

Lecture 3: Recursive Functions

Models of Computation

Clemens Grabmayer

Ph.D. Program, Advanced Courses Period

Gran Sasso Science Institute

L'Aquila, Italy

July 9, 2025

Course overview

Monday, July 7 10.30 – 12.30	Tuesday, July 8 10.30 – 12.30	Wednesday, July 9 10.30 – 12.30	Thursday, July 10 10.30 – 12.30	Friday, July 11
<i>intro</i>	<i>classic models</i>			<i>additional models</i>
Introduction to Computability	Machine Models	Recursive Functions	Lambda Calculus	
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	λ -terms, β -reduction, λ -definable functions, partial recursive = λ -definable = Turing computable	
	<i>imperative programming</i>	<i>algebraic programming</i>	<i>functional programming</i>	
				14.30 – 16.30
				Three more Models of Computation
				Post's Correspondence Problem, Interaction-Nets, Fractran
				comparing computational power

Today

Recursive functions

- ▶ primitive recursive functions

Today

Recursive functions

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

Today

Recursive functions

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

Today

Recursive functions

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

Today

Recursive functions

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

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

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

$$q_0 \langle n_1 \rangle \text{\textit{h}} \langle n_2 \rangle \text{\textit{h}} \dots \text{\textit{h}} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \dots, n_k) \rangle$$

Recursive Functions

Functions defined by recursive equations:

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

$$n + 0 = n$$

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

Recursive Functions

Functions defined by recursive equations:

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

$$n + 0 = n$$

$$n \cdot 0 = 0$$

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

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

Recursive Functions

Functions defined by recursive equations:

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

$$n + 0 = n$$

$$n \cdot 0 = 0$$

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

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

$$n^0 = 1$$

$$n^{m+1} = n^m \cdot n$$

Recursive Functions

Functions defined by recursive equations:

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

$$n + 0 = n$$

$$n \cdot 0 = 0$$

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

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

$$n^0 = 1$$

$$0! = 1$$

$$n^{m+1} = n^m \cdot n$$

$$(n + 1)! = (n + 1) \cdot n!$$

Recursive Functions

Functions defined by recursive equations:

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

$$n + 0 = n$$

$$n \cdot 0 = 0$$

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

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

$$n^0 = 1$$

$$0! = 1$$

$$n^{m+1} = n^m \cdot n$$

$$(n + 1)! = (n + 1) \cdot n!$$

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

Recursive Functions

Functions defined by recursive equations:

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

$$n + 0 = n$$

$$n \cdot 0 = 0$$

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

$$n^0 = 1$$

$$0! = 1$$

$$n^{m+1} = n^m \cdot n$$

$$(n + 1)! = (n + 1) \cdot n!$$

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 $+, \cdot, (\cdot)' : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, and $(\cdot)! : \mathbb{N} \rightarrow \mathbb{N}$:

$$n + 0 = n$$

$$n \cdot 0 = 0$$

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

$$n^0 = 1$$

$$0! = 1$$

$$n^{m+1} = n^m \cdot n$$

$$(n + 1)! = (n + 1) \cdot n!$$

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

General recursive functions: defined by more general systems of equations

μ -Recursive (partial recursive) functions: extend the primitive recursive functions by a μ -operator that allows to construct partial functions

Rósa Péter



Rósa Péter (1905–1977)

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

Base functions:

- ▶ $\mathcal{O} : \mathbb{N}^0 = \{\emptyset\} \rightarrow \mathbb{N}, \emptyset \mapsto 0$ (0-ary constant-0 function)
- ▶ $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}, x \mapsto x + 1$ (successor function)
- ▶ $\pi_i^n : \mathbb{N}^n \rightarrow \mathbb{N}, \vec{x} = \langle x_1, \dots, x_n \rangle \mapsto x_i$ (projection function)

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

Base functions:

- ▶ $\mathcal{O} : \mathbb{N}^0 = \{\emptyset\} \rightarrow \mathbb{N}, \emptyset \mapsto 0$ (0-ary constant-0 function)
- ▶ $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}, x \mapsto x + 1$ (successor function)
- ▶ $\pi_i^n : \mathbb{N}^n \rightarrow \mathbb{N}, \vec{x} = \langle x_1, \dots, x_n \rangle \mapsto x_i$ (projection function)

Closed under operations:

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

$$h(\vec{x}) = f(g_1(\vec{x}), \dots, g_k(\vec{x}))$$

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

Base functions:

- ▶ $\mathcal{O} : \mathbb{N}^0 = \{\emptyset\} \rightarrow \mathbb{N}$, $\emptyset \mapsto 0$ (0-ary constant-0 function)
- ▶ $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$, $x \mapsto x + 1$ (successor function)
- ▶ $\pi_i^n : \mathbb{N}^n \rightarrow \mathbb{N}$, $\vec{x} = \langle x_1, \dots, x_n \rangle \mapsto x_i$ (projection function)

Closed under operations:

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

$$h(\vec{x}) = f(g_1(\vec{x}), \dots, g_k(\vec{x}))$$
- ▶ **primitive recursion**: if $f : \mathbb{N}^n \rightarrow \mathbb{N}$, $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ are **prim. rec.**, then so is $h = \text{pr}(f; g) : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$:

$$h(\vec{x}, 0) = f(\vec{x})$$

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

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

Base functions:

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

Closed under operations:

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

$$h(\vec{x}) = f(g_1(\vec{x}), \dots, g_k(\vec{x}))$$
- ▶ **primitive recursion**: if $f : \mathbb{N}^n \rightarrow \mathbb{N}^l$, $g : \mathbb{N}^{n+l+1} \rightarrow \mathbb{N}^l$ are prim. rec., then so is $h = \text{pr}(f; g) : \mathbb{N}^{n+1} \rightarrow \mathbb{N}^l$:

$$h(\vec{x}, 0) = f(\vec{x})$$

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

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*:

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

② definition by *bounded recursion*:

$$\mu_{z \leq y} \cdot [P(x_1, \dots, x_n, z)] := \begin{cases} z & \dots \neg P(x_1, \dots, x_n, i) \text{ for } 0 \leq i < z \leq y, \\ & \text{and } P(x_1, \dots, x_n, z) \\ y + 1 & \dots \neg \exists z. \wedge 0 \leq z \leq y P(x_1, \dots, x_n, z) \end{cases}$$

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-computable functions are closed under the schemes [composition](#) and [primitive recursion](#)



Turing-computable (total) functions

Definition

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

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

$$q_0 \langle n_1 \rangle \text{\textit{h}} \langle n_2 \rangle \text{\textit{h}} \dots \text{\textit{h}} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \dots, n_k) \rangle$$

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 **not primitive recursive**.*

Proof.

By diagonalisation.



Not primitive recursive (II): Ackermann function



Wilhelm Ackermann (1896–1962)

Not primitive recursive (II): Ackermann function

Ackermann function $A : \mathbb{N}^2 \rightarrow \mathbb{N}$ (simplified version by Rózsa 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**:

$$A(0, n) = n + 1$$

$$A(1, n) = n + 2$$

$$A(2, n) = 2n + 3$$

$$A(3, n) = 2^{n+3} - 2$$

$$A(4, n) = \underbrace{2^{2^{\cdot^{\cdot^{2^{16}}}}}}_n - 3$$

...

Not primitive recursive (II): Ackermann function

Ackermann function $A : \mathbb{N}^2 \rightarrow \mathbb{N}$ (simplified version by Rózsa 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 grows faster than every primitive recursive function:

Theorem

For every primitive recursive $f : \mathbb{N} \rightarrow \mathbb{N}$ 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:

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

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

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

Gödel–Herbrand general recursive function

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

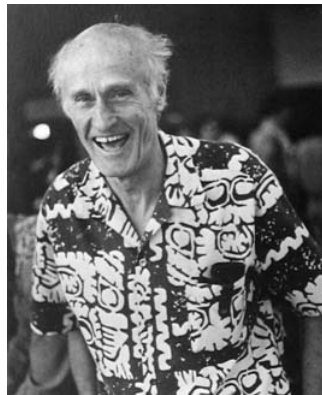
$$\begin{aligned} A(0, x) &= \text{Succ}(x) \\ A(\text{Succ}(x), 0) &= A(x, \text{Succ}(0)) \\ A(\text{Succ}(x), \text{Succ}(y)) &= A(x, A(\text{Succ}(x), y)) \end{aligned}$$

Numerals: $\langle 0 \rangle := 0$, and $\langle n \rangle := \underbrace{\text{Succ}(\dots \text{Succ}(0))}_n$ for $n > 1$.

Definition

A function $f : \mathbb{N}^k \rightarrow \mathbb{N}$ 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 $n_1, \dots, n_k \in \mathbb{N}$, the expression $F(\langle n_1 \rangle, \dots, \langle n_k \rangle)$ evaluates according to $\mathcal{S}$ to a **unique numeral** $\langle n \rangle$, and such that furthermore: $n = f(n_1, \dots, n_k)$.

Stephen Cole Kleene



Stephen Cole Kleene (1906–1994)

Unbounded minimisation (μ -recursion)

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

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

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

is called the **unbounded minimisation** of f .

Unbounded minimisation (μ -recursion)

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

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

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

is called the **unbounded minimisation of f** .

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

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

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

is called the **unbounded minimisation of f** .

Partial, and total, recursive functions

Definition

A **partial function** $f : \mathbb{N}^n \rightarrow \mathbb{N}^l$ is called **partial recursive** if it can be specified from base functions ($\mathcal{O}$, **succ**, π_i^n , and 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** $f : \mathbb{N}^n \rightarrow \mathbb{N}^l$ is called **partial recursive** if it can be specified from base functions ($\mathcal{O}$, **succ**, π_i^n , and **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.

Turing-computable functions

Definition

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

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

$$q_0 \langle n_1 \rangle \text{\textit{h}} \langle n_2 \rangle \text{\textit{h}} \dots \text{\textit{h}} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \dots, n_k) \rangle$$

Turing-computable functions

Definition

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

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

$$M \text{ accepts } \langle n_1 \rangle \perp \langle n_2 \rangle \perp \dots \perp \langle n_k \rangle \iff f(n_1, \dots, n_k) \downarrow$$

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

$$f(n_1, \dots, n_k) \downarrow \implies q_0 \langle n_1 \rangle \perp \langle n_2 \rangle \perp \dots \perp \langle n_k \rangle \vdash_M^* q \langle f(n_1, \dots, n_k) \rangle$$

Turing-computable functions

Definition

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

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

$$q_0 \langle n_1 \rangle \text{halt} \langle n_2 \rangle \text{halt} \dots \text{halt} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \dots, n_k) \rangle$$
- ② A **partial function** $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is **Turing-computable** if there exists a Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \text{halt}, F \rangle$ and a **calculable** coding function $\langle \cdot \rangle : \mathbb{N} \rightarrow \Sigma^*$ such that:

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

$$M \text{ accepts } \langle n_1 \rangle \text{halt} \langle n_2 \rangle \text{halt} \dots \text{halt} \langle n_k \rangle \iff f(n_1, \dots, n_k) \downarrow$$
 - for all $n_1, \dots, n_k \in \mathbb{N}$ there exists $q \in F$ such that:

$$f(n_1, \dots, n_k) \downarrow \implies q_0 \langle n_1 \rangle \text{halt} \langle n_2 \rangle \text{halt} \dots \text{halt} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \dots, n_k) \rangle$$

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) $h : \mathbb{N}^k \rightarrow \mathbb{N}$ there exist primitive recursive functions $f : \mathbb{N} \rightarrow \mathbb{N}$ and $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ such that:

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

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)

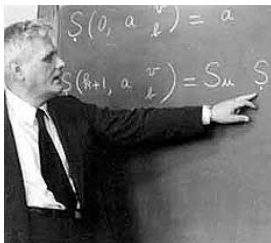
*For every **Turing-computable, partial** function (and hence for every **partial recursive** function) $h : \mathbb{N}^k \rightarrow \mathbb{N}$ there exist **primitive recursive** functions $f : \mathbb{N} \rightarrow \mathbb{N}$ and $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ such that:*

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

Theorem

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

Church's Thesis

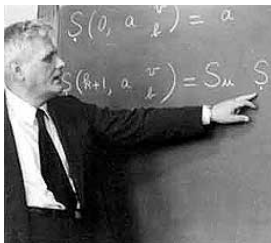


Alonzo Church (1903 –1995)

Thesis (Church, 1936)

Every effectively calculable function is general recursive.

λ -calculus



Alonzo Church (1903 –1992)

Theorem (Kleene/Church, 1935)

Every λ -definable function is general recursive, and vice versa.

Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)

Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)

Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification

Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification
 - ▶ of (immediately accessible) stored data

Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification
 - ▶ of (immediately accessible) stored data
 - ▶ of control state

Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification
 - ▶ of (immediately accessible) stored data
 - ▶ of control state
- ▶ conditionals

Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification
 - ▶ of (immediately accessible) stored data
 - ▶ of control state
- ▶ conditionals
- ▶ loop

Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification
 - ▶ of (immediately accessible) stored data
 - ▶ of control state
- ▶ conditionals
- ▶ loop
- ▶ stopping condition

Summary



▶ A-hierarchy



Recommended reading

1 Recursive and primitive-recursive functions:

Chapter 3, The Lambda Calculus of the book:

- ▶ Maribel Fernández [1]: *Models of Computation (An Introduction to Computability Theory)*, Springer-Verlag London, 2009.

Course overview

Monday, July 7 10.30 – 12.30	Tuesday, July 8 10.30 – 12.30	Wednesday, July 9 10.30 – 12.30	Thursday, July 10 10.30 – 12.30	Friday, July 11
<i>intro</i>	<i>classic models</i>			<i>additional models</i>
Introduction to Computability	Machine Models	Recursive Functions	Lambda Calculus	
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	λ -terms, β -reduction, λ -definable functions, partial recursive = λ -definable = Turing computable	
	<i>imperative programming</i>	<i>algebraic programming</i>	<i>functional programming</i>	
				14.30 – 16.30
				Three more Models of Computation
				Post's Correspondence Problem, Interaction-Nets, Fractran
				comparing computational power

Example suggestions

Examples

- 1.
- 2.
- 3.

References



Maribel Fernández.

Models of Computation (An Introduction to Computability Theory).

Springer, Dordrecht Heidelberg London New York, 2009.