Introduction to Type Theory

Silvia Ghilezan

University of Novi Sad
Mathematical Institute SANU
Serbia

EUTypes Summer School
Ohrid, July 2017

About me...

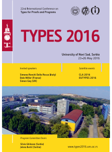
PhD University of Novi Sad

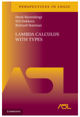
advised by Henk Barendregt and Kosta Došen

Postdoc University of Nijmegen

Research and teaching

- University of Turin
- McGill University
- École Normale Supérieure Lyon





Untyped lambda calculus

Simply lambda calculus

Intersection types

$\lambda o \gamma o \varsigma$

“a ground", “a plea", “an opinion", “an expectation", “a word", “a speech", “an account", “a reason".

- Aristotle: Organon, syllogisms 24, 4th BC
- Stoic logic: syllogisms, deductive logic
- Euclid: Elements
- Roman period
- Middle Age: Scholastic Thomas Aquinus, 12th AD
- Renesanse: Francis Bacon, inductive logic, scientific method
- Gotfrid Leibnitz: logic as a universal language
- Emanuel Kant: “laws of thinking"

- George Boole: “The Laws of Thought"
- Gotlab Frege: predicate logic
- Giuseppe Peano: axiomatization of natural numbers
- George Cantor: naive set theory
- Bertrand Russell:
  - Principia Mathematicae
  - paradox: $x \notin x$
  - **Theory of Types**
- Alonzo Church:
  - theory of functions - formalisation of mathematics (inconsistent),
  - successful model for computable functions
  - **Simply typed λ-calculus**
- Per Martin-Löf: **Type Theory**
- HoTT - **Homotopy Type Theory**

- David Hilbert:
  - problem of **consistency, completeness, decidability** (Entscheidungsproblem)
  - program: to provide secure foundations of all mathematics
- Luitzen Egbertus Jan Brouwer, Andrey Kolmogorov:
  - intuitionism, constructivism
- Gerhard Gentzen:
  - proof theory
- Kurt Gödel:
  - incompleteness theorems (PA is not complete)

Let f be a function given by

$$f(x) = x + 42$$

$$f(5) = 5 + 42 = \ \rightarrow 47$$

What is the function f

$$f = ?$$

function	domain	type
$f(x) = x + 42$	$\mathbb{R}$	$f : \mathbb{R} \to \mathbb{R}$
$g(x) = \frac{1}{x}$	$\mathbb{R}\backslash\{0\}$	$g : \mathbb{R}\backslash\{0\} \to \mathbb{R}$
$h(x) = \sqrt{x}$	$\mathbb{R}^+ \cup \{0\}$	$f : \mathbb{R}^+ \cup \{0\} \to \mathbb{R}$
$s(x) = \sin x$	$\mathbb{R}$	$f : \mathbb{R} \to [-1, 1]$

What are the type of f

$$f : A \to B$$

Untyped lambda calculus

Simply lambda calculus

Intersection types

Informal syntax

λ-terms are divided into three categories:

1. variables: $x, y, z, z_1, ...$
   - free or bound
2. application: MN
   - function application, "apply M to N"
3. abstraction: $\lambda x.M$
   - function generation by binding a variable, thus creating the parameter of the function.

Example

λ-term representing the function defined by $f(x) = x + 42$:

$$f = \lambda x.x + 42$$

Syntax - more formal

Definition

The set Λ of λ-terms is generated by a countable set of variables $V = \{x, y, z, x_1, ...\}$ using application and abstraction:

$$x \in V \text{ then } x \in \Lambda$$
$$M, N \in \Lambda \text{ then } (MN) \in \Lambda$$
$$M \in \Lambda, x \in V \text{ then } (\lambda x.M) \in \Lambda$$

Conventions for minimizing the number of the parentheses:

- $M_1 M_2 M_3 \ldots M_n$ stands for $((((M_1)M_2)M_3)\ldots M_n)$;
- $\lambda x.M_1 M_2 ... M_n \equiv \lambda x.(M_1 M_2 ... M_n)$;
- $\lambda x_1 ... x_n.M \equiv \lambda x_1.(\lambda x_2.(...(\lambda x_n.M)...))$.

$$M ::= x \mid MM \mid \lambda x.M$$

Running example

$$xyzx$$

$$\lambda x.zx$$

$$\mathbf{I} = \lambda x.x$$

$$\mathbf{K} = \lambda xy.x$$

$$\Delta = \lambda x.xx$$

$$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

$$\Omega = \Delta\Delta = (\lambda x.xx)(\lambda x.xx)$$

Free and bound variables

Definition

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

$$\begin{aligned} FV(x) &= \{x\} \\ FV(MN) &= FV(M) \cup FV(N) \\ FV(\lambda x.M) &= FV(M) \setminus \{x\} \end{aligned}$$

A variable in M is bound if it is not free.

(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.
- Term $\lambda xy.xxy$ is closed.

Running example

M	$Fv(M)$
$xyzx$	$\{x, y, z\}$
$\lambda x.zx$	$\{z\}$
$\mathbf{I} = \lambda x.x$	$\emptyset$
$\mathbf{K} = \lambda xy.x$	$\emptyset$
$\Delta = \lambda x.xx$	$\emptyset$
$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	$\emptyset$
$\Omega = \Delta\Delta = (\lambda x.xx)(\lambda x.xx)$	$\emptyset$

α-conversion

Definition

α-reduction:

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

- Bound variables could be renamed in order to avoid name clashing;
- Barendregt's 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

$$\lambda x.fx =_\alpha \lambda y.fy$$

“A rose by any other name would smell as sweet"

William Shakespeare, “Romeo and Juliet"

β-reduction

The principal reduction rule pf λ-calculus:

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

provided that $M, N \in \Lambda$ and $M[x := N]$ is valid substitution.

- Represents an evaluation of the function M with N being the value of the parameter x.

Example

$$(\lambda x.x + 42)5 \longrightarrow_\beta 5 + 42 \rightarrow 47$$

Substituion

Implicit substitution, meta notion

$$
\begin{array}{lcl}
x[x := M] & = & M \\
y[x := M] & = & y \\
(PN)[x := M] & = & P[x := M]\ N[x := M] \\
(\lambda x.N)[x := M] & = & \lambda x.N \\
(\lambda y.N)[x := M] & = & \lambda y.(N[x := M]),\ y \notin FV(M) \\
(\lambda y.N)[x := M] & = & (\lambda z.N[y := z])[x := M]), \\
 & & y \in FV(M)\ \wedge\ z \notin FV(M)
\end{array}
$$

λ-calculus with explicit substitution.

η-conversion

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

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

Running example

M	
$xyzx$	normal form NF
$\mathbf{I} = \lambda x.x$	normal form NF
$\mathbf{K} = \lambda xy.x$	normal form NF
$\mathbf{KI}(\mathbf{KII})$	strongly normalizing SN
$\mathbf{KI}\Omega$	normalizing N
$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	solvable HN
$\Omega = \Delta\Delta = (\lambda x.xx)(\lambda x.xx)$	unsolvable

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

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

$\mathbf{KI}\Omega \ \rightarrow \ \mathbf{I}$

$\mathbf{KI}\Omega \ \rightarrow \ \mathbf{KI}\Omega \ \rightarrow \ \ldots \rightarrow \ \mathbf{KI}\Omega \ \rightarrow \ \mathbf{I}$ stop

$\mathbf{KI}\Omega \ \rightarrow \ \mathbf{KI}\Omega \ \rightarrow \ \ldots \rightarrow \ \mathbf{KI}\Omega \ \rightarrow \ \ldots$ infinite loop

Properties - Confluence

- **Church-Rosser (CR) theorem:** If $M \to N$ and $M \to P$, then there exists S such that $N \longrightarrow S$ and $P \longrightarrow S$ *(confluence)*.

  The proof is deep and involved.

- The corollaries of CR theorem:
  - the order of the applied reductions is arbitrary and always leads to the same result (contractum);
  - every λ-term has at most one normal form (uniqueness of NF);
  - reductions can be executed in parallel (parallel computing).

Properties - Normalisation theorem

Definition

(i) λ-term is in the *head normal form* if its form is:

$$\lambda x_1 x_2 ... x_m . y M_1 ... M_k, \quad m, k \geq 0$$

(ii) λ-term is in the *normal form* if it doesn't contain any β nor η redexes:

$$\lambda x_1 x_2 ... x_m . y N_1 ... N_k, \quad N_i \in NF \text{ and} m, k \geq 0$$

- Informally, λ-term is in the normal form if it is completely evaluated.
- **Normalisation theorem:** If M has normal form, then the leftmost strategy (the sequence of left β- and η-reductions, is terminating, and the result is the normal form of M.

Properties - Fixed point theorems

- **Fixedpoint theorem:** There is a fixed point combinator

$$\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

  such that $\forall F,\ F(\mathbf{Y}F) = \mathbf{Y}F,\ F \in \Lambda$.
- **Multiple fixedpoint theorem:** Let $F_1, F_2, ..., F_n \in \Lambda$. There exist $X_1, X_2, ..., X_n$ such that

$$X_1 = F_1 X_1 X_2 ... X_n, \quad ... , \quad X_n = F_n X_1 X_2 ... X_n.$$

- These properties enable the representation of the recursive functions in λ-calculus.

Logic, conditionals, pairs

- Propositional logic in λ-calculus:

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

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:

$$(E \to M_1|M_2) := MM_1M_2,$$
$$\mathbf{fst} := \lambda x.x\top, \qquad \mathbf{snd} := \lambda x.x\bot, \qquad (M_1, M_2) := \lambda x.xM_1M_2$$

Example

$$(\top \to M_1|M_2) \longrightarrow \top M_1M_2 \longrightarrow (\lambda xy.x)M_1M_2 \longrightarrow M_1$$

Arithmetic

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

$$
\begin{array}{lcl}
\underline{0} & := & \lambda fx.x, \\
\underline{1} & := & \lambda fx.fx, \\
\underline{n} & := & \lambda fx.f^n x, \\
\textbf{succ} & := & \lambda nfx.nf(fx), \\
\textbf{add} & := & \lambda mnfx.mf(nfx), \\
\textbf{iszero} & := & \lambda n.n(\lambda x.\bot)\top, \\
\textbf{pre} & := & \lambda nfx.\textbf{snd}(n(\textbf{prefn}\ f)(\top, x)) \\
\textbf{mult} & := & \textbf{Y multifn}
\end{array}
$$

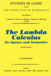
- $\textbf{multifn} \ := \ \lambda\ f\ m\ n.(\textbf{iszero}\ m\ \rightarrow\ 0 \mid \textbf{add}\ n\ (f\ (\textbf{pre}\ m)\ n))$
- $\textbf{add}\ \underline{n}\ \underline{m}\ =_\beta\ \underline{n+m}$
- $\textbf{mult}\ \underline{n}\ \underline{m}\ =_\beta\ \underline{n \times m}$

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

Untyped lambda calculus

Simply lambda calculus

Intersection 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 present in the early work of Church on untyped lambda calculus.
- two typing paradigms:
  - *à la* Curry - implicit type assignment (*lambda calculus with types*);
  - *à la* Church - explicit type assignment (*typed lambda calculus*).

Simply typed λ-calculus - syntax of types

Definition

- The alphabet consists of
  - $\mathrm{V} = \{\alpha, \beta, \gamma, \alpha_1, \ldots\}$, a countable set of type variables
  - $\rightarrow$, a type forming operator
  -), (auxiliary symbols
- The language is the set of types **T** defined as follows
  - If $\alpha \in \mathrm{V}$ then $\alpha \in$ **T**
  - If $\sigma, \tau \in$ **T** then $(\sigma \rightarrow \tau)$ *in* **T**.
- The abstract grammar that generates the language

$$\sigma \ ::= \ \alpha \mid \sigma \rightarrow \sigma$$

Conventions for minimizing the number of the parentheses:

- $\sigma_1 \rightarrow \sigma_2 \rightarrow \ldots \sigma_{n-1} \rightarrow \sigma_n$ stands for $(\sigma_1 \rightarrow (\sigma_2 \rightarrow \ldots (\sigma_{n-1} \rightarrow \sigma_n) \ldots))$;

$\lambda \rightarrow$ - the language

$$M : \sigma$$

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 type assignment in which the term is a variable.
- **Basis (environment)** $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ is a set of declarations in which every variable is assigned as most one type.

$\lambda \rightarrow$ - the type system

- Axiom

(*Ax*1) $$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}$$

- Rules

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

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

Running example

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

- **Subject reduction, type preservation under reduction**

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

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

- **Strong normalization**

  If $M : \sigma$, then M is strongly normalizing.

  - Tait 1967
  - reducibility method (reducibility candidates)
  - arithmetic proofs

Typability (type inference): given M

$$M : ?$$

Inhabitation: given σ

$$? : \sigma$$

Type checking: given M and σ

$$(M : \sigma)?$$

Typability

$\lambda \rightarrow$ typability is decidable

$$M :? \quad \text{decidable}$$

- principal type scheme (Hindley)
- Hindley-Milner-Damas algorithm

Inhabitation

Intuitionistic logic - Natural deduction, Gentzen 1930s

- Axiom

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

- Rules

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

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

Inhabitation

Intuitionistic logic - Natural deduction, Gentzen 1930s

- Axiom

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

- Rules

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

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

Curry-Howard correspondence

Intuitionistic logic vs computation

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

A formula is provable in *LI* 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

formulae	–as–	types
proofs	– as –	terms
proofs	–as–	programs
proof normalisation	–as–	term reduction

- BHK - Brouwer, Heyting, Kolmogorov interpretation of logical connectives is formalized by CH-

Curry-Howard correspondence

Consistency Completeness
Decidability

- Intuitionistic propositional logic *LI* is consistent, complete and decidable.
- Due to CH, decidability of provability in *LI* implies decidability of inhabitation in $\lambda \rightarrow$.
- $\lambda \rightarrow$ inhabitation decidable

$? : \sigma$ decidable

Curry-Howard paradigm

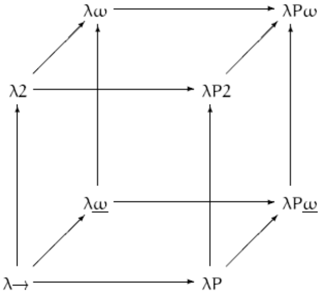
“If, in some cataclysm, all of scientific knowledge were to be destroyed, and only one sentence passed on to the next generation of creatures, what statement would contain the most information in the fewest words?

I believe it is the atomic hypothesis that
all things are made of atoms." - Richard Feynman

In this domain, proposal:

Logical deduction and computation embody each other

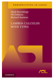
Lambda cube



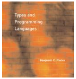
Theorem
M is typable $\Longrightarrow$ *M strongly normalizing.*

- More type systems: intersection types, recursive types

References

H.P. Barendregt, W. Dekkers, R. Statman. 

Lambda Calculus with Types.
Cambridge University Press 2013.

B. C. Pierece. 

Types and programming languages.
MIT Press 2002.

There is no perfect world

In $\lambda\to$

$$\lambda x.xx : \textit{NO}$$

Untyped lambda calculus

Simply lambda calculus

Intersection types

Intersection types

- The abstract grammar that generates the language

$$\sigma \ ::= \ \alpha \mid \sigma \to \sigma \mid \sigma \cap \sigma$$

- Axiom

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

- Rules

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\,(elim \to) \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\,(intr \to)$$

$$\frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \sigma}\,(elim\cap) \frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \tau}\,(elim\cap) \qquad \frac{\Gamma, \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \cap \tau}\,(intr\cap)$$

$$\frac{\Gamma, \vdash M : \sigma \quad \sigma \leq \tau}{\Gamma \vdash M : \tau}\,(\leq)$$

Intersection types

Introduced in the 1980s, to overcome the limitations of simple types

- Coppo, Dezani
- Pottinger
- Sallé

- Intersection types do not correspond to inuitionistic conjunction

$$\sigma \to \tau \to \sigma \cap \tau$$

  intuitionistically provable, but not inhabited in $\lambda\cap$

$$\dfrac{\dfrac{\dfrac{x : (\sigma \to \tau) \cap \sigma \vdash x : (\sigma \to \tau) \cap \sigma}{x : (\sigma \to \tau) \cap \sigma \vdash x : \sigma \to \tau}\,(\mathit{elim}\cap) \qquad \dfrac{x : (\sigma \to \tau) \cap \sigma \vdash x : (\sigma \to \tau) \cap \sigma}{x : (\sigma \to \tau) \cap \sigma \vdash x : \sigma}}{x : (\sigma \to \tau) \cap \sigma \vdash xx : \tau}\,(\mathit{elin}}{\vdash \lambda x.xx : ((\sigma \to \tau) \cap \sigma) \to \tau}\,(\mathit{intr} \to)$$

Intersection types - SN

Complete characterization of strong normalization

Theorem

$$M \text{ is typable} \Longleftrightarrow M \text{ is SN}$$

Proof.

Typability $\Longrightarrow$ SN

- reducibility method

SN $\Longrightarrow$ Typability

- typability of normal forms
- head subject expansion

Complete characterization of different normalization properties: solvable, weakly solvable terms and standardization, among others.

- Typability is undecidable

Intersection types - Inhabitation

- Inhabiation is undecidable - 1996 P. Urzyczyn, J. Tyurin

$$? : \sigma$$

  - Automated theorem proving
  - Program synthesis
  - Fragments of intersection types with decidable inhabitation
    - J. Rehof
  - Complexity

Intersection types - models of λ-calculus

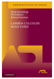
Filter models built by means of intersection types, enable to prove **completeness** of type assignment.

Theorem

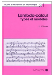
$$\Gamma \vdash M : \sigma \iff \Gamma \models M : \sigma$$

Filter models became powerful tool for proving completeness of type assignment in different framework.

References

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

H.P. Barendregt, M. Coppo, M. Dezani-Ciancaglini.
A filter lambda model and the completeness of type assignment.
Journal of Symbolic Logic 48(4):931–940, 1983.

J.-L. Krivine. 

Lambda-calcul, types et modèls
Masson 1990
Lambda calculus types and models, English translation

Take away

- Types have gained an important role in the analysis of formal systems.
- A type system
  - splits elements of the language (terms)
  - into sets (types)
  - proves absence of certain undesired behaviours.
- Formal calculi: subject reduction (type preservation under reduction, characterization of reduction.
- Programming languages: types represent a well-established technique to ensure program correctness.
- Concurrent systems: types have become a powerful means for security and access control and privacy issues.