's O-machines have largely been forgotten, and instead a myth has taken hold. According to this apocryphal account, Turing demonstrated in the mid-1930s that hypermachines are impossible. He and Alonzo Church, the logician who was Turing's doctoral adviser at Princeton, are mistakenly credited with having enunciated a principle to the effect that a universal Turing machine can exactly simulate the behavior

Copyright 1999 Scientific American, Inc.



electricity.) The value of τ is an irrational number; its written representation would be an infinite string of binary digits, such as 0.00000001101...

The crucial property of τ is that its individual digits happen to represent accurately which programs terminate and which do not. So, for instance, if the integer representing a program were 8,735,439, then the oracle could by measurement obtain the 8,735,439th digit of τ (counting from left to right after the decimal point). If that digit were 0, the oracle would conclude that the program will process forever.

Obviously, without τ the oracle would be useless, and finding some physical variable in nature that takes this exact value might very well be impossible. So the search is on for some practicable way of implementing an oracle. If such a means were found, the impact on the field of computer science could be enormous. —*B.J.C. and D.P.*

TOM MOORE

of any other information-processing machine. This proposition, widely but incorrectly known as the Church-Turing thesis, implies that no machine can carry out an information-processing task that lies beyond the scope of a universal Turing machine. In truth, Church and Turing claimed only that a universal Turing machine can match the behavior of any human mathematician working with paper and pencil in accordance with an algorithmic method—a considerably weaker claim that certainly does not rule out the possibility of hypermachines.

Even among those who are pursuing the goal of building hypercomputers, Turing's pioneering theoretical contributions have been overlooked. Experts routinely talk of carrying out information processing "beyond the Turing limit" and describe themselves as attempting to "break the Turing barrier." A recent review in *New Scientist* of this emerging field states that the new machines "fall outside Turing's conception" and are "computers of a type never envisioned by Turing," as if the British genius had not conceived of such devices more than half a century ago. Sadly, it appears that what has already occurred with respect to Turing's ideas on connectionism is starting to happen all over again.

The Final Years

In the early 1950s, during the last years of his life, Turing pioneered the field of artificial life. He was trying to simulate a chemical mechanism by which the genes of a fertilized egg cell may determine the anatomical structure of the resulting animal or plant. He described this research as "not altogether unconnected" to his study of neural networks, because "brain structure has to be ... achieved by the genetical embryological mechanism, and this theory that I am now working on may make clearer what restrictions this really implies." During this period, Turing achieved the distinction of being the first to engage in the computer-assisted exploration of nonlinear dynamical systems. His theory used nonlinear differential equations to express the chemistry of growth.

But in the middle of this groundbreaking investigation, Turing died from cyanide poisoning, possibly by his own hand. On June 8, 1954, shortly before what would have been his 42nd birthday, he was found dead in his bedroom. He had left a large pile of handwritten notes and some computer programs. Decades later this fascinating material is still not fully understood.

The Authors

B. JACK COPELAND and DIANE PROUDFOOT are the directors of the Turing Project at the University of Canterbury, New Zealand, which aims to develop and apply Turing's ideas using modern techniques. The authors are professors in the philosophy department at Canterbury, and Copeland is visiting professor of computer science at the University of Portsmouth in England. They have written numerous articles on Turing. Copeland's *Turing's Machines* and *The Essential Turing* are forthcoming from Oxford University Press, and his *Artificial Intelligence* was published by Blackwell in 1993. In addition to the logical study of hypermachines and the simulation of B-type neural networks, the authors are investigating the computer models of biological growth that Turing was working on at the time of his death. They are organizing a conference in London in May 2000 to celebrate the 50th anniversary of the pilot model of the Automatic Computing Engine, an electronic computer designed primarily by Turing.

Further Reading

X-Machines and the Halting Problem: Building a Super-Turing Machine. Mike Stannett in *Formal Aspects of Computing,* Vol. 2, pages 331–341; 1990.

Intelligent Machinery. Alan Turing in *Collected Works of A. M. Turing: Mechanical Intelligence*. Edited by D. C. Ince. Elsevier Science Publishers, 1992.

Computation beyond the Turing Limit. Hava T. Siegelmann in *Science,* Vol. 268, pages 545–548; April 28, 1995.

On Alan Turing's Anticipation of Connectionism. B. Jack Copeland and Diane Proudfoot in *Synthese,* Vol. 108, No. 3, pages 361–377; March 1996.

Turing's O-Machines, Searle, Penrose and the Brain. B. Jack Copeland in *Analysis,* Vol. 58, No. 2, pages 128–138; 1998.

The Church-Turing Thesis. B. Jack Copeland in *The Stanford Encyclopedia of Philosophy.* Edited by Edward N. Zalta. Stanford University, ISSN 1095-5054. Available at http://plato.stanford.edu on the World Wide Web.

Copyright 1999 Scientific American, Inc.