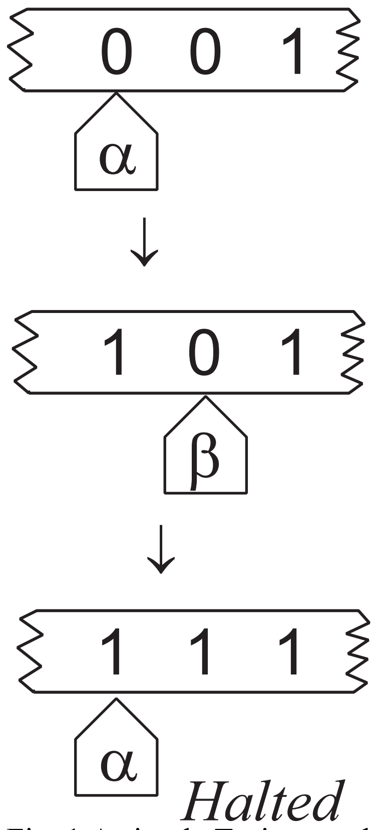
TURING MACHINE: a simple, idealized kind of digital computer devised in 1936 by the Alan Turing (qv) to give exact mathematical meaning to the notion of computation, and explore its strengths and limitations. Informally, computation is a stepwise manipulation of symbols following fixed rules without the intervention of judgment or chance. A Turing machine consists of an infinite *tape*, on which a finite set of symbols (typically 0, 1 and blank) can be read and written by a *head*, which itself is capable of existing in only some finite number of internal states. At each step, the machine's action is determined by the currently scanned tape symbol and current head state, and consists either of halting forthwith, or of writing a new tape symbol, shifting one space left or right along the tape, and changing to a new head state. This cycle of operations is repeated ad infinitum, or until the machine halts (cf Fig.1). Although their design resembles the von Neumann architecture (qv) widely used in electronic computers, with the tape corresponding to memory and the head to the processor, Turing machines were not intended to be practical, and few have ever been built. Instead they have been used very fruitfully ever since their invention to elucidate the theory of computation, including computability, computational complexity, algorithmic information theory, and the physics of computation.



$\alpha 0 \rightarrow 1\ R\ \beta$

$\beta 0 \rightarrow 1\ L\ \alpha$

$\alpha 1 \rightarrow \mathit{Halt}$

$\beta 1 \rightarrow 1\ L\ \beta$

$\alpha \_ \rightarrow \_\ R\ \alpha$

$\beta \_ \rightarrow \_\ L\ \beta$

Fig. 1 A simple Turing machine, obeying the transition rules at the right, transforms input 001 to output 111 and halts.

The Turing machine of Fig. 1 can only do trivial computations, but some more complicated machines, with a few dozen head states, are *universal*—able to perform any computation and simulate any computer, even one much larger and more complicated than itself. A universal Turing machine works by keeping on its tape a coded representation of the head state, head position, and tape contents of the larger machine being simulated, and breaks each step of the larger machine into many steps simple enough to be executed by the universal machine. Besides being able to simulate other Turing machines, universal machines have been found to be able to simulate a great many other models of computation, including ones with multidimensional tapes, random-access memory, sequential arrays of logic gates, and various kinds of parallel processing. The doctrine that any reasonable model of computation can be simulated by a Turing machine, called the Church-Turing thesis, is generally accepted on the strength of this evidence.

Like other general-purpose computers, Turing machines can embark on open-ended searches that might not terminate. The lack of any general procedure for deciding which computations terminate, called the *unsolvability of the halting problem,* is the central negative fact about computation. It is closely related to the seemingly opposite phenomenon of universality: although a universal machine can simulate the step-by-step behavior of all Turing machines, no Turing machine could foresee their ultimate behavior (halting or not halting) without falling into a contradiction. More precisely, consider the function of two integer variables $h(x,y)$ defined to be 1 if the x'th Turing machine (in some standard list of Turing machines) halts on input y, and 0 otherwise. If this function were computable by some Turing machine, then so would the one-variable function $g(x)=h(x,x)$ which equals 1 if the x'th Turing machine halts when fed *its own* index number x as input. Turing showed that any Turing machine that computes $g(x)$ can be slightly modified to get another Turing machine (call it m) that *fails* to halt on input x if and only if the x'th machine halts on x. This machine, when fed its own index m as input, would then both halt and fail to halt, contradicting the assumed computability of $h(x,y)$. Thus there is no general technique for solving all instances of the halting problem, even though specific cases can often be decided easily (eg Fig 1). Turing went on to ask what would happen if a universal machine were enhanced by a hypothetical agency, or *oracle,* able to supply correct values for $h(x,y)$, and showed that the enhanced machine's halting problem would remain unsolvable even with the help of the oracle; repeating this argument one obtains an infinite hierarchy of unsolvable problems. *Computability theory,* also known as *recursive function theory,* deals with the classification of solvable and unsolvable problems, and relations such as Turing reducibility, a problem A being called *Turing-reducible* to another problem B if A can be solved with the help of an oracle for B.

Originally used to demonstrate the existence of unsolvable problems, Turing machines were used, beginning in the 1960s, to classify the level of difficulty of solvable problems, and to prove the existence of arbitrarily hard-to-solve problems. This field is called *computational complexity theory*, and one if its main notions is the complexity class **P**, defined as the set of problems which can be solved by a Turing machine using time (number of cycles) growing no faster than some polynomial in the size of the input (measured in bits). The problem of multiplying two binary numbers, and the problem of testing whether a string of parentheses like ()(((((()))()()))()())) is balanced or not are both in **P**, because there are Turing machines that solve them efficiently. Another important complexity class, called **NP**, consists of problems for which, although there might be no efficient way to *find* a solution, the validity of a proposed solution can be *checked* in polynomial time by a Turing machine. The most famous problem in **NP**, called the traveling salesman problem, asks, given a list of cities and distances between them, for a path visiting them all but having less than a prescribed total length. Any proposed path can be easily tested to verify that it indeed visits all the cities and is shorter than the prescribed length, but finding such a path, or proving that none exists, is believed by many to be exponentially hard. Indeed one of the most important open questions in mathematics, called the **P=NP** question, is whether there is an (as yet undiscovered) efficient way of solving the traveling salesman problem and a large class of problems equivalent to it. Beyond **P** and **NP**, computational complexity theory has shown that some problems are definitely very hard; in the sense that for every computable function of input size, there is a solvable problem requiring more than that amount of time and tape to solve.

Experience with computational complexity led to a *strong Church-Turing thesis* asserting that any reasonable model of computation (eg not involving exponentially growing parallelism) can be simulated in polynomial time by a Turing machine, supplemented by coin tossing to allow the execution of probabilistic algorithms. Recently this thesis has had to be modified to take account of quantum computers, which can be considered reasonable since they obey the laws of physics as currently understood. Quantum computers, while they cannot solve unsolvable problems, appear able to solve a few solvable problems exponentially faster than conventional computers can. Taking this into account, the current form of the strong Church-Turing thesis holds that any reasonable model of computation can be simulated in polynomial time by a quantum Turing machine, ie a Turing machine able to generate and coherently process arbitrary quantum superpositions of ordinary Turing machine states. Quantum computers are the latest of several instances where Turing machines have been used to connect mathematics and physics; earlier examples include the original concept of the Turing machine, which Turing based on the kinds of steps a human might do in performing a computation, and the theory of reversible Turing machines, which have been used to study the thermodynamics of computation and to show that simple mechanical systems, such as a single particle bouncing around in a finitely complicated three dimensional enclosure, can be computationally universal.

Another important application of Turing machines also dating from the 1960's is *algorithmic information theory,* which addresses the question of what it means for an individual bit string like 100100010101011011111010100 to be "random". Intuitively, a random sequence is one so patternless that it cannot be described more concisely than by explicitly listing its digits. Algorithmic information theory formalizes the notion of description as the input:output relation of a standard universal Turing machine, and goes on to define an *algorithmically random* bit string as one not computable as output from any input shorter (in number of bits) than the string itself. Algorithmic randomness has many tantalizing features, for example, while most bit strings are algorithmically random, hardly any them can be proved so within any given consistent axiomatic system. This fact is a version of *Gödel's incompleteness theorem* (qv) in mathematical logic.

For further study see

Hartley Rogers *Theory of Recursive Functions and Effective Computability* (MIT Press 1987, ISBN: 0262680521).
http://www.wikipedia.org/wiki/Theory_of_computation
Christos H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.
Charles H. Bennett, "Undecidable Dynamics" *Nature* **346**, 606-607 (1990).
G.J. Chaitin, “Randomness and mathematical proof,” *Scientific American*, **232** (5) (May 1975)