Evolution of types and functions in 20th century

Fairouz Kamareddine
Heriot-Watt University
Joint work with Twan Laan and Rob Nederpelt

22 July 2002

Abstract

We provide a prehistory of type theory up to 1903 and its development between Russell and Whitehead's *Principia Mathematica* (1910–1912) and Church's simply typed λ-calculus of 1940. We also show that the notion of function adopted in the λ-calculus and type theory is not satisfactory.

Summary

- *Prehistory of types*
- *1902*: Russell's letter to Frege about the paradox in *Begriffsschrift*.
- *1903*: Russell gives the first theory of types: the *Ramified Type Theory* (RTT).
- *simple theory of types* (STT): Ramsey *1926*, Hilbert and Ackermann *1928*.
- *1940*: Church's own *simply typed λ-calculus* (known as $\lambda{\to}$) is based on STT.
- The unsatisfactory notion of function in Church's work.

Prehistory of Types (Euclid)

- Euclid's *Elements* (circa 325 B.C.) begins with:

  1. A *point* is that which has no part;
  2. A *line* is breadthless length.

     ⋮

  15. A *circle* is a plane figure contained by one line such that all the straight lines falling upon it from one point among those lying within the figure are equal to one another.

- Although the above seems to merely *define* points, lines, and circles, it shows more importantly that Euclid *distinguished* between them. Euclid always mentioned to which *class* (points, lines, etc.) an object belonged.

Prehistory of Types (Euclid)

- By distinguishing classes of objects, Euclid prevented undesired situations, like considering whether two points (instead of two lines) are parallel.
- *Undesired* results? Euclid himself would probably have said: *impossible* results. When considering whether two objects were parallel, intuition implicitly forced him to think about the *type* of the objects. As intuition does not support the notion of parallel points, he did not even *try* to undertake such a construction.
- In this manner, types have always been present in mathematics, although they were not noticed explicitly until the late 1800s. If you have studied geometry, then you have some (implicit) understanding of types.

Prehistory of Types (Paradox Threats)

- Starting in the 1800s, mathematical systems became less intuitive, for several reasons:
  - Very complex or abstract systems.
  - Formal systems.
  - Something with less intuition than a human using the systems: a computer.
- These situations are *paradox threats*. An example is Frege's Naive Set Theory. In such cases, there is not enough intuition to activate the (implicit) type theory to warn against an impossible situation. Reasoning proceeds within the impossible situation and then obtains a result that may be wrong or paradoxical.

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* [13].
- 1872: Due to the more *exact definition of real numbers* given by Dedekind [19], the rules for reasoning with real numbers became even more precise.
- 1895-1897: Cantor began formalizing *set theory* [11, 12] and made contributions to *number theory*.

Prehistory of Types (formal systems in 19th century)

- 1889: Peano formalized *arithmetic* [48], but did not treat logic or quantification.

- 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* [20], the first formalisation of logic giving logical concepts via symbols rather than natural language.

Prehistory of Types (formal systems in 19th century)

> "[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)

- 1892-1903 Frege's *Grundgesetze der Arithmetik* [22, 26], could handle elementary arithmetic, set theory, logic, and quantification.

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*.
- Similarly, the *result of applying* a function to an argument did not necessarily have to be a number.

Prehistory of Types (Begriffsschrift's functions)

Functions of more than one argument were constructed by a method that is very close to the method presented by Schönfinkel [58] in 1924:

Abstraction Principle 2.

> *"If, given a function, we think of a sign*[1] *that was hitherto regarded as not replaceable as being replaceable at some or all of its occurrences, then by adopting this conception we obtain a function that has a new argument in addition to those it had before."*

(*Begriffsschrift*, Section 9)

[1] We can now regard a sign that previously was considered replaceable as replaceable also in those places in which up to this point it was considered fixed. [footnote by Frege]

Prehistory of Types (Begriffsschrift's functions)

With this definition of function, two of the three possible paradox threats occurred:

1. The generalisation of the concept of function made the system more abstract and *less intuitive*. The fact that functions could have different types of arguments is at the basis of the Russell Paradox;

2. Frege introduced a *formal* system instead of the informal systems that were used up till then. Type theory, that would be helpful in distinguishing between the different types of arguments that a function might take, was left informal.

So, Frege had to proceed with caution. And so he did, at this stage.

Prehistory of Types (Begriffsschrift's functions)

Frege was aware of some typing rule that does not allow to substitute functions for object variables or objects for function variables:

> "if the [. . .] letter [sign] occurs as a function sign, this circumstance [should] be taken into account."
>
> (*Begriffsschrift*, Section 11)

> " 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 (Begriffsschrift's functions)

In *Function and Concept* he was aware of the fact that making a difference between first-level and second-level objects is essential to prevent paradoxes:

> "The ontological proof of God's existence suffers from the fallacy of treating existence as a first-level concept."
>
> (*Function and Concept*, p. 27, footnote)

The above discussion on functions and arguments show that Frege did indeed avoid the paradox in his Begriffsschrift.

Prehistory of Types (Grundgesetze's functions)

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

- In Grundlagen der Arithmetik [21] he argued that mathematics can be seen as a branch of logic.

- In Grundgesetze der Arithmetik [22, 26] 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.

Prehistory of Types (Grundgesetze's functions)

Frege defined "the function $\Phi(x)$ has the same *course-of-values* as the function $\Psi(x)$" by

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

- Note that functions $\Phi(x)$ and $\Psi(x)$ may have equal courses-of-values even if they have different definitions.
- 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)

Frege denoted the course-of-values of a function $\Phi(x)$ by $\grave{\varepsilon}\Phi(\varepsilon)$. The definition of equal courses-of-values could therefore be expressed as

$$\grave{\varepsilon}f(\varepsilon) = \grave{\varepsilon}g(\varepsilon) \longleftrightarrow \forall a[f(a) = g(a)]. \tag{1}$$

In modern terminology, we could say that the functions $\Phi(x)$ and $\Psi(x)$ have the same course-of-values if they have the same *graph*.

Prehistory of Types (Grundgesetze's functions)

- The notation $\grave{\varepsilon}\Phi(\varepsilon)$. may be the origin of Russell's notation $\hat{x}\Phi(x)$ for the class of objects that have the property Φ.

- According to a paper by Rosser [55], the notation $\hat{x}\Phi(x)$ has been at the basis of the current notation $\lambda x.\Phi$.

- Church is supposed to have written $\wedge x\Phi(x)$ for the function $x \mapsto \Phi(x)$, writing the hat in front of the x in order to distinguish this function from the class $\hat{x}\Phi(x)$.

Prehistory of Types (Grundgesetze's functions)

- Frege treated *courses-of-values* as *ordinary objects*.
- As a consequence, *a function that takes objects as arguments could have its own course-of-values as an argument*.
- In modern terminology: a function that takes objects as arguments can have its own graph as an argument.

Prehistory of Types (Grundgesetze's functions)

- All essential information of a function is contained in its graph.
- So intuitively, 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* from his system 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*.

Prehistory of Types (Russell's paradox in *Grundgesetze*)

- In 1902, Russell wrote a letter to Frege [56], informing him that he had *discovered a paradox* in his *Begriffsschrift* (*Begriffsschrift does not suffer from a paradox*).

- Russell gave his well-known argument, defining the propositional function $f(x)$ *by* $\neg x(x)$ (in Russell's words: "to be a predicate that cannot be predicated of itself").

- Russell assumed $f(f)$. Then by definition of f, $\neg f(f)$, *a contradiction*. Therefore: $\neg f(f)$ holds. But then (again by definition of f), $f(f)$ holds. Russell concluded that *both* $f(f)$ *and* $\neg f(f)$ hold, a contradiction.

Prehistory of Types (Russell's paradox in *Grundgesetze*)

- *Only six days later*, Frege answered Russell that *Russell's derivation of the paradox was incorrect* [25]. He explained that the *self-application* $f(f)$ *is not possible in the* Begriffsschrift. $f(x)$ is a function, which requires an *object* as an argument, and a function cannot be an object in the *Begriffsschrift*.

- In the same letter, however, Frege explained that *Russell's argument could be amended to a paradox* in the system of his Grundgesetze, using the *course-of-values* of functions.

- Frege's amendment was shortly explained in that letter, but he added an appendix of eleven pages to the second volume of his *Grundgesetze* in which he provided a very detailed and correct description of the paradox.

Prehistory of Types (Russell's paradox in *Grundgesetze*)

- Let function $f(x)$ be: $\neg\forall\varphi[(\grave{\alpha}\varphi(\alpha) = x) \longrightarrow \varphi(x)]$ and write $K = \grave{\varepsilon}f(\varepsilon)$.
- Which of $f(K)$ or $\neg f(K)$ hold?
- Actually, *both* $f(K)$ and $\neg f(K)$ hold.
- By (1), for any function $g(x)$: $\grave{\varepsilon}g(\varepsilon) = \grave{\varepsilon}f(\varepsilon) \longrightarrow g(K) = f(K)$. This implies $f(K) \longrightarrow ((\grave{\varepsilon}g(\varepsilon) = K) \longrightarrow g(K))$.
- As this holds for any function $g(x)$, we have:
  $$f(K) \longrightarrow \forall\varphi[(\grave{\varepsilon}\varphi(\varepsilon) = K) \rightarrow \varphi(K)] \qquad (a)$$

Prehistory of Types (Russell's paradox in *Grundgesetze*)

- On the other hand, for any function g,
  $\forall\varphi[(\grave{\varepsilon}\varphi(\varepsilon) = K) \to \varphi(K)] \longrightarrow ((\grave{\varepsilon}g(\varepsilon) = K) \to g(K))$.

- Substituting $f(x)$ for $g(x)$ results in:
  $\forall\varphi[(\grave{\varepsilon}\varphi(\varepsilon) = K) \to \varphi(K)] \longrightarrow ((\grave{\varepsilon}f(\varepsilon) = K) \to f(K))$

- and as $\grave{\varepsilon}f(\varepsilon) = K$ by definition of K, $\forall\varphi[(\grave{\varepsilon}\varphi(\varepsilon) = K) \to \varphi(K)] \longrightarrow f(K)$.

- Using the definition of f, we obtain
  $$\forall\varphi[(\grave{\varepsilon}\varphi(\varepsilon) = K) \to \varphi(K)] \longrightarrow \neg\forall\varphi[(\grave{\varepsilon}\varphi(\varepsilon) = K) \to \varphi(K)] \qquad (b)$$

Prehistory of Types (Russell's paradox in *Grundgesetze*)

- by (b) and reductio ad absurdum, $\neg\forall\varphi[(\grave{\alpha}\varphi(\alpha) = K) \rightarrow \varphi(K)]$, or shorthand: $f(K)$ $\hfill (c)$

- Applying (a) results in $\forall\varphi[(\grave{\alpha}\varphi(\alpha) = K) \rightarrow \varphi(K)]$, which implies $\neg\neg\forall\varphi[(\grave{\alpha}\varphi(\alpha) = K) \rightarrow \varphi(K)]$, or shorthand: $\neg f(K)$ $\hfill (d)$

- (c) and (d) *contradict* each other.

Prehistory of Types (How wrong was Frege?)

In the history of the Russell Paradox, Frege is often depicted as the pitiful person whose system was inconsistent. This suggests that Frege's system was the only one that was inconsistent, and that Frege was very inaccurate in his writings. On these points, history does Frege an injustice.

In fact, Frege's system was much more accurate than other systems of those days. Peano's work, for instance, was *less precise* on several points:

- Peano *hardly paid attention to logic* especially quantification theory;
- Peano *did not make a strict distinction* between his *symbolism* and the *objects underlying this symbolism*. Frege was much more accurate on this point (see Frege's paper *Über Sinn und Bedeutung* [23]);

Prehistory of Types (How wrong was Frege?)

- Frege *made a strict distinction* between a *proposition* (as an object) and the assertion *of a proposition*. Frege denoted a *proposition*, by $-A$, and *its assertion* by $\vdash A$. Peano did not make this distinction and simply wrote A.

Nevertheless, Peano's work was very popular, for several reasons:

- Peano had *able collaborators*, and a *better eye for presentation and publicity*.

- Peano bought *his own press* to supervise the printing of his own journal Rivista di Matematica and Formulaire [49]

Prehistory of Types (How wrong was Frege?)

- Peano used a *familiar symbolism* to the notations were used in those days.
- Many of *Peano's notations*, like $\in$ for "is an element of", and $\supset$ for logical implication, are used in *Principia Mathematica*, and are actually still in use.
- Frege's work did not have these advantages and was hardly read before 1902
- When Peano published his formalisation of mathematics in 1889 [48] he clearly did not know Frege's *Begriffsschrift* as he did not mention the work, and was not aware of Frege's formalisation of quantification theory.

Prehistory of Types (How wrong was Frege?)

- Peano considered quantification theory to be “abstruse” in [49]:

> “In this respect my conceptual notion of 1879 is superior to the Peano one. Already, at that time, I specified all the laws necessary for my designation of generality, so that nothing fundamental remains to be examined. These laws are few in number, and I do not know why they should be said to be abstruse. If it is otherwise with the Peano conceptual notation, then this is due to the unsuitable notation.”
>
> ([24], p. 376)

Prehistory of Types (How wrong was Frege?)

- In the last paragraph of [24], Frege concluded:

  "... I observe merely that the Peano notation is unquestionably more convenient for the typesetter, and in many cases takes up less room than mine, but that these advantages seem to me, due to the inferior perspicuity and logical defectiveness, to have been paid for too dearly — at any rate for the purposes I want to pursue."
  (*Ueber die Begriffschrift des Herrn Peano und meine eigene*, p. 378)

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, by defining the class $K \stackrel{\text{def}}{=} \{x \mid x \notin x\}$ and deriving $K \in K \longleftrightarrow K \notin K$.
- In *Cantor's Set Theory* one can derive the paradox via the same class (or *set*, in Cantor's terminology).

Prehistory of Types (paradoxes)

- 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* ([10], 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 [57].

Prehistory of Types (vicious circle principle)

When Russell proved Frege's *Grundgesetze* to be inconsistent, Frege was not the only person in *trouble*. In Russell's letter to Frege (1902), we read:

> "I am on the point of finishing a book on the principles of mathematics"
>
> (*Letter to Frege*, [56])

Russell *had to find a solution* to the paradoxes, before finishing his book.

His paper Mathematical logic as based on the theory of types [57] *(1908)*, in which a first step is made towards the Ramified Theory of Types, started with a description of the most important contradictions that were known up till then, including Russell's own paradox. He then concluded:

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."

(*Ibid.*)

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

Ramified Type Theory

"*Whatever involves all of a collection* must not be one *of the collection.*"

(Mathematical logic as based on the theory of types)

- Russell applies this principle *very strictly*.
- He implemented it using *types*, in particular the so-called ramified *types*.
- The type theory of 1908 was elaborated in Chapter II of the Introduction to the famous *Principia Mathematica* [60] (1910-1912).

Ramified Type Theory and Principia

- In the *Principia*, *mathematics was founded on logic*, as far as possible.

- A very *formal and accurate* build-up of mathematics, avoiding the logical paradoxes.

- The *logical part* of the *Principia* was *based* on the works of *Frege*. This was acknowledged by Whitehead and Russell in the preface, and can also be seen throughout the description of Type Theory.

- The notion of *function is based on Frege's Abstraction Principles* 1 and 2.

Ramified Type Theory and Principia

- The *Principia* *notation* $\hat{x}f(x)$ for a class looks very *similar to Frege's* $\grave{\varepsilon}f(\varepsilon)$ for course-of-values.

- An important difference is that Whitehead and Russell treated functions as first-class citizens. Frege used courses-of-values when speaking about functions.

- In the *Principia* a direct approach was possible.

- The description of the Ramified Theory of Types (RTT) in the *Principia* was, though extensive, *still informal*.

Ramified Type Theory and Principia

- Type Theory had *not yet* become an *independent subject*. The theory

  > "only recommended itself to us in the first instance by its ability to solve certain contradictions. .... it has also a certain consonance with common sense which makes it inherently credible"
  >
  > (*Principia Mathematica*, p. 37)

- Type Theory was not introduced because it was interesting on its own, but because it had to serve as a *tool* for logic and mathematics.

- A *formalisation* of Type Theory, therefore, was *not considered* in those days.

Ramified Type Theory and Principia

- Though the *description* of the ramified type theory in the *Principia* was still informal, it was *clearly present* throughout the work.

- Types in the *Principia* have a double hierarchy: *(simple) types* and *orders*.

- It was *not mentioned very often*, but when necessary, Russell made a remark on the ramified type theory.

Ramified Type Theory and Principia

- There is *no definition of "type"* in the *Principia*, only a definition of *"being of the same type"*:

  "*Definition of* being of the same type*.* The following is a step-by-step definition, the definition for higher types presupposing that for lower types. We say that u and v *are of the same type* if

  1. both are individuals,
  2. both are elementary [propositional] functions (in Principia, they only take elementary propositions as value) taking arguments of the same type,
  3. u is a pf and v is its negation,

4. u is $\varphi\hat{x}$ *$\varphi\hat{x}$ is a pf that has x as a free variable* or $\psi\hat{x}$, and v is $\varphi\hat{x} \vee \psi\hat{x}$, where $\varphi\hat{x}$ and $\psi\hat{x}$ are elementary pfs,
5. u is $(y).\varphi(\hat{x}, y)$ *forall* and v is $(z).\psi(\hat{x}, z)$, where $\varphi(\hat{x}, \hat{y})$, $\psi(\hat{x}, \hat{y})$ are of the same type,
6. both are elementary propositions,
7. u is a proposition and v is $\sim u$*negation* or
8. u is $(x).\varphi x$ and v is $(y).\psi y$, where $\varphi\hat{x}$ and $\psi\hat{x}$ are of the same type."

(*Principia Mathematica*, $*9{\cdot}131$, p. 133)

- There are some omissions in Russell and Whitehead's definition.

Ramsey's Simple Types

- The ideas behind simple types was already explained by Frege (see earlier quotes from *Function and Concept*).

- *Ramsey's Simple types*:

  1. 0 is a simple type, the type of *individuals*.
  2. If $t_1, \ldots, t_n$ are simple types, then also $(t_1, \ldots, t_n)$ is a simple type.[2] $n = 0$ is allowed: then we obtain the simple type $()$ of *propositions*.
  3. All simple types can be constructed using the rules 1 and 2.

[2] $(t_1, \ldots, t_n)$ is the type of pfs that should take n arguments, the ith argument having type t_i.

Ramsey's Simple Types

- The propositional function $\mathtt{R}(\mathtt{x})$ should have type $(\mathtt{0})$, as it takes one individual as argument.

- The proposition $S(a)$ has type $()$.

- We conclude that in $\mathtt{z}(\mathtt{R}(\mathtt{x}), \mathtt{S}(\mathtt{a}))$, we must substitute pfs of type $((\mathtt{0}), ())$ for $\mathtt{z}$. Therefore, $\mathtt{z}(\mathtt{R}(\mathtt{x}), \mathtt{S}(\mathtt{a}))$ has type $(((\mathtt{0}), ()))$.

Whitehead and Russell's Ramified Types

- With *simple types*, the type of a pf only depends on the *types of the arguments* that it can take.

- In the *Principia*, a *second hierarchy* is introduced by regarding also *the types of the variables that are bound by a quantifier* (see *Principia*, pp. 51–55).

- Whitehead and Russell consider, for instance, the propositions $\mathtt{R}(\mathtt{a})$ and $\forall \mathtt{z}{:}()[\mathtt{z}() \vee \neg \mathtt{z}()]$ to be of a *different level*.

- The first is an *atomic proposition*, while the latter is based on the pf $\mathtt{z}() \vee \neg \mathtt{z}()$.

Whitehead and Russell's Ramified Types

- The pf $\mathtt{z}() \vee \neg \mathtt{z}()$ involves an arbitrary proposition $\mathtt{z}$, therefore $\forall \mathtt{z}{:}()[\mathtt{z}() \vee \neg \mathtt{z}()]$ *quantifies over all* propositions $\mathtt{z}$.

- According to the *vicious circle principle*, $\forall \mathtt{z}{:}()[\mathtt{z}() \vee \neg \mathtt{z}()]$ cannot belong to this collection of propositions.

- This problem is solved by dividing types into *orders* which are natural numbers.

- *Basic* propositions are of order 0. In $\forall \mathtt{z}{:}()[\mathtt{z}() \vee \neg \mathtt{z}()]$ we must mention the *order of the propositions over which is quantified*. The pf $\forall \mathtt{z}{:}()^n[\mathtt{z}() \vee \neg \mathtt{z}()]$ quantifies over *all propositions of order* n, and has order $n + 1$.

Whitehead and Russell's Ramified Types

1. 0^0 is a ramified type of order 0;

2. If $t_1^{a_1}, \ldots, t_n^{a_n}$ are ramified types, and $a \in \mathbb{N}$, $a > \max(a_1, \ldots, a_n)$, then $(t_1^{a_1}, \ldots, t_n^{a_n})^a$ is a ramified type of order a (if $n = 0$ then take $a \geq 0$);

3. All ramified types can be constructed using the rules 1 and 2.

0^0; $(0^0)^1$; $\left((0^0)^1, (0^0)^4\right)^5$; and $\left(0^0, ()^2, \left(0^0, (0^0)^1\right)^2\right)^7$ are *all ramified types*.

$\left(0^0, \left(0^0, (0^0)^2\right)^2\right)^7$ is *not a ramified type*.

Predicative Types

- In the type $(0^0)^1$, all *orders are "minimal"*, i.e., not higher than strictly necessary. Unlike $(0^0)^2$ where orders are not minimal.

- Types in which all orders are minimal are called predicative and play a special role in the Ramified Theory of Types.

1. 0^0 is a predicative type;
2. If ${t_1}^{a_1}, \ldots, {t_n}^{a_n}$ are predicative types, and $a = 1 + \max(a_1, \ldots, a_n)$ (take $a = 0$ if $n = 0$), then $(t_1^{a_1}, \ldots, t_n^{a_n})^a$ is a predicative type;
3. All predicative types can be constructed using the rules 1 and 2 above.

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)** $\mathtt{x} =_{\mathrm{L}} \mathtt{y} \stackrel{\text{def}}{\leftrightarrow} \forall \mathtt{z}[\mathtt{z}(\mathtt{x}) \leftrightarrow \mathtt{z}(\mathtt{y})],$.
  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.

Axiom of Reducibility

- 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*.

- An object defined by an impredicative definition is of a higher order than the order of the elements of the class to which this object should belong. This means that the defined object has an *impredicative type*.

- But impredicativity is not allowed by the vicious circle principle.

- Russell and Whitehead tried to solve these problems with the so-called axiom of reducibility.

Axiom of Reducibility

- *(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.

- In the 2nd edition of the *Principia*, Whitehead and Russell admit:

  > "This axiom has a purely pragmatic justification: it leads to the desired results, and to no others. But clearly it is not the sort of axiom with which we can rest content."
  >
  > (*Principia Mathematica*, p. xiv)

Axiom of Reducibility

- Though Weyl [59] 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:

- One group of paradoxes is 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)

  This means that a class can never be a member of itself. The paradoxes solved by introducing the hierarchy of types (but not orders), like the Russell paradox, and the Burali-Forti paradox, are logical or syntactical paradoxes;

Deramification

- The second group of paradoxes is 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.

- Ramsey agrees with the part of the theory that eliminates the syntactic paradoxes. I.e., RTT without the orders of the types.

- The second part, the hierarchy of orders, does not gain Ramsey's support.

Deramification

- By accepting the hierarchy in its full extent one either has to accept the Axiom of Reducibility or reject ordinary real analysis.

- Ramsey is supported in his view by Hilbert and Ackermann [32].

- They all suggest a deramification of the theory, i.e. leaving out the orders of the types.

- When making a proper distinction between language and meta-language, the deramification will not lead to a re-introduction of the (semantic) paradoxes.

Deramification

- *Deramification* and the *Axiom of Reducibility* are both *violations of the Vicious Circle Principle*. Gödel [29] fills the gap why they can be *harmlessly made*

> "it seems that the vicious circle principle [. . .] *applies* only if the entities involved are *constructed by ourselves*. In this case there must clearly exist a *definition* (namely the description of the construction) which does *not refer to a totality* to which the object defined belongs, because the construction of a thing can certainly not be based on a totality of things to which the thing to be constructed itself belongs. If, however, it is a question of objects that *exist independently of our constructions*, there is *nothing in the least absurd in the existence of totalities* containing members, which can be described only by reference to this totality."
>
> (Russell's mathematical logic)

Deramification

- This turns the Vicious Circle Principle into a *philosophical principle* that will be easily *accepted by intuitionists* but that will be *rejected*, at least in its full strength, by mathematicians with a more *platonic* point of view.

- Gödel is supported in his ideas by *Quine* [50], sections 34 and 35.

- Quine's *criticism on impredicative* definitions (for instance, the definition of the least upper bound of a nonempty subset of the real numbers with an upper bound) is *not on the definition* of a special symbol, but rather on the very assumption of the existence of such an object at all.

Deramification

- Quine states that even for *Poincaré*, who was an *opponent of impredicative definitions and deramification*, one of the doctrines of *classes is that they are there "from the beginning"*. So, even for Poincaré *there should be no evident fallacy in impredicative definitions*.

- The *deramification* has played an *important role* in the development of *type theory*. In *1932 and 1933*, Church presented his (untyped) *λ-calculus* [14, 15]. *In 1940* he combined this theory with a deramified version of Russell's theory of types to the system that is known as the simply typed λ-calculus

The Simple Theory of Types

- Ramsey [51], and Hilbert and Ackermann [32], *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 Simply Typed λ-calculus $\lambda{\rightarrow}$

- Types and terms in the original $\lambda{\rightarrow}$ are a bit different from those of [4].
- The *types* of $\lambda{\rightarrow}$ are defined as follows:
  - ι *individuals* and o *propositions* are types;
  - If α and β are types, then so is $\alpha \rightarrow \beta$.
- The *terms* of $\lambda{\rightarrow}$ are the following:
  - $\neg$, $\wedge$, $\forall_\alpha$ for each type α, and $\imath_\alpha$ for each type α, are terms;
  - A *variable* is a term;
  - If A, B are terms, then so is AB;
  - If A is a term, and x a variable, then $\lambda x{:}\alpha.A$ is a term.

Typing rules in Church's Simply Typed λ-calculus $\lambda\!\!\to$

- $\Gamma \vdash \neg : o \to o$;

  $\Gamma \vdash \wedge : o \to o \to o$;

  $\Gamma \vdash \forall_\alpha : (\alpha \to o) \to o$;

  $\Gamma \vdash \imath_\alpha : (\alpha \to o) \to \alpha$;

- $\Gamma \vdash x : \alpha$ if $x{:}\alpha \in \Gamma$;

- If $\Gamma, x{:}\alpha \vdash A : \beta$ then $\Gamma \vdash (\lambda x{:}\alpha.A) : \alpha \to \beta$;

- If $\Gamma \vdash A : \alpha \to \beta$ and $\Gamma \vdash B : \alpha$ then $\Gamma \vdash (AB) : \beta$.

Comparing $\lambda\!\!\to$ with STT and RTT

- Apart from the orders, RTT *is a subsystem of* $\lambda\!\!\to$.

- The rules of RTT, and the method of deriving the types of pfs have a *bottom-up character*: one can only introduce a variable of a certain type in a context Γ, if there is a pf that has that type in Γ. In $\lambda\!\!\to$, one can introduce variables of any type without wondering whether such a type is inhabited or not.

- Church's $\lambda\!\!\to$ is more general than RTT in the sense that Church does not only describe (typable) *propositional* functions. In $\lambda\!\!\to$, also functions of type $\tau \to \iota$ (where ι is the type of individuals) can be described, and functions that take such functions as arguments, etc..

Where to find above details

- Details concerning the abve slides can be downloaded from

  http://www.cee.hw.ac.uk/ fairouz/talks/brasilia2002.ps

- See also
  - Types in Logic and Mathematics before 1940 by Kamareddine, F. and Laan, L. and Nederpelt, R. P. Bulletin of Symbolic Logic, May 2002. and
  - The Evolution of Type Theory in Logic and Mathematics by Twan Laan, PhD thesis 1997.

Item Notation/Lambda Calculus à la de Bruijn

- For those used to classical notation, $\mathcal{I}$ translates to item notation:

$$\mathcal{I}(x) = x, \qquad \mathcal{I}(\lambda x.B) = [x]\mathcal{I}(B), \qquad \mathcal{I}(AB) = (\mathcal{I}(B))\mathcal{I}(A)$$

- For example, $\mathcal{I}((\lambda x.(\lambda y.xy))z) = (z)[x][y](y)x$. The *items* are (z), $[x]$, $[y]$ and (y).

- The *applicator wagon* (z) and *abstractor wagon* $[x]$ occur NEXT to each other.

- In classical notation the β rule is $(\lambda x.A)B \to_\beta A[x := B]$. In item notation, the rule is:

$$(B)[x]A \to_\beta [x := B]A$$

Redexes in Item Notation

Classical Notation	Item Notation
$(\underline{(\lambda_x.(\lambda_y.\lambda_z.zd)c)b)}a$	$(a)\underline{(b)[x]}(c)[y][z](d)z$
$\downarrow_\beta$	$\downarrow_\beta$
$(\underline{(\lambda_y.\lambda_z.zd)c)}a$	$(a)\underline{(c)[y]}[z](d)z$
$\downarrow_\beta$	$\downarrow_\beta$
$\underline{(\lambda_z.zd)a}$	$\underline{(a)[z]}(d)z$
$\downarrow_\beta$	$\downarrow_\beta$
ad	$(d)a$

$$(a)\ (b)\ [x]\ (c)\ [y]\ [z]\ (d)\ \ z$$

Segments, Partners, Bachelors

- The "bracketing structure" of the classical notation $((\lambda_x.(\lambda_y.\lambda_z.--)c)b)a)$, is '$\{_1\ \{_2\ \{_3\ \}_2\ \}_1\ \}_3$', where '$\{_i$' and '$\}_i$' match.

- In item notation, $(a)(b)[x](c)[y][z](d)$ has the simpler bracketing structure $\{\{\ \}\{\ \}\}$.

- An applicator (a) and an abstractor $[x]$ are *partners* when they match like '$\{$' and '$\}$'. Non-partnered items are *bachelors*. A *segment* $\overline{s}$ is *well balanced* when it contains only partnered items.

- Example: Let $\overline{s} \equiv (a)(b)[x](c)[y][z](d)$. Then: The items (a), (b), $[x]$, (c), $[y]$, and $[z]$ are partnered. The item (d) is a bachelor. The segment $(a)(b)[x](c)[y][z]$ is well balanced.

More on Segments, Partners, and Bachelors

Consider some term $\overline{s}x$. Some facts:

- The *main* items in $\overline{s}$ are those at top level, not within some applicator (a).
- Each main bachelor abstractor $[x]$ precedes each main bachelor applicator (a).
- Removing all main bachelor items from $\overline{s}$ yields a well balanced segment.
- Removing all main partnered items from $\overline{s}$ yields a segment $[v_1]\dots[v_n](a_1)\dots(a_m)$ consisting of all main bachelor abstractors followed by all main bachelor applicators.
- If $\overline{s}$ is of the form $\overline{s_1}(b)\overline{s_2}[v]\overline{s_3}$ where $[v]$ and (b) are partnered, then $\overline{s_2}$ must be well balanced.

Even More on Segments, Partners, and Bachelors

Each non-empty segment $\overline{s}$ has a unique *partitioning* into sub-segments $\overline{s} = \overline{s_0}\overline{s_1} \cdots \overline{s_n}$ such that

- For even i, the segment $\overline{s_i}$ is well balanced. For odd i, the segment $\overline{s_i}$ is a bachelor segment, i.e., it contains only bachelor main items.
- All well balanced segments after the first and all bachelor segments are non-empty.
- If $\overline{s_i} = [x_1] \cdots [x_m]$ (only abstractor main items) and $\overline{s_j} = (a_1) \cdots (a_p)$ (only applicator main items), then $i < j$, i.e., $\overline{s_i}$ precedes $\overline{s_j}$ in $\overline{s}$.

Example

$\overline{s} \equiv [x][y](a)[z][x'](b)(c)(d)[y'][z'](e)$, has the following partitioning:

- well-balanced segment $\overline{s_0} \equiv \emptyset$
- bachelor segment $\overline{s_1} \equiv [x][y]$,
- well-balanced segment $\overline{s_2} \equiv (a)[z]$,
- bachelor segment $\overline{s_3} \equiv [x'](b)$,
- well-balanced segment $\overline{s_4} \equiv (c)(d)[y'][z']$,
- bachelor segment $\overline{s_5} \equiv (e)$.

More on Item Notation

- Above discussion and further details of item notation can be found in [35, 36].

- Item notation helped greatly in the study of a one-sorted style of explicit substitutions, the λs-style which is related to $\lambda\sigma$, but has certain simplifications [37, 38].

Canonical Forms

- Item notation helps in finding nice canonical forms. The term

$$[x][y](a)[z][x'](b)(c)(d)[y'][z'](e)$$

  is equivalent to

$$[x][y][x'](a)[z](c)(d)[y'][z'](b)(e)$$

  and also

$$[x][y][x'](a)[z](d)[y'](c)[z'](b)(e)$$

- Nice canonical forms look like:

bachelor [ ]s	()[ ]-pairs, A_i in CF	bachelor ()s, B_i in CF	end var
$[x_1]\ldots[x_n]$	$(A_1)[y_1]\ldots(A_m)[y_m]$	$(B_1)\ldots(B_p)$	x

- In classical notation:

$$\lambda x_1 \cdots \lambda x_n.(\lambda y_1.(\lambda y_2. \cdots (\lambda y_m.xB_p \cdots B_1)A_m \cdots)A_2)A_1$$

Obtaining Canonical Forms

The results of going to normal form for the indicated reduction rules, in the order shown:

θ:		$()[\,]$-pairs mixed with bach. $[\,]$s $(A_1)[x][y][z](A_2)[p]\cdots$	bach. $()$s $(B_1)(B_2)\cdots$	end var x
γ:	bach. $[\,]$s $[x_1][x_2]\cdots$	$()[\,]$-pairs mixed with bach. $()$s $(B_1)(A_1)[x](B_2)\cdots$		end var x
θ, γ:	bach. $[\,]$s $[x_1][x_2]\cdots$	$()[\,]$-pairs $(A_1)[y_1](A_2)[y_2]\ldots(A_m)[y_m]$	bach. $()$s $(B_1)(B_2)\ldots$	end var x
γ, θ:	bach. $[\,]$s $[x_1][x_2]\cdots$	$()[\,]$-pairs $(A_1)[y_1](A_2)[y_2]\ldots(A_m)[y_m]$	bach. $()$s $(B_1)(B_2)\ldots$	end var x

More on Canonical Forms

- Both $\theta(\gamma(A))$ and $\gamma(\theta(A))$ are in *canonical form* and we have that $\theta(\gamma(A)) =_p \gamma(\theta(A))$ where $\rightarrow_p$ is the rule

$$(A_1)[y_1](A_2)[y_2]B \rightarrow_p (A_2)[y_2](A_1)[y_1]B \qquad \text{if } y_1 \notin \mathrm{FV}(A_2)$$

- For a term A, we define: $[A] = \{B \mid \theta(\gamma(A)) =_p \theta(\gamma(B))\}$.

- When $B \in [A]$, we write that $B \approx_{\texttt{equi}} A$.

- One-step class-reduction $\rightsquigarrow_\beta$ is the least compatible relation such that:

$$A \rightsquigarrow_\beta B \quad \text{iff} \quad \exists A' \in [A]. \exists B' \in [B].\ A' \to_\beta B'$$

- Classes ($[A]$) and class reduction ($\rightsquigarrow_\beta$) nicely preserve various strong normalization properties.

- Define $A \rightsquigarrow_\beta^{(E)[x]} B$ iff $\exists A' \in [A]. \exists B' \in [B]. \exists E' \in [E].\ A' \to_\beta^{(E')[x]} B'$.

Theorem 1. *If* $A \approx_{\texttt{equi}} C$ *and* $A \rightsquigarrow_{\beta}^{(E)[x]} B$ *then*

$(\exists D, E')[B \approx_{\texttt{equi}} D, E' \approx_{\texttt{equi}} E,$ *and* $C \rightarrow_{\beta}^{(E')[x]} D]$.

$$
\begin{tikzcd}
A \arrow[r, squiggly, "(E){[x]}"] \arrow[d, phantom, "\approx_{\texttt{equi}}"] & B \arrow[d, phantom, "\approx_{\texttt{equi}}"] \\
C \arrow[r, squiggly, dashed, "(E'){[x]}"] & D
\end{tikzcd}
$$

A Few Uses of Generalised Reduction and Term Reshuffling

- [52] uses term reshuffling and generalized reduction in analyzing perpetual reduction strategies.
- Term reshuffling is used in [41], [39] in analyzing typability problems.
- [46, 18, 40] use generalised reduction and/or term reshuffling in relating SN to WN.
- [2] uses a form of term-reshuffling in obtaining a calculus that corresponds to lazy functional evaluation.
- [35, 8, 7] showed how generalized reduction and term reshuffling could reduce space/time needs.
- [34] shows various strong properties of generalised reduction.

The Low Level approach of functions

- Historically, functions have long been treated as a kind of meta-objects.

- Function *values* have always been important, but abstract functions have not been recognised in their own right until the third of the 20th century.

- In the *low level approach* or *operational* view on functions, there are no functions as such, but only function values.

- E.g., the sine-function, is always expressed together with a value: $\sin(\pi)$, $\sin(x)$ and properties like: $\sin(2x) = 2\sin(x)\cos(x)$.

- It has long been usual to call $f(x)$—and not f—the function and this is still the case in many introductory mathematics courses.

The revolution of treating functions as first class citizens

- In the nowadays accepted view on functions, they are 'first class citizens'.
- Abstraction and application form the basis of the λ-calculus and type theory.
- This is rigid and does not represent the development of logic in 20th century.
- Frege and Russell's conceptions of functional abstraction, instantiation and application do not fit well with the λ-calculus approach.
- In *Principia Mathematica* [60]: If, for some a, there is a proposition ϕa, then there is a function $\phi\hat{x}$, and vice versa.
- The function ϕ is not a separate entity but always has an argument.

λ-calculus does not fully represent functionalisation

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

- Cannot identify the original term from which a function has been abstracted.

$$\texttt{let}\ \texttt{add}_2 = (\lambda x.x + 2)\ \texttt{in}\ \texttt{add}_2(x) + \texttt{add}_2(y)$$

- 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.

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 [43].

- 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 worthwile 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.

- The representation of a mathematical text in AUTOMATH consists of a finite list of *lines* where every line has the following format:

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

  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 [6] revealed that this combined definition and parameter mechanism is vital for keeping proofs manageable and sufficiently readable for humans.

The Barendregt Cube

- $\mathcal{T}_P \ ::= \mathcal{V} \mid \boldsymbol{S} \mid \mathcal{T}_P\mathcal{T}_P \mid \lambda\mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P \mid \Pi\mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P$

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

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

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

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

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

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

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

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

Different type formation conditions

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

- $(\square, *)$ takes care of polymorphism. $\lambda 2$ is weakest on cube satisfying this.

- $(\square, \square)$ takes care of type constructors. $\lambda\underline{\omega}$ is weakest on cube satisfying this.

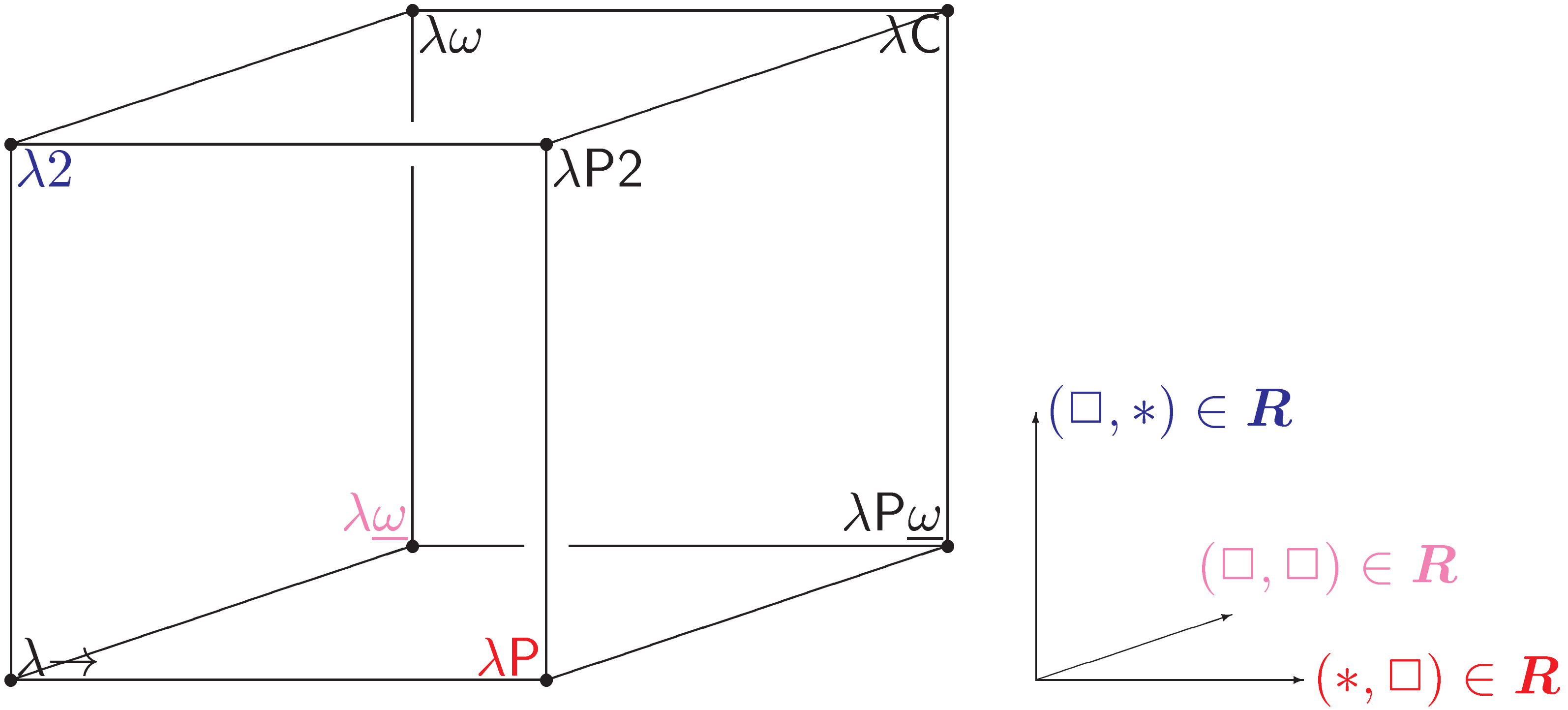
- $(*, \square)$ takes care of term dependent types. λP is weakest on cube satisfying this.

$\lambda\!\to$	$(*, *)$			
$\lambda 2$	$(*, *)$	$(\square, *)$		
λP	$(*, *)$		$(*, \square)$	
$\lambda\underline{\omega}$	$(*, *)$			$(\square, \square)$
λP2	$(*, *)$	$(\square, *)$	$(*, \square)$	
$\lambda\omega$	$(*, *)$	$(\square, *)$		$(\square, \square)$
λP$\underline{\omega}$	$(*, *)$		$(*, \square)$	$(\square, \square)$
λC	$(*, *)$	$(\square, *)$	$(*, \square)$	$(\square, \square)$

Systems of the Barendregt Cube

System	Rel. system	Names, references
$\lambda{\to}$	λ^{τ}	simply typed λ-calculus; [16], [3] (Appendix A), [33] (Chapter 14)
$\lambda 2$	F	second order typed λ-calculus; [28], [54]
λP	AUT-QE	[9]
	LF	[30]
λP2		[44]
$\lambda\underline{\omega}$	POLYREC	[53]
$\lambda\omega$	Fω	[28]
λC	CC	Calculus of Constructions; [17]

The Barendregt Cube



LF

- LF (see [30]) is often described as λP of the Barendregt Cube.

- [27] shows that the use of the Π-formation rule $(*, \Box)$ is very restricted in the practical use of LF.

- This use is in fact based on a parametric construct rather than on Π-formation.

- We will find a more precise position of LF on the Cube (between $\lambda\!\rightarrow$ and λP).

ML

- We only consider an explicit version of a subset of ML.
- In ML, One can define the polymorphic identity by:

$$\texttt{Id}(\alpha{:}*) = (\lambda x{:}\alpha.x) : (\alpha \to \alpha) \tag{2}$$

- But in ML, it is not possible to make an explicit λ-abstraction over $\alpha : *$ by:

$$\texttt{Id} = (\lambda\alpha{:}*.\lambda x{:}\alpha.x) : (\Pi\alpha{:}*.\alpha \to \alpha) \tag{3}$$

- The type $\Pi\alpha{:}*.\alpha \to \alpha$ does not belong to the language of ML and hence the λ-abstraction of equation (3) is not possible in ML.

ML

- Therefore, we can state that ML does not have a Π-formation rule $(\Box, *)$.
- Nevertheless, ML has some parameter mechanism (α parameter of `Id`)
- ML has limited access to the rule $(\Box, *)$ enabling equation (2) to be defined.
- ML's type system is none of those of the eight systems of the Cube.
- We place the type system of ML on our refined Cube (between $\lambda 2$ and $\lambda\underline{\omega}$).

Extending the Cube with parametric constructs

- Parametric constructs are $c(b_1, \ldots, b_n)$ with $b_1, \ldots, b_n$ terms of certain types.

- $\mathcal{T}_P ::= \mathcal{V} \mid S \mid \mathcal{C}(\underbrace{\mathcal{T}_{P_1}, \ldots \mathcal{T}_{P_n}}_{n \geq 0}) \mid \mathcal{T}_P \mathcal{T}_P \mid \lambda \mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P \mid \Pi \mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P$

  $\mathcal{C}$ is a set of constants, $b_1, \ldots, b_n$ are called the *parameters* of $c(b_1, \ldots, b_n)$.

- $\mathfrak{S}$ allows several kinds of Π-constructs. We also use a set ¶ of (s_1, s_2) where $s_1, s_2 \in \{*, \square\}$ to allow several kinds of parametric constructs.

- $(s_1, s_2) \in$ ¶ means that we allow parametric constructs $c(b_1, \ldots, b_n) : A$ 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 $(*, s_2) \in$ ¶ and $(\square, s_2) \in$ ¶ then combinations of parameters allowed. For example, it is allowed that B_1 has type $*$, whilst B_2 has type $\square$.

The Cube with parametric constants

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

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

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

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

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

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 $\mathsf{SN}_{\twoheadrightarrow_\beta}(M)$. I.e. M is strongly normalising with respect to $\twoheadrightarrow_\beta$.

- Other properties such as Uniqueness of types and typability of subterms hold.

- $\lambda\mathfrak{S}\P$ is the system which has Π-formation rules $\boldsymbol{R}$ and parameter rules $\boldsymbol{P}$.

- Let $\lambda\mathfrak{S}\P$ 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 $\lambda\mathfrak{S}\P$.
  - If $\Gamma \vdash_{\mathfrak{S}\P} a : A$ then $\{\Gamma\} \vdash_{\mathfrak{S}} \{a\} : \{A\}$.

Example

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

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

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

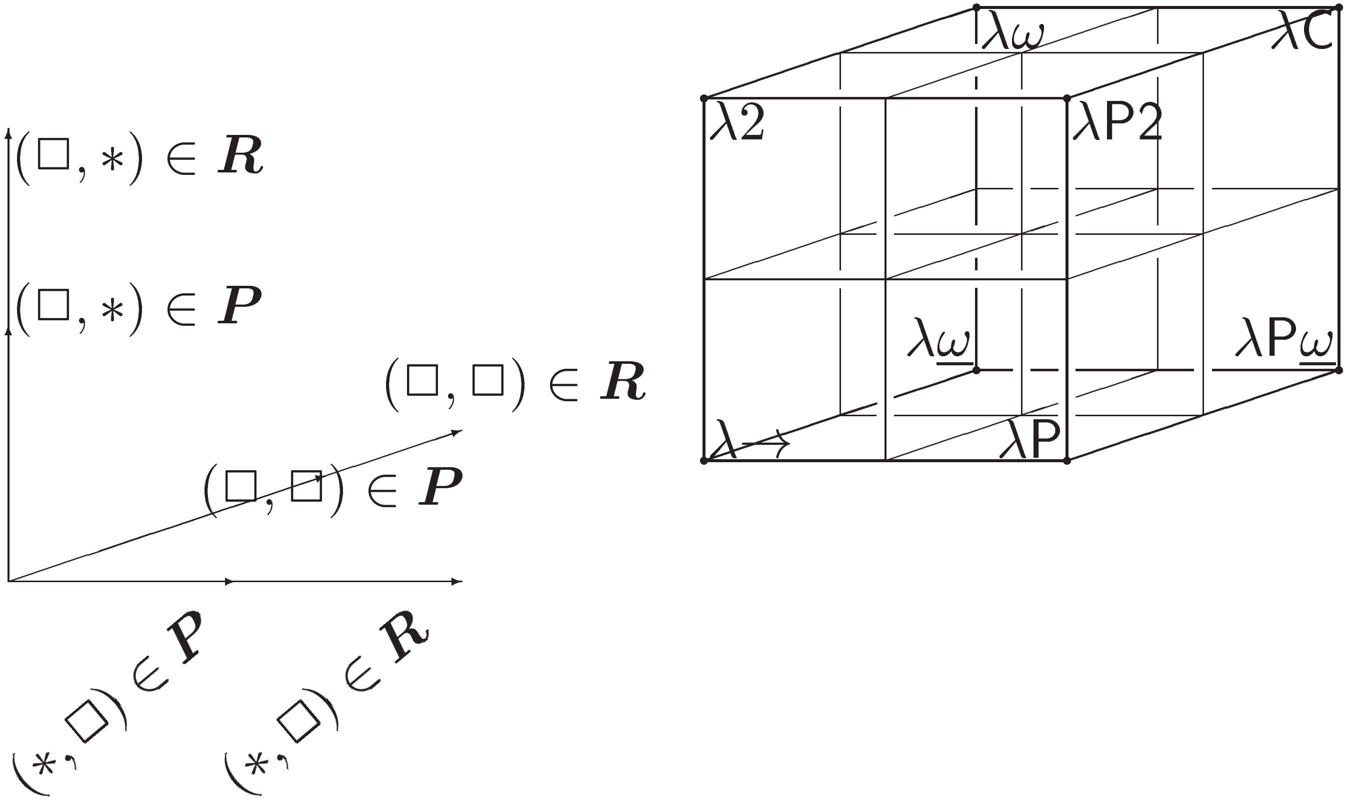
- $\boldsymbol{R}_5 = \{(*, *)\} \qquad \boldsymbol{P}_5 = \{(*, *), (*, \Box)\}$.

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

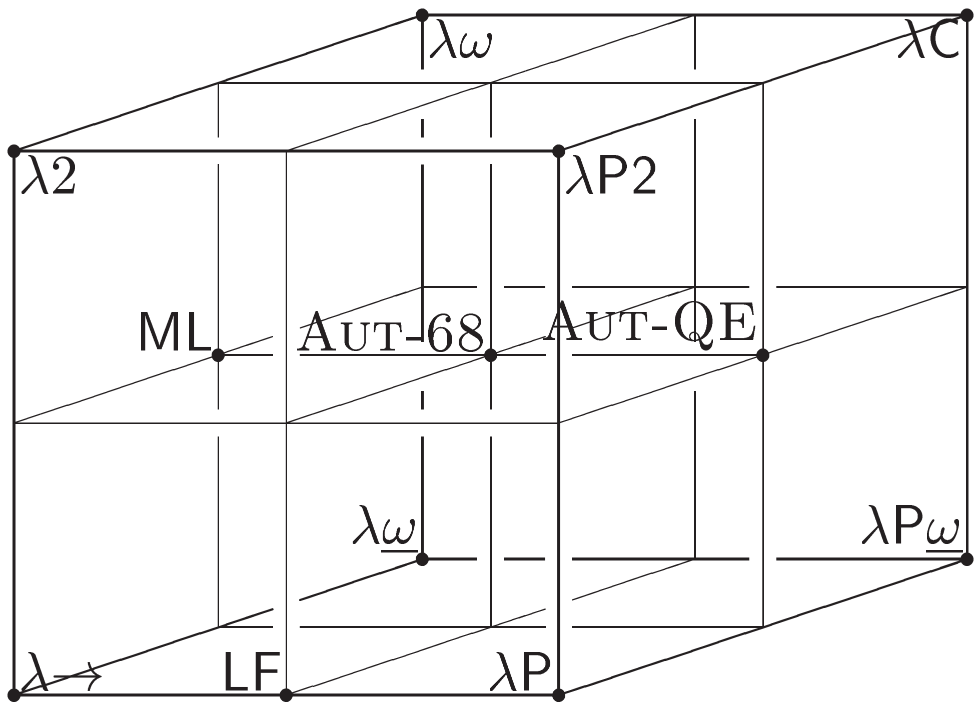
$$
\begin{array}{rcl}
\alpha & : & *, \\
\mathtt{eq}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha) & : & *, \\
\mathtt{refl}(\mathtt{x}{:}\alpha) & : & \mathtt{eq}(\mathtt{x}, \mathtt{x}), \\
\mathtt{symm}(\mathtt{x}{:}\alpha, \mathtt{y}{:}\alpha, \mathtt{p}{:}\mathtt{eq}(\mathtt{x}, \mathtt{y})) & : & \mathtt{eq}(\mathtt{y},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

  eq not possible in $\lambda{\to}$.

The refined Barendregt Cube



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



LF

- [27] initially described LF as the system λP of the Cube.
  However, the Π-formation rule $(*, \Box)$ is restricted in most applications of LF.

- [27] splits λ-formation in two (LF – (λ_P) is called LF$^-$):

$$(\lambda_0)\frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash \Pi x{:}A.B : *}{\Gamma \vdash \lambda_0 x{:}A.M : \Pi x{:}A.B} \qquad (\lambda_0 x{:}A.M)N \to_{\beta_0} M[x{:=}N]$$

$$(\lambda_P)\frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash \Pi x{:}A.B : \Box}{\Gamma \vdash \lambda_P x{:}A.M : \Pi x{:}A.B} \qquad (\lambda_P x{:}A.M)N \to_{\beta_P} M[x{:=}N]$$

- If $M : *$ or $M : A : *$ in LF, then the β_P-normal form of M contains no λ_P;

- If $\Gamma \vdash_{\mathsf{LF}} M : A$, and Γ, M, A do not contain a λ_P, then $\Gamma \vdash_{\mathsf{LF}^-} M : A$;

- If $\Gamma \vdash_{\mathsf{LF}} M : A(: *)$, all in β_P-normal form, then $\Gamma \vdash_{\mathsf{LF}^-} M : A(: *)$.

LF

- Hence: the only need for a type $\Pi x{:}A.B : \Box$ is to declare a variable in it.

- This is only done when the Propositions-As-Types principle PAT is applied during the construction of the type of the operator Prf as follows:

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

- In LF, this is the only point where the Π-formation rule $({*}, \Box)$ is used.

- No λ_P-abstractions are used. Prf is only used when applied to term $p{:}\texttt{prop}$.

- 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.

- This puts LF in between $\lambda{\rightarrow}$ and λP 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 $\boldsymbol{R} = \{(*,*),(*,\Box),(\Box,*)\}$:
$$\texttt{Ind} = \Pi p{:}(\mathbb{N}\to *).p0\to(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn\to Snm\to pm)\to\Pi n{:}\mathbb{N}.pn \tag{4}$$

- 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):

$$\texttt{ind}(p{:}\mathbb{N}\to *, q{:}p0, r{:}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn\to Snm\to pm)) : \Pi n{:}\mathbb{N}.pn \tag{5}$$

- The logician's type `Ind` is not needed by the mathematician and the types that occur in 5 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.

Conclusions

- 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 [43, 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.

References

[1] S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors. *Handbook of Logic in Computer Science, Volume 2: Background: Computational Structures*. Oxford University Press, 1992.

[2] 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.

[3] 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.

[4] H.P. Barendregt. Lambda calculi with types. In [1], pages 117–309. Oxford University Press, 1992.

[5] P. Benacerraf and H. Putnam, editors. *Philosophy of Mathematics*. Cambridge University Press, second edition, 1983.

[6] 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).

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

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

[9] 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 [47], pages 73–100.

[10] C. Burali-Forti. Una questione sui numeri transfiniti. *Rendiconti del Circolo Matematico di Palermo*, 11:154–164, 1897. English translation in [31], pages 104–112.

[11] G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (Erster

Artikel). *Mathematische Annalen*, 46:481–512, 1895.

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

[13] 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.

[14] A. Church. A set of postulates for the foundation of logic (1). *Annals of Mathematics*, 33:346–366, 1932.

[15] A. Church. A set of postulates for the foundation of logic (2). *Annals of Mathematics*, 34:839–864, 1933.

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

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

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

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

[20] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. Also in [31], pages 1–82.

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

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

[23] G. Frege. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik,* new series, 100:25–50, 1892. English translation in [45], pages 157–177.

[24] G. Frege. Ueber die Begriffschrift des Herrn Peano und meine eigene. *Berichte über die Verhandlungen der Königlich Sächsischen Gesellschaft der Wissenschaften zu Leipzig, Mathematisch-physikalische Klasse 48*, pages 361–378, 1896. English translation in [45], pages 234–248.

[25] G. Frege. Letter to Russell. English translation in [31], pages 127–128, 1902.

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

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

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

[29] K. Gödel. Russell's mathematical logic. In P.A. Schlipp, editor, *The Philosophy of Bertrand Russell*. Evanston & Chicago, Northwestern University, 1944. Also in [5], pages 447–469.

[30] 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.

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

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

[33] 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.

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

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

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

[37] Fairouz Kamareddine and Alejandro Ríos. A λ-calculus à la de Bruijn with explicit substitution. In *7th Int'l Symp. Prog. Lang.: Implem., Logics & Programs*, volume 982 of *LNCS*, pages 45–62. Springer-Verlag, 1995.

[38] Fairouz Kamareddine and Alejandro Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *J. Funct. Programming*, 7(4):395–420, 1997.

[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 Computer Sci.*, pages 311–321, 1995.

[41] Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. The hierarchy of finitely typed functional programs. In *Proc. 2nd Ann. Symp. Logic in Computer Sci.*, pages 225–235, 1987.

[42] T. Laan. *The Evolution of Type Theory in Logic and Mathematics*. PhD thesis, Eindhoven University of Technology, 1997.

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

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

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

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

[47] 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.

[48] G. Peano. *Arithmetices principia, nova methodo exposita*. Bocca, Turin, 1889. English translation in [31], pages 83–97.

[49] G. Peano. *Formulaire de Mathématique*. Bocca, Turin, 1894–1908. 5 successive versions; the final edition issued as *Formulario Mathematico*.

[50] W. Van Orman Quine. *Set Theory and its Logic*. Harvard University Press, Cambridge, Massachusetts, 1963.

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

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

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

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

[55] J.B. Rosser. Highlights of the history of the lambda-calculus. *Annals of the History of Computing*, 6(4):337–349, 1984.

[56] B. Russell. Letter to Frege. English translation in [31], pages 124–125, 1902.

[57] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908. Also in [31], pages 150–182.

[58] M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924. Also in [31], pages 355–366.

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

[60] 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.