Introduction to Barendregt's Lambda Cube

Silvia Ghilezan

University of Novi Sad
Mathematical Institute SASA, Serbia

Lecture 1

Oregon Programming Language Summer School
Eugene, June 2023

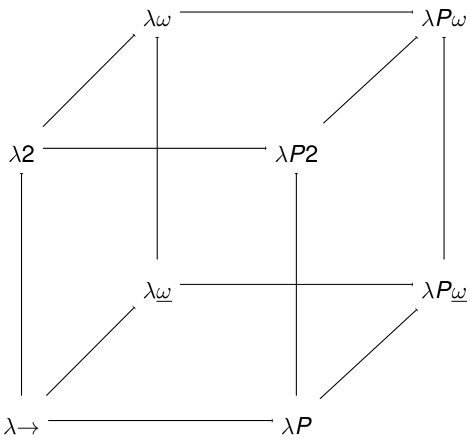


Figure: Lambda cube

- λ Untyped Lambda calculus
- $\lambda\!\rightarrow$ Simple types
- $\lambda 2$ Polymorphic types
- $\lambda\underline{\omega}$
- λP Dependent types
- Lambda cube
- Logic cube

Roadmap

Lambda Calculus

$\lambda \rightarrow$ - Simple types

Roadmap

Lambda Calculus

$\lambda \rightarrow$ - Simple types

λ-calculus, untyped lambda calculus

Decision Problem - Background

- Gottfried Wilhelm Leibniz - Characteristica universalis
- David Hilbert and Wilhelm Ackerman (1928)
  - **Entscheidungsproblem**, or Decision Problem:

    "Given all the axioms of math,
    is there an algorithm that can tell if a proposition is universally vali
    i.e. deducable from the axioms?"
- **Negative answers** (1935/36):
  - Alonzo Church - λ-calculus (equality)
  - Alan Turing - Turing Machines (halting problem)

  - Kurt Gödel - Incompleteness theorems (1931)

λ-calculus - 1930s

- Alonzo Church:
  - theory of functions - formalisation of mathematics (inconsistent)
  - successful model for computable functions - **λ-calculus**
  - **simply typed λ-calculus**
- Haskell Curry:
  - elimination of variables in logic - Moses Schönfinkel (1921)
  - successful model for computable functions - **Combinatory logic**
  - **Combinatory logic with types**
- Alan Turing :
  - formalisation of the concepts of algorithm and computation
  - **Turing Machines**

λ-calculus - expressiveness

- Expressiveness - Effective computability (mid 1930s)
  - **(Curry)** Equivalence of λ-calculus and Combinatory Logic
  - **(Kleene)** Equivalence of λ-calculus and recursive functions
  - **(Turing)** Equivalence of λ-calculus and Turing machines

Syntax

$$M \ ::= \ x \ | \ c \ | \ (MM) \ | (\lambda x.M)$$

x ranges over V, a countable set of variables
c ranges over C, a countable set of constants

Pure λ-calculus, if $C = \emptyset$

Conventions for minimizing the number of the parentheses:

- $M_1 M_2 M_3$ stands for $((M_1 M_2) M_3)$ application associates to left
- $\lambda x.y.M$ stands for $(\lambda x.(\lambda y.(M))$ abstraction associates to right
- $\lambda x.M_1 M_2 \ \equiv \ \lambda x.(M_1 M_2)$; application has priority over abstraction

Running example

$xyzx$	
$\lambda x.zx$	
$\mathbf{I} \equiv \lambda x.x$	combinator **I**
$\mathbf{K} \equiv \lambda xy.x$	combinator **K**
$\mathbf{S} \equiv \lambda xyz.xz(yz)$	combinator **S**
$\Delta \equiv \lambda x.xx$	selfapplication
$\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	fixed point combinator
$\Omega \equiv \Delta\Delta \equiv (\lambda x.xx)(\lambda x.xx)$	higher-order function

Free and bound variables

Definition

(i) The set $FV(M)$ of free variables of M is defined inductively:

- $FV(x) = \{x\}$
- $FV(MN) = FV(M) \cup FV(N)$
- $FV(\lambda x.M) = FV(M) \setminus \{x\}$

(ii) A variable in M is bound if it is not free

- x is bound in M if it appears in a subterm of the form $\lambda x.N$

(ii) M is a closed λ-term (or *combinator*) if $FV(M) = \emptyset$
Λ^o denotes the set of closed λ-terms.

Example

- In $\lambda x.zx$, variable z is free, $FV(M) = \{z\}$
- Term $\lambda xy.xxy$ is closed, $FV(M) = \emptyset$

Reduction rules - operational semantics

α-reduction:

$$\lambda x.M \longrightarrow_\alpha \lambda y.M[x := y], \;\; y \notin FV(M)$$

β-reduction:

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

η-reduction:

$$\lambda x.(Mx) \longrightarrow_\eta M, \;\; x \notin FV(M)$$

α-conversion

Formalisation of the principal that the name of the bound variable is irrelevant

α-reduction:

$$\lambda x.M \longrightarrow_\alpha \lambda y.M[x := y], \quad y \notin FV(M)$$

In math, $f(x) = x^2 + 1$ and $f(y) = y^2 + 1$ same, $f(5) = 26$
$\lambda x.(x^2 + 1)$ and $\lambda y.(y^2 + 1)$ must be considered as equal

Proposition $\twoheadrightarrow_\alpha$ is an equivalence relation, notation $=_\alpha$

Proof.

Symmetry, the interesting case □

β-reduction

Formalisation of function evaluation

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

- $M[x := N]$ represents an evaluation of the function M with N being the value of the parameter x.
- $(\lambda x.M)N$ is a redex and $M[x := N]$ is a contractum
- β-conversion is the symmetric closure of $\twoheadrightarrow_\beta$ is an equivalence (with α-reduction), notation $\equiv_\beta$

- Barendregt's variable convention: If a term contains a free variable which would become bound after *beta*-reduction, that variable should be renamed.
- Renaming could be done also by using De Bruijn name free notation.

Example

$$(\lambda x.x^2 + 1)5 \longrightarrow_\beta 5^2 + 1 \rightarrow 26$$

η-conversion

Formalisation of extensionality

Definition

η-reduction:

$$\lambda x.(Mx) \longrightarrow_\eta M, \quad x \notin FV(M)$$

- This rule identifies two functions that always produce equal results if taking equal arguments.

Example

$$\lambda x.\mathbf{succ}x \longrightarrow_\eta \mathbf{succ}$$

$$(\lambda x.\mathbf{succ}x)2 \longrightarrow_\beta \mathbf{succ}2 \qquad \mathbf{succ}2$$

Properties

- confluence
- normal forms
- normalisation
- strong normalisation
- fixed point theorem
- expressiveness

Properties - Confluence

Theorem (Church-Rosser theorem)

If $M \longrightarrow\!\!\!\!\!\rightarrow N$ and $M \longrightarrow\!\!\!\!\!\rightarrow P$, then there exists S such that $N \longrightarrow\!\!\!\!\!\rightarrow S$ and $P \longrightarrow\!\!\!\!\!\rightarrow S$

The proof is deep and involved.

Corollary

- *If $M \longrightarrow\!\!\!\!\!\rightarrow N$ and $M \longrightarrow\!\!\!\!\!\rightarrow P$, then $N = P$*
- *The order of the applied reductions is arbitrary and always leads to the same result*
- *Reductions can be executed in parallel (parallel computing)*

Proof.

□

Normal forms

- $N \in \Lambda$ is a normal form (NF) if there is no S such that $N \longrightarrow S$
- $P \in \Lambda$ is normalising (has a normal form) if $P \twoheadrightarrow N$ and N is a normal form, then N is a NF of P
- $P \in \Lambda$ is strongly normalising (SN) if all reductions of P are finite

Notation: $\twoheadrightarrow$ will denote $\twoheadrightarrow_\beta \cup \longrightarrow_\alpha$

Theorem (uniqueness of NF)

Every lambda term has at most one normal form

Proof.

Exercise

□

Running example: β-normal forms

$xyzx$	normal form NF
$\mathbf{I} \equiv \lambda x.x$	normal form NF
$\mathbf{K} \equiv \lambda xy.x$	normal form NF
$\mathbf{S} \equiv \lambda xyz.xz(yz)$	normal form NF
$\mathbf{KI}(\mathbf{KII})$	strongly normalizing SN
$\Omega \equiv \Delta\Delta \equiv (\lambda x.xx)(\lambda x.xx)$	unsolvable
$\mathbf{KI}\Omega$	normalizing N
$\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	head normalizing HN (solvable)

$\mathbf{KI}(\mathbf{KII}) \rightarrow \mathbf{KII} \rightarrow \mathbf{I}$
$\mathbf{KI}(\mathbf{KII}) \rightarrow \mathbf{I}$
$\Omega \rightarrow \Omega \rightarrow \Omega \rightarrow \Omega \rightarrow \dots$

Logic, conditionals, pairs

- Propositional logic in λ-calculus:

$$\top := \lambda xy.x \qquad \bot := \lambda xy.y \qquad \neg := \lambda x.x\bot\top$$
$$\wedge := \lambda xy.xy\bot \qquad \vee := \lambda xy.x\top y$$

Example

$$\top \vee A \longrightarrow (\lambda xy.x\top y)(\lambda zu.z)A \longrightarrow (\lambda zu.z)\top A \longrightarrow \top$$

- Conditionals and pairs in λ-calculus:

$$\textbf{if } A \textbf{ then } P \textbf{ else } Q := APQ$$
$$\textbf{fst} := \lambda x.x\top, \qquad \textbf{snd} := \lambda x.x\bot, \qquad (P, Q) := \lambda x.xPQ$$

Example

$$\textbf{if } \top \textbf{ then } P \textbf{ else } Q \equiv \top PQ \to (\lambda xy.x)PQ \to P$$

Arithmetic

- Church's numerals (arithmetics on the Nat set):

$$
\begin{array}{lcl}
\underline{0} & := & \lambda fx.x \\
\underline{1} & := & \lambda fx.fx \\
\underline{n} & := & \lambda fx.f^n x \\
\textbf{add} & := & \lambda xypg.xp(ypq) \\
\textbf{mult} & := & \lambda xyz.x(yz) \\
\textbf{succ} & := & \lambda xyz.y(xyz) \\
\textbf{exp} & := & \lambda xy.yx \\
\textbf{iszero} & := & \lambda n.n(\top\bot)\top
\end{array}
$$

- **add** $\underline{n}\ \underline{m} \;=_\beta\; \underline{n+m}$
- **mult** $\underline{n}\ \underline{m} \;=_\beta\; \underline{n \times m}$

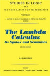
Exercise.

Expressiveness

In the mid 1930s

- **(Kleene)** Equivalence of λ-calculus and recursive functions
- **(Turing)** Equivalence of λ-calculus and Turing machines
- **(Curry)** Equivalence of λ-calculus and Combinatory Logic

References

H.P. Barendregt. 

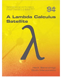
Lambda Calculus: Its syntax and Semantics.

North Holland, 1984.

F. Cardone, J. R. Hindley

History of Lambda-calculus and Combinatory Logic

Handbook of the History of Logic. Volume 5. Logic from Russell to Church Elsevier, 2009, pp. 723-817 (*online 2006*)

H.P. Barendregt, G. Manzonetto 

A Lambda Calculus Satellite

College Publications, 2022.

Roadmap

Lambda Calculus

$\lambda \rightarrow$ - Simple types

$\lambda \rightarrow$ simple (functional) types

Motivation

- "Disadvantages" of the untyped λ-calculus:
  - infinite computation - there exist λ-terms without a normal form
  - meaningless applications - it is allowed to create terms like **sin log**
- Types are syntactical objects that can be assigned to λ-terms
  - Reasoning with types was present in the early work of Church on untyped lambda calculus
- two typing paradigms:
  - *à la* Church - explicit type assignment (*typed lambda calculus*).
  - *à la* Curry - implicit type assignment (*lambda calculus with types*)

$\lambda \rightarrow$ syntax of types

$$\sigma \; ::= \; \alpha \mid (\sigma \rightarrow \sigma)$$

α ranges over TVar, a countable set of type variables

Conventions for minimising the number of the parentheses:

- $\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3$ stands for $(\sigma_1 \rightarrow (\sigma_2 \rightarrow \sigma_3))$

$$M : \sigma$$

Definition

- **Type assignment** is an expression of the form $M : \sigma$, where M is a λ-term and σ is a type
- **Declaration** $x : \sigma$ is a type assignment in which the term is a variable
- **Basis (context, environment)** $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ is a set of declarations in which all term variables are different
- **Statement (sequent)** $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \sigma$ ($\Gamma \vdash M : \sigma$)

$\lambda \rightarrow$ - the type system

à la Church and à la Curry

- Axiom

(*Ax*) $$\overline{\Gamma, x : \sigma \vdash x : \sigma}$$

- Rules

$(\rightarrow_{elim})$ $$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma . M : \sigma \rightarrow \tau} \; (\rightarrow_{intr}) \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x . M : \sigma \rightarrow \tau}$$

à la Church   à la Curry

Running example: types

M	Type
xyz	$x : \sigma \to \tau \to \rho, y : \sigma, z : \tau \vdash xyz : \rho$
$\lambda x.zx$	$z : \sigma \to \rho \vdash \lambda x.zx : \sigma \to \rho$
$\mathbf{I} \equiv \lambda x.x$	$\sigma \to \sigma$
$\mathbf{K} \equiv \lambda xy.x$	$\sigma \to \rho \to \sigma$
$\mathbf{S} \equiv \lambda xyz.xz(yz)$	$\sigma \to \rho \to \tau \to (\sigma \to \tau) \to (\sigma \to \rho)$
$\Delta \equiv \lambda x.xx$	NO
$\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	NO
$\Omega \equiv \Delta\Delta \equiv (\lambda x.xx)(\lambda x.xx)$	NO

I. $\lambda \rightarrow$ Fundamental properties

- Uniqueness of types
  If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma = \tau$
- Church-Rosser property holds in $\lambda \rightarrow$
- Subject reduction, type preservation under reduction

$$\text{If } M \longrightarrow P \text{ and } M : \sigma, \text{ then } P : \sigma.$$

  - Broader context: evaluation of terms (expressions, programs, processes) does not cause the type change.
  - type soundness
  - type safety = progress and preservation

II. $\lambda \rightarrow$ Strong normalisation

- Strong normalization

  If *M* : σ, then *M* is strongly normalizing.

  - Tait 1967
  - reducibility method (reducibility candidates, logical relations)
  - arithmetic proofs

III. $\lambda\to$ expressiveness

Selfapplication is not typable $\nvdash \lambda x.xx : \sigma$

Numerals are typeable

$\underline{n} \equiv \lambda f.\lambda x.f^n x : (\alpha\to\alpha)\to\alpha\to\alpha$ (exercise)

$\underline{n} \equiv \lambda x.\lambda f.f^n x : \alpha\to(\alpha\to\alpha)\to\alpha$ (exercise)

Definition (Extended polynomials)

The smallest class of functions over $\mathbb{N}$

- constant functions 0 and 1
- projections
- addition
- multiplication
- $\mathit{ifzero}(n, m, p) := \mathit{if}\ n = 0\ \mathit{then}\ m\ \mathit{else}\ p$

closed under composition

Theorem

M is typeable in $\lambda\to$ if and only if M is an extended polynomial

$\lambda \to$ and logic

Intuitionistic logic (minimal) - Natural deduction, Gentzen 1930s

- Axiom

(*Ax*) $$\frac{}{\Gamma, \sigma \vdash \sigma}$$

- Rules

$(\to_{elim})$ $$\frac{\Gamma \vdash \sigma \to \tau \qquad \Gamma \vdash \sigma}{\Gamma \vdash \tau}$$

$(\to_{intr})$ $$\frac{\Gamma, \sigma \vdash \tau}{\Gamma \vdash \sigma \to \tau}$$

$\lambda \to$ and logic

Intuitionistic logic (minimal) - Natural deduction, Gentzen 1930s

- Axiom

(*Ax*) $$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}$$

- Rules

($\to_{elim}$) $$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

($\to_{intr}$) $$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}$$

IV. $\lambda \rightarrow$ Curry-Howard correspondence

Intuitionistic logic vs computation

$$\vdash \sigma \quad \Leftrightarrow \vdash M : \sigma$$

A formula is provable in minimal intuitionistic logic
if and only if it is inhabited in $\lambda \rightarrow$.

- 1950s Curry
- 1968 (1980) Howard formulae-as-types

- 1970s Lambek - CCC Cartesian Closed Categories
- 1970s de Bruijn AUTOMATH

- 1970s Martin-Löf Type Theory

formulae (propositions)	–as–	types
proofs	– as –	terms
proofs	–as–	programs
proof normalisation	–as–	term reduction

- BHK - Brouwer, Heyting, Kolmogorov interpretation of logical connectives is formalized by the Curry-Howard correspondence

3 Type?

Type checking: given M and σ

$$(M : \sigma)?$$

Type inference (typability, type synthesis): given M

$$M : ?$$

Type inhabitation (term, program synthesis) : given σ

$$? : \sigma$$

$\lambda \rightarrow$ 3 Type?

Theorem

In $\lambda \rightarrow$

- *Type checking ($(M : \sigma)$?) is decidable*
- *Type inference (M :?) is decidable*
- *Type inhabitation (? : σ) is decidable*

$\lambda\to$ sum up

Advantages

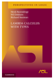
- All terms are SN
- Typability, inhabitation, type checking decidable
- Types exactly all extended polynomials

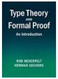
Shortcomings

- no self-application
- no recursion
- no factorial
- no total functions
- not Turing complete

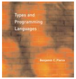
References

H.P. Barendregt 
Lambda calculi with types
Handbook of Logic in Computer Science, Oxford University Press, 1993

H.P. Barendregt, W. Dekkers, R. Statman 
Lambda Calculus with Types
Cambridge University Press 2013

R. Nederpelt, H. Geuvers 

Type Theory and Formal Proof
Cambridge University Press 2014

B. C. Pierece 

Types and programming languages
MIT Press 2002