# REKURSIONSTHEORIE

RECURSION THEORY

WINTERSEMESTER 2021/22

# Lecture I

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

— Lectures will not be scheduled regularly, but picked month by month from a list of possible dates. [Moodle.]

— We continue until we have covered the planned material (approximately 13 to 15 lectures)

— Oral exams will take place once all lectures have been given.

Hand-written lecture notes will be uploaded to Moodle after each lecture.

# RECURSION THEORY VS COMPUTABILITY THEORY

They are the same. | Berechenbarkeits-theorie |

What is it about ?

Mathematical theory of computation
in particular, its limits.

---

Repetition from Logic about definitions, precision & negative results:

NOTION OF PROOF

Theorem ⟵ Proof

For giving a proof, I only need sufficient conditions for being a proof.

If you want to show negative results, i.e., that something is not provable, you need necessary conditions for being a proof.

Negative results require an ability to check that we covered all possible proofs.

## A non-logical example

GALOIS Theory.

Example: "There is no solution formula for quintics."

"There is no geometric construction of $\sqrt[3]{2}$ by ruler & compass."

DELIC problem

## Computation

We have many positive results in mathematics:

"There is an algorithm that computes ..."

....

This does not need necessary conditions for being an algorithm.

But: if we want to show that something is UNCOMPUTABLE, we need to have such a definition.

§ 11. Das Entscheidungsproblem im Funktionenkalkül und seine Bedeutung.

Die mathematische Logik leistet aber noch mehr als eine Verschärfung der Sprache durch die symbolische Darstellung der Schlußweisen. Nachdem einmal der logische Formalismus feststeht, kann man erwarten, daß eine systematische, sozusagen rechnerische Behandlung der logischen Formeln möglich ist, die etwa der Theorie der Gleichungen in der Algebra entsprechen würde.

Eine ausgebildete „Algebra der Logik" begegnete uns im Aussagenkalkül (man vgl. insbesondere § 4–9 des I. Kapitels). Die wichtigsten der dort erwähnten und gelösten Probleme waren das der *Allgemeingültigkeit* und der *Erfüllbarkeit* eines logischen Ausdrucks. Beide Probleme zusammen pflegt man auch kurz als das *Entscheidungsproblem* zu bezeichnen.

Bei dem Problem der *Allgemeingültigkeit* handelt es sich um die folgende Frage: *Wie kann man bei einem beliebigen vorgelegten logischen Ausdruck, der keine individuellen Zeichen enthält, feststellen, ob der Ausdruck bei beliebigen Einsetzungen für die vorkommenden Variablen eine richtige Behauptung darstellt oder nicht?*

Bei dem Problem der *Erfüllbarkeit* handelt es sich um die Frage, *ob es überhaupt eine Einsetzung für die Variablen gibt, so daß durch den betreffenden Ausdruck eine richtige Behauptung dargestellt wird.*

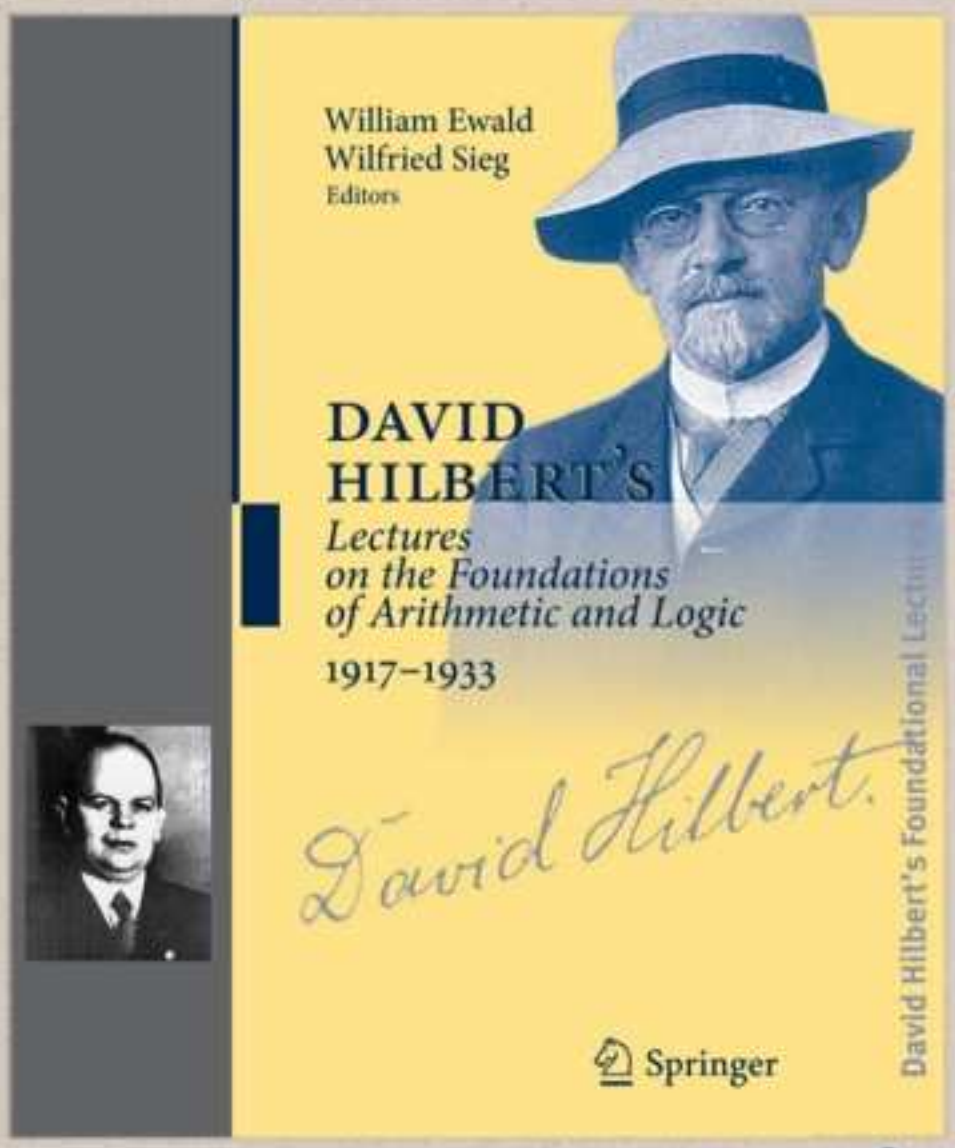
Beide Probleme sind zueinander dual. Ist ein Ausdruck nicht allgemeingültig, so ist das Gegenteil erfüllbar und umgekehrt.

from:
Hilbert-Ackermann,
Grundzüge der theoretischen Logik (1928)
excerpt from: Ewald & Sieg (eds.),
David Hilbert's Lectures on the Foundations of Arithmetic & Logic 1917-1933

The most important problems mentioned and solved there are those of *Validity* and *Satisfiability* of a logical expression. Both problems together are usually called in short the *Entscheidungsproblem*.

73

The problem of *Validity* is the following question: *How can we determine for an arbitrary given logical expression [...] whether it represents under arbitrary assignments for the variables a correct claim or not?*




MOTIVATION

In propositional logic, we have decision algorithms:

e.g., TRUTH TABLE ALGORITHM

$(p \wedge (q \rightarrow \neg p)) \vee q$

| p | q | $q \rightarrow \neg p$ | $p \wedge (q \rightarrow \neg p)$ | $(p \wedge (q \rightarrow \neg p)) \vee q$ |
|---|---|---|---|---|
| T | T | F | F | T |
| T | F | T | T | T |
| F | T | T | F | F |
| F | F | T | F | F |

Hilbert's 1928 question was supposed to have a positive answer ("Wir müssen wissen, wir werden wissen"). There sufficient conditions for what an algorithm is would would have sufficed.

But: the answer was NO.

So we first needed to come up with a precise definition of

DECISION PROCEDURE

[ COMPUTATION ALGORITHM ].

**Alonzo Church**

Alonzo Church (1903–1995)

| | |
|---|---|
| **Born** | June 14, 1903<br>Washington, D.C., US |
| **Died** | August 11, 1995 (aged 92)<br>Hudson, Ohio, US |
| **Citizenship** | United States |
| **Alma mater** | Princeton University |
| **Known for** | Lambda calculus<br>Simply typed lambda calculus<br>Church encoding<br>Church's theorem<br>Church–Kleene ordinal<br>Church–Turing thesis<br>Frege–Church ontology<br>Church–Rosser theorem<br>Intensional logic |

Theory of recursive functions

Based on functions definitions using recursion.

**Alan Turing**

OBE FRS

Turing c. 1928 at age 16

| | |
|---|---|
| **Born** | Alan Mathison Turing<br>23 June 1912<br>Maida Vale, London, England |
| **Died** | 7 June 1954 (aged 41)<br>Wilmslow, Cheshire, England |



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

*By* A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

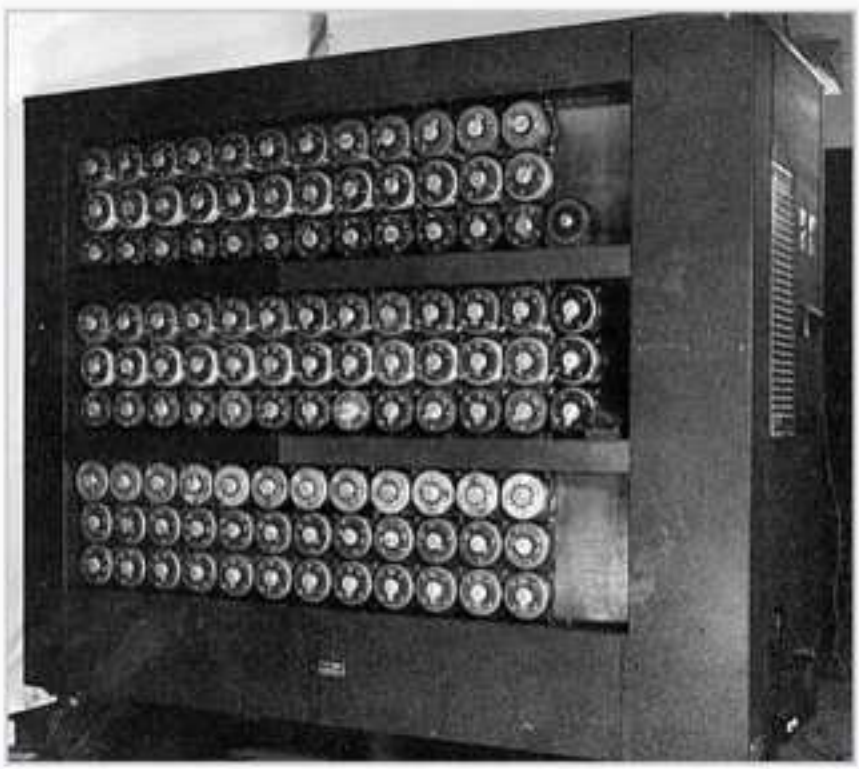
The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers $\pi$, $e$, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.

Military Enigma machine, model "Enigma I", used during the late 1930s and during the war; displayed at Museo Nazionale Scienza e Tecnologia Leonardo da Vinci, Milan, Italy



A wartime picture of a Bletchley Park Bombe

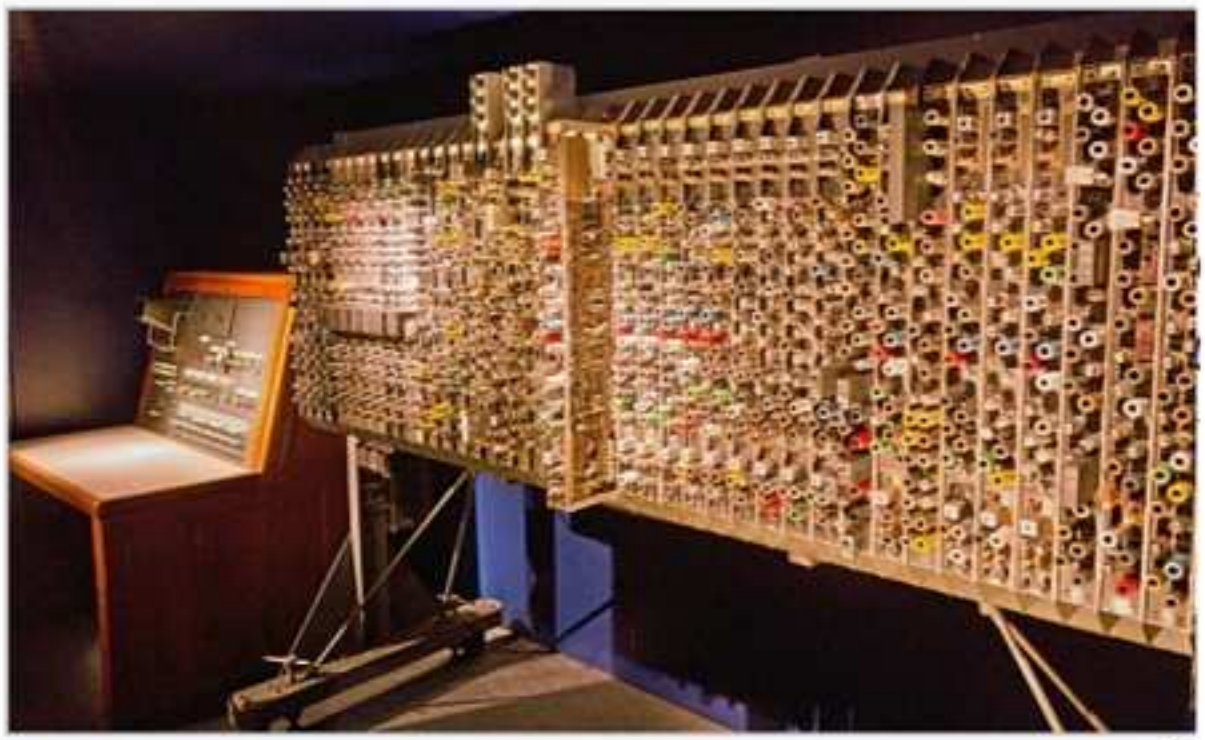
**Bletchley Park**

The mansion in 2017


| | |
|---|---|
| **Established** | 1938 (as a code-breaking centre); 1993 (as a museum) |
| **Location** | Bletchley, Milton Keynes, Buckinghamshire, England, United Kingdom |
| **Coordinates** | 51.998°N 0.741°W |

The Bombe was still custom-built for one purpose and did not follow the SOFTWARE PRINCIPLE (one machine for several purpose, dependizing on the program).

# Automatic Computing Engine



Pilot ACE

ACE

# §1 BASICS

**Def.** A set is called an alphabet if it is nonempty and finite.

Examples: 1. $\{0,1\}$

2. $\{a,b,c,d,\dots,z\}$

3. $\{a,b,c,d,\dots,z,0,1,\dots,9\}$

4. $\{a,A,b,B,c,C,\dots,z,Z,\sqcup,0,1,\dots 9,!,",\S,\$,\%,\dots\}$

↑ CONTENT OF A STANDARD KEYBOARD

**Def.** A word (more precisely, a $\Sigma$-word) is a finite sequence of elements of $\Sigma$. The set of words is $\Sigma^{<\omega}$ (set-theoretic notation) or $\Sigma^*$.

**Observation** $\Sigma^*$ is always countably infinite.

The empty word, set-theoretically $\emptyset$, is denoted by $\varepsilon$.

Set-theoretically, $w \in \Sigma^*$ is a function with $\mathrm{dom}(w) \in \mathbb{N}$. We call it the length of $w$, in symbols, $\mathrm{lh}(w)$.

**Def.** A function $w : \mathbb{N} \longrightarrow \Sigma$ is called an *infinite word*.

The set of infinite words is denoted by $\Sigma^\omega$ and is uncountable if $\Sigma$ has at least two symbols.

If $v, w \in \Sigma^*$ with $lh(v) = n$ and $lh(w) = m$, we write $vw$ for the *concatenation* of $v$ and $w$ defined by

$$vw : n+m \longrightarrow \Sigma$$

$$k \longmapsto v(k) \quad \text{if } k < n$$

$$n+l \longmapsto w(l) \quad \text{if } l < m$$

**Def.** If $v \in \Sigma^*$ with $lh(v) = n$, we say that $w$ is a *subword* of $v$, if there are

$$i_0 < i_1 < \dots < i_m < n$$

$$w(k) = v(i_k) \qquad k < m$$

**Remark** Our definition of "word" is general enough to cover, e.g., digital images:

screen resolution $1000 \times 800$

so 800,000 pixels

colour resolution 64,000 ← alphabet

Then if $\Sigma$ is the set of colours, the set of words of length 800,000 corresponds to all possible image files.

# Partial functions

$f : X \longrightarrow Y$ FUNCTION (*)

$\left[\iff \boxed{\forall x \forall y \forall y' \; f(x)=y \wedge f(x)=y' \longrightarrow y=y'} \wedge \forall x \exists y \; f(x)=y\right]$

TOTALITY

Partial functions only satisfy (*), so they are precisely the functions

$g : Z \longrightarrow Y$

where $Z = \mathrm{dom}(g) \subseteq X$.

---

Let's introduce notation for this:

$f : X \dashrightarrow Y$

for "$f$ is a partial function from $X$ to $Y$"

$\iff \exists Z \subseteq X$ s.t. $f : Z \longrightarrow Y$.

---

If you don't like partial functions, think of them as functions

$f : X \longrightarrow Y \cup \{\uparrow\}$ ← new symbol representing "undefined"

where $\hat{f} : x \longmapsto f(x)$ if $x \in \mathrm{dom}(f)$

$x \longmapsto \uparrow$ if $x \notin \mathrm{dom}(f)$

We are going to use the following notation:

if $f : X \dashrightarrow Y$

and $x \in X$, we write

$f(x)\uparrow$ for $x \notin \mathrm{dom}(f)$

"DIVERGES"
"IS UNDEFINED"

$f(x)\downarrow$ for $x \in \mathrm{dom}(f)$

"CONVERGES"

Sometimes $f(x)\downarrow = y$ for "$f(x)$ converges and is equal to $y$".

CONVENTION If $f, g : X \dashrightarrow Y$ and $x \in X$ we write

$$f(x) = g(x)$$

to mean if $f(x)\downarrow$, then $g(x)\downarrow$ and vice versa and $f(x) = g(x)$

and if $f(x)\uparrow$, then $g(x)\uparrow$ and vice versa.

Equivalently, $\hat{f}(x) = \hat{g}(x)$.

## Concatenation

If $f: X \dashrightarrow Y$ and $g: Y \to Z$, we write

$g \circ f : X \dashrightarrow Z$ and we say

if $f(x)\downarrow$, then $g \circ f(x) := \hat{g}(f(x))$

if $f(x)\uparrow$, then $g \circ f(x) \uparrow$.

---

# §2 Models of Computation

**Def.** A **MODEL OF COMPUTATION** is a triple

$$M = (\Sigma, \Phi, h)$$

where

$\Sigma$ is an alphabet

$\Phi$ is a transition function

$h$ is an output function.

$$\Phi : \Sigma^* \times \Sigma^* \longrightarrow \Sigma^*$$

where $lh(\Phi(v,w)) \leq lh(w) + 1$

PROGRAM

INPUT STORAGE

This already incorporates the software principle.

$h$ is a partial function

$$h: \Sigma^* \dashrightarrow \Sigma^*$$

$H := \mathrm{dom}(h)$ ; its elements are called the halting words

and $h(w)$ is a subword of $w$.

In words: The transition function $\overline{\Phi}$ gets a program $P \in \Sigma^*$ and an input $w \in \Sigma^*$. It does one step of the computation. If the new content of the storage $\overline{\Phi}(P,w) \in H$, it halts and outputs $h(\overline{\Phi}(P,w))$. If $\overline{\Phi}(P,w) \notin H$, it continues.

Recursive definition:

$$w_0 \rightsquigarrow C_M(P,w,0) := w$$

$$w_{n+1} \rightsquigarrow C_M(P,w,n+1) := \overline{\Phi}(P, C(P,w,n))$$

Called the M-computation with program $P$ and input $w$.

**Def.** The $\mathbb{M}$-computation with program $P$ and input $w$ halts if there is some $i \in \mathbb{N}$ s.t.

$$C_{\mathbb{M}}(P,w,i) \in H.$$

In this case, we say that the $\mathbb{M}$-computation with $P/w$ outputs $v$ if for the least $i$ s.t.

$$C_{\mathbb{M}}(P,w,i) \in H$$

we have

$$v = h\left(C_{\mathbb{M}}(P,w,i)\right).$$

Let's write $O_{\mathbb{M}}(P,w)$ for this.

The computation gives us a partial function for each program $P$:

$$f_P : \Sigma^* \dashrightarrow \Sigma^*$$

$$f_P(w) := \begin{cases} O_{\mathbb{M}}(P,w) & \text{if the comp. halts} \\ \uparrow & \text{o/w.} \end{cases}$$

NEXT LECTURE : Friday 22 October 4 pm