Two fundamental concepts

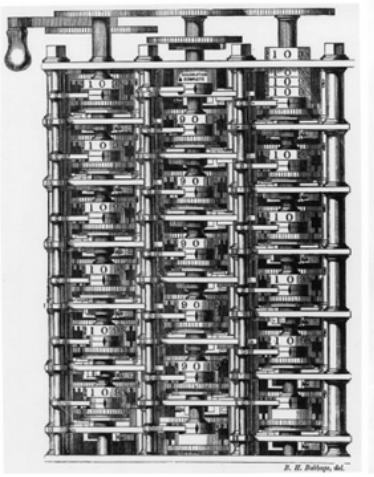
- Algorithms/Programs
  Algorithms were used since antiquity, but became an object of study only early in the 20th century.

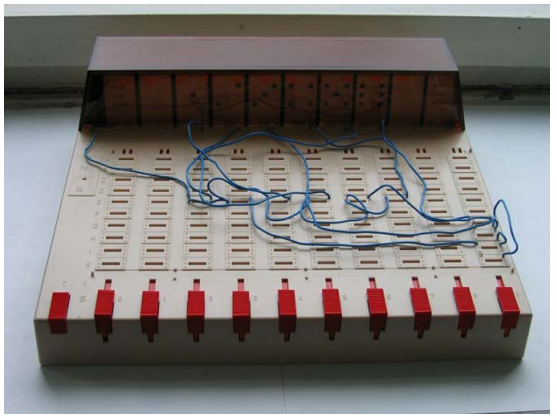
- Universal Computer
  In early "computers", the program was part of the hardware. You had to rewire the computer to run a different program.
  In contrast, a universal computer is a computer that can run any program.

1936 was the *annus mirabilis* of computation, the year where everything came to fit together.

Before universal computers

Charles Babbage's Differential Engine for computing tables of trigonometric functions, logarithms, etc.





Toy computer

Universal Computer

> *What shall we do to get rid of Mr. Babbage and his calculating machine? Surely if completed it would be worthless as far as science is concerned?*
>
> British Prime Minister Sir Robert Peel, 1842

Punched cards already existed in the early 19th century. Babbage's Analytical Engine used them to design a computer that can execute any program given to it.

A universal computer can execute an interpreter ("universal program"): It reads the source code of another program and executes it.

$$U(\langle P \rangle, x) = P(x)$$

So we can define a program V as follows: $V(\langle P \rangle) = \overline{P(\langle P \rangle)}$.

Hilbert's problems

1900 David Hilbert gave a speech to the International Congress of Mathematicians presenting 23 open problems.

Hilbert's 10th Problem:

Specify a procedure which, in a finite number of operations, enables one to determine whether or not a given Diophantine equation with an arbitrary number of variables has an integer solution.

polynomial equation with integer coefficients

1928, Hilbert posed the Entscheidungsproblem:

The Entscheidungsproblem is solved if one knows a procedure that allows one to decide the validity of a given logical expression by a finite number of operations.

What is a "procedure"

Hilbert didn't say what he meant be a procedure, and computers were still decades away.

The early 20th century saw mathematicians struggle with putting mathematics on a foundation of set theory and logic.

Russell's paradox: $R = \{S \mid S \notin S\}$.

In the early 20th century models of computation were defined using ideas from set theory, but it was not clear whether any of them captured the universe of all conceivable computations.

In 1936, all these models turned out to be equivalent, giving rise to the Church-Turing Thesis.

RE and coRE

$$\textsc{Halts}(P, x) = \exists t\ \textsc{HaltsInTime}(P, x, t)$$

(Recursively) enumerable problems (RE) are exactly the problems A of the form $A(x) = \exists w\ B(x, w)$, where B is decidable.

coRE is the family of complements of enumerable languages. A language A is in coRE if and only if it can be expressed as $A(x) = \forall w\ B(x, w)$, where B is decidable.

We have shown

$$\text{Decidable} = \text{RE} \cap \text{coRE}$$

The next level

Consider the problem: Given a program P, does P hold for every possible input x?

We can express this as

$$\forall x \exists t\ \textsc{HaltsInTime}(P, x, t)$$

Imagine a hypercomputer that has access to a black box that solves the halting problem (in finite time).

A hypercomputer cannot solve the halting problem for hypercomputers.

> Imagine a hyperhypercomputer that has access to a black box that solves the halting problem for hypercomputers.
>
> > Imagine a hyperhyperhypercomputer that has access to a black box that solves the halting problem for hyperhypercomputers.

Unprovable truth

A formal system has a set of axioms and rules of inference.

A theorem is a statement that can be proved, with some finite chain of reasoning, from the axioms.

A formal system is consistent if there is no statement T such that both T and $\overline{T}$ are theorems, and complete if, for all T, at least one of T or $\overline{T}$ is a theorem.

In 1931, Kurt Gödel proved that no sufficiently powerful formal system is both consistent and complete.

His proof constructs a self-referential statement that says:

This statement cannot be proved.

Defining computation

There were several attempts to define computation. Mathematicians concentrated on computable functions.

- Primitive recursive functions are functions built from simpler functions (with zero and the successor function as the basis) and primitive recursion. It is equivalent to a straight-line program with for-loops.

The Ackermann-function is not primitive recursive.

- Partial recursive functions add μ-recursion (equivalent to while-loops).
- λ-calculus defines computation by functions that operate on strings:

$$(\lambda x : xax)bc \to bcabc$$

Church-Turing Thesis

[A Turing machine] is able to imitate any automaton, even a much more complicated one. . . It has reached a certain minimum level of complexity. . . an automaton of this complexity can, when given suitable instrcutions, do anything that can be done by automata at all.

John von Neumann

Any system that can simulate a Turing machine can carry out any computation at all. It is computationally universal.

The grand unification

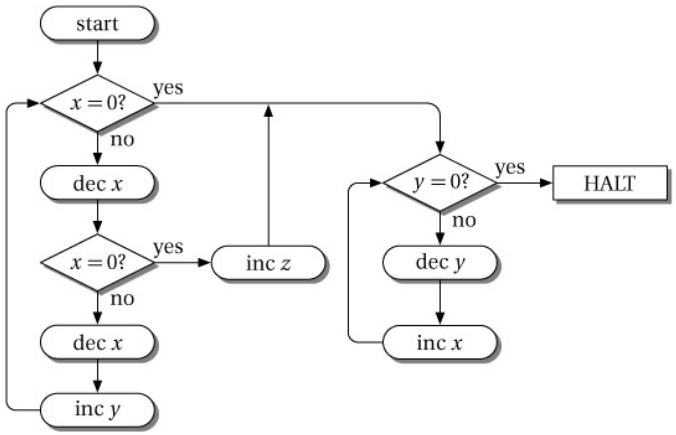
Partial recursive functions and the λ-calculus turned out to be equivalent and are powerful. But do they cover everuthing that Hilbert would have called a "finite procedure"?

There were other definitions of recursive functions, and Gödel thought the λ-calculus to be "thoroughly unsatisfactory".

The Turing machine was more convincing to many people, and it is equivalent to partial recursive functions and to the λ-calculus.

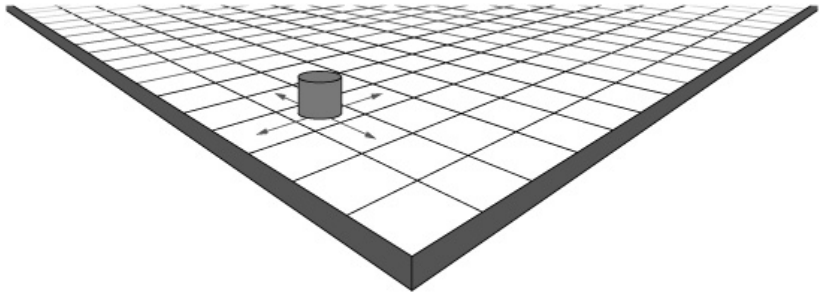
Counter machines

A counter machine is a finite automaton with a finite number of integer counters. It can increment and decrement the counters, and test if they are zero.



A two-counter machine can simulate a Turing machine.

Two-dimensional finite automaton



It cannot read or write, only move its position and test if it is on the edge.

This is a two-counter machine in disguise.

Fractran

This is a program in the programming language Fractran:

$$\frac{17}{91}\ \frac{78}{85}\ \frac{19}{51}\ \frac{23}{38}\ \frac{29}{33}\ \frac{77}{29}\ \frac{95}{23}\ \frac{77}{19}\ \frac{1}{17}\ \frac{11}{13}\ \frac{13}{11}\ \frac{15}{14}\ \frac{15}{2}\ \frac{55}{1}$$

The state is an integer n. In each stop, find the first fraction p/q in the list where $m = (p/q)n$ is an integer, and replace n by m.

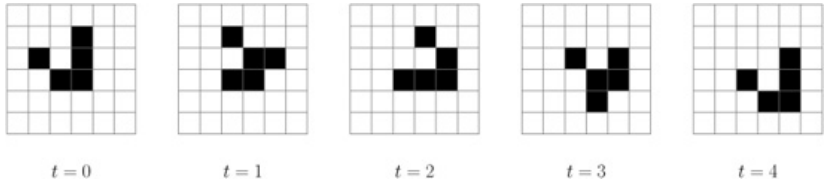
This program computes the prime numbers.

Fractran can simulate a counter machine, and therefore a Turing machine.

It is a special case of a Collatz sequence, which shows that the Collatz problem is undecidable.

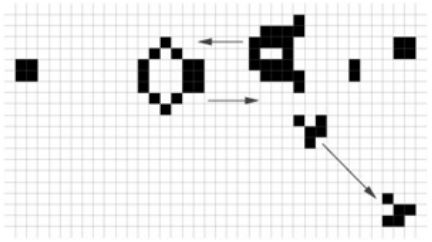
The Meaning of Life

Game of Life is a cellular automaton on the two-dimensional infinite lattice invented by John Horton Conway in 1970. A cell is born when it has exactly three live neighbors, and dies if it has less than two or more than three live neighbors (loneliness or overcrowding).



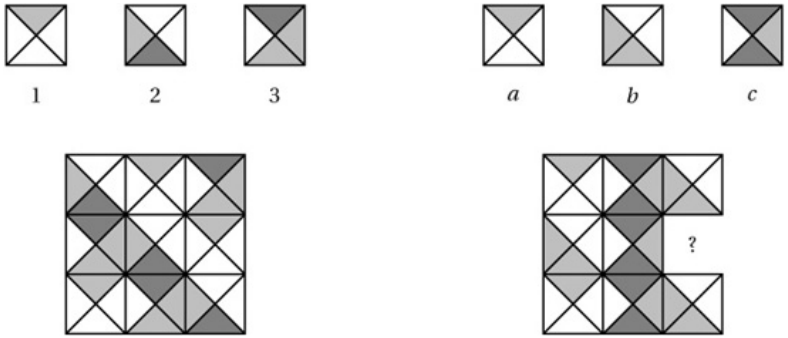
Life can compute

A glider gun:



Tilings

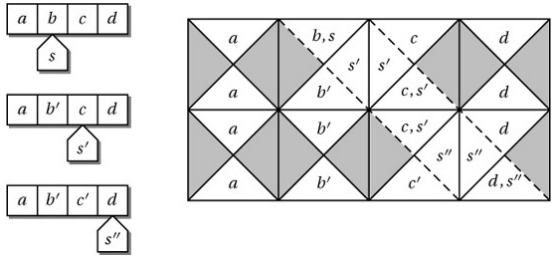
The Tiling problem is to decide if one can tile the two-dimensional plane with tiles of a given set of types.



Wang tiles have a color along each edge, and the color of neighboring tiles have to match.

Tiling is undecidable

Given a TM M, we design a set of Wang tiles s.t. each row is one step of M's computation.



If there is no tile with the HALT state, then the plane will be tiled if and only if M runs forever.