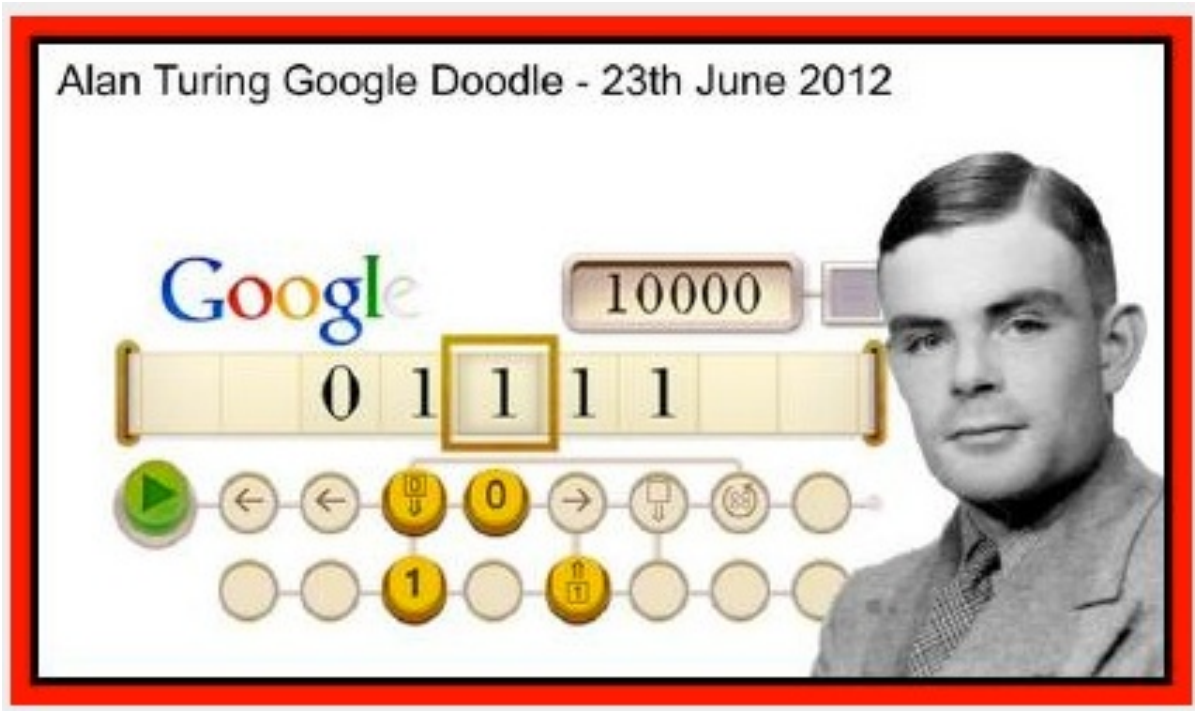
Turing Machines



October 24, 2025 Handouts:
Notes on Assignment 16
Assignment 17

David Hilbert

Born: January 23, 1862 in Königsberg, Prussia
(now Kaliningrad, Russia)
Died: February 14, 1943 in Göttingen, Germany

Hilbert Biography Link

"No one shall expel us from the Paradise that Cantor has created."

Turing Machines Algorithms, and Computing

Turing machines (TMs) can compute any function normally considered "computable." In fact, we can define **computable** to mean "computable by a TM".

At the International Congress of Mathematicians (Paris, 1900), David Hilbert identified 23 problems in Mathematics and posed them as a challenge for the coming century

Hilbert's 10th Problem

Devise an *algorithm** that determines whether a given polynomial (with one or more variables) with integer coefficients has an integral root.

* In Hilbert's words: "... a process according to which it can be determined by a finite number of operations."

For example $p(x, yz) = 6x^3yz^2 + 3xy^2 - x^3 - 10$ has four terms over the three variables x, y, z and has a root at $(x, y, z) = (5, 3, 0)$.

The polynomial $p(x) = x^2 - 3x + 2$ over the single variable x has two roots $\{1,2\}$ which can be found by factoring $p(x) = x^2 - 3x + 2 = (x - 1)(x - 2)$ or by using the quadratic formula.

The polynomial $p(x) = x^2 - 2 = (x - \sqrt{2})(x + \sqrt{2})$ over the single variable x has **no** integer roots.

Perhaps Hilbert assumed that an algorithm must exist (similar to the quadratic formula for single variable polynomials). Someone only needed to find it.

It soon became clear that mathematics lacked a clear definition for just what an "**algorithm**" was as well as just what it meant for something to be "**computable function**". At the turn of the century, these two terms (algorithm and computable) were not seen as essentially being the same.

Not having a precise definition for important terms is generally a problem for mathematicians!

During the 1930s, mathematicians were still trying to come to grips with the notion of just what was meant by the term "**computable function**" (as well as the notion of "**algorithm**" itself). Several alternative notions had been proposed each with its own peculiarities and all very different.

Here is a list of the most well known attempts:

λ-calculus	Alonzo Church, Stephen Kleene (1936)
μ-recursive functions	Kurt Gödel (1933)
combinatory logic	Haskell Curry (1930)
Post systems	Emil Post (1928)
Turing machines	Alan Turing (1936)

All these systems had a notion of "computation" in one form or another. They deal with various types of data, however. For example, Turing machines manipulate strings over a finite alphabet, μ-recursive functions manipulate the natural numbers, the lambda calculus manipulates "lambda" terms, and combinatory logic manipulates strings of "combinatory symbols".

It is easy to encode just about anything as strings in $\{0,1\}$, after all, we do this all the time when using modern day computers. Surprisingly, with suitable encodings as strings, it turns out that all the above formalisms can simulate each other. So despite their superficial differences, they are all equivalent models of "computation".

Of the classical systems listed above, the one that most closely resembles modern computers is the Turing machine.

The Church – Turing Thesis

When it was discovered that all these researchers were essentially talking about the same thing, Alonzo Church went out on a limb and declared that the intuitive notion of "computable by some algorithm", which mathematicians had sought to capture formally for some time, is captured precisely by Turing machines.

This assertion is known as **Church's Thesis** or **The Church – Turing Thesis**. It is not a theorem but rather a declaration that the formalism exactly captures our intuition of what it means to "compute" something

The Church – Turing Thesis

Intuitive notion of what it means to be computable via an algorithm

$\cong$ (is equivalent to) computable via a Turing machine

or

intuitive notion of algorithm $\cong$ TM

or

Any computation that can be carried out by mechanical means can be performed by some Turing machine.

Also, since there is a "**Universal Turing Machine**" that can simulate all other TMs, one can take the liberty of viewing a TM as being an algorithm and a universal TM as being a computer that "executes" algorithms (all other TMs).

Algorithm $\cong$ TM $\cong$ computer program (say in Java or Python)

computer $\cong$ universal TM

Turing Machines Algorithms, and Computing

Turing machines (TMs) can compute any function normally considered "computable." In fact, we can define **computable** to mean "computable by a TM".

Self-Reference

With "self-reference", one can end up with rather strange situations. In logic, for example, one might want all declarative sentences to either be true or false (XOR, "exclusive or"). Can there be a sentence that is **both**?

This sentence is false

Self-Reference

I am not provable

Gödel (in his famous "Incompleteness Theorem") actually showed that in any "sufficiently large" system of mathematics (containing the natural numbers or "axioms of Peano arithmetic", for example), there are statements that are true but not provable. He did this by essentially encoding the sentence "I am not provable" within his mathematical system using "Gödel numbers".