Languages for Formalisation

Fairouz Kamareddine
Heriot-Watt University
Edinburgh, UK

EBL17, Brasil, 12 May 2017

Guy Steele's discussion of most popular programming language in computer science

Computer Science Metanotation (CSM)

- Data Types:
  - Built-in: numbers, arrays, lists, etc.
  - User-defined: Records, Abstract Data Types or Symbolic Expressions (written in BNF).
- Code: Inference rules (written in Gentzen notation)
- Conditionals: rule dispatch via nondeterministic pattern-matching
- Repetition: overlines and/or ellipsis notations, and sometimes iterators
- Primitive expressions: logic and mathematics
- Special operation: capture-free substitution within a symbolic expression

Guy Steele, Its Time for a New Old Language

According to Steele

Early contributors include

- Gentzen
- Bakus
- Naur
- Church

Steel focuses around difficulties of use of BNF notation:

- Substitution
- Overline and Ellipsis
- Formalisation and Mechanisation of CSM.

Gentzen

- Gerhard Gentzen with his rule metanotation for natural deduction:

  "3.1. Eine Schluβfigur läβt sich in der Form Screiben:

$$\frac{\mathfrak{N}_1 \ldots \mathfrak{N}_\nu}{\mathfrak{B}} \qquad (\nu \geq 1),$$

  wobei $\mathfrak{N}_1, \ldots, \mathfrak{N}_\nu$ Formeln sind. $\mathfrak{N}_1, \ldots, \mathfrak{N}_\nu$ heiβen dann die Oberformlen, $\mathfrak{B}$ heiβt die Unterformel der Scluβfigur."

  (Gerhard Gentzen, 1934 [18])

Backus/Naur

- John Backus influenced by Emil Post's productions gives a syntax to write production rules *with multiple alternatives*
  for a context-free grammar for the International Algorithmic Language:

  "$<$ *digit* $>::\equiv$ 0 $\overline{or}$ 1 $\overline{or}$ 2 $\overline{or}$ 3 $\overline{or}$ 4 $\overline{or}$ 5 $\overline{or}$ 6 $\overline{or}$ 7 $\overline{or}$ 8 $\overline{or}$ 9
  $<$ *integer* $>::\equiv<$ *digit* $>$ $\overline{or}$ $<$ *integer* $><$ *digit* $>$"

  (John Backus, 1959)

- Peter Naur uses Backus notation where
  - $::\equiv \Longrightarrow ::=$ and
  - $\overline{or} \Longrightarrow |$ and *gave nonterminals the same names used in the text.*

  " $<$ *unsigned integer* $>::=<$ *digit* $> | <$ *unsigned integer* $><$ *digit* $>$
  $<$ *integer* $>::=<$ *unsigned integer* $> |+ <$ *unsigned integer* $> |- <$
  unsigned integer $>$ "

  (Naur, report on Algol 60, CACM)

Data Declaration à la Alonzo Church

Type Free

- $A ::= x \mid AB \mid \lambda x.B$
- $(\lambda x.B)C \rightarrow_\beta B[x := C]$.

With simple types:

- $\sigma ::= T \mid \sigma \rightarrow \tau$
- $A ::= x \mid AB \mid \lambda x{:}\sigma.B$
- $(\lambda x : \sigma.B)C \rightarrow_\beta B[x := C]$.

With dependent types:

- $A ::= x \mid * \mid \Box \mid AB \mid \lambda x{:}A.B \mid \Pi x{:}A.B$
- $(\lambda x : A.B)C \rightarrow_\beta B[x := C]$.
- Sometimes also with $(\Pi x : A.B)C \rightarrow_\Pi B[x := C]$.

Code à la Alonzo Church

With dependent/polymorphic types

$$(\lambda) \qquad \frac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash \Pi_{x:A}.B : s}{\Gamma \vdash \lambda_{x:A}.b : \Pi_{x:A}.B}$$

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

$$(\text{N-app}_\Pi) \qquad \frac{\Gamma \vdash F : \Pi_{x:A}.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}.B)a}$$

With simple types:

$$(\lambda_\to) \qquad \frac{\Gamma, x{:}\sigma \vdash b : \tau}{\Gamma \vdash \lambda_{x:A}.b : \sigma \to \tau} \qquad (\Gamma \vdash \sigma \to \tau : *)$$

$$(\text{app}) \qquad \frac{\Gamma \vdash F : \tau \to \sigma \qquad \Gamma \vdash a : \tau}{\Gamma \vdash Fa : \sigma}$$

In this talk we concentrate on the development of calculi rather than the development of the notation

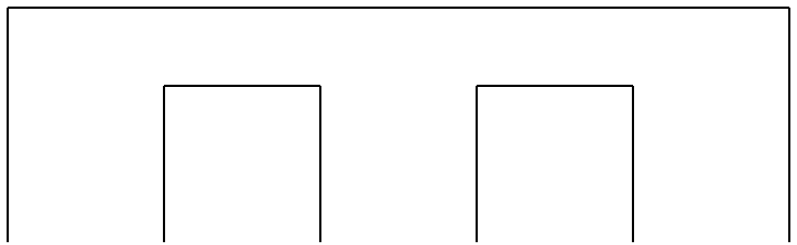
- The challenge is to develop expressive calculi that have clear syntax, semantics, and the desirable properties (Church-Rosser, correctness, termination).
- Nonetheless, notation is important.
- For example, simply changing the order of functions and arguments, and restructuring parenthesis, enable us to:
  - Express things that would hard to do in the old notation.
  - Reduce proofs of strong normalisation to proofs of weak normalisation.
  - Make computations more efficient.
  - Avoid unnecessary/redundant computations and allow for free lazy, local, or global reductions.

Lambda Calculus à la de Bruijn

- $A := x \mid AB \mid \lambda x.B$ $\qquad$ $A := x \mid \ < B > A \mid [x]B$
- $\mathcal{I}(x) = x, \qquad \mathcal{I}(\lambda x.B) = [x]\mathcal{I}(B), \qquad \mathcal{I}(AB) = \langle \mathcal{I}(B) \rangle \mathcal{I}(A)$
- $(\lambda x.\lambda y.xy)z$ 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 $((\ \lambda_x.(\lambda_y.\lambda_z.\text{–})c)ba)$ 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	$\langle d\rangle a$



$\langle a\rangle \quad \langle b\rangle \quad [x] \quad \langle c\rangle \quad [y] \quad [z] \quad \langle d\rangle \quad z$

- This maks it easy to study local/global/mini reductions into the λ-calculus, Kamareddine etal [29, 30]

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

- Regnier [42] uses θ and γ in analyzing perpetual reduction strategies.
- Term reshuffling is used by Kfoury, Tiuryn, Urzyczyn, Wells in [34, 32] in analyzing typability problems.
- Nederpelt [38], de Groote [11], Kfoury+ Wells [33], and Kamareddine [27] use generalised reduction and/or term reshuffling in relating SN to WN.
- Ariola etal [1] uses a form of term-reshuffling in obtaining a calculus that corresponds to lazy functional evaluation.
- Kamareddine etal [29, 24, 31, 3] show that they could reduce space/time needs in computation.

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

$$
\begin{array}{lll}
(\beta) & (\lambda_x.M)N \to M[x := N] & \\
(\beta_I) & (\lambda_x.M)N \to M[x := N] & \text{if } x \in FV(M) \\
(\beta_K) & (\lambda_x.M)N \to M & \text{if } x \notin FV(M) \\
(\theta) & (\lambda_x.N)PQ \to (\lambda_x.NQ)P & \\
(\beta_e) & (M)\overline{s}[x]N \to \overline{s}\{N[x := M] & \text{for } \overline{s} \text{ well-balanced.}
\end{array}
$$

- Kamareddine [27] shows that β_e satisfies *Church Rosser*, *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 $M \twoheadrightarrow^{+}_{\beta_I\theta_K} P \twoheadrightarrow_{\beta_K} O$
(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 [11]

- 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.
- Kamareddine [27] shows that:

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

λ-calculus and Type Theory: When and Why?

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* [7].
- 1872: Due to the more *exact definition of real numbers* given by Dedekind [12], the rules for reasoning with real numbers became even more precise.
- 1895-1897: Cantor began formalizing *set theory* [5, 6] and made contributions to *number theory*.
- 1889: Peano formalized *arithmetic* [40], but did not treat logic or quantification.

λ-calculus and Type Theory: When and Why?

- 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* [16], 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)*

Abstraction principle was fundamental to Frege's formalisation

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

> *"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*)*

The richness of functions required care to avoid paradox

- 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)*

Richness of functions required a form of self-application

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

- In Grundlagen der Arithmetik [13] he argued that mathematics can be seen as a branch of logic.
- In Grundgesetze der Arithmetik [14, 17] he described the elementary parts of arithmetic 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 *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.

Despite extreme care, Frege had paradox

- 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 [45] 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* [15]. 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.

Paradox also 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 $K =_{\mathtt{def}} \{x \mid x \notin x\}$ 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* ([4], 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 was discovered by Cantor around 1895, but was not published before 1932.

Russell suggests types to avoid 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 [47].

Types and 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)** $\mathrm{x} =_{\mathrm{L}} \mathrm{y} \overset{\mathrm{def}}{\leftrightarrow} \forall \mathrm{z}[\mathrm{z}(\mathrm{x}) \leftrightarrow \mathrm{z}(\mathrm{y})]$.
  In order to express this general notion in RTT, we have to incorporate *all* pfs $\forall \mathrm{z} : (0^0)^n[\mathrm{z}(\mathrm{x}) \leftrightarrow \mathrm{z}(\mathrm{y})]$ 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 [49] 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 and Church's simply typed λ-calculus

- Ramsey [41], and Hilbert and Ackermann [22], *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).

Church's λ-calculus

- Church wanted the λ-calculus to be a theory of *functions* and *logic.*
- The logic part turned out to be inconsistent, so Church restricted the λ-calculus to a theory of functions (Church 1932).
- $A := x \mid AB \mid \lambda x.A$
- This type free λ-calculus is the language of the computable function: f is computable iff f can be written in the type free λ-calculus.
- To incorporate logic, Church added the simple types of Ramsey's STT, giving us the simply typed λ-calculus (Church 1940).
- $\lambda\!\!\rightarrow$ is very restrictive.
- Numbers, booleans, identity, etc., 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*).

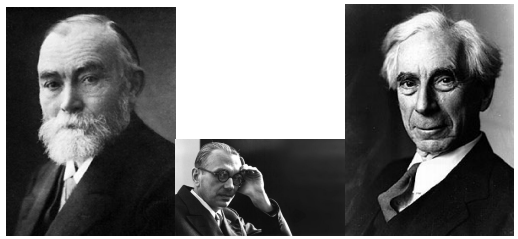
The evolution of functions with Frege, Russell and Church

- Historically, *functions* have been treated as *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 $f(x)$—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.

Limitation of the notion of function in the λ-calculus

- We can enrich types to avoid (some of) the limitations of the λ-calculus.
- But is this enough?
- The answer is no, and this is witnessed by the numerous extensions of logic and computational languages based on the λ-calculus.
- E.g., $A := x \mid AB \mid \lambda x.A \mid c(A_1, \ldots, A_n) \mid A[x := B]$

Summary



- *General definition of function 1879* [16] is key to Frege's *formalisation of logic*.
- *Self-application of functions* was at the heart of *Russell's paradox 1902* [45].
- To *avoid paradox* Russell controled function application via *type theory*.
- Russell [46] *1903* gives the first type theory: the *Ramified Type Theory* (RTT).
- RTT is used in Russell and Whitehead's Principia Mathematica [50] 1910–1912.
- *Simple theory of types* (STT): Ramsey [41] *1926*, Hilbert and Ackermann [22] *1928*.

Summary

- Church's *simply typed λ-calculus* $\lambda\!\!\rightarrow$ [8] 1940 = λ-calculus + STT.
- The hierarchies of types/orders in RTT and STT are *unsatisfactory*.
- Frege's functions ≠ Principia's functions ≠ *λ-calculus* functions.
- The *notion of function adopted in the λ-calculus* is *unsatisfactory* [26].
- 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, computability, etc.) without losing the flexibility of the higher-order aspects.
- Hence, birth of *different systems of functions and types*, each with *different functional power*.

The Challenge

- Is to design calculi that work well for all the features needed for expressive computation languages:
  - Data Types,
  - Inference rules,
  - logic and mathematics,
  - capture-free substitution within a symbolic expression,
- while having a clear syntax, semantics and the desired properties (Correctness, Termination, Computerisation).

Church's Simply Typed λ-calculus in modern notation

- Terms $A ::= x \mid AB \mid \lambda x{:}\sigma.B$
- Types $::= T \mid \mid \sigma \to \tau$
- Γ is an environment (set of declaration).
- Rules:

$$\frac{}{\text{(start)} \quad \frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}}$$

$$(\lambda) \quad \frac{\Gamma, x{:}\sigma \vdash A : \tau}{\Gamma \vdash \lambda_{x:\sigma}.A : \sigma \to \tau}$$

$$(\text{app}\Pi) \quad \frac{\Gamma \vdash A : \sigma \to \tau \qquad \Gamma \vdash B : \sigma}{\Gamma \vdash AB : \tau}$$

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:

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2} \qquad \textit{if } (s_1, s_2) \in \boldsymbol{R}$$

The β-cube: $\rightarrow_\beta$ + conv_β + app_Π

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

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

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

(Π) $$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2 \quad (s_1, s_2) \in \boldsymbol{R}}{\Gamma \vdash \Pi_{x:A}.B : s_2}$$

(λ) $$\frac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash \Pi_{x:A}.B : s}{\Gamma \vdash \lambda_{x:A}.b : \Pi_{x:A}.B}$$

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

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

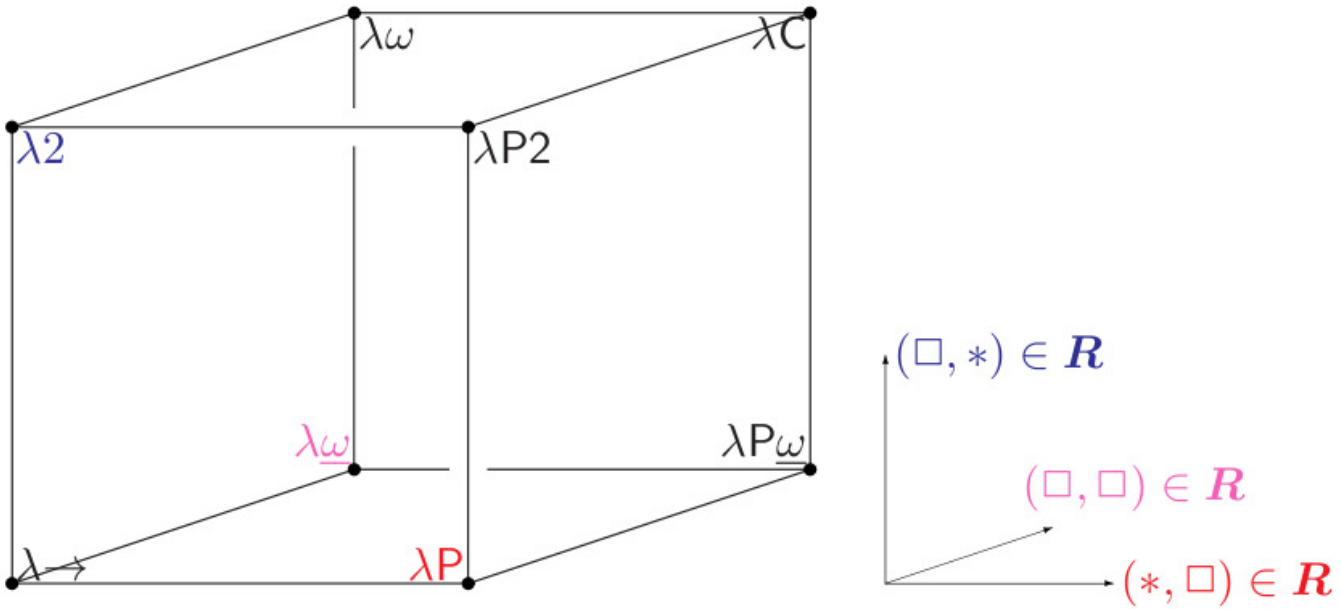
6 desirable properties of a type system with reduction r

- *Types are correct (TC)*
  If $\Gamma \vdash A : B$ then $B \equiv \Box$ or $\Gamma \vdash B : s$ for $s \in \{*, \Box\}$.
- *Subject reduction (SR)* If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_r A'$ then $\Gamma \vdash A' : B$.
- *Preservation of types (PT)* If $\Gamma \vdash A : B$ and $B \twoheadrightarrow_r B'$ then $\Gamma \vdash A : B'$.
- *Strong Normalisation (SN)* If $\Gamma \vdash A : B$ then $\mathrm{SN}_{\to_r}(A)$ and $\mathrm{SN}_{\to_r}(B)$.
- *Subterms are typable (STT)* If A is $\vdash$-legal and if C is a sub-term of A then C is $\vdash$-legal.
- *Unicity of types*
  - *(UT1)* If $\Gamma \vdash A_1 : B_1$ and $\Gamma \vdash A_2 : B_2$ and $A_1 =_r A_2$, then $\Gamma \vdash B_1 =_r B_2$.
  - *(UT2)* If $\Gamma \vdash B_1 : s$, $B_1 =_r B_2$ and $\Gamma \vdash A : B_2$ then $\Gamma \vdash B_2 : s$.

	Simple	Poly-morphic	Depend-ent	Constr-uctors	Related system	Refs.
$\lambda\!\!\rightarrow$	$(*,*)$				λ^{τ}	[8, 2, 23]
$\lambda 2$	$(*,*)$	$(\Box,*)$			F	[20, 44]
λP	$(*,*)$		$(*,\Box)$		AUT-QE, LF	[10, 21]
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$	POLYREC	[43]
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$			[36]
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$	$F\omega$	[20]
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$		
λC	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$	CC	[9]

The 8 Systems

The Barendregt Cube



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

- $$\frac{y : * \vdash y : * \qquad y : *, x{:}y \vdash y : *}{y : * \vdash \Pi x{:}y.y : *}\ \text{by}\ \ (\Pi)\ (*, *)$$
- $$\frac{y : *, x : y \vdash x : y \qquad y : * \vdash \Pi x{:}y.y : *}{y : * \vdash \lambda x : y.x : \Pi x{:}y.y}\ \text{by}\ \ (\lambda)$$
- $$\frac{\vdash * : \Box \qquad y : * \vdash \Pi x{:}y.y : *}{\vdash \Pi y : *.\Pi x{:}y.y : *}\ \text{by}\ \ (\Pi)\ \ \text{by}\ (\Box, *)$$
- $$\frac{y : * \vdash \lambda x : y.x : \Pi x{:}y.y \qquad \vdash \Pi y : *.\Pi x{:}y.y : *}{\vdash \lambda y : *.\lambda x : y.x : \Pi y : *.\Pi x{:}y.y}\ \text{by}\ \ (\lambda)$$

LF and Frege's low level notion of function

- *LF* [21] is often described as λP of the Barendregt Cube. However, *Use of* Π*-formation rule* $(*, \Box)$ *is restricted in LF* [19].
- We only need a type $\Pi x{:}A.B : \Box$ when PAT is applied during construction of the type $\Pi\alpha{:}\texttt{prop}.*$ of the operator Prf where for a proposition Σ, Prf(Σ) is the type of proofs of Σ.

$$\frac{\texttt{prop}{:}* \vdash \texttt{prop}: * \qquad \texttt{prop}{:}*, \alpha{:}\texttt{prop} \vdash *{:}\Box}{\texttt{prop}{:}* \vdash \Pi\alpha{:}\texttt{prop}.* : \Box}(*, \Box) \in \boldsymbol{R}.$$

- In LF, this is the only point where the Π-formation rule $(*, \Box)$ is used. But, Prf is only used when applied to $\Sigma{:}\texttt{prop}$. 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.
- Kamareddine etal [25] precisely locate *LF* (*between* $\lambda{\rightarrow}$ *and* λP).

The Cube with parametric constants

- Let $(*,*) \subseteq \mathbf{R}, \boldsymbol{P} \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$.
- $\lambda\mathbf{R}\boldsymbol{P} = \lambda\mathbf{R}$ and the two rules $\vec{\mathbf{C}}$**-weak)** and $\vec{\mathbf{C}}$**-app)**:

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

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

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

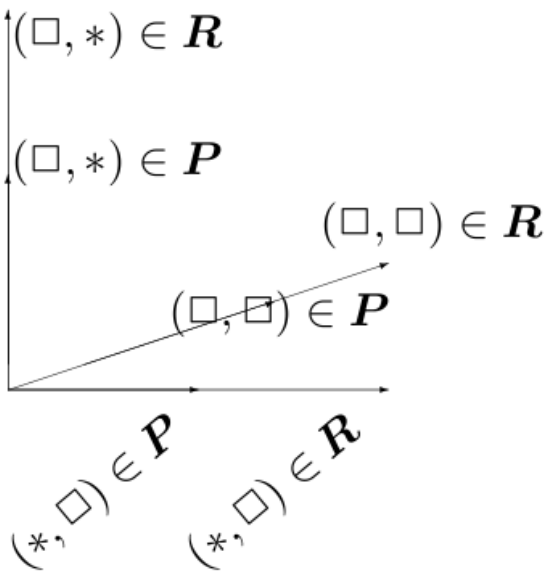
Hence LF is $\lambda_{\rightarrow}$ with $\mathrm{Prf}(b)$ a new syntactic entity such that

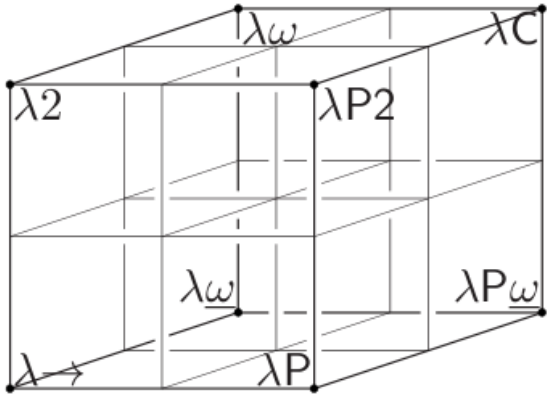
$$\mathtt{prop}{:}*, \mathrm{Prf}(\alpha : \mathtt{prop}) : *, \Sigma{:}\mathtt{prop} \vdash \mathrm{Prf}(\Sigma) : *$$

Properties of the Refined Cube

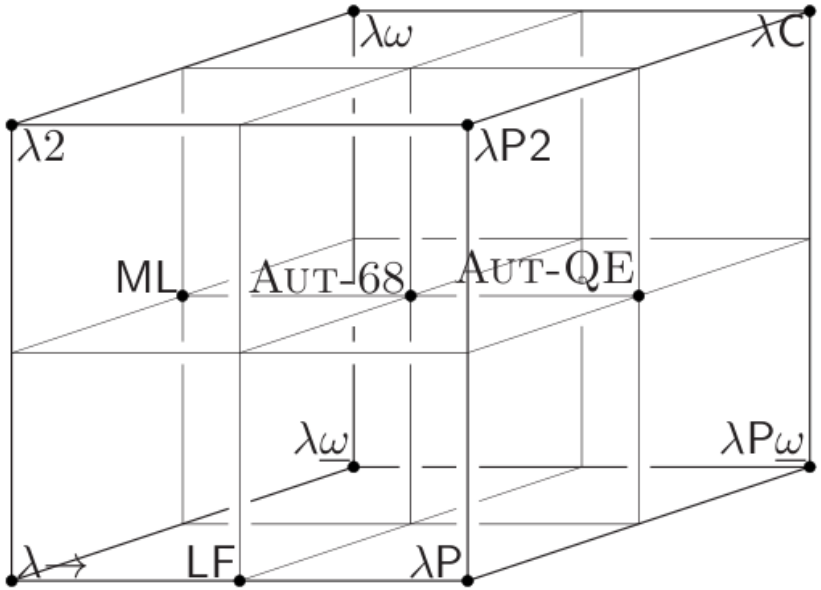
- *(Correctness of types)* If $\Gamma \vdash A : B$ then ($B \equiv \Box$ or $\Gamma \vdash B : S$ for some sort S).
- *(Subject Reduction SR)* If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_\beta A'$ then $\Gamma \vdash A' : B$
- *(Strong Normalisation)* For all $\vdash$-legal terms M, we have $\mathrm{SN}_{\twoheadrightarrow_\beta}(M)$.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

The refined Barendregt Cube





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



Logicians versus mathematicians: induction over numbers

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

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

- Mathematician uses $\mathtt{ind}$ 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 $(\mathtt{ind}PQR){:}(\Pi n{:}\mathbb{N}.Pn)$.
- 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):

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

- The logician's type $\mathtt{Ind}$ 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: induction over numbers

- *Mathematician 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: 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* [35].
- 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.

The π-cube: $R_\pi = R_\beta \setminus (\text{conv}_\beta) \cup (\text{conv}_{\beta\Pi}), \rightarrow_{\beta\Pi}$

- $(\lambda x{:}\alpha.A)B \rightarrow_\beta A[x := B]$
- $(\Pi x{:}\alpha.A)B \rightarrow_\Pi A[x := B]$

$$(\text{axiom}) \qquad (\text{start}) \qquad (\text{weak}) \qquad (\Pi) \qquad (\lambda) \qquad (\text{app}_\Pi)$$

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

Lemma: $\Gamma \vdash_\beta A : B$ iff $\Gamma \vdash_\pi A : B$

Lemma: The β-cube and the π-cube satisfy the six properties that are desirable for type systems.

The π_i-cube: $R_{\pi_i} = R_\pi \setminus (\text{app}_\Pi) \cup (\text{i-app}_\Pi)$, $\rightarrow_{\beta\Pi}$

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

(axiom) (start) (weak) (Π) (λ)

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

$$(\text{i-app}_\Pi) \quad \frac{\Gamma \vdash F : \Pi_{x:A}.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}.B)a}$$

Lemma:

- If $\Gamma \vdash_\beta A : B$ then $\Gamma \vdash_{\pi_i} A : B$.
- If $\Gamma \vdash_{\pi_i} A : B$ then $\Gamma \vdash_\beta A : [B]_\Pi$
  where $[B]_\Pi$ is the Π-normal form of B.

The π_i-cube

- The π_i-cube loses three of its six properties
  Let $\Gamma = z : *, x : z$. We have that $\Gamma \vdash_{\pi_i} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$.
  - *We do not have TC* $(\Pi_{y:z}.z)x \not\equiv \Box$ and $\Gamma \not\vdash_{\pi_i} (\Pi_{y:z}.z)x : s$.
  - *We do not have SR* $(\lambda_{y:z}.y)x \to_{\beta\Pi} x$ but $\Gamma \not\vdash_{\pi_i} x : (\Pi_{y:z}.z)x$.
  - *We do not have UT2* $\vdash_{\pi_i} * : \Box$, $* =_{\beta\Pi} (\Pi_{z:*}.*)\alpha$, $\alpha : * \vdash_{\pi_i} (\lambda_{z:*}.*)\alpha : (\Pi_{z:*}.*)\alpha$ and $\not\vdash_{\pi_i} (\Pi_{z:*}.*)\alpha : \Box$
- But we have:
  - *We have UT1*
  - *We have STT*
  - *We have PT*
  - *We have SN*
  - *We have a weak form of TC* If $\Gamma \vdash_{\pi_i} A : B$ and *B does not have a* Π*-redex* then either $B \equiv \Box$ or $\Gamma \vdash_{\pi_i} B : s$.
  - *We have a weak form of SR* If $\Gamma \vdash_{\pi_i} A : B$, *B is not a* Π*-redex* and $A \twoheadrightarrow_{\beta\Pi} A'$ then $\Gamma \vdash_{\pi_i} A' : B$.

The problem can be solved by re-incorporating Frege and Russell's notions of low level functions (which was lost in Church's notion of function)

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

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

Figure 1: Basic abbreviation rules BA

$$(\text{let}_{\backslash})\quad \frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\backslash_{x:A}.C)B : D[x := B]}$$

Figure 2: $(\text{let}_{\backslash})$ where $\backslash = \lambda$ or $\backslash = \Pi$

The β_a-cube: $R_{\beta_a} = R_\beta + \mathrm{BA} + \mathrm{let}_\beta, \rightarrow_\beta$

(axiom) (start) (weak) (Π) (λ) (app_Π) (conv_β)

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

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

$$\text{(let}_\beta\text{)}\quad \frac{\Gamma, x = B{:}A \vdash C : D}{\Gamma \vdash (\lambda_{x:A}.C)B : D[x := B]}$$

Lemma: The β_a-cube satisfies the desirable properties except for typability of subterms.

If A is $\vdash$-legal and B is a subterm of A such that every bachelor $\lambda_{x:D}$ in B is also bachelor in A, then B is $\vdash$-legal.

The π_a-cube: $R_{\pi_a} = R_\pi + \mathrm{BA} + \mathrm{let}_\beta + \mathrm{let}_\Pi, \to_{\beta\Pi}$

(axiom) (start) (weak) (Π) (λ) (app_Π) ($\mathrm{conv}_{\beta\Pi}$)

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

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

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

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

Lemma: The π_a-cube satisfies the same properties as the β_a.

The π_{ai}-cube: $R_{\pi_{ai}} = R_{\pi_a} \setminus \text{app}_\Pi + \text{i-app}_\Pi$, $\rightarrow_{\beta\Pi}$

Let $\Gamma = z : *, x : z$. We have that $\Gamma \vdash_{\pi_{ai}} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$.

- *We NOW have TC* although $\Gamma \not\vdash_{\pi_i} (\Pi_{y:z}.z)x : s$, we have $\Gamma \vdash_{\pi_{ai}} (\Pi_{y:z}.z)x : s$
  By (weak-a) $z : *, x : z, y = x : z \vdash_{\pi_{ai}} z : *$.
  Hence by (let_Π) $z : *, x : z \vdash_{\pi_{ai}} (\Pi_{y:z}.z)x : *[y := x] \equiv *$.
- *We NOW have SR* $(\lambda_{y:z}.y)x \rightarrow_{\beta\Pi} x$.
  Although $\Gamma \not\vdash_{\pi_i} x : (\Pi_{y:z}.z)x$, we have $\Gamma \vdash_{\pi_{ai}} x : (\Pi_{y:z}.z)x$
  Since $z : *, x : z \vdash_{\pi_{ai}} x : z$, and $z : *, x : z \vdash_{\pi_{ai}} (\Pi_{y:z}.z)x : *$ and $z : *, x : z \Vdash z =_{\beta\Pi} (\Pi_{y:z}.z)x$, we use ($\text{conv}_{\beta\Pi}$) to get:
  $z : *, x : z \vdash_{\pi_{ai}} x : (\Pi_{y:z}.z)x$.

Identifying λ and Π (see [28])

- In the cube, 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

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

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

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

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

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

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

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

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 $\rightarrow_\flat$ [28]

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 $B_1 \overset{\diamond}{=}_\flat B_2$.
2. If $\Gamma \vdash_\flat A_1 : B_1$ and $\Gamma \vdash_\flat A_2 : B_2$ and $A_1 =_\flat A_2$, then $B_1 \overset{\diamond}{=}_\flat B_2$.
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:
   1. either $\Gamma \vdash_\flat A : B_2$, $(\Gamma \vdash_\flat A : B_2)^{-1} = (\Gamma', A'', B_2')$ and $B_1' =_\beta B_2'$,
   2. or $\Gamma \vdash_\flat C : B_2$, $(\Gamma \vdash_\flat C : B_2)^{-1} = (\Gamma', C', B_2')$ and $A' =_\beta C'$.

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 (appl$\flat\flat$) 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 (appl$\flat$): $\Gamma \vdash A : B$ 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:

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

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

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

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

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

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 [50].*

> "**$*9{\cdot}14$.** *If 'φx' is significant, then if x is of the same type as a, 'φa' is significant, and vice versa.*
>
> **$*9{\cdot}15$.** *If, for some a, there is a proposition φa, then there is a function $\varphi\hat{x}$, and vice versa*"
>
> (Principia Mathematica, *p. 133)*

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 $(\lambda_{y:z}.y)x \to_\beta x$ 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 $z =_\beta (\flat_{y:z}.z)x$)

Consequences of unifying λ and Π

- A term can have many distinct types. E.g., in λP we have:

  $\alpha : * \vdash_\beta (\lambda x{:}\alpha.\alpha) : (\Pi x{:}\alpha.*)$ and $\alpha : * \vdash_\beta (\Pi x{:}\alpha.\alpha) : *$

  which, when we give up the difference between λ and Π, result in:
  - $\alpha : * \vdash_\beta [x{:}\alpha]\alpha : [x{:}\alpha] *$ and
  - $\alpha : * \vdash_\beta [x{:}\alpha]\alpha : *$

- More generally, in AUT-QE we have the dervived rule:

$$\frac{\Gamma \vdash_\beta [x_1{:}A_1]\cdots[x_n{:}A_n]B : [x_1{:}A_1]\cdots[x_n{:}A_n]*}{\Gamma \vdash_\beta [x_1{:}A_1]\cdots[x_n{:}A_n]B : [x_1{:}A_1]\cdots[x_m{:}A_m]*} \qquad 0 \leq m \leq n \quad (3)$$

This derived rule (3) has the following equivalent derived rule in λP (and hence in the higher systmes like $\lambda P\omega$):

$$\frac{\Gamma \vdash_\beta \lambda x_1{:}A_1.\cdots\lambda x_n{:}A_n.B : \Pi x_1{:}A_1.\cdots\Pi x_n{:}A_n.* \qquad 0 \leq m \leq n}{\Gamma \vdash_\beta \lambda x_1{:}A_1.\cdots\lambda x_m{:}A_m.\Pi x_{m+1}{:}A_{m+1}.\cdots\Pi x_n{:}A_n.B : \Pi x_1{:}A_1.\cdots\Pi x_m{:}A_m.*}$$

However, $\textsc{Aut}$-QE goes further and generalises (3) to a rule of *type inclusion*:

$$\frac{\Gamma \vdash_\beta M : [x_1{:}A_1]\cdots[x_n{:}A_n]*}{\Gamma \vdash_\beta M : [x_1{:}A_1]\cdots[x_m{:}A_m]*} \qquad 0 \leq m \leq n \qquad (\mathrm{Q})$$

The β_Q-cube = β-cube + (Q_β)

$$(Q_\beta) \qquad \frac{\Gamma \vdash \lambda^{i:1..k}_{x_i:A_i}.A : \Pi^{i:1..n}_{x_i:A_i}.*}{\Gamma \vdash \lambda^{i:1..m}_{x_i:A_i}.\Pi^{i:m+1..k}_{x_i:A_i} A : \Pi^{i:1..m}_{x_i:A_i}.*} \qquad 0 \leq m \leq n,\ A \not\equiv \lambda_{x:B}$$

- *Lemma:*
  - *The β_Q-cube enjoys all the properties of the cube except the unicity of types.*
  - *Rule Q_β and rule $(s, \Box)$ for $s \in \{*, \Box\}$ imply rule $(s, *)$.*
    This means that the type systems $\lambda_Q\underline{\omega}$ and $\lambda_Q\omega$ are equal, and that $\lambda_Q P\underline{\omega}$ and $\lambda_Q P\omega$ are equal as well.
- Unicity of types fails for the β_Q-cube. Take:
  $A : *, x : \Pi_{y:A}.* \vdash x : \Pi_{y:A}.*$ and hence by Q_β,
  $A : *, x : \Pi_{y:A}.* \vdash x : *$.

Cubes								
β	$\rightarrow_\beta$	BT	conv_β	app				
π	$\rightarrow_{\beta\Pi}$	BT	$\mathsf{conv}_{\beta\Pi}$	app				
β_a	$\rightarrow_\beta$	BT	conv_β	app	BA	let_λ		
π_a	$\rightarrow_{\beta\Pi}$	BT	$\mathsf{conv}_{\beta\Pi}$	app	BA	let_λ	let_Π	
π_i	$\rightarrow_{\beta\Pi}$	BT	$\mathsf{conv}_{\beta\Pi}$	i-app				
π_{ai}	$\rightarrow_{\beta\Pi}$	BT	$\mathsf{conv}_{\beta\Pi}$	i-app	BA	let_λ	let_Π	
β_Q	$\rightarrow_\beta$	BT	conv_β	app				Q
π_{iQ}	$\rightarrow_{\beta\Pi}$	BT	$\mathsf{conv}_{\beta\Pi}$	i-app				Q
β_{aQ}	$\rightarrow_\beta$	BT	conv_β	app	BA	let_λ		Q
π_{aiQ}	$\rightarrow_{\beta\Pi}$	BT	$\mathsf{conv}_{\beta\Pi}$	i-app	BA	let_λ	let_Π	Q
π_Q	$\rightarrow_{\beta\Pi}$	BT	$\mathsf{conv}_{\beta\Pi}$	app				Q
π_{aQ}	$\rightarrow_{\beta\Pi}$	BT	$\mathsf{conv}_{\beta\Pi}$	app	BA	let_λ	let_Π	Q
$c\pi$	$\rightarrow_{\beta\Pi}$	BT_c		appc				
$c\pi_a$	$\rightarrow_{\beta\Pi}$	BT_c		appc	BA_c	letc_λ	letc_Π	
$c\pi_Q$	$\rightarrow_{\beta\Pi}$	BT_c		appc				Qc
$c\pi_{aQ}$	$\rightarrow_{\beta\Pi}$	BT_c		appc	BA_c	letc_λ	letc_Π	Qc

Canonical and Non Canonical Type Systems

Properties	β-cube	$\flat$-cube	$\flat_i$-cube	$\flat_d$-cube	$\flat_{ai}$-cube
Church-Rosser	yes	yes	yes	yes	yes
Correctness of types	yes	yes	restr.	yes	yes
Typability of subterms	yes	yes	restr.	restr.	restr.
Subject reduction	yes	yes	restr.	yes	yes
Unicity of types	yes	restr. but patterned	restr.	yes	yes
Strong normalisation	yes	yes	yes	yes	yes
types more terms	no	no	yes	yes	yes

- Also, PTSs with de Bruijn indices.
- Also, PTSs with Curry style typing.
- PTSs with intersection types.

From Frege's low level functions to PTSs that capture strong normalisation

- Kamareddine and Wells 2017, has incorporated Frege's low level of functions to create PTSs with intersection types which contain all the ordinary PTSs (including the β-cube given above and its extensions with parameters/Frege's functions.
- The f-cube is the β-cube extended with finite set declarations in the form of ordinary mathematical notion of function.
- **Theorem:** If $\Gamma \vdash_f A : B$ then A and B are strongly normalising.
- **Theorem:** If a type free term of the λ-calculus M is strongly Normalising then M is typable in the f-cube.
- Urzyczyn proved $U = (\lambda r.\, h(r(\lambda f \lambda s.\, f\ s))(r(\lambda q.\lambda g.\, g\ q)))(\lambda o.\, o\ o\ o)$ is untypable in Fω. Hence U is untypable in any system of the cube.
- But U is strongly normalising.
- Kamareddine and Wells 2017 prove that U is typable in the f-cube: There are Γ, A such that $\Gamma \vdash_f U : A$.

Conclusion

- A hierarchy of systems that classify important properties of CR, SN, SR.
- Not only types are used to derive important properties and avoid paradoxes and non termination, but also types classify non termination.
- We are far away still from having computer help to prove these properties. It would be nice to have a system that can help us with the proofs.

[1] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. pages 233–246.

[2] H[endrik] P[ieter] Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[3] R. Bloo, F. Kamareddine, and R. Nederpelt. The Barendregt Cube with Definitions and Generalised Reduction. *Information and Computation*, 126(2):123–143, 1996.

[4] C. Burali-Forti. Una questione sui numeri transfiniti. *Rendiconti del Circolo Matematico di Palermo*, 11:154–164, 1897. English translation in [48], pages 104–112.

[5] Georg Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (part 1). *Mathematische Annalen*, 46:481–512, 1895.

[6] Georg Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (part 2). *Mathematische Annalen*, 49:207–246, 1897.

[7] Augustin-Louis Cauchy. *Cours d'Analyse de l'École Royale Polytechnique*. Debure, Paris, 1821. Also in *Œuvres Complètes* (2), volume III, Gauthier-Villars, Paris, 1897.

[8] Alonzo Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

[9] Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Inform. & Comput.*, 76:95–120, 1988.

[10] N.G. de Bruijn. The mathematical language Automath – its usage and some of its extensions. In L. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 29–61, Heidelberg, 1970. Springer-Verlag. Reprinted in [39, A.2].

[11] Philippe de Groote. The conservation theorem revisited. pages 163–178.

[12] Richard Dedekind. *Stetigkeit und irrationale Zahlen*. Vieweg & Sohn, Braunschweig, 1872. Fourth edition published in 1912.

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

[14] G. Frege. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik,* new series, 100:25–50, 1892. English translation in [37], pages 157–177.

[15] G. Frege. Letter to Russell. English translation in [48], pages 127–128, 1902.

[16] Gottlob Frege. *Begriffsschrift: eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. Can be found on pp. 1–82 in [48].

[17] Gottlob Frege. *Grundgesetze der Arithmetik*, volume 2. Hermann Pohle, Jena, 1903. Republished 1962 (Olms, Hildesheim).

[18] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934.

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

[20] J[ean]-Y[ves] Girard. *Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur*. Thèse d'Etat, Université de Paris VII, 1972.

[21] 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.

[22] D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Die Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen, Band XXVII. Springer Verlag, Berlin, first edition, 1928.

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

[24] 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.

[25] F. Kamareddine, T. Laan, and R. P. Nederpelt. Refining the Barendregt cube using parameters. In *Proc. 5th Int'l Symp. Functional & Logic Programming*, volume 2024 of *LNCS*, pages 375–389, 2001.

[26] F. Kamareddine, T. Laan, and R. P. Nederpelt. Revisiting the λ-calculus notion of function. *J. Algebraic & Logic Programming*, 54:65–107, 2003.

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

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

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

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

[31] 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.

[32] A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. pages 196–207.

[33] A. J. Kfoury and J. B. Wells. New notions of reduction and non-semantic proofs of β-strong normalization in typed λ-calculi. pages 311–321.

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

[35] Twan Laan and Michael Franssen. Embedding first-order logic in a pure type system with parameters. *J. Log. Comput.*, 11(4):545–557, 2001.

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

[37] B. McGuinness, editor. *Gottlob Frege: Collected Papers on Mathematics, Logic, and Philosophy*. Basil Blackwell, Oxford, 1984.

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

[39] Rob Nederpelt, J. H. Geuvers, and Roel C. de Vrijer. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1994.

[40] Giuseppe Peano. *Árithmetices Principia, Nova Methodo Exposita*. Bocca, Turin, 1889. An English translation can be found on pp. 83–97 in [48].

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

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

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

[44] J. C. Reynolds. Towards a theory of type structure. In *Colloque sur la Programmation*, volume 19 of *LNCS*, pages 408–425. Springer-Verlag, 1974.

[45] B. Russell. Letter to Frege. English translation in [48], pages 124–125, 1902.

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

[47] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908. Also in [48], pages 150–182.

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

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

[50] Alfred North Whitehead and Bertrand Russel. *Principia Mathematica*. Cambridge University Press, 1910–1913. In three volumes published from 1910 through 1913. Second edition published from 1925 through 1927. Abridged edition published in 1962.