# XVI

Sixteenth lecture 21 November 2024
Automata & Formal Languages

## Where are we?

### Chapter 4. COMPUTABILITY

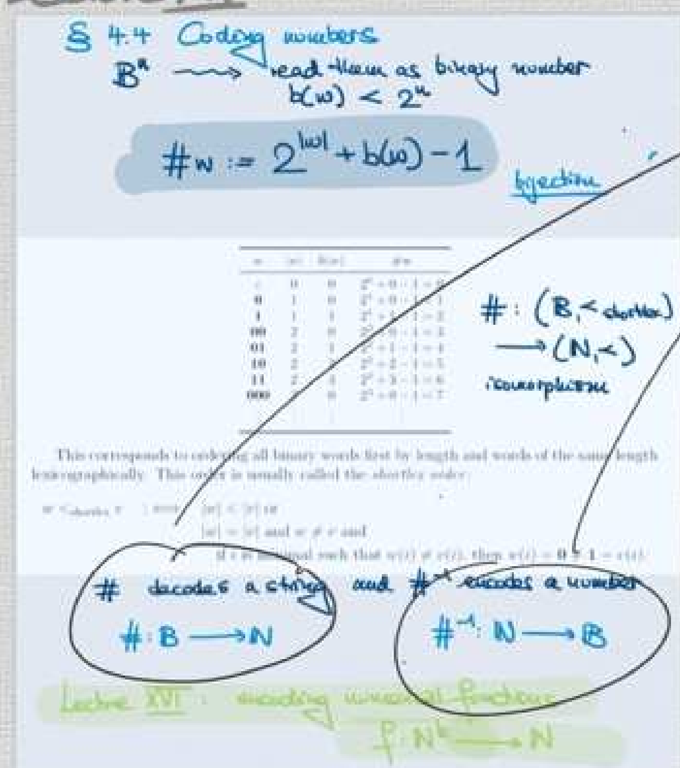
§ 4.1 Register Machines

§ 4.2 What can RMs do?
PERFORM OPERATIONS
ANSWER QUESTIONS

§ 4.3 Computable (partial) functions
Computable & c.e. sets
EXAMPLES Constant functions & projections

Currently § 4.4 CODING NUMBERS

Lecture XV



§ 4.4 Coding numbers

$\mathbb{B}^n \rightsquigarrow$ read them as binary number $b(w) < 2^n$

$$\#w := 2^{|w|} + b(w) - 1$$

bijection

| $w$ | $\lvert w\rvert$ | $b(w)$ | $\#w$ |
|---|---|---|---|
| $\varepsilon$ | 0 | 0 | $2^0+0-1=0$ |
| 0 | 1 | 0 | $2^1+0-1=1$ |
| 1 | 1 | 1 | $2^1+1-1=2$ |
| 00 | 2 | 0 | $2^2+0-1=3$ |
| 01 | 2 | 1 | $2^2+1-1=4$ |
| 10 | 2 | 2 | $2^2+2-1=5$ |
| 11 | 2 | 3 | $2^2+3-1=6$ |
| 000 | 3 | 0 | $2^3+0-1=7$ |

$\# : (\mathbb{B}, <_{\text{shortlex}}) \longrightarrow (\mathbb{N}, <)$ isomorphism

This corresponds to ordering all binary words first by length and words of the same length lexicographically. This order is usually called the shortlex order:

$w <_{\text{shortlex}} v :\iff |w| < |v|$ or $|w| = |v|$ and $w \neq v$ and if $i$ is minimal such that $w(i) \neq v(i)$, then $w(i) = \mathbf{0} \neq \mathbf{1} = v(i)$.

$\#$ decodes a string and $\#^{-1}$ encodes a number

$\# : \mathbb{B} \longrightarrow \mathbb{N}$ $\#^{-1} : \mathbb{N} \longrightarrow \mathbb{B}$

Lecture XVI: encoding unusual functions $f : \mathbb{N}^k \longrightarrow \mathbb{N}$

$\# : \mathbb{B} \longrightarrow \mathbb{N}$ DECODES

$\#^{-1} : \mathbb{N} \longrightarrow \mathbb{B}$ ENCODES

Counting

$\varepsilon$, 0, 1, 00, 01, 10,
11, 000, 001, 010,
...

$$\begin{tikzcd} f: \mathbb{N}^k \arrow[r, dashed] & \mathbb{N} \arrow[d, "\#^{-1}"] \\ \mathbb{B}^k \arrow[u, "\#"] & \mathbb{B} \end{tikzcd}$$

* Use that # extends to $\mathbb{B}^k$.

$$f^{\#}(\vec{w}) := \#^{-1}(f(\#(\vec{w})))$$

the encoding of $f$

We say that $f: \mathbb{N}^k \dashrightarrow \mathbb{N}$ is <u>computable</u> if $f^{\#}$ is.

We say that $A \subseteq \mathbb{N}^k$ is <u>computable</u> or <u>c.e.</u> if $\{\vec{w};\ \#(\vec{w}) \in A\}$ is.

<u>Examples</u>

1. Constant fns & projections are computable.
2. $id = \pi_{1,0}$ is computable.
3. Function $n \longmapsto n+1$ is computable. "successor function".

The successor function is computable.

Consider $s : \mathbb{B} \longrightarrow \mathbb{B}$ s.t. $s(w)$ is the $<_{\text{shortlex}}$-immediate successor of $w$.

Take unused register $k$. Empty it.

(Reverse the content of reg. 0.)

Repeat [ Check if final letter of 0 is 1; if so, write 0 in $k$ and remove the 1 from 0.

until reg. 0 is empty; in that case write 0 into reg 0,

or the final letter of reg. 0 is 0; in that case replace that 0 by 1.

After that copy content of $k$ to the end of 0.

Halt.

q.e.d.

# How to use numerical information in register machines?

## Example 1 REFERRING TO PARTICULAR DIGITS

**Applications: numerical information in computation & truncations.** Being able to refer to numbers via their codes allows us to represent several operations that require the use of numerical information.

**Proposition 4.15.** The question "*If $v$ is the register content of register $i$, what is the $\#v$th letter of register $j$?*" can be answered by a register machine.

*Proof.* Take unused registers $k$ and $\ell$ and empty them. Copy the content of register $j$ in reverse order to register $k$ by Example 4.9 (11). Repeat the following subroutine until register $\ell$ contains $v$: Remove one letter from register $k$ and apply the successor function $s$ to register $\ell$. When register $\ell$ contains $v$, answer the question "What is the final letter of register $k$?" by Example 4.9 (6). Q.E.D.

## COUNT-THROUGH ARGUMENT

Basic idea - Empty a register. Do some procedure repeatedly while applying $s$ (successor fn) to that register until the register meets a requirement.

Algorithm for P 4.15

Un used registers k, l. Empty them.

Copy j to k in reverse order.

Repeat [ Remove the last letter of k

Apply s to l

until the content of l is v.

After that answer what the last letter of k is.

q.e.d.

# Example 2 Referring to computation steps

Similarly, referring to numerical information can be used to refer to computation steps asking a given machine to run for a fixed number of steps; we call this *truncation* and it will play a very important role in §4.8. If $M$ is a register machine and $k, n \in \mathbb{N}$, we can define sets $T_{M,k,n} \subseteq \mathbb{B}^k$, $T_{M,k} \subseteq \mathbb{B}^{k+1}$, and $\widehat{T}_{M,k} \subseteq \mathbb{B}^{k+2}$ as follows:

$$
\begin{aligned}
T_{M,k,n} &:= \{\vec{w}\,;\ M \text{ has halted with input } \vec{w} \text{ after at most } n \text{ steps}\},\\
T_{M,k} &:= \{(\vec{w}, u)\,;\ M \text{ has halted with input } \vec{w} \text{ after at most } \#u \text{ steps}\}, \text{ and}\\
\widehat{T}_{M,k} &:= \{(\vec{w}, u, v)\,;\ (\vec{w}, u) \in T \text{ and } v \text{ is the content of register } 0\\
&\qquad \text{at time of halting}\}.
\end{aligned}
$$

**Proposition 4.17.** The sets $T_{M,k,n}$, $T_{M,k}$, and $\widehat{T}_{M,k}$ are computable.

Proof. Let's only do it for $\hat{T}_{M,k}$.
Describe machine that computes $\chi_{\hat{T}_{M,k}}$.
Input: $(\vec{w}, u, v)$.

Take unused reg. $\ell$. Empty it.
Run $M$ on input $\vec{w}$; after each step of the $M$-computation do the following subroutine:

[ Check whether $\ell$ is equal to $u$;
if so, write $\mathbb{1}$ in reg. 0 and halt;
if not, apply $s$ to reg. $\ell$.

If $M$ reaches its halting state at any point, check whether reg. 0 is equal to $v$; if so, delete reg. 0, write $\mathbb{1}$ in reg. 0, halt.
if not; delete reg. 0, write 0 in reg 0, halt. q.e.d.

# § 4.5 Primitive recursive functions

**Kurt Gödel**

Gödel c. 1926

| | |
|---|---|
| **Born** | Kurt Friedrich Gödel<br>April 28, 1906<br>Brünn, Austria-Hungary (now Brno, Czech Republic) |
| **Died** | January 14, 1978 (aged 71)<br>Princeton, New Jersey, U.S. |
| **Citizenship** | Austria<br>Czechoslovakia<br>Germany<br>United States |
| **Alma mater** | University of Vienna (PhD, 1930) |

**Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I**[1]).

Von **Kurt Gödel** in Wien.

1.

Die Entwicklung der Mathematik in der Richtung zu größerer Exaktheit hat bekanntlich dazu geführt, daß weite Gebiete von ihr formalisiert wurden, in der Art, daß das Beweisen nach einigen wenigen mechanischen Regeln vollzogen werden kann. Die umfassendsten derzeit aufgestellten formalen Systeme sind das System der Principia Mathematica (PM)[2]) einerseits, das Zermelo-Fraenkel-

1931

Gödel's Incompleteness Theorem

Wir schalten nun eine Zwischenbetrachtung ein, die mit dem formalen System $P$ vorderhand nichts zu tun hat, und geben zunächst folgende Definition: Eine zahlentheoretische Funktion[25]) $\varphi(x_1, x_2 \ldots x_n)$ heißt rekursiv definiert aus den zahlentheoretischen Funktionen $\psi(x_1, x_2 \ldots x_{n-1})$ und $\mu(x_1, x_2 \ldots x_{n+1})$, wenn für alle $x_2 \ldots x_n$, $k$[26]) folgendes gilt:

$$\begin{aligned} \varphi(0, x_2 \ldots x_n) &= \psi(x_2 \ldots x_n) \\ \varphi(k+1, x_2 \ldots x_n) &= \mu(k, \varphi(k, x_2 \ldots x_n), x_2 \ldots x_n). \end{aligned} \tag{2}$$

Eine zahlentheoretische Funktion $\varphi$ heißt rekursiv, wenn es eine endliche Reihe von zahlentheor. Funktionen $\varphi_1, \varphi_2 \ldots \varphi_n$ gibt, welche mit $\varphi$ endet und die Eigenschaft hat, daß jede Funktion $\varphi_k$ der Reihe entweder aus zwei der vorhergehenden rekursiv definiert ist oder

– p. 179

"recursively defined from"

"recursive"

**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 |

HOWEVER

Alonzo Church later defined a (more important) and slightly larger class he called RECURSIVE

So, nowadays we call Gödel's functions the primitive recursive functions.

Suppose $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$, $g\colon \mathbb{N}^{k+2} \dashrightarrow \mathbb{N}$, and $g_1, ..., g_k\colon \mathbb{N}^\ell \dashrightarrow \mathbb{N}$ are partial numerical functions, then the partial numerical function $c$ defined by

$$c(\vec{n}) := f(g_1(\vec{n}), ..., g_k(\vec{n}))$$

is called the *composition of $f$ with $(g_1, ..., g_k)$* and the partial numerical function $r$ defined by

$$r(\vec{n}, 0) = f(\vec{n}) \text{ and}$$
$$r(\vec{n}, m+1) = g(\vec{n}, m, r(\vec{n}, m))$$

is called the *recursion result of $f$ and $g$.* A class $\mathcal{C}$ of numerical functions is closed under composition or recursion if, whenever $f, g, g_1, ..., g_m$ are in $\mathcal{C}$, then the composition of $f$ with $(g_1, ..., g_m)$ or the recursion result of $f$ and $g$, respectively, are in $\mathcal{C}$. The numerical functions listed in Proposition 4.14, i.e., the identity function, the constant functions, the projection functions, and the successor function are called *basic functions.* Proposition 4.14 proved that they are all computable.

**Definition 4.18.** The class of *primitive recursive functions* is the smallest class of partial functions containing all basic functions that is closed under composition and recursion.

$$r(\vec{u}, 0) = f(\vec{u})$$
$$r(\vec{u}, m+1) = g(\vec{u}, m, r(\vec{u}, m))$$

We can do proof by induction!

Induction on the length of the shortest "primitive recursive derivation".

p 57 of the typed notes

Note that we can prove statements about the class of primitive recursive functions by *induction* due to its definition as the "smallest class closed under operations": a sequence $(f_0, ..., f_n)$ of numerical functions is called a *primitive recursive derivation* if for every $i \leq n$

(i) either $f_i$ is a basic function,

(ii) or there are $j, j_1, ..., j_k < i$ such that $f_i$ is the composition of $f_j$ with $(f_{j_1}, ..., f_{j_k})$,

(iii) or there are $j, k < i$ such that $f_i$ is the recursion result of $f_j$ and $f_k$.

Clearly, every function that occurs in a primitive recursive derivation is primitive recursive and the class of such functions contains all basic functions and is closed under composition and recursion. Thus, because the primitive recursive functions are the smallest class with these closure properties, we get that a function is primitive recursive if and only if it occurs in a primitive recursive derivation. Therefore we can prove statements by induction on the length of the shortest primitive recursive derivation or by proving that the three conditions (i) to (iii) preserve the validity of the induction hypothesis.

We could have defined the same on functions $\mathbb{B}^k \dashrightarrow \mathbb{B}^k$ with recursion analogue.

$$h(\vec{w}, \varepsilon) = f(\vec{w})$$
$$h(\vec{w}, s(v)) = g(\vec{w}, v, h(\vec{w}, v))$$

$f$ is prim. rec. as num. function $\Longleftarrow$ $f^\#$ is prim. rec. in that alternative def.

# Examples

**Example 4.20.** We build addition using the basic functions and operations.

(a) The identity function is a basic function.

(b) The function $\pi_{3,2}(n, m, k) = k$ is a basic function.

(c) The function $s(n) = n + 1$ is a basic function.

(d) The function $s \circ \pi_{3,2}$ is a concatenation of the functions in (b) and (c): $s \circ \pi_{3,2}(n, m, k) = k + 1$.

(e) We now apply recursion to the functions in (a) and (d), i.e.,

$$\begin{aligned} +(n, 0) &= \mathrm{id}(n) \\ +(n, m+1) &= s(\pi_{3,2}(n, m, +(n, m))) = +(n, m) + 1 \end{aligned}$$

The addition function $+$ defined by these recursion equations is primitive recursive.

The recursion equations given in (e) are the so-called *Grassmann equations for addition.*

$$+(n, 0) := n$$
$$+(n, m+1) := +(n, m) + 1$$

Grassmann equations for multiplication & exponentiation:

$$\times(n, 0) := 0$$
$$\times(n, m+1) := \times(n, m) + n$$

$$\exp(n, 0) := 1$$
$$\exp(n, m+1) := \exp(n, m) \times n$$