Modern eyes on λ-calculus✍

Patrik Eklund[a], Robert Helgesson[a]

[a] *Department of Computing Science, Umeå University, Sweden*

Abstract

In this paper we consider levels of signatures as underlying typed λ terms, i.e., being very observant about where particular types and related operators reside especially before and after λ-abstraction. In this framework, we start with a (many-sorted) signature identifying the underlying primitive operations on level one. On level two, building on the signature of primitive operations we form the type-level signature with associated type terms and type algebras. Finally, on level three we construct a signature of lambda terms. We are in this notion of λ signatures able to show, e.g., how problems with variable renaming can be avoided. We will also show how these notions, involving a strict use of term monads also for many-sorted signatures, can be used as 'modern eyes' in order to provide insight into intuitive approaches appearing e.g. in fundamental work by Schönfinkel, Curry and Church.

Key words: lambda terms, term monads, signatures

1. Introduction

Hilbert's view was that mathematical logic is only part of *Foundations of Mathematics*. This is confirmed by Curry in his review of the 3rd edition of Hilbert's and Ackermann's *Grundzüge* [1], Curry there states

> The book was intended as an introductory textbook of mathematical logic in a narrow sense. The Hilbert school never subscribed to the identification of mathematics and logic, and regarded "mathematical logic", "theoretical logic", and "logical calculus" as synonymous designations for a preliminary stage in the subject of "foundations of mathematics", which many Americans prefer to call "mathematical logic" in a broader sense (cf. Quine's book of 1940). Anything depending on an axiom of infinity or similar assumption

✍This paper is based on an unpublished manuscript, content of which was presented at the International Workshop on 75 Years of the λ-Calculus, St Andrews (Scotland), 15 June 2012.

Email addresses: peklund@cs.umu.se (Patrik Eklund), rah@cs.umu.se (Robert Helgesson)

> would belong to the latter subject but not to the former. Nevertheless, the authors regard the narrower subject as an essential step to the broader. [2]

Curry was formally Hilbert's student, even if more supervised by Bernays, so Curry was well aware of the *Hilbert school* in Göttingen.

Grundzüge's *engere Funktionenkalkül* was indeed first-order predicate calculus but as logic by Hilbert seen in a narrow (engere) sense. Interesting is that *Grundzüge* brings up the importance of types, and that *Grundzüge* basically is based on Hilbert's lectures during 1917–1922. Schönfinkel presented his *Bausteine* in Göttingen 1920, and his paper [3] was eventually published in 1924. Schönfinkel didn't use types, but apparently since he wanted to avoid them rather than not being aware of them. Abstracting away from variables was perhaps seen as abstracting away from types as well. Schönfinkel's aim was therefore to provide an abstraction for this function calculus rather than aiming at bringing this abstraction to some level above logic and being a foundation for logic. Types were implicit in functions and propositions. Further abstractions and dealing with propositions as types turned out to be more difficult than expected as seen, as Schönfinkel's realized when dealing with his J function. Later efforts to move in direction of types as foundations is seen e.g. the Brouwer–Heyting–Kolmogorov interpretation and the Curry–Howard isomorphism, with "propositions are types" and "proofs are programs". However, later developments and also implementations of these efforts reveal that additional abstract level types need to be introduced, being kept outside the machinery. These machineries seem still to suffer, at least partly, from what Curry stated[1] already in his [4].

In what follows we will show how our strict categorical machinery is able to say clearly what belongs and what doesn't. We will provide some remarks, respectively, for Schönfinkel's [3], Curry's [4] and Church's [5]. This has bearing for providing remarks elsewhere as well, both on earlier work like Whitehead's and Russell's *Principia Mathematica*, and on later developments of type theory in various functional programming languages, none mentioned and none forgotten[2]. Also, *Principia Mathematica* is more about 'types of sets' than about 'types of functions', so it leans more on 'set theory' than 'type theory'. Functional programming on the other hand comes with interference of language implementation issues, and even worse, with allowance of imperative constructions as ad hoc to functional calculi, which opens the door also to entirely different 'logic considerations' in the style of Hoare and Dijkstra.

---

[1] *For many of these contradictions appear to arise from applying the rules, appropriate to a certain category of entities, to an entity which seems to belong to that category but in reality does not do so.*

[2] It is hard to find the name *Moisei* as the name for a functional programming language or a fraction thereof, given that Moses Ilyich Schönfinkel also is known as Moisei Isaievich Sheinfinkel.

Therefore, for the purpose of this paper we focus basically only on these three papers [3, 4, 5], that are selected as representatives for the path starting from Schönfinkel's *Bausteine* indeed providing inspiration for Curry's *Combinators* in his *combinatory logic* [6], in turn followed by Church's initiating work [7] on the λ-*calculus*.

2. Types and operators - Signature and terms

Informal definitions of the terms do not necessarily lead to ambiguities, but they lead to an "end-point" so that the term set is not formally formulated for further manipulation and extension. Formal categorical definitions and constructions term monads were given in [8, 9], and opens up possibilities to generalize term monads in various directions. On the one hand, we may compose monads (possibly also partially ordered monads) with term monads, using distributive laws [10]. This intuitively gives us a means to manage rich sets of terms, and yet providing substitutions and compositions of substitutions appropriately.

Concerning λ-terms, the situation with informal definitions about what is and isn't a λ-term is less obvious, in particular in the typed case. We can make this situation explicit by considering levels of signatures, i.e. being very observant about where particular types and related operators reside especially before and after λ-abstraction. Type constructors also need to be handled formally, and their respective algebras must be identified with utmost care.

Starting from a usual signature $\Sigma = (S, \Omega)$, identifying the underlying primitive operations, we have the term monad $\mathbf{T}_\Sigma$, over $\mathtt{Set}$, the category of sets and functions. This situation is *signatures, terms and algebras at level one*.

Then we may create a new signature $\mathtt{S}_\Sigma = (\{\mathtt{type}\}, \Omega')$, on *signature level two*, with $\mathtt{type}$ as the only sort, and operators in Ω' to be understood as type constructors. Interesting on level two is the algebra of $\mathtt{type}$, namely, $\mathfrak{A}(\mathtt{type})$ is the underlying category of your choice.

Now we can make $\mathsf{T}_{\mathtt{S}_\Sigma}\varnothing$ the sort set for *signature level three*, and the interesting part is defining some operators into this signature. Here $\mathsf{T}_{\mathtt{S}_\Sigma}$ is the term functor, where term functors in general indeed require a formal treatment.

In this separation of levels it is very transparent how e.g. operators at level one are shifted over to level three. The most important observation at this stage is that λ is not a 'term transformer' but an 'operator mover' between level one and level three.

All this notions can be made precise, and we are able to show e.g. how problems with variable renaming can be avoided. Our paper touches also upon foundational questions, such to be or not to be a predicativist. In this paper (at least) we are not. We see *logic for mathematics* being the first-order approach developing hand in hand with axiomatic set theory, providing ZFC, Zermelo–Fraenkel's set theory including the Axiom of Choice, as the metalanguage (including appropriate intuitions for *conglomerates* and *universe*) for the object language *category theory*. In turn, when we move over to defining our

view of *mathematical logic*, as represented e.g. by our substitution logic, category theory becomes the metalanguage for the object language *substitution logic*.

In this strictly hierarchical approach we forbid moving back and forth, as Gödel frequently did, and indeed remain strict when representation terms, sentences and proofs in logic. Gödel's numbering indeed comes to a proof calculus, using proof trees and provides numberings for sentences appearing in proof trees, then producing predicates involving these numberings, and goes back to the set of sentences and throws this new sentences into the bag of old sentences. This was allowed one hundred years ago, and some still allow it. We don't, and indeed for our families of logic enabled by the framework of substitution logic. Whatever happened before ZFC became ZFC, is here not of our concern. We trust ZFC and we trust ZFC as the metalanguage, not *a* metalanguage, for category theory. Substitution logic must use categorical notions only. No by-passing of this principle is allowed. Expectedly, this will echo to other areas of type theory as well, e.g. including the Curry–Howard isomorphism, and modern approaches to "types as propositions".

3. Type constructors, term monads and λ-abstractions

This section relies on the term construction and notation as presented in [11]. For type theory in general, we need to integrate type constructors that appear as operators in such a 'different level' signature to a particular many-sorted signature $\Sigma = (S, \Omega)$. To be more precise, a (Σ-)*superseding type signature* is a one-sorted signature $\mathtt{S}_\Sigma = (\{\mathtt{type}\}, \mathtt{Q})$, where $\mathtt{Q}$ is a set of *type constructors* satisfying

(i) $\mathtt{s} :\to \mathtt{type}$ is in $\mathtt{Q}$ for all $\mathtt{s} \in S$

(ii) there is a $\Rrightarrow\colon \mathtt{type} \times \mathtt{type} \to \mathtt{type}$ in $\mathtt{Q}$

If $\mathtt{Q}$ does not contain any other type constructors, apart from those given by (i)-(ii), we say that $\mathtt{S}_\Sigma$ is a (Σ-)*superseding simple type signature*.

For any Σ-superseding type signature $\mathtt{S}_\Sigma$ we obviously have the term monad, called the *type term monad*, $\mathbf{T}_{\mathtt{S}_\Sigma}$, over $\mathtt{Var}$, so that $\mathsf{T}_{\mathtt{S}_\Sigma} X$, $X \in \mathrm{Ob}(\mathtt{Var})$, contains all terms which we call *type terms*.

We write $\mathtt{s} \Rrightarrow \mathtt{t}$ for the type term $\Rrightarrow(\mathtt{s}, \mathtt{t})$, which we call an *arrow type term*.

We are now finally in the position to discuss 'lambda terms', i.e. terms over signatures $\Sigma' = (S', \Omega')$, where $S' = \mathsf{T}_{\mathtt{S}_\Sigma}\varnothing$. We will express λ, the abstractor, as an operator in Ω'.

At this point note that traditional λ-calculus allows identification of terms in $\mathsf{T}_\Sigma X$ with terms in $\mathsf{T}_{\Sigma'} X$, which is not allowed. Moreover, we can't use one and the same 'X', as the X in the expression $\mathsf{T}_\Sigma X$ actually is of form $\{X_\mathtt{s}\}_{\mathtt{s}\in S}$, and X in $\mathsf{T}_{\Sigma'} X$ is of form $\{X_{\mathtt{s}'}\}_{\mathtt{s}'\in S'}$. Further, we shouldn't take any liberties concerning 'term lookalikes', given that $S \subseteq S'$.

The situation is very apparent even the simple case of the original signature NAT for natural numbers. We will have $\mathtt{nat}$ also in S', but for terms we have to pay attention. We have $\mathtt{0} : \to \mathtt{nat}$ in Ω, so $\mathtt{0}$ is a term in $\mathsf{T}_\Sigma\{X_\mathtt{s}\}_{\mathtt{s}\in S}$, since $\mathtt{0} \in \mathsf{T}_{\Sigma,\mathtt{nat}}\{X_\mathtt{s}\}_{\mathtt{s}\in S}$. We are inclined to write $\mathtt{0} :: \mathtt{nat}$, but be aware, the literature is not unambiguous concerning the use of '::'.

We may now syntactically select $\lambda^0_0 :\to \mathtt{nat}$ to be an operator in Ω'. The intuitive reading is 'the λ-abstraction of the term $\mathtt{0}$ at the 0th position of the outmost operator in the term expression'.

Let $\mathfrak{A}_\Sigma$ and $\mathfrak{A}_{\Sigma'}$ be algebras, respectively, for Σ and Σ'. Assume $\mathfrak{A}_\Sigma(\mathtt{nat}) = \mathbb{N}$, and $\mathfrak{A}_\Sigma(\mathtt{0}) = 0 \in \mathbb{N}$. Note how it would be a *definition* to say that also $\mathfrak{A}_{\Sigma'}(\mathtt{nat}) = \mathbb{N}$ and possible also that $\mathfrak{A}_{\Sigma'}(\lambda^0_0) = \mathfrak{A}_\Sigma(\mathtt{0})$. Clearly, the algebras could have been defined differently for Σ and Σ' and their related terms.

For $\mathtt{succ} : \mathtt{nat} \to \mathtt{nat}$ we have $\mathtt{succ}(\mathtt{0}) \in \mathsf{T}_\Sigma\{X_\mathtt{s}\}_{\mathtt{s}\in S}$, and we may select $\lambda^{\mathtt{succ}}_1 :\to (\mathtt{nat} \Rrightarrow \mathtt{nat})$ to be another operator in Ω', with the intuitive reading 'the λ-abstraction of $\mathtt{succ}$ the 1st position of the outmost operator in the term expression'.

It is obviously not a bad model to have $\mathfrak{A}_{\Sigma'}(\lambda^{\mathtt{succ}}_1) \in \mathrm{Hom}(\mathfrak{A}_{\Sigma'}(\mathtt{nat}), \mathfrak{A}_{\Sigma'}(\mathtt{nat}))$, as it seems natural to consider $\mathfrak{A}_{\mathtt{S}_\Sigma}(\mathtt{s} \Rrightarrow \mathtt{t}) = \mathrm{Hom}(\mathfrak{A}_{\mathtt{S}_\Sigma}(\mathtt{s}), \mathfrak{A}_{\mathtt{S}_\Sigma}(\mathtt{t}))$.

The general principle of abstraction is then that each 'primitive' operator $\omega : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s} \in \Omega$ has a corresponding abstraction as a constant $\lambda^\omega_{i_1,\ldots,i_n} : \to (\mathtt{s}_{i_1} \Rrightarrow \cdots \Rrightarrow (\mathtt{s}_{i_{n-1}} \Rrightarrow (\mathtt{s}_{i_n} \Rrightarrow \mathtt{s})))$ in Ω', where $i_1, \ldots, i_n$ is some selected permutation of $1, \ldots, n$. An interesting observation here is that there is obviously an 'well understood morphism' between the signatures Σ and Σ', but the standard definition of signature morphisms does not identify this situation.

We now obviously want to add *application* as an operator, and this means we have a $\mathtt{app}_{\mathtt{s},\mathtt{t}} : (\mathtt{s} \Rrightarrow \mathtt{t}) \times \mathtt{s} \to \mathtt{t}$ in Ω', assuming $\mathtt{s}, \mathtt{t} :\to \mathtt{type}$, so that $\mathtt{app}_{\mathtt{nat},\mathtt{nat}}(\lambda^{\mathtt{succ}}_1, \lambda^0_0) \in \mathsf{T}_{\Sigma'}\{X_{\mathtt{s}'}\}_{\mathtt{s}'\in S'}$.

Algebraic values can be unfolded accordingly, but we leave these aspects mostly outside the scope of this paper.

Note what abstraction really and precisely is doing. Abstraction takes an operator in Ω (in Σ), 'waits' for the related sorts to 'move over' to constants in the superseding level signature, and for these constants as terms (on the superseding level) to 'move over' to appear as sorts in S' (in Σ'). An important point and observation is then that we do not have anything like an 'abstraction of an abstraction', which, of course, will turn out to be the key solution to avoiding the need for renaming. For instance, if we would have an operator like $\mathtt{add} : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}$, we have the abstractions $\lambda^{\mathtt{add}}_1 :\longrightarrow \mathtt{nat} \Rrightarrow (\mathtt{nat} \Rrightarrow \mathtt{nat})$ and $\lambda^{\mathtt{add}}_2 : \mathtt{nat} \Rrightarrow (\mathtt{nat} \Rrightarrow \mathtt{nat})$.

Note also that we are indeed including $\lambda^{\mathtt{succ}}_1 :\to (\mathtt{nat} \Rrightarrow \mathtt{nat})$ into Ω', but not something like $\kappa^{\mathtt{succ}}_1 : \mathtt{nat} \to \mathtt{nat}$ into Ω'. It would be possible, but whereas $\lambda^{\mathtt{succ}}_1$ is an abstraction, $\kappa^{\mathtt{succ}}_1$ leaves $\mathtt{succ}$ basically as it is and as it appears in Ω.

On the renaming issue we now arrive at the observation we mentioned earlier. To start with, note that the renaming issue is really about being 'hygienic' (Barendregt has used this word) and careful when selecting free and bound variables so as to avoid variable overlap in the sense that a term with a free

variable being included by substitution into another term leads to the free variable becoming bound. The definition of substitutions must then consider these distinctions, so that full 'hygiene' is achieved. Such definitions, however, are not suitable for functorial treatments.

Before providing more examples, let us also be clear about substitutions. In our lingua substitutio logica, substitutions are always and nothing but morphisms in the Kleisli category of the selected term monad. A substitution in a substitution logic over a signature Σ (regardless of where it appears), is thus a morphism $\sigma : X \longrightarrow \mathsf{T}_\Sigma Y$, where we may have $X = Y$, but must not have. This also means that $\mu_Y \circ \mathsf{T}_\Sigma\sigma : \mathsf{T}_\Sigma X \longrightarrow \mathsf{T}_\Sigma Y$ is the morphism that transforms terms to terms, given the particular substitution σ. So for $t \in \mathsf{T}_\Sigma X$, $\mu_Y \circ \mathsf{T}_\Sigma\sigma(t)$ is the result of the overall substitution. Frequently, the literature provides notation for what happens when only one $x \in X$ is affected by a particular $\sigma : X \longrightarrow \mathsf{T}_\Sigma X$. In this case, the assumption is that $\sigma(y) = y$, always when $y \neq x$. A typical notation for $(\mu_X \circ \mathsf{T}_\Sigma\sigma)(t)$ in this case is $t[x := \sigma(x)]$ or $t[\sigma(x)/x]$, frequently used in functional programming, or, $t\{x \mapsto \sigma(x)\}$, more used in logic programming. The logic programming notation in a more general form, for any σ, is then $t\{x_1 \mapsto \sigma(x_1) \ldots x_n \mapsto \sigma(x_n)\}$, meaning $\sigma(y) = y$, always when $y \neq x_i, i = 1, \ldots, n$, and in its most general form $t\{x \mapsto \sigma(x) \mid x \in X\}$, the latter being the intuitive notation corresponding always and precisely to $(\mu_X \circ \mathsf{T}_\Sigma\sigma)(t)$.

In this view, we do not have to select one-sortedness or many-sortedness, but we may for the time being think on one-sortedness in sentences below, so that the underlying category is $\mathtt{Set}$, and not $\mathtt{Set}_S$. For $\Sigma = \mathtt{NAT}$, note that $\mathtt{succ}(x)$, $x \in X$, is a term, i.e. a member of $\mathsf{T}_{\mathtt{NAT}}X$, but $\mathtt{succ}$ isn't, so it doesn't make any sense e.g. to write $\mathtt{succ}[\mathtt{0}/x]$, whereas $\mathtt{succ}(x)[\mathtt{0}/x] = \mathtt{succ}(\mathtt{0})$, $x \in X$, and $\mathtt{succ}(y)[\mathtt{0}/x] = \mathtt{succ}(y)$, $x, y \in X$.

If confusion is at no risk, we may use any of these notations, as long as we can assure that they coincide with the purely functorial meaning of substitution.

It is now very important also to discuss what happens with variables. If we continue using the one-sorted $\mathtt{NAT}$ signature, and then use $S' = \mathsf{T}_{\mathtt{S}_\Sigma}\varnothing$, so that S' contains (at least) $\mathtt{nat}$, $\mathtt{nat} \Rrightarrow \mathtt{nat}$, $\mathtt{nat} \Rrightarrow (\mathtt{nat} \Rrightarrow \mathtt{nat})$, and so on. We are then in a many-sorted $\Sigma' = (S', \Omega')$, and suppose we have $\lambda_0^{\mathtt{0}} :\rightarrow \mathtt{nat}$, $\lambda_1^{\mathtt{succ}} :\rightarrow (\mathtt{nat} \Rrightarrow \mathtt{nat})$, and $\mathtt{app}_{\mathtt{nat},\mathtt{nat}} : (\mathtt{nat} \Rrightarrow \mathtt{nat}) \times \mathtt{nat} \rightarrow \mathtt{nat}$ in Ω'.

For $X \in \mathrm{Ob}(\mathtt{Set})$[3], $x \in X$ is a term in $\mathsf{T}_\Sigma X$. We may now select $X' = \{X_{\mathtt{s}}\}_{\mathtt{s} \in S'}$ so that $X_{\mathtt{nat}} = X$. In this case, $x \in X$ will appear, not *as* a term in $\mathsf{T}_{\Sigma'}X'$, but *as part of* a term in $\mathsf{T}_{\Sigma'}X'$, i.e. we have some term $\{x_{\mathtt{s}}\}_{\mathtt{s} \in S'} \in \{X_{\mathtt{s}}\}_{\mathtt{s} \in S'}$, so that $\{x_{\mathtt{s}}\}_{\mathtt{s} \in S'} \in \mathsf{T}_{\Sigma'}X'$, where $x_{\mathtt{nat}} = x$.

We are now ready to demonstrate how we avoid the renaming issue. In classical λ-calculus, consider the term $\lambda y.\mathtt{succ}(x)$, and the substitution $[x := \mathtt{succ}(y)]$. There must be a renaming of either y before substitution takes place, so that e.g. renaming y to z in the term leads to the desired result that $\lambda z.\mathtt{succ}(x)[x := \mathtt{succ}(y)]$ becomes $\lambda z.\mathtt{succ}(\mathtt{succ}(y))$.

[3]Note that $\mathtt{Set}_S$, $S = \{\mathtt{nat}\}$, and $\mathtt{Set}$ are isomorphic.

Before proceeding with the example, note the following

Proposition 3.1. *Let t_1 and t_2 be terms in $\mathsf{T}_{\Sigma'}X'$, and let $\sigma : X' \longrightarrow \mathsf{T}_{\Sigma'}X'$ be a substitution, where $\sigma = \{\sigma_{\mathtt{s}}\}_{\mathtt{s}\in S'}$, and $\sigma_{\mathtt{s}} : X_{\mathtt{s}} \longrightarrow \mathsf{T}_{\Sigma',\mathtt{s}}X'$. Then*

$$((\mu_{\mathtt{s}})_{X_{\mathtt{s}}} \circ \mathsf{T}_{\Sigma,\mathtt{s}}\sigma)(\mathtt{app}_{\mathtt{nat},\mathtt{nat}}(t_1, t_2)) = \mathtt{app}_{\mathtt{nat},\mathtt{nat}}(((\mu_{\mathtt{s}})_{X_{\mathtt{s}}} \circ \mathsf{T}_{\Sigma,\mathtt{s}}\sigma)(t_1), ((\mu_{\mathtt{s}})_{X_{\mathtt{s}}} \circ \mathsf{T}_{\Sigma,\mathtt{s}}\sigma)(t_2)).$$

Proof. Immediate from the term monad construction. □

In classical λ-calculus, this is a definition, and it is usually, for 'λ-terms' M and N, written as $(M\ N)[x := t] = (M[x := t]\ N[x := t])$.

To proceed with our example, the substitution $\sigma = \{\sigma_{\mathtt{s}}\}_{\mathtt{s}\in S}$, for which $\sigma_{\mathtt{s}}(x_{\mathtt{nat}}) = \mathtt{app}_{\mathtt{nat},\mathtt{nat}}(\lambda_1^{\mathtt{succ}}, y_{\mathtt{nat}})$, where $x_{\mathtt{nat}}, y_{\mathtt{nat}} :: \mathtt{nat}$, and otherwise $\sigma_{\mathtt{s}}(x) = x$, where $x \neq x_{\mathtt{nat}}$ and $x :: \mathtt{nat}$, then gives the following, with $\widetilde{\sigma}$ as a short form for $((\mu_{\mathtt{s}})_{X_{\mathtt{s}}} \circ \mathsf{T}_{\Sigma,\mathtt{s}}\sigma)$,

$$\begin{aligned}\widetilde{\sigma}(\mathtt{app}_{\mathtt{nat},\mathtt{nat}}(\lambda_1^{\mathtt{succ}}, x_{\mathtt{nat}})) &= \mathtt{app}_{\mathtt{nat},\mathtt{nat}}(\widetilde{\sigma}(\lambda_1^{\mathtt{succ}}), \widetilde{\sigma}(x_{\mathtt{nat}})) \\ &= \mathtt{app}_{\mathtt{nat},\mathtt{nat}}(\lambda_1^{\mathtt{succ}}, \mathtt{app}_{\mathtt{nat},\mathtt{nat}}(\lambda_1^{\mathtt{succ}}, y_{\mathtt{nat}}))\end{aligned}$$

Note also how that the difference between $\lambda_1^{\mathtt{succ}}$ and $\mathtt{app}_{\mathtt{nat},\mathtt{nat}}(\lambda_1^{\mathtt{succ}}, x_{\mathtt{nat}})$ is now very transparent.

Readers may have already noted that λ could be avoided in $\lambda_1^{\mathtt{succ}}$. We could simply write $\mathtt{succ}_1$ for the term (and constant operator) within the Σ' signature, and at the same time always pay attention to make the necessary distinction from its origin $\mathtt{succ}$ being the operator within the Σ signature. However, we prefer to keep the letter λ in that meme in order to support those readers already having created a strong intuition about the power of λ-calculus.

Somehow λ abstraction, when understood traditionally and intuitively, is like the *mentha piperita* plant. It's a perennial and easy to grow, but, given its very long and persistent root system, it is really hard to kill in particular if it has been allowed to grow without any attention and control.

4. Bausteine, functionality and simple typing

4.1. Schönfinkel's Bausteine *(1920)*

In the following we provide a 'simple typing' of Schönfinkel's *Bausteine*. We provide it at this point without support of Church's [5] notation of results, so as to demonstrate the role of the superceding signature to decide clearly "what belongs and what doesn't".

In what follows, we will refer to the English version of [3] appearing in [12].

From start, Schönfinkel makes no distinction between syntax and semantics. Functions $F(x, y)$ are transformed to $G_x(y)$, and then Schönfinkel says "we can regard this function G itself – its form, so tho speak – as the value (function

value) of a new function f, so that $G = fx$"[4], and thereby arrives at the notation fxy. In these views, f is more like an operator in a signature and F is its semantic counterpart, so we actually would have something like $G = \mathfrak{A}(fx)$. To be precise, we would have an operator $\omega : \mathtt{s}_1 \times \mathtt{s}_2 \to \mathtt{s}$ within the signature Σ so that $f = \lambda^{\omega}_{1,2} :\to (\mathtt{s}_1 \Rrightarrow (\mathtt{s}_2 \Rrightarrow \mathtt{s}))$, and $\mathfrak{A}_{\Sigma'}(fxy) = F(\mathfrak{A}_{\Sigma'}(x), \mathfrak{A}_{\Sigma'}(y))$.

Schönfinkel's functions I, C, T, Z and S can now be 'simply typed' as follows.

The *identity function* I, by Schönfinkel defined as $Ix = x$, can then be seen as an operator $\mathtt{I} : \mathtt{type} \to \mathtt{type}$, an *identity type modifier*, within the signature $\mathtt{S}_{\Sigma}$, so that $\mathfrak{A}_{\mathtt{S}_{\Sigma}}(\mathtt{I}) : \mathtt{Set} \longrightarrow \mathtt{Set}$ is the identify functor and $\mathfrak{A}(\mathtt{type}) = \mathtt{Set}$. The 'requirement'[5] $I(\mathtt{s}) = \mathtt{s}$ can be seen in the meaning of $\mathfrak{A}_{\mathtt{S}_{\Sigma}}(I(\mathtt{s})) = \mathfrak{A}_{\mathtt{S}_{\Sigma}}(I)(\mathfrak{A}_{\mathtt{S}_{\Sigma}}(\mathtt{s})) = \mathfrak{A}_{\mathtt{S}_{\Sigma}}(\mathtt{s})$. We may also view I as an identity for each type $\mathtt{s}$ in the signature Σ' so that the identity is the constant operator $I_{\mathtt{s}} :\to (\mathtt{s} \Rrightarrow \mathtt{s})$. We then see $I_{\mathtt{s}}$ in the meaning of $\mathfrak{A}_{\Sigma'}(I_{\mathtt{s}})$ is the identity morphism in $\mathfrak{A}_{\Sigma'}(\mathtt{s} \Rrightarrow \mathtt{s})$. Schönfinkel's "$II$ would be equal to I" then is '$I_{\mathtt{s}\Rrightarrow\mathtt{s}}(I_{\mathtt{s}}) = I_{\mathtt{s}}$'.

The *constancy function* C, defined as $(Ca)y = a$, can be seen as the type constructor $\mathtt{C} : \mathtt{type} \times \mathtt{type} \to \mathtt{type}$ fulfilling the 'equational condition' $\mathtt{C}(\mathtt{s}, \mathtt{t}) = \mathtt{s}$, and $\mathfrak{A}_{\mathtt{C}_{\Sigma}}$ would again be a functor fulfilling the corresponding criteria. Additionally, C can also be seen as an operator within Σ' as $\mathtt{C}_{\mathtt{s},\mathtt{t}} :\to (\mathtt{s} \Rrightarrow (\mathtt{t} \Rrightarrow \mathtt{s}))$, with $\mathfrak{A}_{\Sigma'}(\mathtt{C}_{\mathtt{s},\mathtt{t}}) \in \mathrm{Hom}(\mathfrak{A}_{\Sigma'}(\mathtt{s}), \mathrm{Hom}(\mathfrak{A}_{\Sigma'}(\mathtt{t}), \mathfrak{A}_{\Sigma'}(\mathtt{s})))$ so that $\mathfrak{A}_{\Sigma'}(\mathtt{C}_{\mathtt{s},\mathtt{t}})(x)(y) = x$ for $x \in \mathfrak{A}_{\Sigma'}(\mathtt{s})$ and $y \in \mathfrak{A}_{\Sigma'}(\mathtt{t})$. A sentence, in equational type logic, prescribing the constancy function condition would then look like $\mathtt{app}_{\mathtt{t},\mathtt{s}}(\mathtt{app}_{\mathtt{s},\mathtt{t}\Rrightarrow\mathtt{s}}(\mathtt{C}_{\mathtt{s},\mathtt{t}}, x), y) = x$, where $x :: \mathtt{s}$ and $y :: \mathtt{t}$ are terms.

The *interchange function* T, defined as $(T\varphi)xy = \varphi yx$, has no immediate counterpart as a type constructor within $\mathtt{S}_{\Sigma}$. For being an operator within Σ', we would have $\mathtt{T} :\to ((\mathtt{s} \Rrightarrow (\mathtt{t} \Rrightarrow \mathtt{u})) \Rrightarrow (\mathtt{t} \Rrightarrow (\mathtt{s} \Rrightarrow \mathtt{u})))$ so that for any $\varphi :\to \mathtt{s} \Rrightarrow (\mathtt{t} \Rrightarrow \mathtt{u})$ we require $\mathtt{app}_{\mathtt{s},\mathtt{u}}(\mathtt{app}_{\mathtt{t},\mathtt{s}\Rrightarrow\mathtt{u}}(\mathtt{app}_{\mathtt{s}\Rrightarrow(\mathtt{t}\Rrightarrow\mathtt{u}),\mathtt{t}\Rrightarrow(\mathtt{s}\Rrightarrow\mathtt{u})}(\mathtt{T}, \varphi), x), y) = \mathtt{app}_{\mathtt{t},\mathtt{u}}(\mathtt{app}_{\mathtt{s},\mathtt{t}\Rrightarrow\mathtt{u}}(\varphi, y), x)$.

In particular, for $\varphi = \lambda^{\omega}_{i_1,i_2}$, an operator within Σ', where $\omega : \mathtt{s}_1 \times \mathtt{s}_2 \to \mathtt{s}$ is an operator within Σ, we then have $\mathtt{app}_{\mathtt{s}_2\Rrightarrow(\mathtt{s}_1\Rrightarrow\mathtt{s}),\mathtt{s}_1\Rrightarrow(\mathtt{s}_2\Rrightarrow\mathtt{s})}(\mathtt{T}, \lambda^{\omega}_{i_1,i_2}) = \lambda^{\omega}_{i_2,i_1}$, for the permutation $i_1 = 1$ and $i_2 = 2$.

Note also that for $n > 2$ in $\lambda^{\omega}_{i_1,\ldots,i_n}$, a similar equation can be given involving two-valued versions of $\mathtt{T}$, and then the equation reflects the shifting from one permutation to another, i.e., kind of reflects a sorting algorithm.

The *composition function* Z, defined as $((Z\varphi)\chi)x = \varphi(\chi x)$, again has no immediate counterpart as a type constructor within $\mathtt{S}_{\Sigma}$. For being an operator within Σ', we have $\mathtt{Z}_{\mathtt{s},\mathtt{t},\mathtt{u}} :\to ((\mathtt{t} \Rrightarrow \mathtt{u}) \Rrightarrow ((\mathtt{s} \Rrightarrow \mathtt{t}) \Rrightarrow (\mathtt{s} \Rrightarrow \mathtt{u})))$ so that for any

[4]Schönfinkel indeed writes G in $G = fx$ where one would expect to see G_x.

[5]Note that '=' in the "identification" $\mathtt{I}(\mathtt{s}) = \mathtt{s}$ leans towards viewing $\mathtt{I}(\mathtt{s}) = \mathtt{s}$ as an equation, i.e. a sentence in some equational like logic, which, however, hasn't been defined at this point.

$\varphi :\to (\mathtt{t} \Rrightarrow \mathtt{u})$ and $\chi :\to (\mathtt{s} \Rrightarrow \mathtt{t})$ we require

$$\mathtt{app}_{\mathtt{s},\mathtt{u}}(\mathtt{app}_{\mathtt{s}\Rrightarrow\mathtt{t},\mathtt{s}\Rrightarrow\mathtt{u}}(\mathtt{app}_{\mathtt{t}\Rrightarrow\mathtt{u},(\mathtt{s}\Rrightarrow\mathtt{t})\Rrightarrow(\mathtt{s}\Rrightarrow\mathtt{u})}(\mathtt{Z}_{\mathtt{s},\mathtt{t},\mathtt{u}}, \varphi), \chi), x) = \mathtt{app}_{\mathtt{t},\mathtt{u}}(\varphi, \mathtt{app}_{\mathtt{s},\mathtt{t}}(\chi, x)).$$

The *fusion function* S, defined as $((S\varphi)\chi)x = (\varphi x)(\chi x)$, again has no counterpart within $\mathtt{S}_\Sigma$, but within Σ' we have $\mathtt{S}_{\mathtt{s},\mathtt{t},\mathtt{u}} :\to ((\mathtt{s} \Rrightarrow (\mathtt{t} \Rrightarrow \mathtt{u})) \Rrightarrow ((\mathtt{s} \Rrightarrow \mathtt{t}) \Rrightarrow (\mathtt{s} \Rrightarrow \mathtt{u})))$, so that for any $\varphi :\to (\mathtt{s} \Rrightarrow (\mathtt{u} \Rrightarrow \mathtt{u}))$ and $\chi :\to (\mathtt{s} \Rrightarrow \mathtt{t})$ we require

$$\mathtt{app}_{\mathtt{s},\mathtt{u}}(\mathtt{app}_{\mathtt{s}\Rrightarrow\mathtt{t},\mathtt{s}\Rrightarrow\mathtt{u}}(\mathtt{app}_{\mathtt{s}\Rrightarrow(\mathtt{t}\Rrightarrow\mathtt{u}),(\mathtt{s}\Rrightarrow\mathtt{t})\Rrightarrow(\mathtt{s}\Rrightarrow\mathtt{u})}(\mathtt{S}_{\mathtt{s},\mathtt{t},\mathtt{u}}, \varphi), \chi), x) = \mathtt{app}_{\mathtt{t},\mathtt{u}}(\mathtt{app}_{\mathtt{s},\mathtt{t}\Rrightarrow\mathtt{u}}(\varphi, x), \mathtt{app}_{\mathtt{s},\mathtt{t}}(\chi, x)).$$

Schönfinkel then makes the observation that C and S suffice to define the others, and proceeds to use the *compatibility function* U for propositional functions f and g in $Ufg = fx \mid^x gx$ to show how "every formula of logic can be expressed by means of C, S and U", making use of the generalized stroke symbol. In doing that, we need these expressions to be based on Σ', i.e., `bool` may have appeared in Σ and 'brought down' to reside within Σ' so that unary operators would look like $f, g :\to (\mathtt{s} \Rrightarrow \mathtt{bool})$.

The generalized stroke symbol '|', or $\mid^x$ when specified with the 'binding variable', universal quantification and negation is connected by $fx \mid^x gx = (x)\overline{fx \;\&\; gx}$. Here & and $\overline{[]}$ would be naturally viewed, respectively, as operators $\& :\to (\mathtt{bool} \Rrightarrow (\mathtt{bool} \Rrightarrow \mathtt{bool}))$ and $\overline{[]} :\to (\mathtt{bool} \Rrightarrow \mathtt{bool})$ within Σ'. However, then the abstraction $(x)(fx)$ cannot be done in any comparison with λ abstractions. On the other hand, & and $\overline{[]}$ could be viewed, respectively, as operators $\& : (\mathtt{bool} \times \mathtt{bool} \to \mathtt{bool}$ and $\overline{[]} : \mathtt{bool} \to \mathtt{bool}$ within Σ', and further introducing a type `prop` into a (Σ'-)superseding type signature $\mathtt{S}_{\Sigma'}$. This indeed works syntactically up to a certain point, but choices for $\mathfrak{A}_{\mathtt{S}_{\Sigma'}}(\mathtt{prop})$ are not easy to motivate. Something like a category of cylindric algebras might seem natural for these purposes, but we run into another type of mystery concerning "what belongs and what doesn't". This pastiche, however, might be a first step towards further exploration of what really happens in "supercedings of supercedings".

Note that $\mathtt{S}_\Sigma$ is a one-sorted signature with `type` as its only sort. Even if we mentioned the possibility of "supercedings of supercedings" we thereby by no means exclude the possibility to investigate the situation where $\mathtt{S}_\Sigma$ is many-sorted so that a type like `prop` is included as another sort alongside with `type`. This option is mentioned when dealing with Church's simple types.

It is now interesting to see what happens in situation when using the J function, but we leave it outside the scope of this paper.

4.2. Curry's functionality *(1934)*

Curry's *functionality* [4] deals with what Curry calls *categories of logic*, namely, *proposition* and *propositional function* apparently in the sense of an

intuition of sets of terms, since Curry speaks about the *intuitions by which we tell what entities belong to them. Combinatory logic* was introduced already in [6], but in [4] *entities* are defined as apparently related to Schönfinkel's *Bausteine* even if reference to Schönfinkel's paper is not given.

Curry speaks about *primitive frame* including *set of primitive ideas, axioms and rules of procedure*, i.e., terms and sentences, axioms and theory, and proof calculi. Furthermore, in [6] he *does not postulate any such notions as variable*, and variables indeed appear not until in [13], where Curry admits that it is not convenient to continue the study of logic in a manner where variables do not appear explicitly in formal developments of symbolic theory. Curry also believes at that point that variables *may be introduced into the formal developments without loss of precision.* This, in our view, is the "what belongs and what doesn't" of variables, leading to fear about 'loss of precision'. Variables were at that time mostly viewed as 'distinct from constants'. Curry writes further that *variables are not the names of any entities whatever, but are "incomplete symbols", whose function is to indicate possibilities of substitution.* In our framework variables are entities, they are terms, and as terms they change form when passing through the superceding stage. Moreover, variables are very tightly connected with substitution, and in [13] Curry describes situations for variable substitutions in terms and predicates. What Curry obviously never realized was that 'structured sets of variables' in our sense can be equally well used in substitutions without loss of generality nor specificity, i.e., he never formulated substitution as any kind of morphism, and moreover, composable morphisms.

In [4], Curry then moves over to discussing the *function concept*, where is back to the initial position of Schönfinkel. Curry's statement, that *contradictions appear to arise from applying the rules, appropriate to a certain category of entities, to an entity which seems to belong to that category but in reality does not do so*, is, in our view, repeating the concern about "what belongs and what doesn't", and really acknowledging that intuitive crossboarder traversal between intuitive takes on "category" cannot be always advisable. Categories as seen somehow as structired sets of terms and sentences even is structure is not defined and distinction between term and sentence is not necessarily provided.

Curry then proceeds to say that F is an entity and that FXY is the category of functions on X and Y. Curry, like Schönfinkel, is weak on making distinction between syntax and semantics, so F on Σ-superseding level would be $F\ = \Rrightarrow$: $\mathtt{type} \to \mathtt{type}$ so that FXY is the term $X \Rrightarrow Y$, with $X, Y :: \mathtt{type}$. Thus, Curry's $\vdash FXYf$, *representing the statement that* f *belongs to that category*, means f is the constant $f : X \Rrightarrow Y$. Both F and f is by Curry called 'entities', but they are operators within different signatures. Curry's definition $F_0 = I$ must be written out in the same way as for Schönfinkel, since I is taken for granted. If F is placed as an operator within the $\mathtt{S}_\Sigma$ signature, then so is I. The interpretation of Curry's $F_{n+1} \equiv [x_1, x_2, \cdots, x_n, y, z]F_n x_1 x_2 \cdots x_n(Fyz)$ is obvious, but it should also be noted that $x_i :: X_i$ means x_i is in $T^\iota_{\mathtt{S}_\Sigma, X_i} U$ for either $\iota = 0$ or $\iota > 0$.

Axioms for F are then given, and the last axiom is said to be *of a somewhat*

more dubious character than the others.

4.3. Church's simple typing *(1940)*

Church's *simple typing* [5] has as its starting statement that *a complete incorporation of the calculus of λ-conversion into the theory of types is impossible if we require that λ and juxtaposition shall retain their respective meanings as an abstraction operator and as denoting the application of function to argument.*

Church then introduces the type constructor which in effect is our $\Rrightarrow$, so that $(\beta \Rrightarrow \alpha)$ is his $(\beta\alpha)$. Concerning Church's ι and o, the situation is less clear. An interpretation of ι to be `type` is clearly less controversial, but for the interpreation of o there are a number of alternative intuitions.

Church says, *o is the type of propositions*, but at that point nothing is said about *proposition*. In fact, Church states the following: *We purposely refrain from making more definite the nature of the types o and ι, the formal theory admitting of a variety of interpretations in this regard. Of course the matter of interpretation is in any case irrelevant to the abstract construction of the theory, and indeed other and quite different interpretations are possible (formal consistency assumed).*

One intuition and interpretation of proposition is that proposition is in the sense e.g. of Schönfinkel, Bernays, Ackermann and Hilbert. Then, *proposition* is a term or sentence with a *truth value* as output. Our interpretration here must be that of a term, rather than viewing it as a sentence, for which truth values belongs to another story even if they may be identified.

What is now important concerning *proposition* in that sense is where it actually belongs, namely, within which signature has in been constructed. There is no ambiguity concerning the Σ level, since then o would be basically `bool`, or something similar in its semantics representing "the (possibly structured) set of truth values". This, however, seems not entirely in parity with Church's '*o is the type of propositions*'. On the other, viewing *proposition* within the Σ-superceding level, is not similarly straightforward. If the Σ-superceding signature is one-sorted with `type` as its only sort (or type), then '*o is the type of propositions*' cannot mean anything but terms on the superceding level are always of type `type`. This is not the appropriate intuition, and therefore one might introduce a second type `prop` into the set of sorts on that superceding level, as we mentioned in connection with Schönfinkel's *Bausteine*. However, then we must understand that $\Rrightarrow$ still is reserved to be $\Rrightarrow$: $\texttt{type} \times \texttt{type} \to \texttt{type}$ and introduction of similar type constructors allowing the use of `bool` must then be considered so that we will have terms, on this superceding level, e.g. of the form $\Rrightarrow_{\texttt{prop,type,type}}$: $\texttt{type} \times \texttt{type} \to \texttt{prop}$ if written in the spirit of Church's notation $o\iota\iota$. In modern and intuitive language, a *well-formed formula (wff)* is then a term of sort `prop` on the Σ-superceding level, i.e., an element of $\mathsf{T}_{\mathsf{s}_\Sigma,\texttt{prop}}\{X_{\texttt{type}}, X_{\texttt{prop}}\}$.

Having `prop` on the Σ-superceding level raises the question about which constants and operators really involve the use of `prop` as sort. A quantifiers would look like $\Pi : \texttt{type} \times \texttt{prop} \to \texttt{prop}$, like Church's $\Pi_{o(o\alpha)}$, but it is hard to

imagine any operators e.g. like $P : \texttt{type} \times \texttt{prop} \to \texttt{type}$. Therefore, `type` and `prop` are really very different in nature.

Further, and in the most simple case, there is only one constant of sort `prop`, and there would be the temptation to intuitively 'identity' `prop` with that constant. Then we must be very careful not intuitively to identify that constant to be a constant also of sort `type`. Polymorphism is usually seen as advantageous, but in polymorphism involving `type` and `prop` can semantically be quite disastrous. Note indeed that in this interpretation, keeping `prop` on that level, we cannot escape the question about what `prop` should be, and Church is obviously very well aware of this situation.

For readers now very confused about 'sort', 'type', 'sort `type`', etc., it may be helpful continuously to note the distinction between 'type' and '`type`', observing also that '`prop` is a type', but not 'of type `type`', and understanding the difference between "hierarchy of types" and "hierarchy of `type`s".

The alternative to keeping `prop` on the Σ-superceding level is to place `prop` on the level of "supercedings of supercedings", i.e., on the Σ'-superceding level, thereby viewing quantifiers, as abstractions, as something "coming after" or "being above" λ-abstractions, and not necessarily "side-by-side". Our preference concerning these alternatives is irrelevant for the purpose of this paper.

Church's listing of *primitive symbols* includes $\lambda, (,), N_{o\iota\iota}, A_{ooo}, \Pi_{o(o\alpha)}, \iota_{\alpha(o\alpha)}, a_\alpha$ and b_α, where $N_{o\iota\iota}$ (negation), A_{ooo} (disjunction), $\Pi_{o(o\alpha)}$ (universal quantification) and $\iota_{\alpha(o\alpha)}$ are called *constants*, a_α and b_α *variables*. Interesting is now that λ has no indexing, and moreover, that Church calls λ an *improper symbols*, where also '(' and ')' are improper symbols, as contrary to $N_{o\iota\iota}, A_{ooo}, \Pi_{o(o\alpha)}$ and $\iota_{\alpha(o\alpha)}$ being *proper symbols*. This is then the main difference to our view of λ being indexed e.g. by the operator it abstracts, and indeed that λ as a symbol in fact thereby can be avoided entirely.

Note also how Church says that if $\mathbf{M}_\alpha$ *is a well-formed formula of type* α, *then* $\lambda x_\beta \mathbf{M}_\alpha$ *is a well-formed formula having the type* $\beta \Rrightarrow \alpha$. This is now one of the main positions where our framework departs from Church's, since $\mathbf{M}_\alpha$ is a term on the Σ level and $\lambda x_\beta \mathbf{M}_\alpha$ appears on the Σ' level.

Church's 'variable binding' operator, or *choice function*, $\iota_{\alpha(o\alpha)}$, is influence e.g. by Hilbert's ϵ-operator in the ϵ-calculus culminating in Ackermann's thesis 1924. The $\iota_{\alpha(o\alpha)}$ operator obviously has its counterpart in our framework as well, but appears differently since variables are only implicitly pointed at by the indices appearing in $\lambda^{\omega}_{i_1,\ldots,i_n}$.

This discussion leads inevitably also to considerations of sets, when identified with term expressions, appearing at different levels given "supercedization" of types and operators. Whatever the situation, we should note that *Axiom of Choice* is a sentence, and the formulation of sentences has been mostly avoided in this paper.

Church's $I_{\alpha\alpha}$ operator is Schönfinkels identity function I, and Church's $K_{\alpha\beta\alpha}$ operator is Schönfinkels constancy function C. His syntactic definitions of natural numbers $0_{\alpha'}$, $1_{\alpha'}$, $2_{\alpha'}$, $3_{\alpha'}$, etc., is then kind of assuming that the topmost signature Σ is the empty signature, so that the sort α' is introduced on the Σ' level, since $\alpha' :\to \texttt{type}$ then would be assumed to be a operator within the

$\mathtt{S}_\Sigma$ signature. This means that α' becomes "natural number type `nat` which is non-existing in the empty signature Σ".

Church then proceeds with logical considerations and this is beyond our scope.

5. Conclusions

Our intuition is that we can abstract away from 'variables' but emphasis is then on 'abstract away from', so variables must have been there in one form or another from start, indeed at least in the underlying metalanguage as provided by set theory. We are again back to our conviction that if there are no foundations, you will, being tricky enough, end up precisely anywhere you want to be. Untyped theories tend to be like this, and also typed theories, where certain difficult issues are solved by allowing 'metatypes' of various kind, being placed outside the formal or semiformal discussions.

Renaming is thereby solved since the concept of *binding variables* has been dismissed and nullified in this framework, and our abstraction differs from the de Bruijn notation [14], which is nameless on variables but the abstractor still has its global power over the operator domain.

Our abstractors are expected to provide more insight to Schönfinkel's and Curry's efforts with respect to combinators, and to Church's management of simple types. In fact, first-order logic at *fons et origo* also suffers from these aspects. Natural numbers are similar. They are intuitively used, and somewhere along the line, they are formally defined, and redefined. Set theory is the same, and Hilbert's view seems not have changed during decades of his work towards *Grundlagen*, namely, that set theory and logic were developed kind of hand in hand, being sometimes guided by paradoxes, sometimes just by pure hard work of developing foundations.

As we have previously shown, see [15, 11], the underlying categories for variables are selected e.g. for internalized modeling of uncertainty. This opens up avenues for similar consideration in type theory, namely, considering type constructor and λ term monads, or 'combinator monads' in general for that matter, over suitable underlying categories representing the desired character of underlying information for respective purposes in various applications.

`Fuzzy λ-calculus is now just behind the corner!`

References

[1] D. Hilbert, W. Ackermann, Grundzüge der theoretischen Logik:, Springer-Verlag, 1928.

[2] H. B. Curry, Review: D. Hilbert and W. Ackermann, Grundzüge der theoretischen Logik, Bulletin (New Series) of the American Mathematical Society 59 (3) (1953) 263–267.

[3] M. Schönfinkel, Über die bausteine der mathematischen logik, Mathematische Annalen 92 (3) (1924) 305–316.

[4] H. B. Curry, Functionality in combinatory logic, Proceedings of the National Academy of Sciences of the United States of America 20 (11) (1934) 584–590.

[5] A. Church, A formulation of the simple theory of types, The journal of symbolic logic 5 (2) (1940) 56–68.

[6] H. B. Curry, Grundlagen der kombinatorischen logik, American journal of mathematics 52 (4) (1930) 789–834.

[7] A. Church, An unsolvable problem of elementary number theory, American journal of mathematics 58 (2) (1936) 345–363.

[8] E. G. Manes, Algebraic Theories, Vol. 26 of Graduate Texts in Mathematics, Springer-Verlag, 1976.

[9] W. Gähler, Monads and convergence, in: Generalized Functions, Convergences Structures, and Their Applications, Plenum Press, New York, 1988, pp. 29–46.

[10] J. Beck, Distributive laws, in: Seminar on Triples and Categorical Homology Theory, 1966/67, Lecture Notes in Mathematics **80**, Springer-Verlag, 1969, pp. 119–140.

[11] P. Eklund, M. Galán, R. Helgesson, J. Kortelainen, Fuzzy terms, Fuzzy Sets and Systems, in press.

[12] J. Heijenoort, From Frege to Gödel: a source book in mathematical logic, 1879-1931, Source books in the history of the sciences, Harvard University Press, 1967.

[13] H. Curry, Apparent variables from the standpoint of combinatory logic, The Annals of Mathematics 34 (3) (1933) 381–404.

[14] N. G. De Bruijn, Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem, Indagationes Mathematicae (1972) 381–392.

[15] P. Eklund, M. Galán, R. Helgesson, J. Kortelainen, Paradigms for many-sorted non-classical substitutions, in: 2011 41st IEEE International Symposium on Multiple-Valued Logic (ISMVL 2011), 2011, pp. 318–321.