# Lecture 3: Recursive Functions

## Models of Computation

https://clegra.github.io/moc/moc.html

### Clemens Grabmayer

Teaching Mobility Program (PNRR-TNE DESK)
University of Novi Sad
Novi Sad, Serbia

March ., 2026

# Course overview

| | | | | |
|---|---|---|---|---|
| *intro* | *classic models* | | | *additional models* |
| **Introduction to Computability** | **Machine Models** | **Recursive Functions** | **Lambda Calculus** | **Three more Models of Computation** |
| computation and decision problems, from logic to computability, overview of models of computation relevance of MoCs | Post Machines, typical features, Turing's analysis of human computers, Turing machines, basic recursion theory | primitive recursive functions, Gödel–Herbrand recursive functions, partial recursive funct's, partial recursive = = Turing-computable, Church's Thesis | $\lambda$-terms, $\beta$-reduction, $\lambda$-definable functions, partial recursive = $\lambda$-definable = Turing computable | Post's Correspondence Problem, Interaction-Nets, Fractran |
| | *imperative programming* | *algebraic programming* | *functional programming* | |

# Calculable function?

### Questions/Exercises

**2** Let $f : \mathbb{N} \to \mathbb{N}$ defined by

$$n \longmapsto \begin{cases} 0 & \dots n = 0 \ \& \ \text{Goldbach's conjecture is false} \\ 1 & \dots n = 0 \ \& \ \text{Goldbach's conjecture is true} \\ n + 1 & \dots n > 0 \end{cases}$$

Is $f$ calculable?

# Calculable function?

### Questions/Exercises

**2** Let $f : \mathbb{N} \to \mathbb{N}$ defined by

$$n \longmapsto \begin{cases} 0 & \ldots n = 0 \ \& \ \text{Goldbach's conjecture is false} \\ 1 & \ldots n = 0 \ \& \ \text{Goldbach's conjecture is true} \\ n + 1 & \ldots n > 0 \end{cases}$$

Is $f$ calculable?

Answer: Yes, because it is one of two calculable functions.
        (We just do no know which one.)

# Summary

Recursive functions

▶ primitive recursive functions

## Summary

Recursive functions

- ▶ primitive recursive functions
- ▶ Gödel–Herbrand(–Kleene) general recursive functions

# Summary

Recursive functions

- ▶ primitive recursive functions
- ▶ Gödel–Herbrand(–Kleene) general recursive functions
- ▶ partial recursive functions

# Summary

Recursive functions

- ▶ primitive recursive functions
- ▶ Gödel–Herbrand(–Kleene) general recursive functions
- ▶ partial recursive functions
    - ▶ defined with $\mu$-recursion (unbounded minimisation)

# Summary

Recursive functions

- ▶ primitive recursive functions
- ▶ Gödel–Herbrand(–Kleene) general recursive functions
- ▶ partial recursive functions
  - ▶ defined with $\mu$-recursion (unbounded minimisation)
- ▶ Partial recursive functions = Turing computable functions

## Summary

Recursive functions

▶ primitive recursive functions

▶ Gödel–Herbrand(–Kleene) general recursive functions

▶ partial recursive functions

    ▶ defined with $\mu$-recursion (unbounded minimisation)

▶ Partial recursive functions = Turing computable functions

▶ Church's thesis

# Summary

Recursive functions

- ▶ primitive recursive functions
- ▶ Gödel–Herbrand(–Kleene) general recursive functions
- ▶ partial recursive functions
    - ▶ defined with $\mu$-recursion (unbounded minimisation)
- ▶ Partial recursive functions = Turing computable functions
- ▶ Church's thesis
    - ▶ effectively calculable functions $\stackrel{\triangle}{=}$ partial-recursive functions

# Summary

Recursive functions

- ▶ primitive recursive functions
- ▶ Gödel–Herbrand(–Kleene) general recursive functions
- ▶ partial recursive functions
    - ▶ defined with $\mu$-recursion (unbounded minimisation)
- ▶ Partial recursive functions = Turing computable functions
- ▶ Church's thesis
    - ▶ effectively calculable functions $\triangleq$ partial-recursive functions
    - ▶ some debate

# Timeline: From logic to computability

| | |
|---|---|
| 1900 | Hilbert's 23 Problems in mathematics |
| 1910/12/13 | Russell/Whitehead: Principia Mathematica |
| 1928 | Hilbert/Ackermann: formulate completeness/decision problems for the predicate calculus (the latter called 'Entscheidungsproblem') |
| 1929 | Presburger: completeness/decidability of theory of addition on $\mathbb{Z}$ |
| 1930 | Gödel: completeness theorem of predicate calculus |
| 1931 | Gödel: incompleteness theorems for first-order arithmetic |
| 1932 | Church: $\lambda$-calculus |
| 1933/34 | Herbrand/Gödel: general recursive functions |
| 1936 | Church/Kleene: $\lambda$-definable ~ general recursive |
| | Church Thesis: 'effectively calculable' be defined as either |
| | Church shows: the 'Entscheidungsproblem' is unsolvable |
| | Post: machine model; Church's thesis as 'working hypothesis' |
| 1937 | Turing: convincing analysis of a 'human computer' leading to the 'Turing machine' |

# Turing-computable (total) functions

## Definition

A total function $f : \mathbb{N}^k \to \mathbb{N}$ is Turing-computable if there exists a
Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \textit{b}, F \rangle$ and a calculable coding
function $\langle \cdot \rangle : \mathbb{N} \to \Sigma^*$ such that:

  ▶ for all $n_1, \ldots, n_k \in \mathbb{N}$ there exists $q \in F$ such that:

  $$q_0 \langle n_1 \rangle \textit{b} \langle n_2 \rangle \textit{b} \ldots \textit{b} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \ldots, n_k) \rangle$$

## Recursive Functions

Functions defined by recursive equations:

like e.g. functions $+, \cdot, (\cdot)^{\cdot} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $(\cdot)! : \mathbb{N} \to \mathbb{N}$:

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

## Recursive Functions

Functions defined by recursive equations:

like e.g. functions $+, \cdot, (\cdot)^{\cdot} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $(\cdot)! : \mathbb{N} \to \mathbb{N}$:

$$n + 0 = n \qquad\qquad n \cdot 0 = 0$$
$$n + (m + 1) = (n + m) + 1 \qquad n \cdot (m + 1) = n \cdot m + n$$

## Recursive Functions

Functions defined by recursive equations:

like e.g. functions $+, \cdot, (\cdot)^\cdot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $(\cdot)! : \mathbb{N} \to \mathbb{N}$:

$$
\begin{aligned}
n + 0 &= n & n \cdot 0 &= 0 \\
n + (m + 1) &= (n + m) + 1 & n \cdot (m + 1) &= n \cdot m + n \\
n^0 &= 1 \\
n^{m+1} &= n^m \cdot n
\end{aligned}
$$

## Recursive Functions

Functions defined by recursive equations:

like e.g. functions $+, \cdot, (\cdot)\dot{} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $(\cdot)! : \mathbb{N} \to \mathbb{N}$:

$$n + 0 = n \qquad\qquad n \cdot 0 = 0$$
$$n + (m + 1) = (n + m) + 1 \qquad n \cdot (m + 1) = n \cdot m + n$$
$$n^0 = 1 \qquad\qquad 0! = 1$$
$$n^{m+1} = n^m \cdot n \qquad\qquad (n + 1)! = (n + 1) \cdot n!$$

## Recursive Functions

Functions defined by recursive equations:

like e.g. functions $+, \cdot, (\cdot)^{\cdot} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $(\cdot)! : \mathbb{N} \to \mathbb{N}$:

$$n + 0 = n \qquad\qquad\qquad n \cdot 0 = 0$$
$$n + (m + 1) = (n + m) + 1 \qquad n \cdot (m + 1) = n \cdot m + n$$
$$n^0 = 1 \qquad\qquad\qquad 0! = 1$$
$$n^{m+1} = n^m \cdot n \qquad\qquad (n + 1)! = (n + 1) \cdot n!$$

Primitive recursive functions: defined by such equations (termination of the evaluation process guaranteed)

## Recursive Functions

Functions defined by recursive equations:

like e.g. functions $+, \cdot, (\cdot)^{\cdot} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $(\cdot)! : \mathbb{N} \to \mathbb{N}$:

$$n + 0 = n \qquad\qquad n \cdot 0 = 0$$
$$n + (m + 1) = (n + m) + 1 \qquad n \cdot (m + 1) = n \cdot m + n$$
$$n^0 = 1 \qquad\qquad 0! = 1$$
$$n^{m+1} = n^m \cdot n \qquad\qquad (n + 1)! = (n + 1) \cdot n!$$

Primitive recursive functions: defined by such equations (termination of the evaluation process guaranteed)

General recursive functions: defined by more general systems of equations

# Recursive Functions

Functions defined by recursive equations:

like e.g. functions $+, \cdot, (\cdot)^{\cdot} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $(\cdot)! : \mathbb{N} \to \mathbb{N}$:

$$n + 0 = n \qquad\qquad n \cdot 0 = 0$$
$$n + (m + 1) = (n + m) + 1 \qquad n \cdot (m + 1) = n \cdot m + n$$
$$n^0 = 1 \qquad\qquad 0! = 1$$
$$n^{m+1} = n^m \cdot n \qquad\qquad (n + 1)! = (n + 1) \cdot n!$$

Primitive recursive functions: defined by such equations (termination of the evaluation process guaranteed)

General recursive functions: defined by more general systems of equations

$\mu$-Recursive (partial recursive) functions:
  extend the primitive recursive functions by a $\mu$-operator
    that allows to obtain partial functions

# Rósza Péter



Rósza Péter (1905–1977)

# Primitive recursive functions ($\mathbb{N}^k \to \mathbb{N}$)

Base functions:

- $\mathcal{O} : \mathbb{N}^0 = \{\varnothing\} \to \mathbb{N}$, $\varnothing \mapsto 0$ (0-ary constant-0 function)
- $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$, $x \mapsto x + 1$ (successor function)
- $\pi_i^n : \mathbb{N}^n \to \mathbb{N}$, $\vec{x} = \langle x_1, \ldots, x_n \rangle \mapsto x_i$ (projection function)

# Primitive recursive functions ($\mathbb{N}^k \to \mathbb{N}$)

Base functions:

- $\mathcal{O} : \mathbb{N}^0 = \{\varnothing\} \to \mathbb{N}$, $\varnothing \mapsto 0$ (0-ary constant-0 function)
- $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$, $x \mapsto x + 1$ (successor function)
- $\pi_i^n : \mathbb{N}^n \to \mathbb{N}$, $\vec{x} = \langle x_1, \ldots, x_n \rangle \mapsto x_i$ (projection function)

Closed under operations:

- composition: if $f : \mathbb{N}^k \to \mathbb{N}$, and $g_i : \mathbb{N}^n \to \mathbb{N}$ are prim. rec., then so is $h = f \circ (g_1 \times \ldots \times g_k) : \mathbb{N}^n \to \mathbb{N}$ :
$$h(\vec{x}) = f(g_1(\vec{x}), \ldots, g_k(\vec{x}))$$

# Primitive recursive functions ($\mathbb{N}^k \to \mathbb{N}$)

Base functions:

- $\mathcal{O} : \mathbb{N}^0 = \{\varnothing\} \to \mathbb{N}$, $\varnothing \mapsto 0$ (0-ary constant-0 function)
- $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$, $x \mapsto x + 1$ (successor function)
- $\pi_i^n : \mathbb{N}^n \to \mathbb{N}$, $\vec{x} = \langle x_1, \ldots, x_n \rangle \mapsto x_i$ (projection function)

Closed under operations:

- composition: if $f : \mathbb{N}^k \to \mathbb{N}$, and $g_i : \mathbb{N}^n \to \mathbb{N}$ are prim. rec., then so is $h = f \circ (g_1 \times \ldots \times g_k) : \mathbb{N}^n \to \mathbb{N}$:
$$h(\vec{x}) = f(g_1(\vec{x}), \ldots, g_k(\vec{x}))$$

- primitive recursion: if $f : \mathbb{N}^n \to \mathbb{N}$, $g : \mathbb{N}^{n+2} \to \mathbb{N}$ are prim. rec., then so is $h = \mathrm{pr}(f; g) : \mathbb{N}^{n+1} \to \mathbb{N}$:

$$h(\vec{x}, 0) = f(\vec{x})$$
$$h(\vec{x}, y + 1) = g(\vec{x}, h(\vec{x}, y), y)$$

# Primitive recursive functions ($\mathbb{N}^n \to \mathbb{N}^l$)

Base functions:

- $\mathcal{O} : \mathbb{N}^0 = \{\varnothing\} \to \mathbb{N}$, $\varnothing \mapsto 0$ (0-ary constant-0 function)
- $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$, $x \mapsto x + 1$ (successor function)
- $\pi_i^n : \mathbb{N}^n \to \mathbb{N}$, $\vec{x} = \langle x_1, \ldots, x_n \rangle \mapsto x_i$ (projection function)
- for $n > 1$: $\mathrm{id}^n : \mathbb{N}^n \to \mathbb{N}^n$, $\vec{x} = \langle x_1, \ldots, x_n \rangle \mapsto \vec{x}$ ($n$-ary identity f.)

Closed under operations:

- composition: if $f : \mathbb{N}^{km} \to \mathbb{N}^l$, and $g_i : \mathbb{N}^n \to \mathbb{N}^m$ are prim. rec., then so is $h = f \circ (g_1 \times \ldots \times g_k) : \mathbb{N}^n \to \mathbb{N}^l$ :
$$h(\vec{x}) = f(g_1(\vec{x}), \ldots, g_k(\vec{x}))$$

- primitive recursion: if $f : \mathbb{N}^n \to \mathbb{N}^l$, $g : \mathbb{N}^{n+l+1} \to \mathbb{N}^l$ are prim. rec., then so is $h = \mathrm{pr}(f; g) : \mathbb{N}^{n+1} \to \mathbb{N}^l$ :

$$h(\vec{x}, 0) = f(\vec{x})$$
$$h(\vec{x}, y + 1) = g(\vec{x}, h(\vec{x}, y), y)$$

# Primitive recursive functions (exercises)

## Exercise

Show that the following functions are primitive recursive:

- ▶ addition
- ▶ constant functions
- ▶ multiplication
- ▶ (positive) sign-function
- ▶ the representing functions $\chi_=$ and $\chi_<$ for the predicates = and <.

## Try-yourself-Examples

Show that the following functions are primitive recursive:

- ▶ exponentiation
- ▶ factorial

# Admissible operations for primitive recursive functions

### Proposition

**1** *definition by case distinction:*

$$f(\vec{x}) := \begin{cases} f_1(\vec{x}) & \dots P_1(\vec{x}) \\ f_2(\vec{x}) & \dots P_2(\vec{x}) \land \neg P_1(\vec{x}) \\ \dots \\ f_k(\vec{x}) & \dots P_k(\vec{x}) \land \neg P_{k-1}(\vec{x}) \land \dots \land \neg P_1(\vec{x}) \\ f_{k+1}(\vec{x}) & \dots \neg P_k(\vec{x}) \land \dots \land \neg P_1(\vec{x}) \end{cases}$$

**2** *definition by bounded recursion:*

$$\mu z_{\leq y}. \left[ P(x_1, \dots, x_n, z) \right] :=$$

$$\begin{cases} z & \dots \neg P(x_1, \dots, x_n, i) \text{ for } 0 \leq i < z \leq y, \\ & \text{and } P(x_1, \dots, x_n, z) \\ y+1 & \dots \neg \exists z. \left( 0 \leq z \leq y \land P(x_1, \dots, x_n, z) \right) \end{cases}$$

# Properties of primitive recursive functions

### Proposition

1. *Every primitive recursive function is total.*
2. *Every primitive recursive function is Turing-computable.*

# Properties of primitive recursive functions

### Proposition

1. *Every primitive recursive function is total.*
2. *Every primitive recursive function is Turing-computable.*

### Proof.

For (2):

- ▶ the base functions are Turing-computable
- ▶ the Turing-computible functions are closed under the schemes composition and primitive recursion

□

# Turing-computable (total) functions

## Definition

A total function $f : \mathbb{N}^k \to \mathbb{N}$ is Turing-computable if there exists a Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not{b}, F \rangle$ and a calculable coding function $\langle \cdot \rangle : \mathbb{N} \to \Sigma^*$ such that:

- for all $n_1, \ldots, n_k \in \mathbb{N}$ there exists $q \in F$ such that:

$$q_0 \langle n_1 \rangle \not{b} \langle n_2 \rangle \not{b} \ldots \not{b} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \ldots, n_k) \rangle$$

# Features of computationally complete MoC's present?

- ▶ storage (unbounded)

- ▶ control (finite, given)

- ▶ modification
    - ▶ of (immediately accessible) stored data
    - ▶ of control state

- ▶ conditionals

- ▶ loop (unbounded)

- ▶ stopping condition

# Features of computationally complete MoC's present?

- ▶ storage (unbounded) ✓

- ▶ control (finite, given) ✓

- ▶ modification ✓
  - ▸ of (immediately accessible) stored data
  - ▸ of control state

- ▶ conditionals ✓

- ▶ loop ✓ (unbounded)

- ▶ stopping condition ✓

# Features of computationally complete MoC's present?

- ▶ storage (unbounded) ✓

- ▶ control (finite, given) ✓

- ▶ modification ✓
  - ▶ of (immediately accessible) stored data
  - ▶ of control state

- ▶ conditionals ✓

- ▶ loop ✓ (unbounded) ✗

- ▶ stopping condition ✓

# Not primitive recursive (I)

## Proposition

*There exist calculable/Turing-computable functions*
*that are <span style="color:red">not primitive recursive</span>.*

## Proof.

By diagonalisation. ☐

# Not primitive recursive (II): Ackermann function



Wilhelm Ackermann (1896–1962)

# Not primitive recursive (II): Ackermann function

Ackermann function $A : \mathbb{N}^2 \to \mathbb{N}$ (simplified version by Rósza Péter):

$$A(0, x) = \mathsf{Succ}(x)$$

$$A(x + 1, 0) = A(x, \mathsf{Succ}(0))$$

$$A(x + 1, y + 1) = A(x, A(x + 1, y))$$

# Not primitive recursive (II): Ackermann function

Ackermann function $A : \mathbb{N}^2 \to \mathbb{N}$ (simplified version by Rósza Péter):

$$A(0, x) = \text{Succ}(x)$$
$$A(x + 1, 0) = A(x, \text{Succ}(0))$$
$$A(x + 1, y + 1) = A(x, A(x + 1, y))$$

A is not primitive recursive, it grows too fast:

$$A(0, n) = n + 1$$
$$A(1, n) = n + 2$$
$$A(2, n) = 2n + 3$$
$$A(3, n) = 2^{n+3} - 2$$
$$A(4, n) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^{16}}}}}}}_{n} - 3$$

. . .

# Not primitive recursive (II): Ackermann function

Ackermann function $A : \mathbb{N}^2 \to \mathbb{N}$ (simplified version by Rósza Péter):

$$A(0, y) = \mathsf{Succ}(y)$$

$$A(x + 1, 0) = A(x, \mathsf{Succ}(0))$$

$$A(x + 1, y + 1) = A(x, A(x + 1, y))$$

A grows faster than every primitive recursive function:

# Not primitive recursive (II): Ackermann function

Ackermann function $A : \mathbb{N}^2 \to \mathbb{N}$ (simplified version by Rósza Péter):

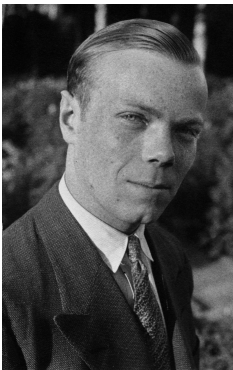$$A(0, y) = \text{Succ}(y)$$
$$A(x + 1, 0) = A(x, \text{Succ}(0))$$
$$A(x + 1, y + 1) = A(x, A(x + 1, y))$$

A grows faster than every primitive recursive function:

**Theorem**

*For every primitive recursive $f : \mathbb{N} \to \mathbb{N}$ there exists some $i \in \mathbb{N}$ such that $f(i) < A(i, i)$.*

# Jacques Herbrand



Jacques Herbrand (1908–1931)

# Kurt Gödel



Kurt Gödel (1906–1978)

## Gödel–Herbrand general recursive function

Defined by systems of recursion equations like that for the Ackermann function:

$$A(0, y) = \mathsf{Succ}(y)$$
$$A(\mathsf{Succ}(x), 0) = A(x, \mathsf{Succ}(0))$$
$$A(\mathsf{Succ}(x), \mathsf{Succ}(y)) = A(x, A(\mathsf{Succ}(x), y))$$

# Gödel–Herbrand general recursive function

Defined by systems of recursion equations like that for the Ackermann function:

$$A(0, y) = \mathsf{Succ}(y)$$

$$A(\mathsf{Succ}(x), 0) = A(x, \mathsf{Succ}(0))$$

$$A(\mathsf{Succ}(x), \mathsf{Succ}(y)) = A(x, A(\mathsf{Succ}(x), y))$$

Numerals: $\langle 0 \rangle := 0$, and $\langle n \rangle := \underbrace{\mathsf{Succ}(\ldots \mathsf{Succ}(0))}_{n}$ for $n > 1$.

---

**Definition**

A function $f : \mathbb{N}^k \to \mathbb{N}$ is called general recursive if it can be defined by (such a) system $\mathcal{S}$ of recursion equations via a function symbol $F$ if for all $n_1, \ldots, n_k \in \mathbb{N}$, the expression $F(\langle n_1 \rangle, \ldots, \langle n_k \rangle)$ evaluates according to $\mathcal{S}$ to a unique numeral $\langle n \rangle$, and such that furthermore: $n = f(n_1, \ldots, n_k)$.

---

# Stephen Cole Kleene
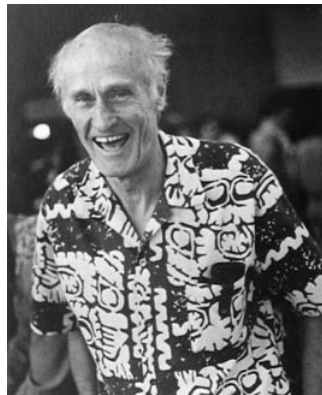


Stephen Cole Kleene (1906–1994)

# Unbounded minimisation ($\mu$-recursion)

Let $f : \mathbb{N}^{k+1} \to \mathbb{N}$ total. Then the partial function defined by:

$$\mu(f) : \mathbb{N}^k \rightharpoonup \mathbb{N}$$

$$\vec{x} \mapsto \begin{cases} \min\{y \in \mathbb{N} \mid f(\vec{x}, y) = 0\} & \ldots \exists y\,(f(\vec{x}, y) = 0) \\ \uparrow & \ldots \text{else} \end{cases}$$

is called the unbounded minimisation of $f$.

# Unbounded minimisation ($\mu$-recursion)

Let $f : \mathbb{N}^{k+1} \to \mathbb{N}$ total. Then the partial function defined by:

$$\mu(f) : \mathbb{N}^k \rightharpoonup \mathbb{N}$$

$$\vec{x} \mapsto \begin{cases} \min\{y \in \mathbb{N} \mid f(\vec{x}, y) = 0\} & \dots \exists y \, (f(\vec{x}, y) = 0) \\ \uparrow & \dots \text{else} \end{cases}$$

is called the unbounded minimisation of $f$.

Let $f : \mathbb{N}^{k+1} \rightharpoonup \mathbb{N}$ partial. Then the partial function $\mu(f)$:

$$\mu(f) : \mathbb{N}^k \rightharpoonup \mathbb{N}$$

$$\vec{x} \mapsto \begin{cases} z & \dots f(\vec{x}, z) = 0 \, \wedge \, \forall y \, \big(0 \leq y < z \to (f(\vec{x}, y) \!\downarrow \neq 0)\big) \\ \uparrow & \dots \neg \exists y \, \big(f(\vec{x}, y) = 0 \, \wedge \, \forall z \, (0 \leq z < y \to (f(\vec{x}, z) \!\downarrow)\big) \end{cases}$$

is called the unbounded minimisation of $f$.

# Partial, and total, recursive functions

### Definition

A partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}^l$ is called partial recursive if it can be specified from base functions ($\mathcal{O}$, $\mathrm{succ}$, $\pi_i^n$, and $\mathrm{id}^n$) by successive applications of composition, primitive recursion, and unbounded minimisation.

A partial recursive function is called (total) recursive if it is total.

# Partial, and total, recursive functions

### Definition

A partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}^l$ is called partial recursive if it can be specified from base functions ($\mathcal{O}$, $\mathrm{succ}$, $\pi_i^n$, and $\mathrm{id}^n$) by successive applications of composition, primitive recursion, and unbounded minimisation.

A partial recursive function is called (total) recursive if it is total.

### Proposition

*Every partial recursive function is Turing-computable.*

# Primitive recursive

- storage (unbounded) ✓

- control (finite, given) ✓

- modification ✓
  - of (immediately accessible) stored data
  - of control state

- conditionals ✓

- loop ✓ (unbounded) ✗

- stopping condition ✓

# Partial recursive = prim. rec. + unbounded minimization

- ▶ storage (unbounded) $\checkmark$

- ▶ control (finite, given) $\checkmark$

- ▶ modification $\checkmark$
    - of (immediately accessible) stored data
    - of control state

- ▶ conditionals $\checkmark$

- ▶ loop $\checkmark$ (unbounded) $\checkmark$

- ▶ stopping condition $\checkmark$

# Turing-computable functions

## Definition

**1** A total function $f : \mathbb{N}^k \to \mathbb{N}$ is Turing-computable if there exists a Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \mathfrak{b}, F \rangle$ and a calculable coding function $\langle \cdot \rangle : \mathbb{N} \to \Sigma^*$ such that:

- for all $n_1, \ldots, n_k \in \mathbb{N}$ there exists $q \in F$ such that:

$$q_0 \langle n_1 \rangle \mathfrak{b} \langle n_2 \rangle \mathfrak{b} \ldots \mathfrak{b} \langle n_k \rangle \vdash^*_M q \langle f(n_1, \ldots, n_k) \rangle$$

# Turing-computable functions

### Definition

2. A partial function $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$ is Turing-computable if there exists a Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, b\!\!\!/, F \rangle$ and a calculable coding function $\langle \cdot \rangle : \mathbb{N} \to \Sigma^*$ such that:

- for all $n_1, \ldots, n_k \in \mathbb{N}$:

  $$M \text{ accepts } \langle n_1 \rangle b\!\!\!/ \langle n_2 \rangle b\!\!\!/ \ldots b\!\!\!/ \langle n_k \rangle \iff f(n_1, \ldots, n_k)\!\downarrow$$

- for all $n_1, \ldots, n_k \in \mathbb{N}$ there exists $q \in F$ such that:

  $$f(n_1, \ldots, n_k)\!\downarrow \implies q_0 \langle n_1 \rangle b\!\!\!/ \langle n_2 \rangle b\!\!\!/ \ldots b\!\!\!/ \langle n_k \rangle \vdash_M^* q \langle f(n_1, \ldots, n_k) \rangle$$

# Turing-computable functions

## Definition

**1** A total function $f : \mathbb{N}^k \to \mathbb{N}$ is Turing-computable if there exists a Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not{b}, F \rangle$ and a calculable coding function $\langle \cdot \rangle : \mathbb{N} \to \Sigma^*$ such that:

- for all $n_1, \ldots, n_k \in \mathbb{N}$ there exists $q \in F$ such that:
$$q_0 \langle n_1 \rangle \not{b} \langle n_2 \rangle \not{b} \ldots \not{b} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \ldots, n_k) \rangle$$

**2** A partial function $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$ is Turing-computable if there exists a Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not{b}, F \rangle$ and a calculable coding function $\langle \cdot \rangle : \mathbb{N} \to \Sigma^*$ such that:

- for all $n_1, \ldots, n_k \in \mathbb{N}$:
$$M \text{ accepts } \langle n_1 \rangle \not{b} \langle n_2 \rangle \not{b} \ldots \not{b} \langle n_k \rangle \iff f(n_1, \ldots, n_k) \downarrow$$

- for all $n_1, \ldots, n_k \in \mathbb{N}$ there exists $q \in F$ such that:
$$f(n_1, \ldots, n_k) \downarrow \implies q_0 \langle n_1 \rangle \not{b} \langle n_2 \rangle \not{b} \ldots \not{b} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \ldots, n_k) \rangle$$

# Partial recursive vs. Turing-computable functions

### Lemma

*Every Turing-computable function is partial recursive.*

# Partial recursive vs. Turing-computable functions

### Lemma

*Every Turing-computable function is partial recursive.*

Proof by arithmetization of Turing machines, showing:

### Theorem (Kleene's normal form theorem)

*For every Turing-computable, partial function (and hence for every partial recursive function) $h : \mathbb{N}^k \to \mathbb{N}$ there exist primitive recursive functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N}^{k+1} \to \mathbb{N}$ such that:*

$$h(x_1, \ldots, x_n) = (f \circ \mu(g))(x_1, \ldots, x_n)$$

# Partial recursive vs. Turing-computable functions

## Lemma

*Every Turing-computable function is partial recursive.*

Proof by arithmetization of Turing machines, showing:
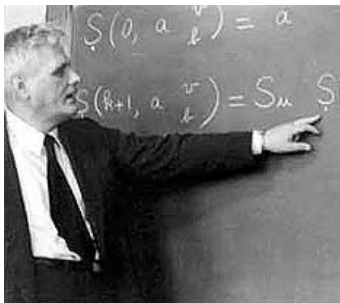
## Theorem (Kleene's normal form theorem)

*For every Turing-computable, partial function (and hence for every partial recursive function) $h : \mathbb{N}^k \to \mathbb{N}$ there exist primitive recursive functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N}^{k+1} \to \mathbb{N}$ such that:*

$$h(x_1, \ldots, x_n) = (f \circ \mu(g))(x_1, \ldots, x_n)$$

## Theorem

*The Turing-computable (partial) functions coincide with the partial recursive functions.*

# Alonzo Church



Alonzo Church (1903 –1995)

# Effectively calculable functions

Alonzo Church (1936):

> *"We now define the notion [. . . ] of an effectively calculable function of positive integers by identifying it with the notion of a recursive function of positive integers (or a $\lambda$-definable function of positive integers). This definition is thought to be justified by the considerations which follow, so far as positive justification can ever be obtained for the selection of formal definition to correspond to an intuitive notion."*

# Effectively calculable functions

Alonzo Church (1936):

> *"We now define the notion [. . . ] of an effectively calculable function of positive integers by identifying it with the notion of a recursive function of positive integers (or a $\lambda$-definable function of positive integers). This definition is thought to be justified by the considerations which follow, so far as positive justification can ever be obtained for the selection of formal definition to correspond to an intuitive notion."*

### Definition (Church)

For every total function $f : \mathbb{N} \to \mathbb{N}$, and partial function $g : \mathbb{N} \rightharpoonup \mathbb{N}$,

$$f \text{ is effectively calculable } : \iff f \text{ is recursive}$$
$$g \text{ is effectively calculable } : \iff g \text{ is partial-recursive}$$

# Church's Thesis

Alonzo Church (1936):

> *"We now define the notion [. . . ] of an effectively calculable function of positive integers by identifying it with the notion of a recursive function of positive integers (or a $\lambda$-definable function of positive integers). This definition is thought to be justified by the considerations which follow, so far as positive justification can ever be obtained for the selection of formal definition to correspond to an intuitive notion."*
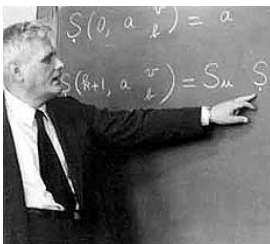
### Definition (Church)

For every total function $f : \mathbb{N} \to \mathbb{N}$, and partial function $g : \mathbb{N} \rightharpoonup \mathbb{N}$,

$$f \text{ is effectively calculable} \ : \iff \ f \text{ is recursive}$$
$$g \text{ is effectively calculable} \ : \iff \ g \text{ is partial-recursive}$$

# $\lambda$-calculus



Alonzo Church (1903 –1992)

### Theorem (Kleene/Church, 1935)

*Every $\lambda$-definable function is general recursive, and vice versa.*

# Recommended reading

1. Recursive and primitive-recursive functions:

   Chapter 3, The Lambda Calculus of the book:

   ▸ Maribel Fernández [2]: *Models of Computation (An Introduction to Computability Theory)*, Springer-Verlag London, 2009.

# Post's 'working hypothesis'

E.L. Post in his 1936 article (Post machines):

*"The writer expects the present formulation to turn out to be logically equivalent to recursiveness in the sense of the Gödel–Church development. Its purpose, however, is not only to present a system of a certain logical potency but also, in its restricted field, of psychological fidelity. In the latter sense wider and wider formulations are contemplated. On the other hand, our aim will be to show that all such are logically reducible to formulation 1 [Post machines]. We offer this conclusion at the present moment as a working hypothesis. And to our mind such is Church's identification of effective calculability with recursiveness."*

# Church on Post's 'working hypothesis'

Alonzo Church in his review (1937) of Post's 1936 article:

> "The author proposes a definition of "finite 1-process" which is similar in formulation, and in fact equivalent, to computation by a Turing machine (see the preceding review). He does not, however, regard his formulation as certainly to be identified with effectiveness in the ordinary sense, but takes this identification as a "working hypothesis" in need of continual verification. To this the reviewer would object that effectiveness in the ordinary sense has not been given an exact definition, and hence the working hypothesis in question has not an exact meaning. To define effectiveness as computability by an arbitrary machine, subject to restrictions of finiteness, would seem to be an adequate representation of the ordinary notion, and if this is done the need for a working hypothesis disappears."

## Church on Turing's paper

A. Church in his review (1937) of Turing's 1936 article:

> *"The author proposes as a criterion that an infinite sequence of digits 0 and 1 be "computable" that it shall be possible to devise a computing machine, occupying a finite space and with working parts of finite size, which will write down the sequence to any desired number of terms if allowed to run for a sufficiently long time. As a matter of convenience, certain further restrictions are imposed on the character of the machine, but these are of such a nature as obviously to cause no loss of generality—in particular, a human calculator, provided with pencil and paper and explicit instructions, can be regarded as a kind of Turing machine. It is thus immediately clear that computability, so defined, can be identified with (especially, is no less general than) the notion of effectiveness as it appears in certain mathematical problems [. . . ].*

# Summary

Recursive functions

- ▶ primitive recursive functions
- ▶ Gödel–Herbrand(–Kleene) general recursive functions
- ▶ partial recursive functions
    - ▶ defined with $\mu$-recursion (unbounded minimisation)
- ▶ Partial recursive functions = Turing computable functions
- ▶ Church's thesis
    - ▶ effectively calculable functions $\stackrel{\triangle}{=}$ partial-recursive functions
    - ▶ some debate

# Course overview

| intro | classic models | | | additional models |
|---|---|---|---|---|
| **Introduction to Computability** | **Machine Models** | **Recursive Functions** | **Lambda Calculus** | **Three more Models of Computation** |
| computation and decision problems, from logic to computability, overview of models of computation relevance of MoCs | Post Machines, typical features, Turing's analysis of human computers, Turing machines, basic recursion theory | primitive recursive functions, Gödel–Herbrand recursive functions, partial recursive funct's, partial recursive = = Turing-computable, Church's Thesis | $\lambda$-terms, $\beta$-reduction, $\lambda$-definable functions, partial recursive = $\lambda$-definable = Turing computable | Post's Correspondence Problem, Interaction-Nets, Fractran |
| | *imperative programming* | *algebraic programming* | *functional programming* | |

# References I

📄 Alonzo Church.
An Unsolvable Problem of Elementary Number Theory.
*American Journal of Mathematics*, 58(2):345–363, April 1936.

📄 Maribel Fernández.
*Models of Computation (An Introduction to Computability Theory)*.
Springer, Dordrecht Heidelberg London New York, 2009.

📄 Emil Leon Post.
Finite Combinatory Processes – Formulation 1.
*Journal of Symbolic Logic*, 1(3):103–105, 1936.
https://www.wolframscience.com/prizes/tm23/images/Post.pdf.

# References II

📄 Alan M. Turing.
On Computable Numbers, with an Application to the
Entscheidungsproblem.
*Proceedings of the London Mathematical Society*,
42(2):230–265, 1936.
http://www.wolframscience.com/prizes/tm23/
images/Turing.pdf.