Types and Functions since Principia and the Computerisation of Language and Mathematics

Fairouz Kamareddine
Heriot-Watt University
Edinburgh, Scotland
`http://www.macs.hw.ac.uk/~fairouz/talks/talks2013/pas13.pdf`

23 October 2013

Summary

- *General definition of function 1879* [17] is key to Frege's *formalisation of logic*.
- *Self-application of functions* was at the heart of *Russell's paradox 1902* [51].
- To *avoid paradox* Russell controled function application via *type theory*.
- Russell [52] *1903* gives the first type theory: the *Ramified Type Theory* (RTT).
- RTT is used in Russell and Whitehead's Principia Mathematica [55] 1910–1912.
- *Simple theory of types* (STT): Ramsey [47] *1926*, Hilbert and Ackermann [26] *1928*.
- Church's *simply typed λ-calculus* $\lambda\!\!\rightarrow$ [12] 1940 = λ-calculus + STT.

- The hierarchies of types (and orders) as found in RTT and STT are *unsatisfactory*.
- The *notion of function adopted in the λ-calculus* is *unsatisfactory* (cf. [30]).
- Hence, birth of *different systems of functions and types*, each with *different functional power*.
- We discuss the evolution of functions and types and their use in logic, language and computation.
- We then concentrate on these notions in mathematical vernaculars (as in Automath) and in logic.
- Frege's functions $\neq$ Principia's functions $\neq$ *λ-calculus* functions (1932).
- Not all functions need to be *fully abstracted* as in the λ-calculus. For some functions, their values are enough.

- *Non-first-class functions* allow us to stay at a lower order (keeping decidability, typability, etc.) without losing the flexibility of the higher-order aspects.

- Furthermore, non-first-class functions allow placing the type systems of modern theorem provers/programming languages like ML, LF and Automath more accurately in the modern formal hierarchy of types.

- Another issue that we touch on is the lessons learned from formalising mathematics in logic (à la Principia) and in proof checkers (à la Automath, or any modern proof checker).

Prehistory of Types (formal systems in 19th century)

In the 19th century, the need for a more *precise* style in mathematics arose, because controversial results had appeared in analysis.

- 1821: Many of these controversies were solved by the work of Cauchy. E.g., he introduced *a precise definition of convergence* in his *Cours d'Analyse* [11].
- 1872: Due to the more *exact definition of real numbers* given by Dedekind [16], the rules for reasoning with real numbers became even more precise.
- 1895-1897: Cantor began formalizing *set theory* [9, 10] and made contributions to *number theory*.
- 1889: Peano formalized *arithmetic* [46], but did not treat logic or quantification.

Prehistory of Types (formal systems in 19th century)

- 1879: Frege was not satisfied with the use of natural language in mathematics:

  > ". . . I found the inadequacy of language to be an obstacle; no matter how unwieldy the expressions I was ready to accept, I was less and less able, as the relations became more and more complex, to attain the precision that my purpose required."
  >
  > (*Begriffsschrift*, Preface)

  Frege therefore presented *Begriffsschrift* [17], the first formalisation of logic giving logical concepts via symbols rather than natural language.

  > "[Begriffsschrift's] first purpose is to provide us with the most reliable test of the validity of a chain of inferences and to point out every presupposition that tries to sneak in unnoticed, so that its origin can be investigated."
  >
  > (*Begriffsschrift*, Preface)

Prehistory of Types (Begriffsschrift's functions)

The introduction of a *very general definition of function* was the key to the formalisation of logic. Frege defined what we will call the Abstraction Principle.

Abstraction Principle 1.

> *"If in an expression, [. . .] a simple or a compound sign has one or more occurrences and if we regard that sign as replaceable in all or some of these occurrences by something else (but everywhere by the same thing), then we call* the part that remains invariant in the expression *a* function, *and* the replaceable part *the* argument *of the function."*
>
> (*Begriffsschrift*, Section 9)

Prehistory of Types (Begriffsschrift's functions)

- Frege put *no restrictions* on what could play the role of *an argument*.
- An argument could be a *number* (as was the situation in analysis), but also a *proposition*, or a *function*.
- the *result of applying* a function to an argument did not have to be a number.
- Frege was aware of some typing rule that does not allow to substitute functions for object variables or objects for function variables:

  " Now just as functions are fundamentally different from objects, so also *functions whose arguments are and must be functions* are fundamentally different from *functions whose arguments are objects and cannot be anything else*. I call the latter *first-level*, the former *second-level*."
  (*Function and Concept*, pp. 26–27)

Prehistory of Types (Grundgesetze's functions)

The *Begriffsschrift*, however, was only a prelude to Frege's writings.

- In Grundlagen der Arithmetik [18] he argued that mathematics can be seen as a branch of logic.
- In Grundgesetze der Arithmetik [19, 21] he described the elementary parts of arithmetics within an extension of the logical framework of *Begriffsschrift*.
- Frege approached the *paradox threats for a second time* at the end of Section 2 of his *Grundgesetze*.
- He did not want to *apply a function to itself*, but to its course-of-values.
- "the function $\Phi(x)$ has the same *course-of-values* as the function $\Psi(x)$" if:

> " $\Phi(x)$ and $\Psi(x)$ always have the same value for the same argument."
> (*Grundgesetze*, p. 7)

- E.g., let $\Phi(x)$ be $x \wedge \neg x$, and $\Psi(x)$ be $x \leftrightarrow \neg x$, for all propositions x.

Prehistory of Types (Grundgesetze's functions)

- All essential information of a function is contained in its graph.
- So a system in which a function can be applied to its own graph should have similar possibilities as a system in which a function can be applied to itself.
- Frege *excluded the paradox threats* by *forbidding self-application*, but due to his *treatment of courses-of-values* these threats were able to *enter his system through a back door*.
- In 1902, Russell wrote to Frege [51] that he had *discovered a paradox* in his *Begriffsschrift* (*Begriffsschrift does not suffer from a paradox*).
- *Only six days later*, Frege answered that *Russell's derivation of the paradox was incorrect* [20]. That *self-application $f(f)$ is not possible in the* Begriffsschrift. And that *Russell's argument could be amended to a paradox* in the system of his Grundgesetze, using the *course-of-values* of functions.

Prehistory of Types (paradox in Peano and Cantor's systems)

- Frege's system was *not the only paradoxical* one.
- The Russell Paradox can be derived in *Peano's system* as well, as well as on *Cantor's Set Theory* by defining the class $K \stackrel{\text{def}}{=} \{x \mid x \notin x\}$ and deriving $K \in K \longleftrightarrow K \notin K$.
- Paradoxes were already widely known in *antiquity*.
- The oldest logical paradox: the *Liar's Paradox* "This sentence is not true", also known as the Paradox of Epimenides. It is referred to in the Bible (Titus $1{:}12$) and is based on the confusion between language and meta-language.
- The *Burali-Forti paradox* ([8], 1897) is the first of the modern paradoxes. It is a paradox within Cantor's theory on ordinal numbers.
- *Cantor's paradox* on the largest cardinal number occurs in the same field. It discovered by Cantor around 1895, but was not published before 1932.

Prehistory of Types (paradoxes)

- Logicians considered these paradoxes to be *out of the scope of logic*:
  The *Liar's Paradox* can be regarded as a problem of *linguistics*.
  The *paradoxes of Cantor and Burali-Forti* occurred in what was considered in those days a *highly questionable* part of mathematics: Cantor's Set Theory.

- The Russell Paradox, however, was *a paradox that could be formulated in all* the systems that were presented at the end of the 19th century (except for Frege's *Begriffsschrift*). It was at the very basics of logic. It could not be disregarded, and a solution to it had to be found.

- In 1903-1908, Russell suggested the use of *types* to solve the problem [53].

Prehistory of Types (vicious circle principle)

> "In all the above contradictions there is a common characteristic, which we may describe as *self-reference* or *reflexiveness*. [...] In each contradiction something is said about all cases of some kind, and from what is said a new case seems to be *generated*, which both *is and is not* of the same kind as the cases of which *all* were concerned in what was said."
>
> (*Mathematical logic as based on the theory of types*)

Russell's plan was, *to avoid the paradoxes* by *avoiding all possible self-references*. He postulated the *"vicious circle principle"*:

> "*Whatever involves all of a collection* must not be one *of the collection.*"
>
> (Mathematical logic as based on the theory of types)

Russell implements this principle *very strictly* using *types*.

Problems of Ramified Type Theory

- The main part of the *Principia* is devoted to the development of logic and mathematics using the legal pfs of the ramified type theory.

- *ramification*/division of simple types into orders make RTT not easy to use.

- **(Equality)** $\mathtt{x} =_{\mathrm{L}} \mathtt{y} \overset{\mathrm{def}}{\leftrightarrow} \forall \mathtt{z}[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})]$.
  In order to express this general notion in RTT, we have to incorporate *all* pfs $\forall \mathtt{z} : (0^0)^n[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})]$ for $n > 1$, and this cannot be expressed in one pf.

- Not possible to give a constructive proof of the theorem of the least upper bound within a ramified type theory.

- It is not possible in RTT to give a definition of an object that refers to the class to which this object belongs (because of the Vicious Circle Principle). Such a definition is called an *impredicative definition*.

Axiom of Reducibility

- Russell and Whitehead tried to solve problems with the axiom of reducibility: *For each formula f, there is a formula g with a* predicative *type such that f and g are (logically) equivalent.*
- The validity of the Axiom of Reducibility has been questioned from the moment it was introduced.
- Though Weyl [54] made an effort to develop analysis within the Ramified Theory of Types (without the Axiom of Reducibility),
- and various parts of mathematics can be developed within RTT and without the Axiom,
- the general attitude towards RTT (without the axiom) was that the system was too restrictive, and that a *better solution* had to be found.

Deramification

- Ramsey considers it essential to divide the paradoxes into two parts:

- logical or syntactical paradoxes (like the Russell paradox, and the Burali-Forti paradox) are removed

  "by pointing out that a propositional function cannot significantly take itself as argument, and by dividing functions and classes into a hierarchy of types according to their possible arguments."

  (The Foundations of Mathematics, p. 356)

- Semantical paradoxes are excluded by the hierarchy of orders. These paradoxes (like the Liar's paradox, and the Richard Paradox) are based on the confusion of language and meta-language. These paradoxes are, therefore, not of a purely mathematical or logical nature. When a proper distinction between object language and meta-language is made, these so-called semantical paradoxes disappear immediately.

The Simple Theory of Types

- Ramsey [47], and Hilbert and Ackermann [26], *simplified* the Ramified Theory of Types RTT by removing the orders. The result is known as the Simple Theory of Types *(*STT*)*.

- Nowadays, STT is known via Church's formalisation in λ-calculus. However, STT *already existed (1926) before λ-calculus did (1932)*, and is therefore not inextricably bound up with λ-calculus.

- How to obtain STT from RTT? Just *leave out all the orders* and the references to orders (including the notions of predicative and impredicative types).

Limitation of the simply typed λ-calculus

- $\lambda\!\!\rightarrow$ is very restrictive.
- Numbers, booleans, the identity function have to be defined at every level.
- We can represent (and type) terms like $\lambda x : o.x$ and $\lambda x : \iota.x$.
- We cannot type $\lambda x : \alpha.x$, where α can be instantiated to any type.
- This led to new (modern) type theories that allow more general notions of functions (e.g, *polymorphic*).

The evolution of functions with Frege, Russell and Church

- Historically, functions have long been treated as a kind of meta-objects.
- Function *values* were the important part, not abstract functions.
- In the *low level/operational approach* there are only function values.
- The sine-function, is always expressed with a value: $\sin(\pi)$, $\sin(x)$ and properties like: $\sin(2x) = 2\sin(x)\cos(x)$.
- In many mathematics courses, one calls $f(x)$—and not f—the function.
- Frege, Russell and Church wrote $x \mapsto x+3$ resp. as $x+3$, $\hat{x}+3$ and $\lambda x.x+3$.
- Principia's *functions are based on Frege's Abstraction Principles* but can be first-class citizens. Frege used courses-of-values to speak about functions.
- Church made every function a first-class citizen. This is rigid and does not represent the development of logic in 20th century.

Functionalisation and Instantiation

[35] assessed evolution of the function concept from two points of vue:

- *Functionalisation*: the construction of a function out of an expression, as in constructing the function $\lambda_x.x \times 3 + x$ from the expression $2 \times 3 + 2$.

- Functionalisation is
  - *Abstraction from a subexpression* e.g., moving from $2 \times 3 + 2$ to $x \times 3 + x$
  - *Function construction* e.g., turning $x \times 3 + x$ into $\lambda_x.x \times 3 + x$.

- *Instantiation*: the calculation of a function value when a suitable argument is assigned to the function,
  as in the construction of $2 \times 3 + 2$ by applying the function $\lambda_x.x \times 3 + x$ to 2.

- Instantiation is:
  - *Application construction* e.g., $(\lambda_x.x \times 3 + x)2$ the application of $\lambda_x.x \times 3 + x$ to 2
  - *Concretisation to a subexpression* e.g., calculating $(\lambda_x.x \times 3 + x)2$ to $2 \times 3 + 2$.

λ-calculus does not fully represent functionalisation

1. Abstraction from a subexpression $2 + 3 \mapsto x + 3$
2. Function construction $x + 3 \mapsto \lambda x.x + 3$
3. Application construction $(\lambda x.x + 3)2$
4. Concretisation to a subexpression $(\lambda x.(x + 3))2 \rightarrow 2 + 3$

- cannot abstract only half way: $x + 3$ is not a function, $\lambda x.x + 3$ is.
- cannot apply $x + 3$ to an argument: $(x + 3)2$ does not evaluate to 2+3.

Common features of modern types and functions

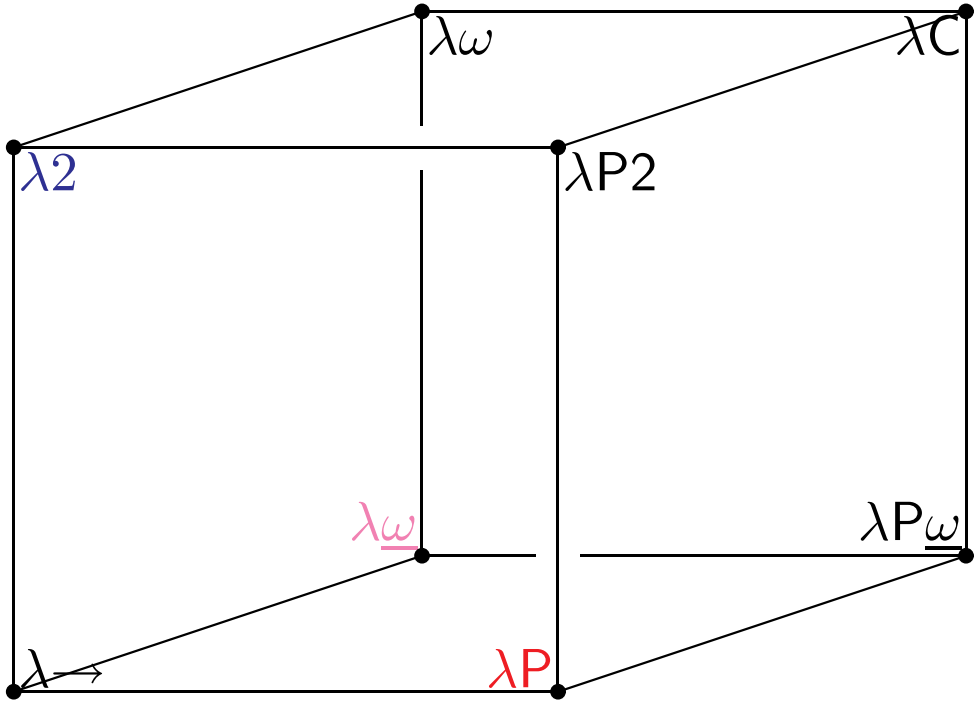
- We can *construct* a type by abstraction. (Write $A : *$ for A *is a type*)
  - $\lambda_{y:A}.y$, the identity over A *has type* $A \to A$
  - $\lambda_{A:*}.\lambda_{y:A}.y$, the polymorphic identity *has type* $\Pi_{A:*}.A \to A$
- We can *instantiate* types. E.g., if $A = \mathbb{N}$, then the identity over $\mathbb{N}$
  - $(\lambda_{y:A}.y)[A := \mathbb{N}]$ *has type* $(A \to A)[A := \mathbb{N}]$ or $\mathbb{N} \to \mathbb{N}$.
  - $(\lambda_{A:*}.\lambda_{y:A}.y)\mathbb{N}$ *has type* $(\Pi_{A:*}.A \to A)\mathbb{N} = (A \to A)[A := \mathbb{N}]$ or $\mathbb{N} \to \mathbb{N}$.
- $(\lambda x{:}\alpha.A)B \to_\beta A[x := B]$ $\qquad$ $(\Pi x{:}\alpha.A)B \to_\Pi A[x := B]$
- Write $A \to A$ as $\Pi_{y:A}.A$ when y not free in A.

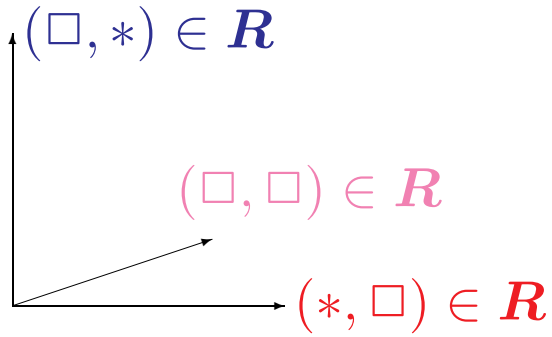
The Barendregt Cube

- Syntax: $A ::= x \mid * \mid \Box \mid AB \mid \lambda x{:}A.B \mid \Pi x{:}A.B$
- Formation rule: $$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2} \quad \text{if } (s_1, s_2) \in \boldsymbol{R}$$

	Simple	Poly-morphic	Depend-ent	Constr-uctors	Related system	Refs.
$\lambda{\to}$	$(*,*)$				λ^τ	[12, 2, 27]
$\lambda 2$	$(*,*)$	$(\Box,*)$			F	[23, 50]
λP	$(*,*)$		$(*,\Box)$		AUT-QE, LF	[6, 24]
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$	POLYREC	[49]
λP2	$(*,*)$	$(\Box,*)$	$(*,\Box)$			[43]
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$	Fω	[23]
λP$\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$		
λC	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$	CC	[13]

The Barendregt Cube





Typing Polymorphic identity needs $(\Box, *)$

- $$\frac{y : * \vdash y : * \quad y : *, x{:}y \vdash y : *}{y : * \vdash \Pi x{:}y.y : *}$$ by (Π) $(*, *)$

- $$\frac{y : *, x : y \vdash x : y \quad y : * \vdash \Pi x{:}y.y : *}{y : * \vdash \lambda x : y.x : \Pi x{:}y.y}$$ by (λ)

- $$\frac{\vdash * : \Box \quad y : * \vdash \Pi x{:}y.y : *}{\vdash \Pi y : *.\Pi x{:}y.y : *}$$ by (Π) $(\Box, *)$

- $$\frac{y : * \vdash \lambda x : y.x : \Pi x{:}y.y \quad \vdash \Pi y : *.\Pi x{:}y.y : *}{\vdash \lambda y : *.\lambda x : y.x : \Pi y : *.\Pi x{:}y.y}$$ by (λ)

The story so far of the evolution of functions and types

- Functions have gone through a long process of evolution involving various degrees of abstraction/construction/instantiation/concretisation/evaluation.
- Types too have gone through a long process of evolution involving various degrees of abstraction/construction/instantiation/concretisation/evaluation.
- During their progress, some aspects have been added or removed.
- The development of types and functions have been interlinked and their abstraction/construction/instantiation/concretisation/evaluation have much in common.
- We also argue that some of the aspects that have been dismissed during their evolution need to be re-incorporated.

From the point of vue of ML

- The language ML is not based on all of system F (2nd order polymorphic λ-calculus).
- This was not possible since it was not known then whether type checking and type finding are decidable.
- ML is based on a fragment of system F for which it was known that type checking and type finding are decidable.
- 23 years later after the design of ML, Wells showed that type checking and type finding in system F are undecidable.
- ML has polymorphism but not all the polymorphic power of system F.

- The question is, what system of functions and types does ML use?
- *A clean answer can be given when we re-incorporate the low-level function notion used by Frege and Russell (and de Bruijn) and dismissed by Church.*
- ML treats let val id $=$ (fn $x \Rightarrow x$) in (id id) end as this Cube term $(\lambda \mathsf{id}{:}(\Pi\alpha{:}*.\ \alpha \to \alpha).\ \mathsf{id}(\beta \to \beta)(\mathsf{id}\,\beta))(\lambda\alpha{:}*.\ \lambda x{:}\alpha.\ x)$

- To type this in the Cube, the $(\Box, *)$ rule is needed (i.e., $\lambda 2$).
- ML's typing rules forbid this expression:
  let val id $=$ (fn $x \Rightarrow x$) in (fn $y \Rightarrow y\,y$)(id id) end
  Its equivalent Cube term is this well-formed typable term of $\lambda 2$:
  $(\lambda \mathsf{id} : (\Pi\alpha{:}*.\ \alpha \to \alpha).$
  $\quad (\lambda y{:}(\Pi\alpha{:}*.\ \alpha \to \alpha).\ y(\beta \to \beta)(y\,\beta))$
  $\quad (\lambda\alpha{:}*.\ \mathsf{id}(\alpha \to \alpha)(\mathsf{id}\,\alpha)))$
  $(\lambda\alpha{:}*.\ \lambda x{:}\alpha.\ x)$
- Therefore, ML should not have the full Π-formation rule $(\Box, *)$.
- ML has limited access to the rule $(\Box, *)$.
- ML's type system is none of those of the eight systems of the Cube.
- [29] places the type system of ML (between $\lambda 2 + \lambda\underline{\omega}$).

LF

- LF [24] is often described as λP of the Barendregt Cube.
  However, Use of Π-formation rule $(*, \Box)$ is restricted in LF [22].
- We only need a type $\Pi x{:}A.B : \Box$ when PAT is applied during construction of the type $\Pi\alpha{:}\mathtt{prop}.*$ of the operator Prf where for a proposition Σ, Prf(Σ) is the type of proofs of Σ.

$$\frac{\mathtt{prop}{:}* \vdash \mathtt{prop}: * \qquad \mathtt{prop}{:}*, \alpha{:}\mathtt{prop} \vdash *{:}\Box}{\mathtt{prop}{:}* \vdash \Pi\alpha{:}\mathtt{prop}.* : \Box}.$$

- In LF, this is the only point where the Π-formation rule $(*, \Box)$ is used.
  But, Prf is only used when applied to $\Sigma{:}\mathtt{prop}$. We never use Prf on its own.
- This use is in fact based on a parametric constant rather than on Π-formation.
- Hence, the practical use of LF would not be restricted if we present Prf in a parametric form, and use $(*, \Box)$ as a parameter instead of a Π-formation rule.
- [29] precisely locate LF (between $\lambda{\to}$ and λP).

Parameters: What and Why

- We speak about *functions with parameters* when referring to functions with variable values in the *low-level* approach. The x in $f(x)$ is a parameter.

- Parameters enable the same expressive power as the high-level case, while allowing us to stay at a lower order. E.g. first-order with parameters versus second-order without [42].

- Desirable properties of the lower order theory (decidability, easiness of calculations, typability) can be maintained, without losing the flexibility of the higher-order aspects.

- This low-level approach is still worthwhile for many exact disciplines. In fact, both in logic and in computer science it has certainly not been wiped out, and for good reasons.

Automath

- The first tool for mechanical representation and verification of mathematical proofs, AUTOMATH, has a parameter mechanism.

- Mathematical text in AUTOMATH written as a finite list of *lines* of the form:

  $x_1 : A_1, \ldots, x_n : A_n \vdash g(x_1, \ldots, x_n) = t : T.$

  Here g is a new name, an abbreviation for the expression t of type T and $x_1, \ldots, x_n$ are the parameters of g, with respective types $A_1, \ldots, A_n$.

- Each line introduces a new definition which is inherently parametrised by the variables occurring in the context needed for it.

- Developments of ordinary mathematical theory in AUTOMATH [4] revealed that this combined definition and parameter mechanism is vital for keeping proofs manageable and sufficiently readable for humans.

Extending the Cube with parametric constants, see [29]

- We add parametric constants of the form $c(b_1, \ldots, b_n)$ with $b_1, \ldots, b_n$ terms of certain types and $c \in \mathcal{C}$.
- $b_1, \ldots, b_n$ are called the *parameters* of $c(b_1, \ldots, b_n)$.
- $\mathfrak{S}$ allows several kinds of Π-constructs. We also use a set $\boldsymbol{P}$ of (s_1, s_2) where $s_1, s_2 \in \{*, \Box\}$ to allow several kinds of parametric constants.
- $(s_1, s_2) \in \boldsymbol{P}$ means that we allow parametric constants $c(b_1, \ldots, b_n) : A$ where $b_1, \ldots, b_n$ have types $B_1, \ldots, B_n$ of sort s_1, and A is of type s_2.
- If both $(*, s_2) \in \boldsymbol{P}$ and $(\Box, s_2) \in \boldsymbol{P}$ then combinations of parameters allowed. For example, it is allowed that B_1 has type $*$, whilst B_2 has type $\Box$.

The Cube with parametric constants

- Let $(*,*) \subseteq \mathfrak{S}, \boldsymbol{P} \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$.

- $\lambda\mathfrak{S}\boldsymbol{P} = \lambda\mathfrak{S}$ and the two rules **($\vec{\mathbf{C}}$-weak)** and **($\vec{\mathbf{C}}$-app)**:

$$\frac{\Gamma \vdash b : B \quad \Gamma, \Delta_i \vdash B_i : s_i \quad \Gamma, \Delta \vdash A : s}{\Gamma, c(\Delta) : A \vdash b : B} \; (s_i, s) \in \boldsymbol{P}, c \text{ is } \Gamma\text{-fresh}$$

$$\frac{\begin{array}{lcll} \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash & b_i{:}B_i[x_j{:=}b_j]_{j=1}^{i-1} & (i = 1, \ldots, n) \\ \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash & A : s & (\text{if } n = 0) \end{array}}{\Gamma_1, c(\Delta){:}A, \Gamma_2 \vdash c(b_1, \ldots, b_n) : A[x_j{:=}b_j]_{j=1}^{n}}$$

$\Delta \equiv x_1{:}B_1, \ldots, x_n{:}B_n.$
$\Delta_i \equiv x_1{:}B_1, \ldots, x_{i-1}{:}B_{i-1}$

Properties of the Refined Cube

- (Correctness of types) If $\Gamma \vdash A : B$ then ($B \equiv \Box$ or $\Gamma \vdash B : S$ for some sort S).
- (Subject Reduction SR) If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_\beta A'$ then $\Gamma \vdash A' : B$
- (Strong Normalisation) For all $\vdash$-legal terms M, we have $\mathsf{SN}_{\twoheadrightarrow_\beta}(M)$.
- Other properties such as Uniqueness of types and typability of subterms hold.
- $\lambda\mathfrak{S}\boldsymbol{P}$ is the system which has Π-formation rules $\boldsymbol{R}$ and parameter rules $\boldsymbol{P}$.
- Let $\lambda\mathfrak{S}\boldsymbol{P}$ parametrically conservative (i.e., $(s_1, s_2) \in \boldsymbol{P}$ implies $(s_1, s_2) \in \boldsymbol{R}$).
  - The parameter-free system $\lambda\mathfrak{S}$ is at least as powerful as $\lambda\mathfrak{S}\boldsymbol{P}$.
  - If $\Gamma \vdash_{\mathfrak{S}\boldsymbol{P}} a : A$ then $\{\Gamma\} \vdash_{\boldsymbol{R}} \{a\} : \{A\}$.

Example

- $\boldsymbol{R} = \{(*,*),(*,\Box)\}$

  $\boldsymbol{P}_1 = \emptyset \qquad \boldsymbol{P}_2 = \{(*,*)\} \qquad \boldsymbol{P}_3 = \{(*,\Box)\} \qquad \boldsymbol{P}_4 = \{(*,*),(*,\Box)\}$

  All $\lambda\boldsymbol{R}\boldsymbol{P}_i$ for $1 \leq i \leq 4$ with the above specifications are all equal in power.

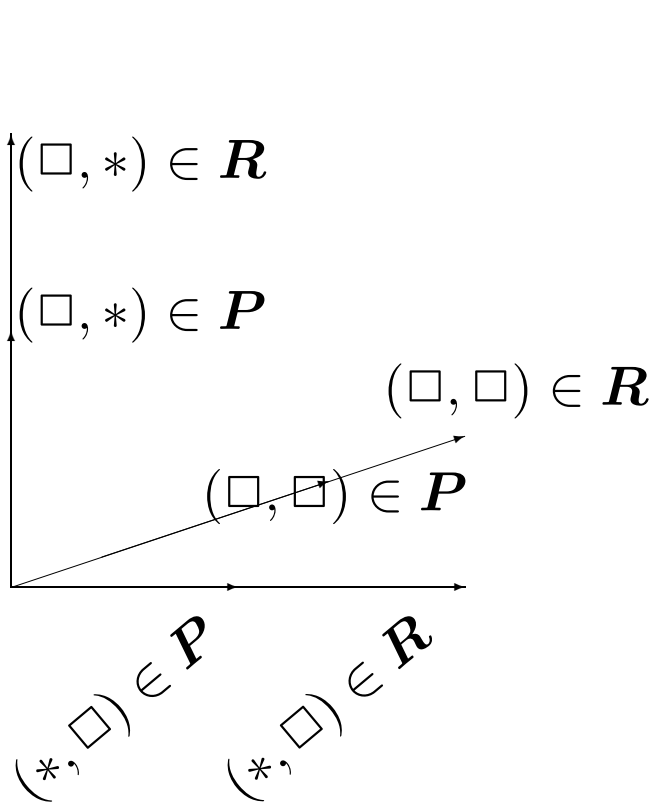
- $\boldsymbol{R}_5 = \{(*,*)\} \qquad \boldsymbol{P}_5 = \{(*,*),(*,\Box)\}.$

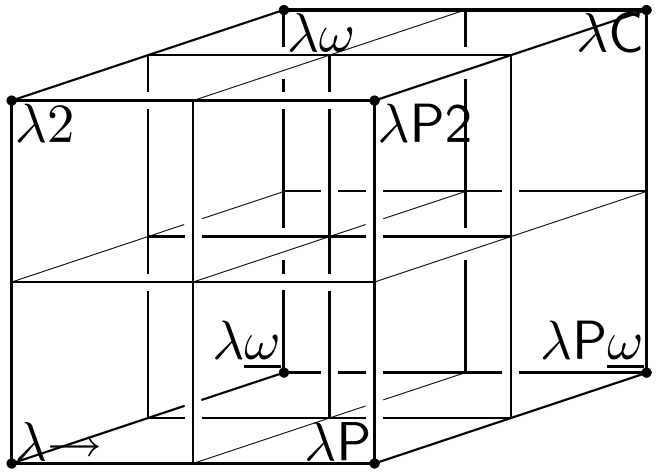
  $\lambda\!\!\rightarrow \;<\; \lambda\boldsymbol{R}_5\boldsymbol{P}_5 \;<\; \lambda$P: we can to talk about predicates:

$$\begin{array}{rcl}
\alpha & : & *, \\
\mathtt{eq}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha) & : & *, \\
\mathtt{refl}(\mathtt{x}{:}\alpha) & : & \mathtt{eq}(\mathtt{x},\mathtt{x}), \ . \\
\mathtt{symm}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x},\mathtt{y})) & : & \mathtt{eq}(\mathtt{y},\mathtt{x}), \\
\mathtt{trans}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{z}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x},\mathtt{y}), \mathtt{q}{:}\mathtt{eq}(\mathtt{y},\mathtt{z})) & : & \mathtt{eq}(\mathtt{x},\mathtt{z})
\end{array}$$

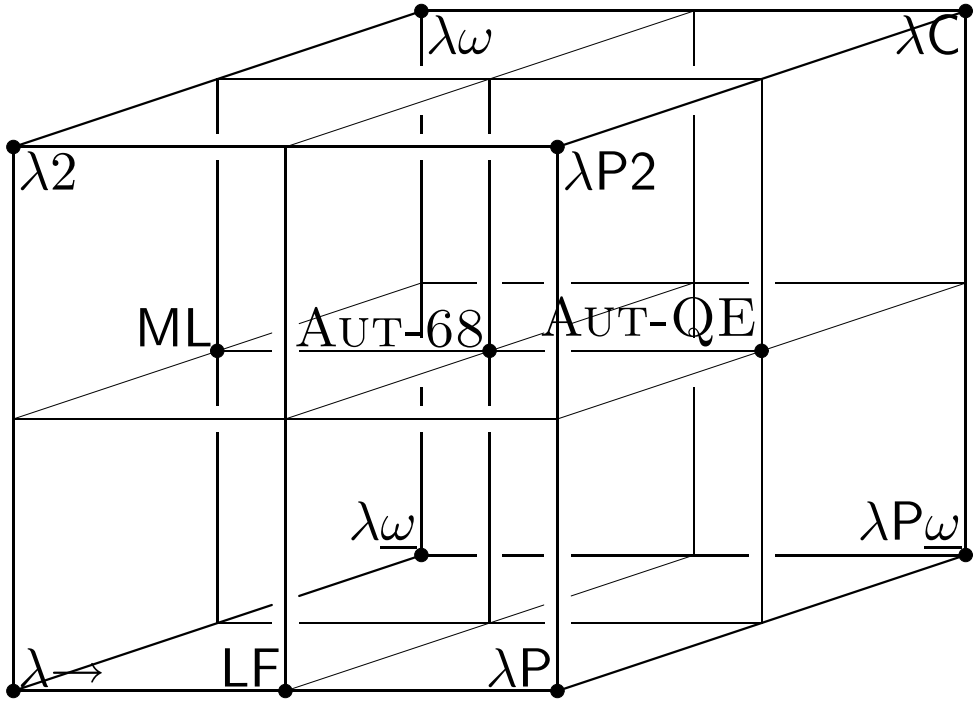
  eq not possible in $\lambda\!\!\rightarrow$.

The refined Barendregt Cube





LF, ML, AUT-68, and AUT-QE in the refined Cube



Logicians versus mathematicians and induction over numbers

- Logician uses `ind: Ind` as proof term for an application of the induction axiom. The type `Ind` can only be described in $\lambda\boldsymbol{R}$ where $\boldsymbol{R} = \{(*,*),(*,\Box),(\Box,*)\}$:

$$\texttt{Ind} = \Pi p{:}(\mathbb{N}\to *).p0\to(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn\to Snm\to pm)\to\Pi n{:}\mathbb{N}.pn \quad (1)$$

- Mathematician uses `ind` only with $P : \mathbb{N}\to *$, $Q : P0$ and $R : (\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.Pn\to Snm\to Pm)$ to form a term $(\texttt{ind}PQR){:}(\Pi n{:}\mathbb{N}.Pn)$.

- The use of the induction axiom by the mathematician is better described by the parametric scheme (p, q and r are the *parameters* of the scheme):

$$\texttt{ind}(p{:}\mathbb{N}\to *, q{:}p0, r{:}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn\to Snm\to pm)) : \Pi n{:}\mathbb{N}.pn \quad (2)$$

- The logician's type `Ind` is not needed by the mathematician and the types that occur in 2 can all be constructed in $\lambda\boldsymbol{R}$ with $\boldsymbol{R} = \{(*,*)(*,\Box)\}$.

Logicians versus mathematicians and induction over numbers

- Mathematician: only *applies* the induction axiom and doesn't need to know the proof-theoretical backgrounds.

- A logician develops the induction axiom (or studies its properties).

- $(\Box, *)$ is not needed by the mathematician. It is needed in logician's approach in order to form the Π-abstraction $\Pi p{:}(\mathbb{N} \to *).\cdots)$.

- Consequently, the type system that is used to describe the mathematician's use of the induction axiom can be weaker than the one for the logician.

- Nevertheless, the parameter mechanism gives the mathematician limited (but for his purposes sufficient) access to the induction scheme.

- Parameters enable the same expressive power as the high-level case, while allowing us to stay at a lower order. E.g. first-order with parameters versus second-order without [42].

- Desirable properties of the lower order theory (decidability, easiness of calculations, typability) can be maintained, without losing the flexibility of the higher-order aspects.

- Parameters enable us to find an exact position of type systems in the generalised framework of type systems.

- Parameters describe the difference between *developers* and *users* of systems.

Identifying λ and Π (see [33])

- In the cube of the generalised framework of type systems, we saw that the syntax for terms (functions) and types was intermixed with the only distinction being λ- versus Π-abstraction.

- We unify the two abstractions into one.
  $\mathcal{T}_\flat ::= \mathcal{V} \mid \boldsymbol{S} \mid \mathcal{T}_\flat \mathcal{T}_\flat \mid \flat \mathcal{V}{:}\mathcal{T}_\flat.\mathcal{T}_\flat$

- $\mathcal{V}$ is a set of variables and $\boldsymbol{S} = \{*, \Box\}$.

- The β-reduction rule becomes $(\flat)$ $\qquad (\flat_{x:A}.B)C \to_\flat B[x := C]$.

- Now we also have the old Π-reduction $(\Pi_{x:A}.B)C \to_\Pi B[x := C]$ which treats type instantiation like function instantiation.

- The type formation rule becomes

$$(\flat_1) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash (\flat x{:}A.B) : s_2} \; (s_1, s_2) \in \boldsymbol{R}$$

(axiom) $$\langle\rangle \vdash * : \Box$$

(start) $$\frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}\ x \notin \mathrm{DOM}\,(\Gamma)$$

(weak) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x{:}C \vdash A : B}\ x \notin \mathrm{DOM}\,(\Gamma)$$

($\flat_2$) $$\frac{\Gamma, x{:}A \vdash b : B \quad \Gamma \vdash (\flat x{:}A.B) : s}{\Gamma \vdash (\flat x{:}A.b) : (\flat x{:}A.B)}$$

(app$\flat$) $$\frac{\Gamma \vdash F : (\flat x{:}A.B) \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$$

(conv) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}$$

Translations between the systems with 2 binders and those with one binder

- For $A \in \mathcal{T}$, we define $\overline{A} \in \mathcal{T}_\flat$ as follows:
  - $\overline{s} \equiv s \qquad \overline{x} \equiv x \qquad \overline{AB} \equiv \overline{A}\,\overline{B}$
  - $\overline{\lambda_{x:A}.B} \equiv \overline{\Pi_{x:A}.B} \equiv \flat_{x:\overline{A}}.\overline{B}$.
- For contexts we define: $\overline{\langle\rangle} \equiv \langle\rangle \quad \overline{\Gamma, x:A} \equiv \overline{\Gamma}, x:\overline{A}$.
- For $A \in \mathcal{T}_\flat$, we define $[A]$ to be $\{A' \in \mathcal{T}$ such that $\overline{A'} \equiv A\}$.
- For context, obviously: $[\Gamma] \equiv \{\Gamma'$ such that $\overline{\Gamma'} \equiv \Gamma\}$.

Isomorphism of the cube and the $\flat$-cube

- If $\Gamma \vdash A : B$ then $\overline{\Gamma} \vdash_\flat \overline{A} : \overline{B}$.
- If $\Gamma \vdash_\flat A : B$ then there are unique $\Gamma' \in [\Gamma]$, $A' \in [A]$ and $B' \in [B]$ such that $\Gamma' \vdash_\pi A' : B'$.
- The $\flat$-cube enjoys all the properties of the cube except the unicity of types.

Organised multiplicity of Types for $\vdash_\flat$ and $\to_\flat$ [33]

For many type systems, unicity of types is not necessary (e.g. Nuprl).

We have however an organised multiplicity of types.

1. If $\Gamma \vdash_\flat A : B_1$ and $\Gamma \vdash_\flat A : B_2$, then $B_1 \stackrel{\diamond}{=}_\flat B_2$.

2. If $\Gamma \vdash_\flat A_1 : B_1$ and $\Gamma \vdash_\flat A_2 : B_2$ and $A_1 =_\flat A_2$, then $B_1 \stackrel{\diamond}{=}_\flat B_2$.

3. If $\Gamma \vdash_\flat B_1 : s_1$, $B_1 =_\flat B_2$ and $\Gamma \vdash_\flat A : B_2$ then $\Gamma \vdash_\flat B_2 : s_1$.

4. Assume $\Gamma \vdash_\flat A : B_1$ and $(\Gamma \vdash_\flat A : B_1)^{-1} = (\Gamma', A', B_1')$. Then $B_1 =_\flat B_2$ if:

   (a) either $\Gamma \vdash_\flat A : B_2$, $(\Gamma \vdash_\flat A : B_2)^{-1} = (\Gamma', A'', B_2')$ and $B_1' =_\beta B_2'$,
   (b) or $\Gamma \vdash_\flat C : B_2$, $(\Gamma \vdash_\flat C : B_2)^{-1} = (\Gamma', C', B_2')$ and $A' =_\beta C'$.

Extending the cube with Π-reduction loses subject reduction [34]

If we change (appl) by (new appl) in the cube we lose subject reduction.

$$\text{(appl)}\quad \frac{\Gamma \vdash F : (\Pi_{x:A}.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]}$$

$$\text{(new appl)}\quad \frac{\Gamma \vdash F : (\Pi_{x:A}.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}.B)a}$$

[34] solved the problem by re-incorporating Frege and Russell's notions of low level functions (which was lost in Church's notion of function).

The same problem and solution can be repeated in our $\flat$-cube.

Adding type instantiation to the typing rules of the $\flat$-cube

If we change (app$\flat$) by (new app$\flat$) in the $\flat$-cube we lose subject reduction.

$$(\text{app}\flat)\quad \frac{\Gamma \vdash_\flat F : (\Pi_{x:A}.B) \qquad \Gamma \vdash_\flat a : A}{\Gamma \vdash_\flat Fa : B[x := a]}$$

$$(\text{app}\flat\flat)\quad \frac{\Gamma \vdash_\flat F : (\flat_{x:A}.B) \qquad \Gamma \vdash_\flat a : A}{\Gamma \vdash_\flat Fa : (\flat_{x:A}.B)a}$$

Failure of correctness of types and subject reduction

- **Correctness of types no longer holds.** With (applbb) one can have $\Gamma \vdash A : B$ without $B \equiv \Box$ or $\exists S \,.\, \Gamma \vdash B : S$.

- For example, $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ yet $(\flat_{y:z}.z)x \not\equiv \Box$ and $\forall s \,.\, z : *, x : z \not\vdash (\flat_{y:z}.z)x : s$.

- **Subject Reduction no longer holds.** That is, with (applb): $\Gamma \vdash A : B$ and $A \twoheadrightarrow A'$ may not imply $\Gamma \vdash A' : B$.

- For example, $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and $(\flat_{y:z}.y)x \to_\flat x$, but one can't show $z : *, x : z \vdash x : (\flat_{y:z}.z)x$.

Solving the problem

Keep all the typing rules of the $\flat$-cube the same except: replace (conv) by (new-conv), (appl$\flat$) by (appl$\flat\flat$) and add three new rules as follows:

(start-def) $$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B{:}A \vdash x : A} \qquad x \notin \mathrm{DOM}(\Gamma)$$

(weak-def) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D{:}C \vdash A : B} \qquad x \notin \mathrm{DOM}(\Gamma)$$

(def) $$\frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\flat x{:}A.C)B : D[x := B]}$$

(new-conv) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad \Gamma \vdash B =_{def} B'}{\Gamma \vdash A : B'}$$

(appl$\flat\flat$) $$\frac{\Gamma \vdash F : \flat_{x:A}.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\flat_{x:A}.B)a}$$

In the conversion rule, $\Gamma \vdash B =_{def} B'$ is defined as:

- If $B =_\flat B'$ then $\Gamma \vdash B =_{def} B'$
- If $x = D : C \in \Gamma$ and B' arises from B by substituting one particular free occurrence of x in B by D then $\Gamma \vdash B =_{def} B'$.
- Our 3 new rules and the definition of $\Gamma \vdash B =_{def} B'$ are trying to re-incorporate low-level aspects of functions that are not present in Church's λ-calculus.
- *In fact, our new framework is closer to Frege's abstraction principle and the principles $*9{\cdot}14$ and $*9{\cdot}15$ of [55].*

Correctness of types holds.

- We demonstrate this with the earlier example.
- Recall that we have $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and want that for some s, $z : *, x : z \vdash (\flat_{y:z}.z)x : s$.
- Here is how the latter formula now holds:

$$
\begin{array}{ll}
z : *, x : z \vdash z : * & \text{(start and weakening)} \\
z : *, x : z.y : z\rangle x \vdash z : * & \text{(weakening)} \\
z : *, x : z \vdash (\flat_{y:z}.z)x : *[y := x] \equiv * & \text{(def rule)}
\end{array}
$$

Subject Reduction holds.

- We demonstrate this with the earlier example.
- Recall that we have $z : *, x : z \vdash (\flat_{y:z}.y)x : (\flat_{y:z}.z)x$ and $(\lambda_{y:z}.y)x \rightarrow_\beta x$ and we need to show that $z : *, x : z \vdash x : (\flat_{y:z}.z)x$.
- Here is how the latter formula now holds:

$a.$	$z : *, x : z \vdash x : z$	(start and weakening)
$b.$	$z : *, x : z \vdash (\flat_{y:z}.z)x : *$	(from 1 above)
	$z : *, x : z \vdash x : (\flat_{y:z}.z)x$	(conversion, a, b, and $z =_\beta (\flat_{y:z}.z)x$)

De Bruijn's typed λ-calculi started with his Automath

- In 1967, an internationally renowned mathematician called N.G. de Bruijn wanted to do something never done before: use the computer to formally check the correctness of mathematical books.
- Such a task needs a good formalisation of mathematics, a good competence in implementation, and extreme attention to all the details so that nothing is left informal.
- Implementing extensive formal systems on the computer was never done before.
- De Bruijn, an extremely original mathematician, did every step his own way.
- He proudly announced at the ceremony of the publications of the collected Automath work: *I did it my way*.
- Dirk van Dalen said at the ceremony: *The Germans have their 3 B's, but we Dutch too have our 3 B's: Beth, Brouwer and de Bruijn*.

There is a fourth B:

They look good together

Theme 1: De Bruijn Indices and Explicit Substitutions [14]

- Classical λ-calculus: $A ::= x \mid (\lambda x.B) \mid (BC)$
  $(\lambda x.A)B \to_\beta A[x := B]$
- $(\lambda x.\lambda y.xy)y \to_\beta (\lambda y.xy)[x := y] \neq \lambda y.yy$
- $(\lambda x.\lambda y.xy)y \to_\beta (\lambda y.xy)[x := y] =_\alpha (\lambda z.xz)[x := y] = \lambda z.yz$
- $\lambda x.x$ and $\lambda y.y$ are the same function. Write this function as $\lambda 1$.

- Assume a free variable list (say $x, y, z, \ldots$).

- $(\lambda\lambda 2\ 1)2 \to_\beta (\lambda 2\ 1)[1 := 2] = \lambda(2[2 := 3])(1[2 := 3]) = \lambda 3\ 1$

Classical λ-calculus with de Bruijn indices

- Let $i, n \geq 1$ and $k \geq 0$
- $A \ ::= \ n \,|\, (\lambda B) \,|\, (BC)$
  $(\lambda A)B \rightarrow_\beta A\{\!\{1 \leftarrow B\}\!\}$
- $$\begin{array}{l} U_k^i(AB) = U_k^i(A)\, U_k^i(B) \\ U_k^i(\lambda A) = \lambda(U_{k+1}^i(A)) \end{array} \qquad U_k^i(\mathtt{n}) = \begin{cases} \mathtt{n} + \mathtt{i} - \mathtt{1} & \text{if} \quad n > k \\ \mathtt{n} & \text{if} \quad n \leq k\,. \end{cases}$$
- $$\begin{array}{lcl} (A_1 A_2)\{\!\{\mathtt{i} \leftarrow B\}\!\} & = & (A_1\{\!\{\mathtt{i} \leftarrow B\}\!\})\,(A_2\{\!\{\mathtt{i} \leftarrow B\}\!\}) \\ (\lambda A)\{\!\{\mathtt{i} \leftarrow B\}\!\} & = & \lambda(A\{\!\{\mathtt{i}+\mathtt{1} \leftarrow B\}\!\}) \end{array}$$
  $$\mathtt{n}\{\!\{\mathtt{i} \leftarrow B\}\!\} = \begin{cases} \mathtt{n} - \mathtt{1} & \text{if} \quad n > i \\ U_0^i(B) & \text{if} \quad n = i \\ \mathtt{n} & \text{if} \quad n < i\,. \end{cases}$$
- Numerous implementations of proof checkers and programming languages have been based on de Bruijn indices.

From classical λ-calculus with de Bruijn indices to substitution calculus λs [31]

- Write $A\{\!\{n \leftarrow B\}\!\}$ as $A\sigma^n\, B$ and $U_k^i(A)$ as $\varphi_k^i A$.
- $A \;::=\; n \mid (\lambda B) \mid (BC) \mid (A\sigma^i B) \mid (\varphi_k^i B) \qquad where \quad i, n \geq 1\,, \quad k \geq 0\,.$

σ-*generation*	$(\lambda A)\, B$	$\longrightarrow$	$A\,\sigma^1\, B$
σ-λ-*transition*	$(\lambda A)\,\sigma^i B$	$\longrightarrow$	$\lambda(A\,\sigma^{i+1}\, B)$
σ-*app-transition*	$(A_1\, A_2)\,\sigma^i B$	$\longrightarrow$	$(A_1\,\sigma^i B)\,(A_2\,\sigma^i B)$
σ-*destruction*	$\mathtt{n}\,\sigma^i B$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \text{if} \quad n > i \\ \varphi_0^i\, B & \text{if} \quad n = i \\ \mathtt{n} & \text{if} \quad n < i \end{cases}$
φ-λ-*transition*	$\varphi_k^i(\lambda A)$	$\longrightarrow$	$\lambda(\varphi_{k+1}^i\, A)$
φ-*app-transition*	$\varphi_k^i(A_1\, A_2)$	$\longrightarrow$	$(\varphi_k^i\, A_1)\,(\varphi_k^i\, A_2)$
φ-*destruction*	$\varphi_k^i\,\mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n+i-1} & \text{if} \quad n > k \\ \mathtt{n} & \text{if} \quad n \leq k \end{cases}$

1. The s-calculus (i.e., λs minus σ-generation) is strongly normalising,

2. The λs-calculus is confluent and simulates (in small steps) β-reduction

3. The λs-calculus preserves strong normalisation PSN.

4. The λs-calculus has a confluent extension with open terms λse.

- The λs-calculus was the first calculus of substitutions which satisfies all the above properties 1., 2., 3. and 4.

$\lambda\upsilon$ [3]

Terms: $\Lambda\upsilon^t ::= \mathsf{IN} \mid \Lambda\upsilon^t\Lambda\upsilon^t \mid \lambda\Lambda\upsilon^t \mid \Lambda\upsilon^t[\Lambda\upsilon^s]$
Substitutions: $\Lambda\upsilon^s ::= \uparrow \mid \Uparrow(\Lambda\upsilon^s) \mid \Lambda\upsilon^t.$

(Beta)	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b/]$
(App)	$(a\, b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a\,[\Uparrow(s)])$
(FVar)	$\mathtt{1}\,[a/]$	$\longrightarrow$	a
(RVar)	$\mathtt{n}+\mathtt{1}\,[a/]$	$\longrightarrow$	$\mathtt{n}$
(FVarLift)	$\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{1}$
(RVarLift)	$\mathtt{n}+\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{n}\,[s]\,[\uparrow]$
(VarShift)	$\mathtt{n}\,[\uparrow]$	$\longrightarrow$	$\mathtt{n}+\mathtt{1}$

$\lambda\upsilon$ satisfies 1., 2., and 3., but does not have a confluent extension on open terms.

$\lambda\sigma_{\Uparrow}$

Terms: $\Lambda\sigma^t_{\Uparrow} ::= \mathsf{IN} \mid \Lambda\sigma^t_{\Uparrow}\Lambda\sigma^t_{\Uparrow} \mid \lambda\Lambda\sigma^t_{\Uparrow} \mid \Lambda\sigma^t_{\Uparrow}[\Lambda\sigma^s_{\Uparrow}]$

Substitutions: $\Lambda\sigma^s_{\Uparrow} ::= id \mid \uparrow \mid \Uparrow(\Lambda\sigma^s_{\Uparrow}) \mid \Lambda\sigma^t_{\Uparrow} \cdot \Lambda\sigma^s_{\Uparrow} \mid \Lambda\sigma^s_{\Uparrow} \circ \Lambda\sigma^s_{\Uparrow}.$

(Beta)	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b \cdot id]$
(App)	$(a\, b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a\,[\Uparrow(s)])$
(Clos)	$(a\,[s])[t]$	$\longrightarrow$	$a\,[s \circ t]$
(Varshift1)	$\mathtt{n}\,[\uparrow]$	$\longrightarrow$	$\mathtt{n+1}$
(Varshift2)	$\mathtt{n}\,[\uparrow \circ s]$	$\longrightarrow$	$\mathtt{n+1}\,[s]$
(FVarCons)	$\mathtt{1}\,[a \cdot s]$	$\longrightarrow$	a
(RVarCons)	$\mathtt{n+1}\,[a \cdot s]$	$\longrightarrow$	$\mathtt{n}\,[s]$
(FVarLift1)	$\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{1}$
(FVarLift2)	$\mathtt{1}\,[\Uparrow(s) \circ t]$	$\longrightarrow$	$\mathtt{1}\,[t]$
(RVarLift1)	$\mathtt{n+1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{n}[s \circ \uparrow]$
(RVarLift2)	$\mathtt{n+1}\,[\Uparrow(s) \circ t]$	$\longrightarrow$	$\mathtt{n}[s \circ (\uparrow \circ t)]$

$\lambda\sigma_{\Uparrow}$ rules continued

(Map)	$(a \cdot s) \circ t$	$\longrightarrow$	$a\,[t] \cdot (s \circ t)$
(Ass)	$(s \circ t) \circ u$	$\longrightarrow$	$s \circ (t \circ u)$
(ShiftCons)	$\uparrow \circ (a \cdot s)$	$\longrightarrow$	s
(ShiftLift1)	$\uparrow \circ \Uparrow(s)$	$\longrightarrow$	$s \circ \uparrow$
(ShiftLift2)	$\uparrow \circ (\Uparrow(s) \circ t)$	$\longrightarrow$	$s \circ (\uparrow \circ t)$
(Lift1)	$\Uparrow(s) \circ \Uparrow(t)$	$\longrightarrow$	$\Uparrow(s \circ t)$
(Lift2)	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\longrightarrow$	$\Uparrow(s \circ t) \circ u$
(LiftEnv)	$\Uparrow(s) \circ (a \cdot t)$	$\longrightarrow$	$a \cdot (s \circ t)$
(IdL)	$id \circ s$	$\longrightarrow$	s
(IdR)	$s \circ id$	$\longrightarrow$	s
(LiftId)	$\Uparrow(id)$	$\longrightarrow$	id
(Id)	$a\,[id]$	$\longrightarrow$	a

$\lambda\sigma_{\Uparrow}$ satisfies 1., 2., and 4., but does not have PSN.

How is λse obtained from λs?

- They said, we can have *open terms (holes in proofs)* in $\lambda\sigma$, can you do so in λs?
- $A ::= X \mid n \mid (\lambda B) \mid (BC) \mid (A\sigma^i B) \mid (\varphi^i_k B) \quad where \quad i, n \geq 1\,, \quad k \geq 0\,.$
- Extending the syntax of λs with open terms without extending the λs-rules loses the confluence (even local confluence):
  $((\lambda X)Y)\sigma^1 1 \to (X\sigma^1 Y)\sigma^1 1 \qquad ((\lambda X)Y)\sigma^1 1 \to ((\lambda X)\sigma^1 1)(Y\sigma^1 1)$
- $(X\sigma^1 Y)\sigma^1 1$ and $((\lambda X)\sigma^1 1)(Y\sigma^1 1)$ have no common reduct.
- But, $((\lambda X)\sigma^1 1)(Y\sigma^1 1) \twoheadrightarrow (X\sigma^2 1)\sigma^1(Y\sigma^1 1)$
- Simple: add de Bruijn's metasubstitution and distribution lemmas to the rules of λs:

σ-σ	$(A\sigma^i B)\,\sigma^j\,C$	$\longrightarrow$	$(A\,\sigma^{j+1}\,C)\,\sigma^i\,(B\,\sigma^{j-i+1}\,C)$	if	$i \leq j$
σ-φ *1*	$(\varphi^i_k\,A)\,\sigma^j\,B$	$\longrightarrow$	$\varphi^{i-1}_k\,A$	if	$k < j < k+i$
σ-φ *2*	$(\varphi^i_k\,A)\,\sigma^j\,B$	$\longrightarrow$	$\varphi^i_k(A\,\sigma^{j-i+1}\,B)$	if	$k+i \leq j$
φ-σ	$\varphi^i_k(A\,\sigma^j\,B)$	$\longrightarrow$	$(\varphi^i_{k+1}\,A)\,\sigma^j\,(\varphi^i_{k+1-j}\,B)$	if	$j \leq k+1$
φ-φ *1*	$\varphi^i_k\,(\varphi^j_l\,A)$	$\longrightarrow$	$\varphi^j_l\,(\varphi^i_{k+1-j}\,A)$	if	$l+j \leq k$
φ-φ *2*	$\varphi^i_k\,(\varphi^j_l\,A)$	$\longrightarrow$	$\varphi^{j+i-1}_l\,A$	if	$l \leq k < l+j$

- These extra rules are the rewriting of the well-known meta-substitution ($\sigma - \sigma$) and distribution ($\varphi - \sigma$) lemmas (and the 4 extra lemmas needed to prove them).

- ($\sigma - \sigma$):
  $A[x := B][y := C] = A[y := C][x := B[y := C]]$ if $x \neq y$ and $x \notin FV(C)$.

- ($\varphi - \sigma$):
  $updatedA[x := B] = updatedA[x := updatedB]$.

Where did the extra rules come from?

In de Bruijn's classical λ-calculus we have the lemmas:

$(\sigma-\varphi\ 1)$ For $k<j<k+i$ we have: $U_k^{i-1}(A)=U_k^i(A)\{\!\{\mathtt{j}\leftarrow B\}\!\}$.

$(\varphi-\varphi\ 2)$ For $l\leq k<l+j$ we have: $U_k^i(U_l^j(A))=U_l^{j+i-1}(A)$.

$(\sigma-\varphi\ 2)$ For $k+i\leq j$ we have: $U_k^i(A)\{\!\{\mathtt{j}\leftarrow B\}\!\}=U_k^i(A\{\!\{\mathtt{j}-\mathtt{i}+\mathtt{1}\leftarrow B\}\!\})$.

$(\sigma-\sigma)$ *[Meta-substitution lemma]* For $i\leq j$ we have:
$A\{\!\{\mathtt{i}\leftarrow B\}\!\}\{\!\{\mathtt{j}\leftarrow C\}\!\}=A\{\!\{\mathtt{j}+\mathtt{1}\leftarrow C\}\!\}\{\!\{\mathtt{i}\leftarrow B\{\!\{\mathtt{j}-\mathtt{i}+\mathtt{1}\leftarrow C\}\!\}\}\!\}$.

- The proof of $(\sigma-\sigma)$ uses $(\sigma-\varphi\ 1)$ and $(\sigma-\varphi\ 2)$ both with $k=0$.
- The proof of $(\sigma-\varphi\ 2)$ requires $(\varphi-\varphi\ 2)$ with $l=0$.

Where did the extra rules come from (continued)?

In de Bruijn's classical λ-calculus we also have the lemmas:

$(\varphi - \varphi\ 1)$ For $j \leq k+1$ we have: $U^i_{k+p}(U^j_p(A)) = U^j_p(U^i_{k+p+1-j}(A))$.

$(\varphi - \sigma)$ *[Distribution lemma]*
For $j \leq k+1$ we have: $U^i_k(A\{\!\{\mathtt{j} \leftarrow B\}\!\}) = U^i_{k+1}(A)\{\!\{\mathtt{j} \leftarrow U^i_{k+1-j}(B)\}\!\}$.

- $(\varphi - \varphi\ 1)$ with $p = 0$ is needed to prove $(\varphi - \sigma)$.

Theme 2: Lambda Calculus à la de Bruijn

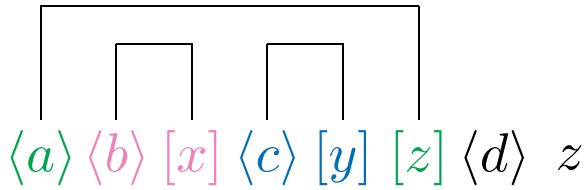
- $\mathcal{I}(x) = x, \qquad \mathcal{I}(\lambda x.B) = [x]\mathcal{I}(B), \qquad \mathcal{I}(AB) = \langle \mathcal{I}(B) \rangle \mathcal{I}(A)$
- $(\lambda x.\lambda y.xy)z$ translates to $\langle z \rangle [x][y]\langle y \rangle x$.
- The *applicator wagon* $\langle z \rangle$ and *abstractor wagon* $[x]$ occur NEXT to each other.
- $(\lambda x.A)B \rightarrow_\beta A[x := B]$ becomes

$$\langle B \rangle [x] A \rightarrow_\beta [x := B]A$$

- The "bracketing structure" of $((\lambda_x.(\lambda_y.\lambda_z.\ -\ -)c)b)a)$, is '$[_1\ [_2\ [_3\ ]_2\ ]_1\ ]_3$', where '$[_i$' and '$]_i$' match.
- The bracketing structure of $\langle a \rangle \langle b \rangle [x] \langle c \rangle [y][z] \langle d \rangle$ is simpler: $[[\,][\,]]$.
- $\langle b \rangle [x]$ and $\langle c \rangle [y]$ are AT-pairs whereas $\langle a \rangle [z]$ is an AT-couple.

Redexes in de Bruijn's notation

Classical Notation	de Bruijn's Notation
$\underline{((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a}$	$\langle a\rangle\underline{\langle b\rangle[x]}\langle c\rangle[y][z]\langle d\rangle z$
$\downarrow_\beta$	$\downarrow_\beta$
$\underline{((\lambda_y.\lambda_z.zd)c)a}$	$\langle a\rangle\underline{\langle c\rangle[y]}[z]\langle d\rangle z$
$\downarrow_\beta$	$\downarrow_\beta$
$\underline{(\lambda_z.zd)a}$	$\underline{\langle a\rangle[z]}\langle d\rangle z$
$\downarrow_\beta$	$\downarrow_\beta$
ad	$\langle d\rangle a$



- This maks it easy to introduce local/global/mini reductions into the λ-calculus [7].
- Further study of de Bruijn's notation can be found in [36, 37]

Some notions of reduction studied in the literature

Name	In Classical Notation	In de Bruijn's notation
(θ)	$((\lambda_x.N)P)Q$ $\downarrow$ $(\lambda_x.NQ)P$	$\langle Q\rangle\langle P\rangle[x]N$ $\downarrow$ $\langle P\rangle[x]\langle Q\rangle N$
(γ)	$(\lambda_x.\lambda_y.N)P$ $\downarrow$ $\lambda_y.(\lambda_x.N)P$	$\langle P\rangle[x][y]N$ $\downarrow$ $[y]\langle P\rangle[x]N$
(γ_C)	$((\lambda_x.\lambda_y.N)P)Q$ $\downarrow$ $(\lambda_y.(\lambda_x.N)P)Q$	$\langle Q\rangle\langle P\rangle[x][y]N$ $\downarrow$ $\langle Q\rangle[y]\langle P\rangle[x]N$
(g)	$((\lambda_x.\lambda_y.N)P)Q$ $\downarrow$ $(\lambda_x.N[y := Q])P$	$\langle Q\rangle\langle P\rangle[x][y]N$ $\downarrow$ $\langle P\rangle[x][y := Q]N$
(β_e)	? $\downarrow$?	$\langle Q\rangle\overline{s}[y]N$ $\downarrow$ $\overline{s}[y := Q]N$

A Few Uses of these reductions/term reshuffling

- [48] uses θ and γ in analyzing perpetual reduction strategies.
- Term reshuffling is used in [41, 39] in analyzing typability problems.
- [44, 15, 40, 32] use generalised reduction and/or term reshuffling in relating SN to WN.
- [1] uses a form of term-reshuffling in obtaining a calculus that corresponds to lazy functional evaluation.
- [36, 28, 38, 5] shows that they could reduce space/time needs.
- All these works have been heavily influenced by de Bruijn's Automath whose λ-notation facilitated the manipulation of redexes.
- All can be represented clearer in de Bruijn's notation.

An impressive thinker and entertainer

Even more: de Bruijn's generalised reduction has better properties

(β)	$(\lambda_x.M)N \to M[x := N]$	
(β_I)	$(\lambda_x.M)N \to M[x := N]$	if $x \in FV(M)$
(β_K)	$(\lambda_x.M)N \to M$	if $x \notin FV(M)$
(θ)	$(\lambda_x.N)PQ \to (\lambda_x.NQ)P$	
(β_e)	$(M)\overline{s}[x]N \to \overline{s}\{N[x := M]$	for $\overline{s}$ well-balanced.

- [32] shows that β_e satisfies PSN, postponment of K-contraction and conservation (latter 2 properties fail for β-reduction).

- Conservation of β_e: If A is $\beta_e I$-normalisable then A is β_e-strongly normalisable.

- Postponment of K-contraction : Hence, discard arguments of K-redexes after I-reduction. This gives flexibility in implementation: unnecessary work can be delayed, or even completely avoided.

- Attempts have been made at establishing some reduction relations for which postponement of K-contractions and conservation hold.

- The picture is as follows (-N stands for normalising and $r \in \{\beta_I, \theta_K\}$).

(β_K-postponement for r)	If $M \to_{\beta_K} N \to_r O$ then $\exists P$ such that $M \to\!\!\to^{+}_{\beta_I\theta_K} P \to\!\!\to_{\beta_K}$	
(Conservation for β_I)	If M is β_I-N then M is β_I-SN	Barendregt's book
(Conservation for $\beta + \theta$)	If M is $\beta_I\theta_K$-N then M is β-SN	[15]

- De Groote does not produce these results for a single reduction relation, but for $\beta + \theta$ (this is more restrictive than β_e).

- β_e is the first single relation to satisfy β_K-postponement and conservation.

- [32] shows that:

(β_{eK}-postponement for β_e)	If $M \to_{\beta_{eK}} N \to_{\beta_{eI}} O$ then $\exists P$ such that $M \to_{\beta_{eI}} P \to\!\!\to^{+}_{\beta_e}$
(Conservation for β_e)	If M is β_{eI}-N then M is β_e-SN

Here is de Bruijn at a lecture

And here is Henk listening to de Bruijn's talk

This is de Bruijn at 9:15 am lecture my students at a short notice

References

[1] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. In *Conf. Rec. 22nd Ann. ACM Symp. Princ. of Prog. Langs.*, pages 233–246, 1995.

[2] H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics 103. North-Holland, Amsterdam, revised edition, 1984.

[3] Z.E.A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.

[4] L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the Automath system*. PhD thesis, Eindhoven University of Technology, 1977.

Published as Mathematical Centre Tracts nr. 83 (Amsterdam, Mathematisch Centrum, 1979).

[5] Roel Bloo, Fairouz Kamareddine, and Rob Nederpelt. The Barendregt cube with definitions and generalised reduction. *Inform. & Comput.*, 126(2):123–143, May 1996.

[6] N.G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In M. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, pages 29–61, IRIA, Versailles, 1968. Springer Verlag, Berlin, 1970. Lecture Notes in Mathematics **125**; also in [45], pages 73–100.

[7] N.G. de Bruijn. Generalising automath by means of a lambda-typed lambda calculus. 1984. Also in [45], pages 313–337.

[8] C. Burali-Forti. Una questione sui numeri transfiniti. *Rendiconti del Circolo Matematico di Palermo*, 11:154–164, 1897. English translation in [25], pages 104–112.

[9] G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (Erster Artikel). *Mathematische Annalen*, 46:481–512, 1895.

[10] G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (Zweiter Artikel). *Mathematische Annalen*, 49:207–246, 1897.

[11] A.-L. Cauchy. *Cours d'Analyse de l'Ecole Royale Polytechnique*. Debure, Paris, 1821. Also as *Œuvres Complètes* (2), volume III, Gauthier-Villars, Paris, 1897.

[12] A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

[13] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[14] N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the chuirch rosser theorem. In *Indagationes Math*, pages 381–392. 1972. Also in [45].

[15] Philippe de Groote. The conservation theorem revisited. In *Proc. Int'l Conf. Typed Lambda Calculi and Applications*, pages 163–178. Springer, 1993.

[16] R. Dedekind. *Stetigkeit und irrationale Zahlen*. Vieweg & Sohn, Braunschweig, 1872.

[17] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. Also in [25], pages 1–82.

[18] G. Frege. *Grundlagen der Arithmetik, eine logisch-mathematische Untersuchung über den Begriff der Zahl*. , Breslau, 1884.

[19] G. Frege. *Grundgesetze der Arithmetik, begriffschriftlich abgeleitet*, volume I. Pohle, Jena, 1892. Reprinted 1962 (Olms, Hildesheim).

[20] G. Frege. Letter to Russell. English translation in [25], pages 127–128, 1902.

[21] G. Frege. *Grundgesetze der Arithmetik, begriffschriftlich abgeleitet*, volume II. Pohle, Jena, 1903. Reprinted 1962 (Olms, Hildesheim).

[22] J.H. Geuvers. *Logics and Type Systems*. PhD thesis, Catholic University of Nijmegen, 1993.

[23] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

[24] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings Second Symposium on Logic in Computer Science*, pages 194–204, Washington D.C., 1987. IEEE.

[25] J. van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, Cambridge, Massachusetts, 1967.

[26] D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Die

Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen, Band XXVII. Springer Verlag, Berlin, first edition, 1928.

[27] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press, 1986.

[28] F. Kamareddine, R. Bloo, and R. Nederpelt. On Π-conversion in the λ-cube and the combination with abbreviations. *Ann. Pure Appl. Logic*, 97(1–3):27–45, 1999.

[29] F. Kamareddine, L. Laan, and R.P. Nederpelt. Refining the Barendregt cube using parameters. In *Proceedings of the Fifth International Symposium on Functional and Logic Programming, FLOPS 2001*, pages 375–389, 2001.

[30] F. Kamareddine, T. Laan, and R. Nederpelt. Revisiting the notion of function. *Logic and Algebraic programming*, 54:65–107, 2003.

[31] F. Kamareddine and A. Ríos. A λ-calculus à la de bruijn with explicit substitutions. *Proceedings of Programming Languages Implementation and the Logic of Programs PLILP'95, Lecture Notes in Computer Science*, 982:45–62, 1995.

[32] Fairouz Kamareddine. Postponement, conservation and preservation of strong normalisation for generalised reduction. *J. Logic Comput.*, 10(5):721–738, 2000.

[33] Fairouz Kamareddine. Typed lambda-calculi with one binder. *J. Funct. Program.*, 15(5):771–796, 2005.

[34] Fairouz Kamareddine, Roel Bloo, and Rob Nederpelt. On pi-conversion in the lambda-cube and the combination with abbreviations. *Ann. Pure Appl. Logic*, 97(1-3):27–45, 1999.

[35] Fairouz Kamareddine, Twan Laan, and Rob Nederpelt. Revisiting the notion of function. *J. Log. Algebr. Program.*, 54(1-2):65–107, 2003.

[36] Fairouz Kamareddine and Rob Nederpelt. Refining reduction in the λ-calculus. *J. Funct. Programming*, 5(4):637–651, October 1995.

[37] Fairouz Kamareddine and Rob Nederpelt. A useful λ-notation. *Theoret. Comput. Sci.*, 155(1):85–109, 1996.

[38] Fairouz Kamareddine, Alejandro Ríos, and J. B. Wells. Calculi of generalised β-reduction and explicit substitutions: The type free and simply typed versions. *J. Funct. Logic Programming*, 1998(5), June 1998.

[39] A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. In *Proc. 1994 ACM Conf. LISP Funct. Program.*, pages 196–207, 1994.

[40] A. J. Kfoury and J. B. Wells. New notions of reduction and non-semantic proofs of β-strong normalization in typed λ-calculi. In *Proc. 10th Ann. IEEE Symp. Logic in Comput. Sci.*, pages 311–321, 1995.

[41] Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. An analysis of ML typability. *J. ACM*, 41(2):368–398, March 1994.

[42] Twan Laan and Michael Franssen. Parameters for first order logic. *Logic and Computation*, 2001.

[43] G. Longo and E. Moggi. Constructive natural deduction and its modest interpretation. Technical Report CMU-CS-88-131, Carnegie Mellono University, Pittsburgh, USA, 1988.

[44] Rob Nederpelt. *Strong Normalization in a Typed Lambda Calculus With Lambda Structured Types*. PhD thesis, Eindhoven, 1973.

[45] R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors. *Selected Papers on Automath*. Studies in Logic and the Foundations of Mathematics **133**. North-Holland, Amsterdam, 1994.

[46] G. Peano. *Arithmetices principia, nova methodo exposita*. Bocca, Turin, 1889. English translation in [25], pages 83–97.

[47] F.P. Ramsey. The foundations of mathematics. *Proceedings of the London Mathematical Society,* 2nd series, 25:338–384, 1926.

[48] Laurent Regnier. *Lambda calcul et réseaux*. PhD thesis, University Paris 7, 1992.

[49] G.R. Renardel de Lavalette. Strictness analysis via abstract interpretation for recursively defined types. *Information and Computation*, 99:154–177, 1991.

[50] J.C. Reynolds. *Towards a theory of type structure*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 1974.

[51] B. Russell. Letter to Frege. English translation in [25], pages 124–125, 1902.

[52] B. Russell. *The Principles of Mathematics*. Allen & Unwin, London, 1903.

[53] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908. Also in [25], pages 150–182.

[54] H. Weyl. *Das Kontinuum*. Veit, Leipzig, 1918. German; also in: Das Kontinuum und andere Monographien, Chelsea Pub.Comp., New York, 1960.

[55] A.N. Whitehead and B. Russell. *Principia Mathematica*, volume I, II, III. Cambridge University Press, 1910^1, 1927^2. All references are to the first volume, unless otherwise stated.