

Lecture 4: Lambda Calculus

Models of Computation

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

Clemens Grabmayer

Ph.D. Program, Advanced Courses Period

Gran Sasso Science Institute

L'Aquila, Italy

July 10, 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

Lambda calculus

- ▶ λ -calculus
 - ▶ syntax
 - ▶ reduction rules

Today

Lambda calculus

- ▶ λ -calculus
 - ▶ syntax
 - ▶ reduction rules
- ▶ λ -definable functions

Today

Lambda calculus

- ▶ λ -calculus
 - ▶ syntax
 - ▶ reduction rules
- ▶ λ -definable functions
- ▶ primitive recursive functions are λ -definable

Today

Lambda calculus

- ▶ λ -calculus
 - ▶ syntax
 - ▶ reduction rules
- ▶ λ -definable functions
- ▶ primitive recursive functions are λ -definable
- ▶ μ -recursive/partial recursive functions are λ -definable

Today

Lambda calculus

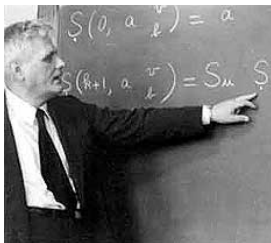
- ▶ λ -calculus
 - ▶ syntax
 - ▶ reduction rules
- ▶ λ -definable functions
- ▶ primitive recursive functions are λ -definable
- ▶ μ -recursive/partial recursive functions are λ -definable
- ▶ λ -definable functions are Turing computable

Today

Lambda calculus

- ▶ λ -calculus
 - ▶ syntax
 - ▶ reduction rules
- ▶ λ -definable functions
- ▶ primitive recursive functions are λ -definable
- ▶ μ -recursive/partial recursive functions are λ -definable
- ▶ λ -definable functions are Turing computable
- ▶ **Hence: λ -definable = partial recursive = Turing-computable**

Church's Thesis



Alonzo Church (1903 –1995)

Thesis (Church, 1936)

- ▶ *Every total effectively calculable function is recursive.*
- ▶ *Every effectively calculable partial function is partial-recursive.*

λ -terms

Definition

- ▶ **variables:** $x, y, z, x_1, y_1, z_1, \dots \in \Lambda$
- ▶ **λ -abstraction:** x a variable, $M \in \Lambda \implies (\lambda x.M \in \Lambda)$
- ▶ **application:** $M, N \in \Lambda \implies (MN) \in \Lambda$

β -reduction

Definition

- **One-step β -reduction** \rightarrow_β is defined as the application of the rule:

$$(\lambda x.M)N \rightarrow_\beta M[x := N]$$

in λ -terms $C[(\lambda x.M)N]$ formed by arbitrary λ -term contexts $C[]$, where $(\lambda x.M)N$ is called a **redex**, and furthermore:

$M[x := N]$:= substitution of N for free occurrences of x in M
(using **α -conversion** to avoid variable capture)

β -reduction

Definition

- **One-step β -reduction** \rightarrow_β is defined as the application of the rule:

$$(\lambda x.M)N \rightarrow_\beta M[x := N]$$

in λ -terms $C[(\lambda x.M)N]$ formed by arbitrary λ -term contexts $C[]$, where $\lambda x.MN$ is called a **redex**, and furthermore:

$M[x := N]$:= substitution of N for free occurrences of x in M
(using **α -conversion** to avoid variable capture)

- **Many-step β -reduction** \rightarrow_β^* is defined as the concatenation of zero, one, or more \rightarrow_β -steps.

β -reduction

Definition

- **One-step β -reduction** \rightarrow_β is defined as the application of the rule:

$$(\lambda x.M)N \rightarrow_\beta M[x := N]$$

in λ -terms $C[(\lambda x.M)N]$ formed by arbitrary λ -term contexts $C[]$, where $(\lambda x.M)N$ is called a **redex**, and furthermore:

$M[x := N]$:= substitution of N for free occurrences of x in M
(using **α -conversion** to avoid variable capture)

- **Many-step β -reduction** \rightarrow_β^* is defined as the concatenation of zero, one, or more \rightarrow_β -steps.
- A λ -term M is a **normal form** if it does not contain a redex.

Church numerals

Definition

For every $n \in \mathbb{N}$, the Church numeral $\ulcorner n \urcorner$ for n is defined by:

$$\ulcorner n \urcorner := \lambda f x. f^n x$$

Church numerals

Definition

For every $n \in \mathbb{N}$, the Church numeral $\ulcorner n \urcorner$ for n is defined by:

$$\begin{aligned}\ulcorner n \urcorner &:= \lambda f x. f^n x \\ &= \lambda f x. f(\underbrace{f(\dots (f x) \dots)}_n)\end{aligned}$$

Church numerals

Definition

For every $n \in \mathbb{N}$, the **Church numeral** $\ulcorner n \urcorner$ for n is defined by:

$$\begin{aligned}\ulcorner n \urcorner &:= \lambda f x. f^n x \\ &= \lambda f x. f(\underbrace{f(\dots (f x) \dots)}_n)\end{aligned}$$

Examples.

$$\ulcorner 0 \urcorner = \lambda f x. x$$

$$\ulcorner 1 \urcorner = \lambda f x. f x$$

$$\ulcorner 2 \urcorner = \lambda f x. f(f x)$$

...



Pairs in λ -calculus

Definition

For all $M, N \in \Lambda$ we define the **pair** $\langle M, N \rangle$ consisting of M and N :

$$\langle M, N \rangle := \lambda x. xMN$$

and the **unpairing projections** ρ_1 and ρ_2 :

$$\rho_1 := \lambda p. p(\lambda xy. x)$$

$$\rho_2 := \lambda p. p(\lambda xy. y)$$

Proposition

For all $M_1, M_2 \in \Lambda$ and $i = 1, 2$:

$$\rho_i \langle M_1, M_2 \rangle \rightarrow_{\beta} M_i$$

True, false, if-then-else, **zero?** in λ -calculus

Definition

$$\mathbf{true} := \lambda xy.x$$
$$\mathbf{false} := \lambda xy.y$$
$$\mathbf{if } P \mathbf{ then } Q \mathbf{ else } R := PQR$$
$$\mathbf{zero?} := \lambda x.x(\lambda y.\mathbf{false})\mathbf{true}$$

Proposition

$$\mathbf{if } \mathbf{true} \mathbf{ then } Q \mathbf{ else } R \twoheadrightarrow_{\beta} Q$$
$$\mathbf{if } \mathbf{false} \mathbf{ then } Q \mathbf{ else } R \twoheadrightarrow_{\beta} R$$
$$\mathbf{zero?} \ulcorner 0 \urcorner \twoheadrightarrow_{\beta} \mathbf{true}$$
$$\mathbf{zero?} \ulcorner n + 1 \urcorner \twoheadrightarrow_{\beta} \mathbf{false}$$

λ -definable functions

Definition

- ▶ Let $f : \mathbb{N}^n \rightarrow \mathbb{N}$ be total.

A λ -term M_f **represents** f if for all $m_1, \dots, m_n \in \mathbb{N}$:

$$M_f \ulcorner m_1 \urcorner \dots \ulcorner m_n \urcorner \rightarrow_{\beta} \ulcorner f(m_1, \dots, m_n) \urcorner$$

f is **λ -definable** if there exists a λ -term that represents f .

λ -definable functions

Definition

- Let $f : \mathbb{N}^n \rightarrow \mathbb{N}$ be total.

A λ -term M_f **represents** f if for all $m_1, \dots, m_n \in \mathbb{N}$:

$$M_f \text{ ' } m_1 \text{ ' } \dots \text{ ' } m_n \text{ ' } \rightarrow_{\beta} \text{ ' } f(m_1, \dots, m_n) \text{ ' }$$

f is **λ -definable** if there exists a λ -term that represents f .

- Let $f : \mathbb{N}^n \rightarrow \mathbb{N}$ be a partial function.

A λ -term M_f **represents** f if for all $m_1, \dots, m_n \in \mathbb{N}$:

$$f(m_1, \dots, m_n) \downarrow \implies M_f \text{ ' } m_1 \text{ ' } \dots \text{ ' } m_n \text{ ' } \rightarrow_{\beta} \text{ ' } f(m_1, \dots, m_n) \text{ ' }$$

$$f(m_1, \dots, m_n) \uparrow \implies M_f \text{ ' } m_1 \text{ ' } \dots \text{ ' } m_n \text{ ' } \text{ has no normal form}$$

f is **λ -definable** if there exists a λ -term that represents f .

λ -definable

Examples.

- ▶ **successor:** $M_{\text{succ}} := \lambda n f x. f(n f x)$
- ▶ **addition:** $M_+ := \lambda m n f x. m f(n f x)$
- ▶ **multiplication:** $M_{\times} := \lambda m n f x. m(n f)x$
- ▶ **exponentiation:** $M_E := \lambda m n f x. m n f x$
- ▶ **unary constant zero function:** $M_{C_0^1} = \lambda m. \ulcorner 0 \urcorner$
- ▶ **projection function:** $M_{\pi_i^k} = \lambda n_1 \dots n_k. n_i$

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
 - ▶ 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

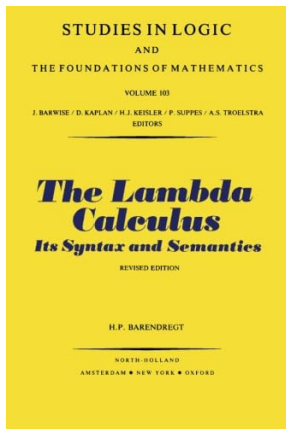
Typical features of 'computationally complete' MoC's

- ▶ 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

The Book



([1]) **Henk**drik Pieter (Henk) Barendregt

Exercises

- ▶ Reduce $(\lambda xy.x)(\lambda x.xx)(\lambda x.xx)$ to normal form.
- ▶ Find two distinct λ -terms representing the successor function on Church-numerals (hint: think of $n + 1$ and $1 + n$). Prove that your λ -terms are not- β -equivalent.
- ▶ Try computing the normal form of the Y -combinator, i.e. of AA where $A = \lambda am.m(aam)$, e.g. by each time selecting the leftmost redex (reducible expression, i.e. subexpression of the shape $(\lambda x.M)N$).

Primitive recursive functions are λ -definable

Proposition

Every primitive recursive function is λ -definable.

Primitive recursive functions are λ -definable

Proposition

Every primitive recursive function is λ -definable.

Proof (The case of primitive recursion).

Let $h := \text{pr}(f; g) : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ for prim.rec. $f : \mathbb{N}^n \rightarrow \mathbb{N}$, $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$:

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

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

Suppose that f and g are represented by $M_f, M_g \in \Lambda$, respectively.

Primitive recursive functions are λ -definable

Proposition

Every primitive recursive function is λ -definable.

Proof (The case of primitive recursion).

Let $h := \text{pr}(f; g) : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ for prim.rec. $f : \mathbb{N}^n \rightarrow \mathbb{N}$, $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$:

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

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

Suppose that f and g are represented by $M_f, M_g \in \Lambda$, respectively.

$$\text{Init} := \langle \ulcorner 0 \urcorner, M_f x_1 \dots x_n \rangle$$

$$\text{Step} := \lambda p. \langle M_{\text{succ}}(\rho_1 p), M_g x_1 \dots x_n (\rho_2 p) (\rho_1 p) \rangle$$

Then the following λ -term M_h represents h :

$$M_h := \lambda x_1 \dots x_n x. \rho_2 (x \text{ Step Init})$$



Primitive recursive functions ($\mathbb{N}^n \cup \mathbb{N}^0 \rightarrow \mathbb{N}$)

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 $h = \text{pr}(f; g) : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$:

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

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

μ -recursion, and partial recursive functions

Definition

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

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

μ -recursion, and partial recursive functions

Definition

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

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

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

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

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

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

μ -recursion, and partial recursive functions

Definition

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

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

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

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

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

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

Reminder: Kleene's normal form theorem

Theorem

For every *partial recursive* function $h : \mathbb{N}^n \rightarrow \mathbb{N}$ there exist *primitive recursive* functions $f : \mathbb{N} \rightarrow \mathbb{N}$ and $g : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ such that:

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

μ -recursive/partial recursive \Rightarrow λ -definable

Theorem

Every μ -recursive/partial recursive function is λ -definable.

Proof.

Let $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ be partial recursive.

μ -recursive/partial recursive \Rightarrow λ -definable

Theorem

Every μ -recursive/partial recursive function is λ -definable.

Proof.

Let $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ be partial recursive.

Then by Kleene's normal form theorem there exist $g : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ and $f : \mathbb{N} \rightarrow \mathbb{N}$ such that:

$$h(\vec{x}) = f \circ \mu(g)(\vec{x}) = f(\mu z. [g(\vec{x}, z) = 0])$$

μ -recursive/partial recursive \Rightarrow λ -definable

Theorem

Every μ -recursive/partial recursive function is λ -definable.

Proof.

Let $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ be partial recursive.

Then by Kleene's normal form theorem there exist $g : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ and $f : \mathbb{N} \rightarrow \mathbb{N}$ such that:

$$h(\vec{x}) = f \circ \mu(g)(\vec{x}) = f(\mu z. [g(\vec{x}, z) = 0])$$

Let M_f and M_g be λ -terms representing f and g , respectively. Let:

$$W := \lambda y. \text{if } (\text{zero? } M_g x_1 \dots x_n y) \text{ then } (\lambda w. M_f y) \text{ else } (\lambda w. w(M_{\text{succ}} y) w)$$

Then the following λ -term M_h represents h :

$$M_h := \lambda x_1 \dots x_n. W \text{ '0' } W$$



A normalizing reduction strategy

Normal order reduction strategy $\overset{n}{\rightarrow}$:

only perform \rightarrow_{β} -steps in left-most positions.

A normalizing reduction strategy

Normal order reduction strategy \xrightarrow{n} :

only perform \rightarrow_β -steps in left-most positions.

Theorem

The *normal order reduction strategy* in is normalizing in λ -calculus, that is:

$$M \twoheadrightarrow_\beta N \wedge N \text{ is a normal form} \implies M \xrightarrow{n} N$$

λ -definable \Rightarrow Turing-computable

Theorem

Every λ -definable function is Turing computable.

λ -definable \Rightarrow Turing-computable

Theorem

Every λ -definable function is Turing computable.

Idea of the Proof.

Let $f : \mathbb{N}^n \rightarrow \mathbb{N}$ be a partial function that is λ -definable. Then there exists a λ -term M_f that represents f .

λ -definable \Rightarrow Turing-computable

Theorem

Every λ -definable function is Turing computable.

Idea of the Proof.

Let $f : \mathbb{N}^n \rightarrow \mathbb{N}$ be a partial function that is λ -definable. Then there exists a λ -term M_f that represents f .

To compute f , one can build a Turing machine M that, for given $m_1, \dots, m_n \in \mathbb{N}$:

- ▶ simulates a normal order rewrite sequence on $M_f \ulcorner m_1 \urcorner \dots \ulcorner m_n \urcorner$

λ -definable \Rightarrow Turing-computable

Theorem

Every λ -definable function is Turing computable.

Idea of the Proof.

Let $f : \mathbb{N}^n \rightarrow \mathbb{N}$ be a partial function that is λ -definable. Then there exists a λ -term M_f that represents f .

To compute f , one can build a Turing machine M that, for given $m_1, \dots, m_n \in \mathbb{N}$:

- ▶ simulates a normal order rewrite sequence on $M_f \ulcorner m_1 \urcorner \dots \ulcorner m_n \urcorner$ to obtain the normal form $\ulcorner f(m_1, \dots, m_n) \urcorner$ □

Summary

Lambda calculus

- ▶ λ -calculus
 - ▶ syntax
 - ▶ reduction rules
- ▶ λ -definable functions
- ▶ primitive recursive functions are λ -definable
- ▶ μ -recursive/partial recursive functions are λ -definable
- ▶ λ -definable functions are Turing computable
- ▶ Hence: λ -definable = partial recursive = Turing-computable

Suggested reading

- ▶ Interaction-Based Models of Computation:

Chapter 7, The Lambda Calculus of the book:

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

Suggested reading

- ▶ Interaction-Based Models of Computation:

Chapter 7, The Lambda Calculus of the book:

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

- ▶ Post's Correspondence Problem

- ▶ see paper link webpage

- ▶ Fractran

- ▶ see paper and video link webpage

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

References



[Henk Pieter Barendregt.](#)

The Lambda Calculus (Its Syntax and Semantics), volume 103 of *Studies in Logic and the Foundations of Mathematics*.
Elsevier, 1984.



[Maribel Fernández.](#)

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