Alan Turing and the Foundations of Computer Science

Waseda University, SILS,
History of Mathematics

Formalism

Around the turn of the 20th century, there was a shift towards formalism among mathematicians. G. Frege and G. Peano began to formalize logic and arithmetic; E. Zermelo established axiomatic set theory; and all of this was continued by A.N. Whitehead and B. Russell in their *Principia Mathematica*.

D. Hilbert gave an address, titled "Axiomatic thought," 1917, in which he argued that mathematicians should use mathematical logic to develop proof theory and metamathematics in order to determine whether or not a given set of axioms are *independent* and *consistent*, and whether or not the theorems developed from them are *complete* and *decidable*.

In the following years, mathematicians like K. Gödel, A. Church and A. Turing began to address these questions using new mathematical approaches.

Giuseppe Peano's axioms of arithmetic (1889)

Unity: 1. $1 \in \mathbb{N}$.

Equality: 2. $n \in \mathbb{N} \Rightarrow n = n$. [reflexive]
3. $n, m \in \mathbb{N}, n = m \Rightarrow m = n$. [symmetric]
4. $n, m, o \in \mathbb{N}, n = m, m = o \Rightarrow n = o$. [transitive]
5. $n \in \mathbb{N}, n = m \Rightarrow m \in \mathbb{N}$. [closed]

Successor: Where $S(n) := n + 1$,
6. $n \in \mathbb{N} \Rightarrow S(n) \in \mathbb{N}$.
7. $n, m \in \mathbb{N}, n = m \iff S(n) = S(m)$. [injection]
8. $n \in \mathbb{N} \Rightarrow S(n) \neq 1$.

Induction: 9. $K := \{1 \in K, n \in \mathbb{N}, n \in K \Rightarrow S(n) \in K\} \Rightarrow K = \mathbb{N}$.

[Peano, did not consider 0 to be a member of the natural numbers. In those places where he used 1, we now use 0.]

Independence

For a set of axioms to be *independent* means that there should be no superfluous axioms. That is, no individual axiom should be demonstrable from any subset of the remaining axioms.

For example, for many centuries some mathematicians believed that Euclid's 5^{th} (parallel) postulate was not independent, and, hence, tried to prove it from the other "axioms." Eventually, with the development of non-Euclidean geometry, and a reinvestigation of the previous attempts to prove the 5^{th} postulate, it was realized that all of the previous proofs of the postulate relied on some assumption that was mathematically equivalent to the 5^{th} postulate. Hence, Hilbert set out an equivalent axiom (Playfair's) as independent and necessary for Euclidean geometry.

Consistency

For an axiomatic system to be *consistent*, there must be no contradictions among the axioms and the theorems that can be derived from them. That is, in a consistent system it cannot be possible to use some subset of the axioms to show a contradiction with another axiom, or to derive two theorems which contradict one another.

If you can use an axiomatic system to derive both P and *not-P*—a contraction—then the system is inconsistent, and mathematically worthless.

Hence, the question arises, given some set of axioms, is there a *general method* that we can use to determine whether or not they are consistent.

Completeness

When an axiomatic system is *complete*, all statements that can be composed of the terms defined in the system can be shown to be either valid or invalid—that is, they can be proved to follow from the axioms, or proved to contradict one or more of the axioms. More loosely, it means that every true theorem can be derived from the axioms.

That is, a system of axioms, $\mathbf{A}$, is complete when given *any* statement P, it is either the case that

$$\mathbf{A} \Rightarrow P, \text{ or } \mathbf{A} \Rightarrow \textit{not-}P.$$

This leads to the final issue, having to do with the question of whether or not we can know whether or not $\mathbf{A} \Rightarrow P$, or $\mathbf{A} \Rightarrow \textit{not-}P$, for any given P.

Decidability

An axiomatic system is *decidable* when we can state a general procedure that can be used to determine, for a given P, whether or not $\mathbf{A} \Rightarrow P$, or $\mathbf{A} \Rightarrow$ *not-P*. That is, irregardless of whether or not a system is complete, does a decidability procedure exist?

For example, if we could state some procedure that would show that $\mathbf{A} \not\Rightarrow P$ or $\mathbf{A} \not\Rightarrow$ *not-P*, then the system might be incomplete, but it could still be *decidable*.

Might there be such a decidability procedure that could determine in a general way whether or not P or *not-P* follows from the axioms by a finite series of steps?

In the early decades of the 20^{th} century, a number of mathematicians, at the urging of D. Hilbert, began to examine these metamathematical questions.

David Hilbert (1862–1943)

- Born to a middle-class Prussian family.
- Educated at Göttingen, and after some years at Königsberg, was professor of mathematics at Göttingen for most of his life.
- He and F. Klein developed an important school of mathematicians, of which he became the leading figure.
- Developed and worked in a broad range of areas.
- Advocated formalism and rigor in mathematics.

The 1900 International Congress of Mathematics

At the 2nd ICM, Paris, Hilbert was asked to give one of the major, general addresses. At the suggestion of his colleague H. Minkoski, he chose to talk about what he thought would be the important goals of mathematics in the 20th century. He set out a list of 10 problems (out of 23 published later) that became deeply influential among mathematicians up to this time and are known as "Hilbert's Problems."

Hilbert's Address, 1900

"[The] conviction of the solvability of every mathematical problem is a powerful incentive to the worker. We hear within us the perpetual call: There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no *ignorabimus*."

Hilbert's Problem 10 (the *Entscheidungsproblem*)

Problem 1 Show that the cardinality of the continuum is the next transfinite number after that of the natural numbers.

Problem 2 Show that the axioms of arithmetic are consistent.

Problem 8 Demonstrate the Riemann hypothosis, Goldbach's conjecture, the twin prime conjecture, etc.

10. Entscheindung der Lösbarkeit einer diophantischen Gleichung. (Determination of the Solvability of a Diophantine Equation.[1])

Given a diophantine equation …: *To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.*

[1] A polynomial equation with integer coefficients.

Foundations and Crisis

The goal of the foundations of mathematics was to set out a system of logical or metamathematical axioms from which all the valid theorems of mathematics could be derived. In order to show that this had been done it would then be necessary to show that these axioms were *independent* and *consistent*, and that the system was *complete* and all theorems were *decidable*.

As time passed, however, it began to become increasingly clear that there might be problems with this project. Bertrand Russel (1872–1970) discovered a paradox concerning the set of all sets that are not members of themselves, $R := \{x \mid x \notin x\}$, and wrote the *Principia Mathematica* with Whitehead, in which they took hundreds of pages to prove that $1 + 1 = 2$. Finally, Gödel and Turing showed that even relatively simple sets of axioms are not complete and not decidable.

Gödel's Completeness Theorem

Kurt Gödel (1906–1978) was an Austrian mathematician who later settled in the US and worked at the Princeton Institute for Advanced Studies.

In 1929, in his PhD thesis, Gödel demonstrated that first-order predicate logic—a logic with relations and quantifiers—is complete. This means that every semantically valid theorem is syntactically derivable from the axioms. A number of simpler versions of this proof were given over the following decades.

Although this was a significant result, it was not unexpected. For mathematics, however, the main issue was that even the most basic set of axioms—like Peano's axioms of arithmetic, or Zermelo's axioms of set theory, required extra, purely mathematical, ideas.

Gödel's Incompleteness Theorems

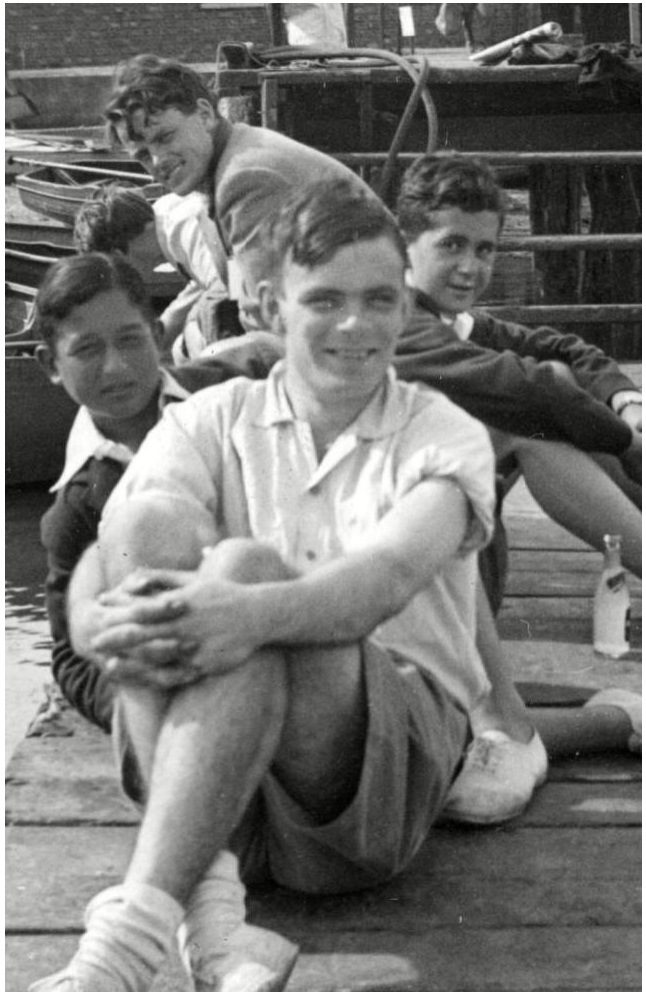
Gödel announced, in 1930, that he had shown that if sufficient axioms were added to first-order logic to allow the derivation of arithmetic, then the system was *incomplete*. In 1931, he published his two incompleteness theorems.

The first theorem states that no consistent set of axioms that can be listed by an *effective procedure* is capable of proving all statements that are true of the natural numbers. The second incompleteness theorem, which is a generalization, states that such a system cannot demonstrate its own consistency.

The key to the proofs was an argument analogous to Cantor's diagonal argument. We assume a complete listing, and show that it is incomplete. (It is often claimed that these theorems show that all axiomatic systems are incomplete, but this is false. Gödel had shown that first-order logic is consistent.)

Alan Turing (1912–1954)

- Turing was born to a British colonial family and his father was often away in India.
- Took a first class in mathematics from King's College, Cambridge.
- Became fellow at Kings; worked with A. Church in Princeton – took his PhD.
- During WWII, played a key role in breaking German codes at Bletchley Park.
- Worked on early electronic computers.
- Was arrested for homosexuality, medicated, and died mysteriously.



"On Computable Numbers…" (1936)

In 1935, in the first year of his fellowship at King's College, he wrote a paper called "On Computable Numbers, with an Application to the Entsheidungsproblem."

- He defines a computable number as a real number whose digits can be computed by any finite means.
- Defines a idealized computing machine (Turing Machine, TM), and introduces the idea of an enumeration of computing machines.
- Argues that there is a universal computing machine (UTM).
- Uses the diagonal argument—as developed by Cantor and Gödel—to show that some numbers are *not computable*, which is another way of saying that they are *undecidable*.

Abstracting the Idea of Computation

"On Computable Numbers…," 1936

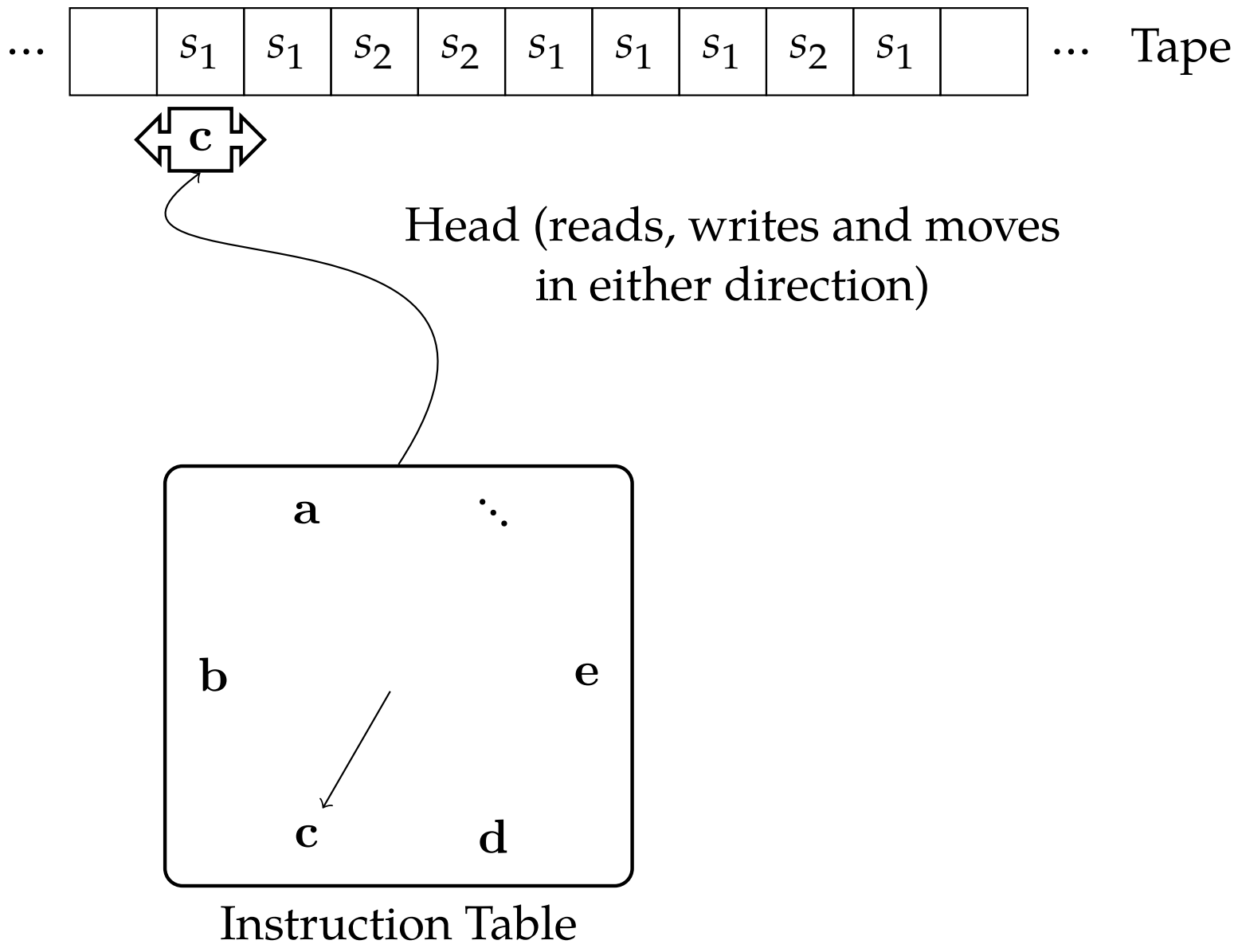
Computing is normally done by writing certain symbols on paper. We may suppose this paper divided into squares like a child's arithmetic book… I think it can be agreed that the two-dimensional character of the paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e. on a tape divided into squares. I shall suppose that the number of symbols which may be printed is finite.… The behavior of the computer at any moment is determined by the symbols that he is computing, and his 'state of mind' at that moment.… Let us imagine that the operations performed by the computer to be split up into 'simple operations' which are so elementary that it is not easy to imagine them further divided.

The Turing Machine

In order to address the question of computability, Turing devised a conceptual model of a computing device, which was later called a "Turing machine" (TM). It consists of a non-finite tape, a head, and an instruction table.

- The tape is divided up into cells, each of which can be blank, or contain a symbol from some finite set, $\{s_0, ..., s_n\}$. The tape may also contain instructions for the machine.
- The head can read and write/erase symbols, $\{s_0, ..., s_n\}$, in the cells of the tape, and can move one cell right or left.
- The instruction table issues the following possible instructions: *move right*, *move left*, *print*, *change state*, *halt*.

No mechanisms for performing any of these functions are described in the paper—the machine is *purely conceptual*.


s_1 s_1 s_2 s_2 s_1 s_1 s_1 s_2 s_1
Tape
c
Head (reads, writes and moves in either direction)
a
b
e
c
d
Instruction Table

A Basic Program

As an example, Turing gives the following "instruction table":

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

This table starts out with a *blank tape*—actually, it will fail if the tape is not blank—and prints the following sequence:

The Universal Turing Machine

He then gives an idealized construction of a general machine—that is, a general instruction table—that can read and execute instructions coded onto the tape. In this way, the machine can be controlled by some set of instructions coded onto the tape itself, which acts as a *stored-program*.

Since any program that we like may be coded onto the tape, the general machine can carry out any computation that an indefinitely large set of instruction tables can carry out. That is, the set of programs that can be run on the general machine is the denumerable set, with cardinality $\aleph_0$.

This conceptual machine was later called a "universal Turing machine" (UTM), or simply a Turning machine.

The *Entscheidungsproblem* (Decidability Problem)

Turing approached the *Entscheidungsproblem* by an argument that was explicitly analogous to the diagonal argument developed by Cantor to show that the reals are non-denumerable. He argued that if we set all of the computable numbers out in a denumerable table—for example, we have a TM to compute 2, another for $\sqrt{2}$, another for π, and so on—then we can define a number by the diagonal process. Then we can ask whether or not another TM can be written which can compute that number. That is, whether or not it is possible to decide if the process of defining that diagonal number halts. Turing showed that this is not always possible.

The generalization of this claim involves the following assumption.

The Church-Turing Thesis

The Church-Turing thesis is the claim that any computation that can be carried out by a rote method—a well-defined effective procedure—can be carried out by a UTM. That is, a UTM "can do anything that could be described as 'rule of thumb' or 'purely mechanical.'"

This is a definition of the concept of "computable" to mean anything that can be carried out on a UTM. An extension of this is the claim that any process that can be carried out on one UTM can also be carried out on any other UTM.

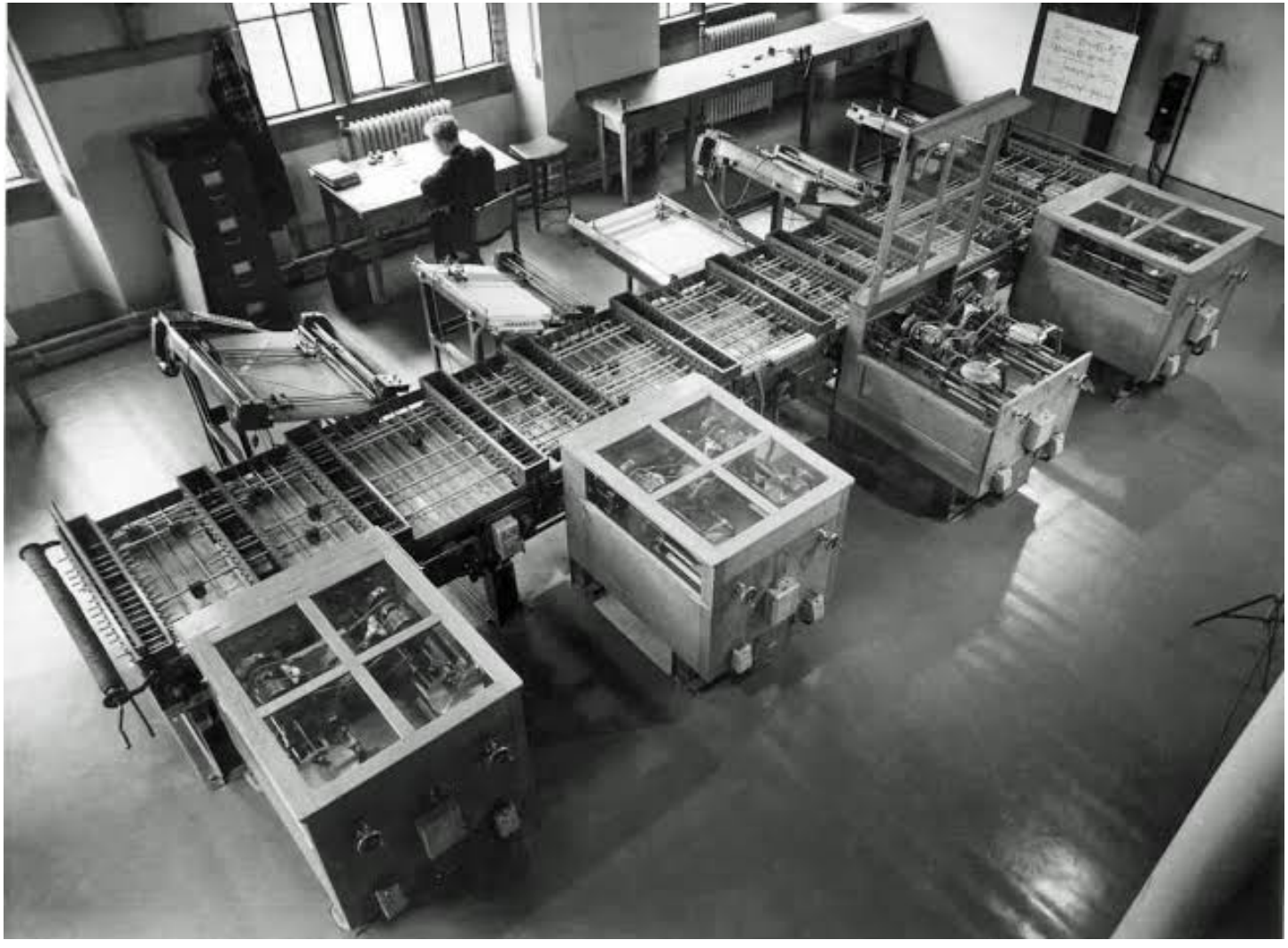
It is sometimes claimed that this means that a UTM can carry out any computation that can be carried out by *any* machine, but this is not true—a quantum computer or analog machine that incorporates a random mechanism are clear counterexamples.

Claude Shannon's Logic Circuits

In 1854, George Boole (1815–1864) had published the *Laws of Thought*, in which he showed that an algebra on binary arithmetic could be used to codify classical logic and extend it to expressing and solving equations.

In 1936, Claude Shannon (1916–2001) enrolled at MIT as a master's student under Vannevar Bush, working on Bush's differential analyzer – an analog computer that had to be manually set for every equation it solved. In 1938, in his MS thesis, *A Symbolic Analysis of Relay and Switching Circuits*, Shannon showed that electric logic circuits could be used to simplify the electromechanical relays then in use in telephone networks, and that, in general, these circuits could solve any problem that Boolean algebra could solve. He also gave a few examples of basic circuits.

Vannevar Bush's Differential Analyser



The Concept of the Stored-Program Computer

Contained in Turing's 1936 paper was a concept that would play a fundamental role in the development of computers as technological objects—namely, the idea of the stored-program calculator, or computer. The key idea is that the logical control and arithmetic functions could be written into the core of the UTM, and any auxiliary program could be written out on the tape and read in as necessary.

This idea was popularized by John von Neumann (1903–1957), a Jewish-Hungarian mathematician who immigrated to the US. He worked closely with American engineers and encouraged them to read Turing's 1936 paper. Von Neumann later went on to write a government report called "First Draft of a Report on the EDVAC." This report introduced the idea of the stored-program computer to a wider readership of engineers.

British Codebreaking in WWII

Starting in 1926, the German military adopted the use of the Enigma encoding machine, made by the firm Scherbius & Ritter. The Dutch, Japanese, and Italian militaries followed.

Before the outbreak of war, the Poles bought an Enigma and their mathematicians designed a machine, the Bomba, to decode it. When they realized the Germans were going to attack, they gave the machines to the British and the Polish mathematicians taught the Brits what they had learned.

The British government set up a code-breaking division at Bletchley Park, where they built "Bombes" and eventually Colossus computers, to decode the new German Lorenz encoders. The first Colossus used 1,500 vacuum tubes and read paper tape at 5,000 characters per second. The Colossi decrypted around 63,000,000 characters of German code.

A German Enigma Machine

The Enigma Machines

The Enigma machines were a series of rotor cipher machines that combined mechanical rotors with electrical sub-systems.

Mechanical: Keyboard for the input letters, rotating disks on a spindle, a stepping mechanism to rotate the disks.

Electrical: A circuit that varied with different rotor positions, a plugboard that could be reconfigured by the operator, and lamps for the output cipher letters.

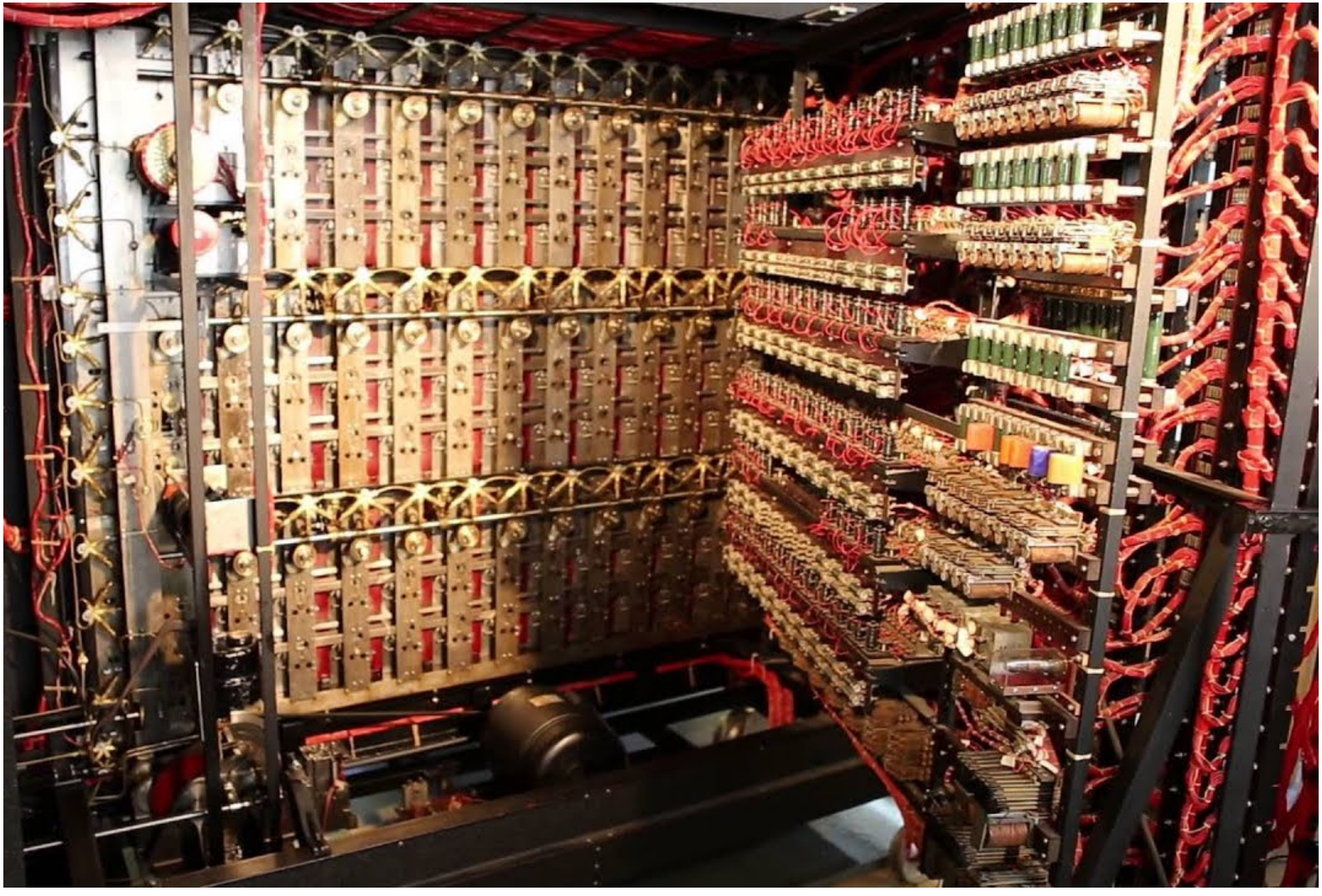
Both German operators had a series of daily keys that they would use to set up the machines. Then the transmitter would enter letters on the keyboard and the lamp would indicate the output characters. This message would be transmitted normally on open radio, and as long as the receiving operator had the same set-up, they could enter the cyphers and receive normal German output.

Turing at Bletchley Park

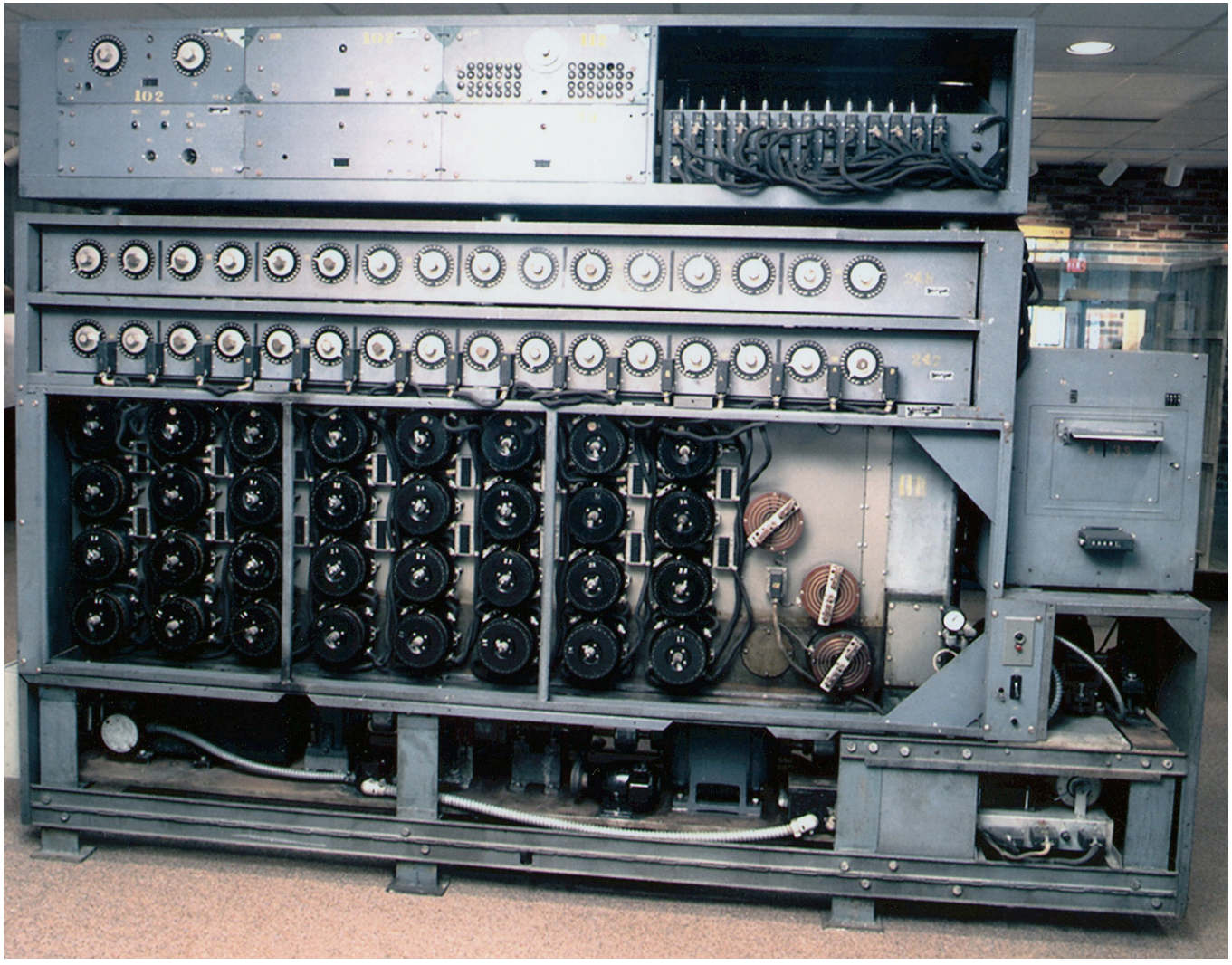
Bletchley Park was a manor house that had been converted to the use of the Government Code and Cypher School. Here Turing worked on a team with a number of other mathematicians and engineers. Based on the previous work of the Poles, in 1939, Turing and Gordon Welchman designed a Bombe that could elecro-mechanically search for a certain string—called a crib—at a rate of 20 positions per second. When the Bombe stopped, the positions would be tried on an Engima to see if it produced German. If not, they would keep going, if so, the messages for that section of the German military could be read in near real time for the rest of the day. The following day the whole process would begin again.

In 1943, T. Flowers built the Colossus, a large-scale electronic computer based on designs by Max Newman, one of Turing's mathematics professors, to decrypt the new Lorentz machines.

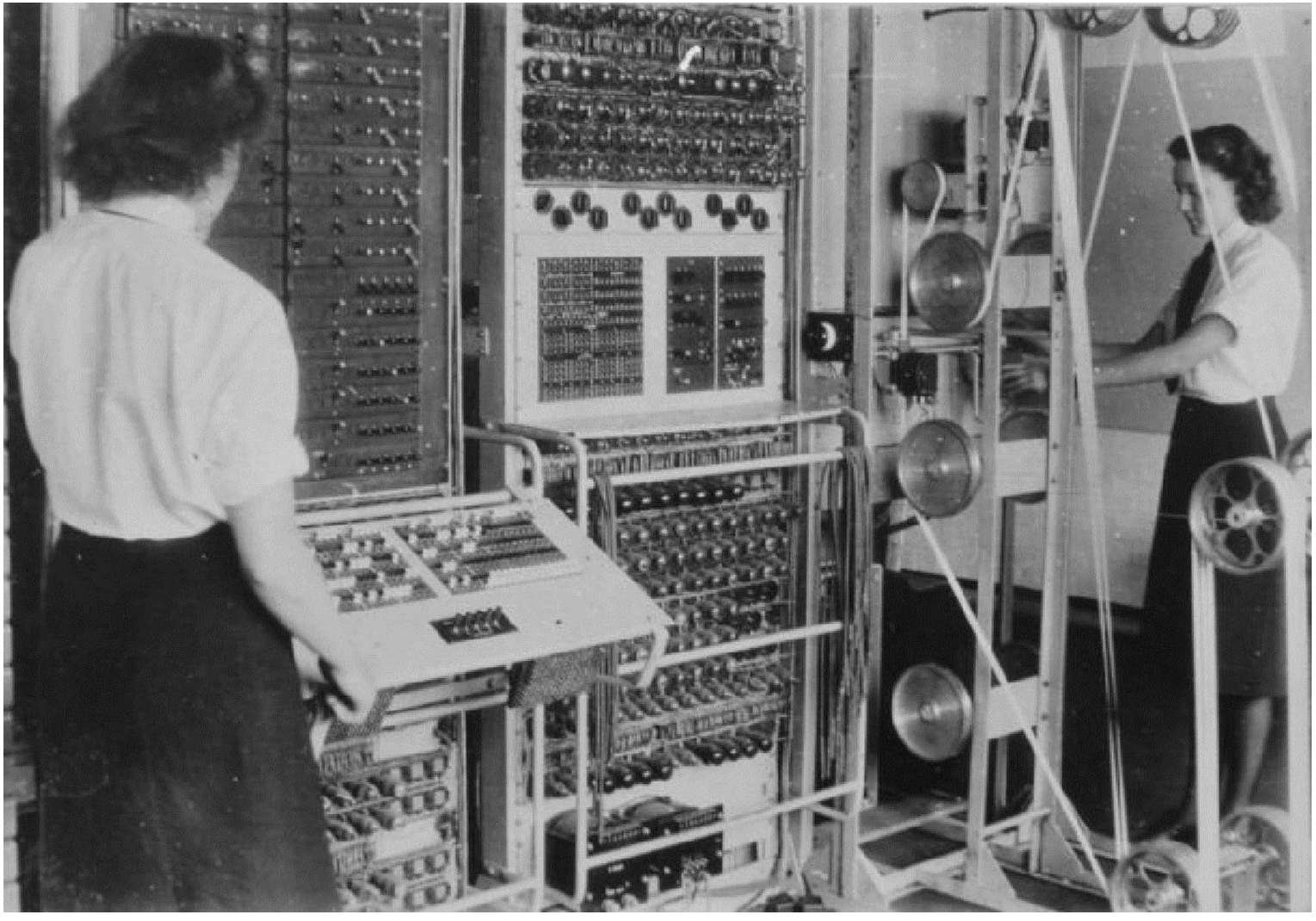
Turing's Version of the Bombe (Reconstruction)



A Bombe machine in the US



The Colossus at Bletchley Park



Turing at the National Physical Laboratory (NPL)

At the NPL in Bushy Park, London, Turing set out the designs for an electronic stored-program computer, called the Automatic Computing Engine (ACE). In 1946, he wrote a report called "Proposed Electronic Calculator" that included many of von Neumann's ideas, but also set out detailed designs for logic control and programming.

However, because of the secrecy about the work at Bletchley Park, Turing was not able to tell the engineers about the Colossus, and they did not believe that it was possible to build a fully electronic devise—also Flowers, who had been at Bletchley Park, could not be employed to do the job. Hence, Turing's full design was not implemented and at first only a pilot ACE was built. In 1948, Turing left the NPL and moved to Manchester.

Turing at Manchester

At Manchester, Turing joined Newman's group, including the engineers Williams and Kilburn, who were working on the Manchester Baby, and the Mark I, which were fully-implemented, electronic, stored-program computers—having worked with and been influenced by the people at both Bletchley Park and the Moore School in the US.

When he arrived at Manchester, Turing finally had a fully functional general-purpose electronic computer to work with. He designed the input mechanism, the programming system and wrote a programming manual. Here, he frustrated the engineers by working directly in the base-32 of the machine, since he found it more efficient to program in this base system than to convert everything back and forth between decimal.

Artificial Intelligence

Before he started at Manchester, Turing took a sabbatical at King's College and wrote a report called "Intelligent Machinery" (IM). This was a highly philosophical paper that summarized many of his ideas on machine intelligence in which he introduced the concepts of logic-based problem solving and genetic or evolutionary search. He claims in this paper that "intellectual activity consists mainly of various kinds of search." (At Manchester, Turing wrote one of the first chess programs—purely on paper.)

A major concept of the IM paper was that unorganized machines could act like neurons and by following simple rules organize themselves into networks—now called Turing nets—and eventually carry out general computation.

Artificial Life

In the final years of his life, Turing worked in the field that is now known as artificial life (A-life). The goal of A-life is to produce a theoretical understanding of the way that organisms self-organize.

Turing used the early Ferranti computers (the commercial versions of the Mark I) to study the spontaneous formation of structure related to early cell division in embryos; symmetrical structures in shell fish, starfish and flowers; the arrangement of leaves and branches in plants; and color patterns, such as spots and striping on animals and fish.

Unfortunately, in the middle of this work, he died. It is generally believed that he committed suicide, but there are problems with this interpretation (coroner's report mentions violence, no evidence of depression, etc.).

The Development of Computer Science

The Turing machine notion of computability, the Church-Turing thesis, and the stored-program concept of the computer were all developed originally to show that certain mathematical problems were undecidable—that is, to show mathematical and logical limits to what is possible in mathematics.

To Turing himself, however, and to the first generation of computer programers—on paper—and engineers, more interesting questions soon began to circulate around what could actually be done with these Turing machines.

The new computer science stimulated work in mathematics in various ways—attention to detail around computation and base systems, new logical theories to study the structural similarities between programs and proofs, new experimental methods for investigating mathematical claims, and so on.