MODELS, INTERPRETATIONS AND THE INITIALLITY CONJECTURES

VLADIMIR VOEVODSKY

Abstract. Work on proving consistency of the intensional Martin-Löf type theory with a sequence of univalent universes ("MLTT+UA") led to the understanding that in type theory we do not know how to construct an interpretation of syntax from a model of inference rules. That is, we now have the concept of a model of inference rules and the concept of an interpretation of the syntax and a conjecture that implies that the former always defines the latter. This conjecture, stated as the statement that the term model is an initial object in the category of all models of a given kind, is called the Initiallity Conjecture. In my talk I will outline the various parts of this new vision of the theory of syntax and semantics of dependent type theories.

1. Introduction

The first few steps in all approaches to the set-theoretic semantics of dependent type theories remain insufficiently understood. The constructions which have been worked out in detail in the case of a few particular type systems by dedicated authors are being extended to the wide variety of type systems under consideration today by analogy. This is not acceptable in mathematics. Instead we should be able to obtain the required results for new type systems by *specialization* of general theorems and constructions formulated for abstract objects the instances of which combine together to produce a given type system.

An approach that follows this general philosophy was outlined in [29]. In this approach the connection between the type theories, which belong to the concrete world of logic and programming, and abstract mathematical concepts such as sets or homotopy types is constructed through the intermediary of C-systems.

C-systems were introduced in [11] (see also [12]) under the name "contextual categories". A modified axiomatics of C-systems and the construction of new C-systems as sub-objects and regular quotients of the existing ones in a way convenient for use in type-theoretic applications are considered in [35]. A C-system equipped with additional operations corresponding to the inference rules of a type theory is called a model or a C-system model of these rules or of this type theory. There are other classes of objects on which one can define operations corresponding to inference rules of type theories most importantly categories with families or CwFs. They lead to other classes of models.

Date: August, 2017.

2000 *Mathematics Subject Classification.* Primary 18C50; Secondary 03B15.

In the approach of [29], in order to provide an interpretation for a type theory one first constructs two C-systems. One C-system, which we will call the proximate or term C-system of a type theory, is constructed from formulas of the type theory using the main construction of [33]. The second C-system is constructed from the category of abstract mathematical objects using the results of [30]. Both C-systems are then equipped with additional operations corresponding to the inference rules of the type theory making them into models of type theory. The model whose underlying C-system is the term C-system is called the term model.

A crucial component of this approach is the expected result that for a particular class of inference rules the term model is an initial object in the category of models. This is known as the Initiallity Conjecture. In the case of the pure Calculus of Constructions with a "decorated" application operation this conjecture was proved in 1988 by Thomas Streicher [27]. The problem of finding an appropriate formulation of the general version of the conjecture and of proving this general version will be the subject of future work.

Note that we do not have much freedom in how we may define the term C-system of a type theory. In particular, we can not use for its definition various approaches that generate a priory an initial model. The term C-system should reflect faithfully the object that is used to implement the type theory in the computer programs of proof assistants. Since such an implementation is based on a subset of "well-typed" sentences in a given system of "raw" syntax we either have to realize this definition mathematically or to provide a theorem stating that the definition that we use is equivalent to the one in terms of the subsets of well-typed sentences in the sets of raw sentences.

Assuming that the initiallity conjecture is proved for a given system of inference rules, there is a unique homomorphism from the term C-system to the abstract C-system that is compatible with the corresponding systems of operations. Such homomorphisms are called representations or interpretations of the type theory. More generally, any functor from the category underlying the term C-system of the type theory to another category may be called a representation of the type theory in that category. Since objects and morphisms of term models are built from formulas of the type theory and objects and morphisms of abstract C-systems are built from mathematical objects such as sets or homotopy types and the corresponding functions, such representations provide a mathematical meaning to formulas of type theory.

The existence of these interpretations in the particular case of the "standard univalent models" of Martin-Löf type theories and of the Calculus of Inductive Constructions (CIC) provides the only known justification for the use of the proof assistants such as Coq for the formalization of mathematics in the univalent style [31], [38].

Only if we know that the initiallity result holds for a given type theory can we claim that a model defines a representation. A similar problem also arises in the predicate logic but there, since one considers only one fixed system of syntax and inference rules, it can and had been solved once without the development of a general theory. The term models for a class of type theories can be obtained by considering slices of the term model of the type theory called Logical Framework (LF), but unfortunately it

is unclear how to extend this approach to type theories that have more substitutional (definitional) equalities than LF itself.

A construction of a model for the version of the Martin-Löf type theory that is used in the UniMath library [38], [31] is sketched in [20]. At the time when that paper was written it was unfortunately assumed that a proof of the initiallity result can be found in the existing body of work on type theory which is reflected in [20, Theorem 1.2.9] (cf. also [20, Example 1.2.3] that claims as obvious everything that is done in tens of different papers by computer scientists, the present paper and in [35]). Since then it became clear that this is not the case and that a mathematical theory leading to the initiallity theorem and providing a proof of such a theorem is lacking and needs to be developed.

As the criteria for what constitutes an acceptable proof were becoming more clear as a result of continuing work on formalization, it also became clear that more detailed and general proofs need to be given to many of the theorems of [20] that are related to the model itself. For the two of the several main groups of inference rules of current type theories it is done in [34], [36] and [32]. Other groups of inference rules will be considered in further papers of that series.

That work concerned the construction of the second, "abstract", C-system model used in the construction of a representation.

The work done in [33] provides the first step in the construction of the term C-system model. The result of our construction is equivalent to the results of constructions sketched many years ago, see e.g. [19]. The main innovation, other than the first careful mathematical proofs of all the required assertions, is the observations that one can take *all* raw sentences (sequents) as the source for the construction and build from them a C-system. Moreover, this C-system can be equipped by operations corresponding to the systems of inference rules of the type theory.

Note that we say "a system of inference rules" instead of "a collection of inference rules". The reason for it is that inference rules of type theories depend on each other and there is a partial order structure on the collection of all inference rules corresponding to this dependence. The simplest example is the pair of inference rules

$$\frac{\Gamma, x : A \rhd B\ type}{\Gamma \rhd \prod(x : A), B\ type} \qquad \frac{\Gamma, x : A \rhd r : B}{\Gamma \rhd \lambda(x : A), r : \prod(x : A), B} \tag{1}$$

Here the first rule introduces a type constructor $\prod$ and the second one introduces an element constructor λ. Without going into any detail about what else these rules "mean" one can see that it is impossible to state the second one until the first one was stated. It is a usual situation with inference rules in dependent type theories with sequences of dependency between the rules themselves that can be arbitrary long.

We do not have yet a mathematical definition of what a general system of inference rules is. It is an active area of current research. What we certainly expect is that any such system defines a system of operations on the sets of raw sentences in the syntax of the type theory. Again, we don't yet have a well defined mathematical concept of what a "system of operations" is. However, we certainly expect that for any such system $\mathcal{S}$ there is the concept of a set of sentences X closed under the operations from

$\mathcal{S}$ and that the intersection of any collection of sets closed under $\mathcal{S}$ is again closed under $\mathcal{S}$.

This very minimal requirement on what systems of operations can be allows one to show that for any set of sentences X there is the smallest set of sentences $Cl_{\mathcal{S}}(X)$ that contains X and is closed under $\mathcal{S}$. In particular, there is the set $Cl_{\mathcal{S}}(\emptyset)$ that is the smallest one among all sets of sentences closed under $\mathcal{S}$.

The term C-system of the type theory with a given raw syntax and the set of inference rules $\mathcal{S}$ should be defined as the C-system *corresponding to* the set $Cl_{\mathcal{S}}(\emptyset)$. This C-system should carry a system of operations $\mathcal{S}_C$ corresponding to $\mathcal{S}$ and with this operations should be, in "good" cases, the initial object in the category of C-systems with operations $\mathcal{S}_C$.

This is the general picture. In this picture we know some of the components. We know what the "raw syntax" is specified by and what are the sets of "raw sentences" of any type system of the Martin-Löf kind.

As we will describe below there are five main kinds of sentences. The first three kinds correspond to objects while the other two to equivalences between objects.

We know what conditions a subset in the set of raw sentences should satisfy to define a C-system. In particular, the set of all raw sentences of the first three kinds defines a C-system described in detail in [33] and the C-systems corresponding to the "good" subsets of sentences are quotients of sub-systems of this C-system.

A set of inference rules should satisfy some conditions in order for the corresponding set $Cl_{\mathcal{S}}(\emptyset)$ of sentences to correspond to a C-system. There is no complete clarity in what this conditions are as there is no complete clarity in what a general system of inference rules is. As I mentioned above this is an active area of current research.

Let me explain now the parts of the picture that we understand. Here by "understand" I mean a very strong statement, namely "know how to formalize in the UniMath". This meaning allows us to treat a very complex concept of understanding in very concrete terms. Experience shows that it is an excellent tool to eliminate arguments and to concentrate on the mathematics.

The five kinds of "sequents" that we call "sentences" were originally introduced by Per Martin-Löf in [22, p.161][1] . If we consider the type theory as a language then sequents are the smallest units that have semantical meaning when an interpretation is chosen. This is why we call them "sentences". The five kinds of sentences considered by Martin-Löf are sequences of expressions of the following forms

$$x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd ok \tag{2}$$

$$x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd T\ type \tag{3}$$

$$x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd t : T \tag{4}$$

$$x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd T \equiv T' \tag{5}$$

$$x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd t \equiv t' : T \tag{6}$$

[1]This paper is highly recommended. It is a foundational one for many ideas of type theory and for the modern approach to constructive mathematics in general.

Here $x_0, \dots, x_{n-1}$ are names of variables, T_i is an expression with free variables from the set $\{x_0, \dots, x_{i-1}\}$, and T and t are expressions with free variables from the set $\{x_0, \dots, x_{n-1}\}$. If one wants to emphasize that a variable x may appear as a free variable in the expression T one writes $T(x)$, but *in most cases the set of allowed free variables in an expression should be inferred from its position in the sentence*.

In many modern papers on type theory the symbol $\vdash$ is used where we use the triangle symbol $\rhd$. We made this choice because the meaning of the former symbol in type theory may conflict with its meaning in logic.

The part of a sentence to the left of $\rhd$ is called the context and the part to the right of this symbol is called the judgement. When the names of variables and the expressions of the context are not important or can be inferred from some data or conventions, it is customary to denote the context by a capital Greek letter such as Γ or Δ.

There are some equivalent versions of the Martin-Löf's approach. For example, Martin Hofmann, in [19], considers six kinds of sentences adding the equality of contexts, $\rhd \Gamma \equiv \Delta$, as a separate kind.

In any approach sentences are sequences of expressions with some restrictions on allowed free variables. Therefore

The first step towards mathematical theory of type theories is to find a way to view "expressions" as mathematical objects.

In practice, what are "expressions" from which the sentences of a type theory are formed is most likely to be specified in detail when this type theory is used as a basis of a computer proof assistant. Depending on the programming language on which the proof assistant is written and on the personal tastes of the developers, "expressions" will be represented as elements of different datatypes. They may be represented as actual strings of characters or as trees with additional labels at nodes and edges or as something else entirely. While each of these representations can be given a precise mathematical form it would be clearly wrong to make the mathematical theory of type theories dependent on which of the representations is chosen. Therefore, we need a concept of an abstract expression, or, as we will see below, two concepts one for abstract element expressions and one for abstract type expressions.

This problem has been addressed by many authors, first in the context of algebraic expressions and later in the context of expressions with binders, that is, expressions that may contain bound variables. As far as we know, the first mathematical abstraction in the case of expressions with binders was described by Fiore, Plotkin and Turi in [15]. Later a different and more convenient for mathematicians abstraction was described by Hirschowitz and Maggesi in a series of papers including [17]. The two approaches were shown to be equivalent in particular in [6], [7] using the concept of a well behaved functor. The proof of equivalence in [6], [7] was based on the important observation that the abstract clones of [15] are particular cases of *relative monads*.

Let us explain this approach in some detail. Let us consider the case of algebraic expressions first. Modern mathematical theory of algebraic expressions has been developed in a multitude of papers which can be summarized by the 1935 paper by

G. Birkhoff [10], followed by the 1963 Ph.D. thesis of Bill Lawvere [21] and a 1966 paper by F. Linton [**?**] connecting the two approaches. However, our interest in having the theory extended later to operations that bound variables, such as the $\forall$ quantifier, together with the need to have our approach adapted for a constructive meta-theory bring forward aspects of this theory that are easy to miss otherwise.

Systems of algebraic expressions are specified by algebraic signatures - pairs, consisting of a set Op, called the set of operations, and a function $Ar : Op \to \mathbb{N}$, called arity. Let us choose a set V from which we will take the names of variables. Since we test our understanding using UniMath and UniMath is a library of constructive mathematics it is convenient to assume that both Op and V are sets with decidable equality. Let further $Fin(V)$ be the set of finite subsets in V. Constructive definition of a finite subset includes the condition that the subset is decidable, that is, it is decidable whether or not a given element of V is in the given finite subset X.

Given a signature Sig and a set V one can define, for any $X \in Fin(V)$, the set $Exp(X)$ of expressions relative to Sig with (free) variables from X. Note that when we write $Exp(X)$ we assume that the signature Sig and the set V have been fixed.

There are many different families of sets $Exp(X)$ whose elements may be called expressions. For example, one can use a subset of the set of sequences (lists) of elements of $Op \amalg X$. Alternatively, one can use some axiomatization of planar rooted trees with labels from $Op \amalg X$ on the nodes. Other possibilities exist as well. We will assume that one of these variants is chosen.

Let U be a universe of sets that contains Op, V, and $Exp(X)$ for all $X \in Fin(V)$. Let $\mathcal{F}in(V)$ be the category of sets whose set of objects is $Fin(V)$, let $Sets(U)$ be the category of sets whose set of objects is U, and $J_V : \mathcal{F}in(V) \to Sets(U)$ the inclusion functor.

We may consider Exp as a function $Fin(V) \to U$. In the chosen representation of expressions one can construct the substitution operation that for any $X, Y \in Fin(V)$ and $f : X \to Exp(Y)$, defines a function $rr_{X,Y}(f) : Exp(X) \to Exp(Y)$. In addition, for any $X \in Fin(V)$ one can define a function $\eta_X : X \to Exp(X)$. The triple (Exp, η, rr) satisfies the conditions making it into a J_V-relative monad.

This is how relative monads appear in the theory of expressions with variables.

Next, following [15], we let $\mathbb{F}$ denote the category with the set of objects $\mathbb{N}$ and the set of morphisms

$$Mor_{\mathbb{F}}(m, n) = Fun(stn(m), stn(n))$$

where $stn(m) = \{i \in \mathbb{N} \,|\, i < m\}$ is our choice for the standard set with m elements and where for two sets X and Y, $Fun(X, Y)$ is the set of functions from X to Y.

It is sometimes convenient to distinguish natural numbers used as objects of $\mathbb{F}$ from their other uses. For this purpose we may write n for n used as an object of $\mathbb{F}$.

Let $Jf : \mathbb{F} \to Sets(U)$ be the functor given by $n \mapsto stn(n)$ on objects and by the identity on the sets of morphisms.

Our previous construction applies to $V = \mathbb{N}$. Consider the functor $\Phi : \mathbb{F} \to \mathcal{F}in(\mathbb{N})$ that takes n to $stn(n)$ and that is again the identity on the sets of morphisms.

Relative monads on a functor $\mathcal{C}_1 \to \mathcal{C}_2$ can be precomposed with functors $\mathcal{C}_0 \to \mathcal{C}_1$. Precomposing the monad of expressions (Exp, η, rr) with Φ and observing that $\Phi \circ J_{\mathbb{N}} = Jf$ we obtain, for any algebraic signature Sig a Jf-relative monad that we denote by $\mathbf{Exp}_{Sig}$.

This is how the Jf-relative monads appear in the theory of algebraic expressions.

Note that that up to this point our constructions were completely elementary.

Suppose now that we want to associate with the family of sets $Exp(X)$ not a relative monad but a usual monad. First we would need to extend, for some V with decidable equality, the function $Exp : \mathcal{F}in(V) \to U$ to a function $U \to U$. There is no way of doing it in the UniMath and I do not know of any way of doing it in any constructive foundation. The best one could achieve is to construct a function $Exp' : U \to U$ and a family of isomorphisms $\phi_V : Exp(X) \to Exp'(X)$ for $X \in Fin(V)$. This requires developing a constructive theory of filtered colimits and functors that commute with such colimits and it is not an obvious task. Modulo these difficulties the category of Jf-relative monads is equivalent to the category of finitary monads on $Sets(U)$.

Alternatively, one can build a (finitary) monad Exp'' corresponding to a signature directly by constructing the set $Exp''(X)$ as an initial algebra over the functor $F_{Sig,X} : Sets(U) \to Sets(U)$ given on objects by the formula

$$F_{Sig,X}(A) := X \coprod (\coprod_{O \in Op} A^{Ar(O)})$$

Constructing initial algebras for $F_{Sig,X}$ also requires the use of colimits, but only ω-colimits, that is, colimits of sequences, see e.g. [2]. The monad structure on the family of sets $Exp''(X)$ can be constructed from the initial algebra structures, see [9] or [23, Th.3, p.161]. One is then left with the task of establishing a family of bijections between $Exp''(X)$ and $Exp(X)$ for $X \in Fin(V)$ that are compatible with the substitution which can be done but requires extra work.

In general, a monad on $Sets(U)$ is a richer object than a Jf-relative monad and there may be situations when a monad associated with a signature is required as an intermediary between the syntax and an abstract mathematical construction. However, in our case, when we want to construct from the syntax a C-system, a Jf-relative monad is precisely what we need, so that even when we have a monad at our disposal we have to restrict it to a Jf-relative monad first in order to perform our construction.

This is why we use Jf-relative monads and not the usual monads.

Let us now explain another very important point. At the very start of our explanation of how the Jf-relative monads are related to expressions we said that we will consider *algebraic* expressions. However, the expressions that appear in the sentences of type theories are often not algebraic because they contain operations that bound some of the variables in their arguments. For example, the expression $\prod(x : A), B$ that appears in (**??**) can be rewritten as $\prod(A, x.B)$ which makes it visible that it is the result of an operation $\prod$ applied to two arguments A and B and that this operation binds one variable, here called x, in its second argument.

Expressions that contain operations that may bound variables in their arguments are called expressions with binders.

Expressions with binders are specified by binding signatures - pairs consisting of a set of operations Op and the arity function $Ar : Op \to Fseq(\mathbb{N})$. Here $Fseq(\mathbb{N})$ is the set of finite sequences of elements of $\mathbb{N}$. The set $\mathbb{N}$ is considered as a subset of $Fseq(\mathbb{N})$ through the embedding taking d to the sequence $(0, \dots, 0)$ of length d. This defines an inclusion of algebraic signatures into binding signatures. The earliest mention of the concept equivalent to the binding arity that we know of is in [1]. The meaning of an operation E with the algebraic arity d is that E has d arguments. The meaning of an operation E with the binding arity $(i_1, \dots, i_d)$ is that E has d arguments and binds i_k variables in its k-th argument.

To apply an operation Op with arity $(n_0, \dots, n_{d-1})$ to expressions $E_0, \dots, E_{d-1}$ one has to specify, in addition to the expressions themselves, d sequences, of lengths $n_0, \dots, n_{d-1}$ respectively, of names of variables. These sequences will show which of the variables are bound in each argument.

The best known examples of operations with binders are the quantifiers $\forall$ and $\exists$ of predicate logic and the λ-abstraction of the (untyped) lambda calculus [13],[8]. All three of these operations have arity (1), that is, they have one argument in which they bind one variable.

To get an example with arity (2) one may consider the operation that one gets by applying an operation of arity (1) twice.

Consider expressions formed by operations with binders applied to variables. For example, consider expressions generated by one operation of arity (1) that we will call λ. Every such expression is of the form

$$E = \lambda\, x_{n-1}.\lambda\, x_{n-2}.\dots.\lambda\, x_0.x$$

Here $x_n, \dots, x_0$ are bound variables. We do not assume that $x_i \neq x_j$ for $i \neq j$ or that $x_i \neq x$. In particular x is a free variable if $x_i \neq x$ for all $i \geq 0$ and a bound one otherwise. The usual, but hard to formulate precisely, rules of α-equivalence (see e.g. [8, Def. 2.1.11, p.26]) imply that if we rename the bound variables in any way that preserves the rightmost occurrence of x among the x_i's then the resulting expression will be α-equivalent to the original one. In particular, we can always rename x_i such that $x_i \neq x_j$ for $i \neq j$ and there is an most one k such that $x_k = x$. If such a k exists then E has no free variables and if it does not then E has one free variable x.

If x is a free variable then we can substitute another expression E' of the same form for x. However, we can not do it directly. Instead we have to use something called the *capture-avoiding substitution* to avoid the "capture" of variable names by binders. For example, let $E = \lambda\, x_0.x$, where x is free, and $E' = x'$. Then we have two cases - if $x_0 \neq x'$ then we can directly substitute E' for x and $E[E'/x] = \lambda\, x_0.x'$. If $x_0 = x'$ we have first to rename x_0 into x'_0 such that $x'_0 \neq x'$ and then to substitute, obtaining $E[E'/x] = \lambda\, x'_0.x'$. If we used direct substitution the resulting operation would not respect the α-equivalence. The capture-avoiding substitution does.

One shows, and it should be clear from the above that it is not easy, that for any binding signature (Op, Ar) one can define, for expressions constructed using operations of this signature and names of variables from a given set V, which occurrences of variable names among the arguments of the operations are free and which are bound. From this one can define, for any subset $X \in Fin(V)$, the set $Exp^{\cdot}(X)$ of expressions with free variables from X. Next one can define the concept of α-equivalence on each of the sets $Exp^{\cdot}(X)$ and define the sets $Exp^{\alpha}(X)$ of α-equivalence classes of expressions with free variables from X. Most definitions of α-equivalence require V to be a set with an additional operation that for every finite subset of V gives an element in the complement to this subset. Let us call it a freshness operation. Some approaches to the α-equivalence and further constructions discussed below, notably the approach through the nominal sets [24], may only require that for any finite subset of V there exist an element in the complement to this subset. In the latter case we will say that V has the freshness property. In the ZFC a set has the freshness property if and only if it is infinite. In constructive meta-theories the situation may be more involved and it is convenient to have a special name for this particular property.

If V has the freshness property one can define, and again it is not at all easy, the simultaneous capture-avoiding substitution of expressions $E_x \in Exp^{\cdot}(Y)$, $x \in X$, for the free variables of an expression $E' \in Exp^{\cdot}(X)$ such that it is compatible with the α-equivalence. After the passage to the α-equivalence classes these constructions become equivalent and one obtains, for any function $X \to Exp^{\alpha}(Y)$ and an element of $Exp^{\alpha}(X)$, an element of $Exp^{\alpha}(Y)$. In addition one has, for any $X \in Fin(V)$, a function $X \to Exp^{\alpha}(X)$.

This brings us again to a structure of the same form as we obtained in the case of algebraic operations - a J_V-relative monad, where J_V is the obvious functor from $\mathcal{F}in(V)$ to $Sets(U)$. Performing the same construction as the one described above in the case of algebraic expressions one obtains from a $J_{\mathbb{N}}$-relative monad, a Jf-relative monad $\mathbf{Exp}_{Sig}$. This is a direct generalization of the construction that we described previously from algebraic expressions to expressions with binders. The main idea behind this generalization goes back to Fiore, Plotkin and Turi [15] where this theory is developed for abstract clones, structures constructively equivalent to the Jf-relative monads.

This is how the Jf-relative monads appear in the theory of expressions with binders.

Let us calculate what we get from this construction when the signature is given by one operation λ with the arity (1). The expressions then are the expressions that we considered above. We have seen that

$$Exp^{\alpha}(\emptyset) = \{a_{n,k} \,|\, n \in \mathbb{N}, k = 0, \ldots, n-1\}$$

where $a_{n,k}$ is (the equivalence class of) the expression with n λ-abstractions such that k is the smallest index satisfying $x_k = x$.

Next, we know that

$$Exp^{\alpha}(\{x\}) = Exp^{\alpha}(\emptyset) \cup \{b_n(x) \,|\, n \in \mathbb{N}\}$$

where $b_n(x)$ is the expression with n λ-abstractions ending with x and such that $x_i \neq x$ for all $n-1 \geq i \geq 0$. We have to add $Exp^{\alpha}(\emptyset)$ because an expression without free variables is an expression with free variables from the set $\{x\}$.

Finally, for a general $X \in Fin(V)$ we have

$$Exp^{\alpha}(X) = Exp^{\alpha}(\emptyset) \cup (\cup_{x \in X} \{b_n(x) \,|\, n \in \mathbb{N}\})$$

and the union on the right hand side is disjoint.

The capture-avoiding substitution in the case of one free variable is of the form

$$b_n(a_{n',k'}/x) = a_{n+n',k'}$$

$$b_n(b_{n'}(x')/x) = b_{n+n'}(x')$$

For many free variables the substitution is determined by the case of one free variable because in any one expression there is at most one free variable.

It is easy to see that the Jf-relative monad that we obtain in this case is isomorphic to the Jf-relative monad defined by the algebraic signature with operations a_k, $k \in \mathbb{N}$ and b where the arity of a_k is 0 and the arity of b is 1. The elements corresponding under this isomorphism to $a_{n,k}$ are $b^n(a_k)$ and the elements corresponding to $b_n(x)$ are $b^n(x)$.

Church's famous λ-calculus starts with the system of abstract expressions corresponding to two operations ap and λ with the arity of ap being $(0,0)$ and the arity of λ being (1). Operation ap is called application and is usually denoted using the infix notation with the empty operation symbol, that is, $ap(E, E')$ is denoted $E\,E'$.

I do not know of an algebraic representation similar to what we have described above for the free Church's λ-expressions, that is, for the Jf-relative monad corresponding to the binding signature

$$Sig_{\Lambda} = (\{ap, \lambda\}, Ar(ap) = (0,0), Ar(\lambda) = (1))$$

More generally, one may ask if for any binding signature Sig one may construct an algebraic signature $Alg(Sig)$ and an isomorphism between the Jf-relative monads corresponding to Sig and $Alg(Sig)$ as we have done in the case when $Sig = (\{\lambda\}, Ar(\lambda) = (1))$.

To obtain the actual λ-calculus, or more specifically, the $\lambda_{\eta\beta}$-calculus, one has to add to the system of expressions defined by Sig_{Λ} two relations that are called the β- and the η-reductions. The fact that one still gets a Jf-relative monad structure after passing to the equivalence classes under the equivalence relation generated by these "reductions" requires a proof.

It appears that the Jf-relative monad, corresponding to the $\lambda_{\eta\beta}$-calculus has an algebraic presentation closely related to the combinatory logic of Schönfinkel [25] (translated in [28]) and Curry [14]. However many subtle difficulties arise in making it precise (cf. [26]) and we know of no theorem asserting such a presentation in terms of relative monads or monads.

This is how the Jf-relative monads corresponding to binding signatures relate to the Jf-relative monads corresponding to algebraic signatures.

What we said about the direct extension of $\mathbf{Exp}_{Sig}$ from a Jf-relative monad to a monad immediately generalizes from the algebraic case to the case of operations with binders.

Also generalizes the discussion about the possibility to construct a monad corresponding to the signature directly using category theory. The beginnings of this generalization can be seen in [15]. It is highly non-trivial. Operations that bind variables change the set of free variables e.g for $x \in X$, the operation $\lambda\, x$ can be seen as an operation from $Exp(X \coprod \{x\})$ to $Exp(X)$. Because of this, the individual sets $Exp^{\alpha}(X)$ do not have universal characterization. Instead, a universal characterization can be given to a functor $Exp : Sets(U) \to Sets(U)$ that will be later given a monad structure. This functor has an initial algebra structure for $\underline{Id} + H_{Sig}$ where H_{Sig} is a *functor of the second order* - a functor from functors to functors and $\underline{Id}$ is the functor of second order that takes any F to the identity functor of $Sets(U)$. The functor H_{Sig} can be directly constructed from the binding signature Sig. Bindings correspond to the operation on functors $F \mapsto F'$ where $F'(X) = F(X \coprod pt)$. The general theory of initial algebras for ω-cocontinuous functors from [2] is applicable here as well and an initial algebra Exp'' for $\underline{Id} + H_{Sig}$ can be constructed as the colimit of the sequence of functors $(\underline{Id} + H_{Sig})^n(\underline{\emptyset})$ where $\underline{\emptyset}$ is the functor $X \mapsto \emptyset$. Since the initial algebras are unique up to a unique algebra isomorphism the sets $Exp''(X)$ constructed by the colimit construction are in a bijective correspondence with the sets $Exp(X)$ of α-equivalence classes of expressions. The sets Exp'' are closely related to the sets that one obtains representing α-equivalence classes using de Bruijn levels or indexes. There is more story to tell here, but it is too much outside of the scope of the present note.

Next one needs to construct a monad structure on Exp''. The corresponding theory is developed in [15, Sec. 4],[23] and with the formalization in the UniMath in [3] and [4]. An outline of the theory that allows one to give a universal characterization to the monad structure itself can be found in [18]. Not all is understood yet and it remains an active area of research. Much of the work that is being done today is being simultaneously formalized in the UniMath. The key question here is what structure on H has to be specified in order to obtain a monad structure on the initial algebra of $\underline{Id} + H$. The main idea was introduced in [23]. In [3] a functor with this structure is called an (abstract) signature. As became understood later in [4], the additional condition of H being ω-cocontinuous allows one to remove the condition of the existence of the right adjoints from the main theorem [23, Th. 15, p.170] leading to [4, Th. 48].

The case that is most important for us, that of the monads defined by the binding signatures, has been fully formalized in the UniMath. There remains the problem of showing that the families of sets of the J_V-relative monads corresponding to this monad are isomorphic to the sets of expressions modulo the α-equivalence and that the monad structure that one obtains satisfies the universality conditions of [18].

The preceding discussion shows how the monads corresponding to the binding signatures can be constructed by methods of category theory.

The raw syntax of a type theory can be specified by a binding signature[2]. For example, the raw syntax of Streicher's formulation of the Calculus of Constructions of G. Huet and T. Coquand (CC-S), when brought into the standard form, consists of six operations $\prod$, $Prop$, $Proof$, λ, app and $\forall$ with the corresponding arities $(0,1)$, $()$, (0), $(0,1)$, $(0,1,0,0)$ and $(0,1)$, see [27, p.157].

In view of the preceding discussion, this suggests that the class of abstract mathematical objects that can be used to most directly model the raw syntax of type theories is the class of Jf-relative monads. However, in [33] we use pairs of a Jf-relative monad $\mathbf{RR}$ and a left module $\mathbf{LM}$ over this monad. Let us explain why we need such pairs and how one can generate them from data similar to binding signatures.

To obtain the binding signature of the raw syntax from the usual presentation of a type theory by a list of inference rules such as (1) one should make the list of operations that these inference rules introduce with their names and their binding arities. Often operations will be given in a non-standard form such as $\prod\, x : A, B$ instead of $\prod(A, x.B)$, but for unambiguous inference rules it should be easy to see what the corresponding standard form should be.

Among those operations will be operations that introduce *types* and operations that introduce *elements* (also called *objects*) of types.

For example, in the type theory CC-S the operations $\prod$, $Prop$ and $Proof$ introduce types while operations λ, app and $\forall$ introduce elements. In addition, some arguments of each operation must be types and some elements. However, *only element variables can be bound*.

Define a restricted 2-sorted binding signature as a signature where arities of operations are given by sequences $((n_0, \epsilon_0), \dots, (n_{d-1}, \epsilon_{d-1}), \epsilon)$ where $\epsilon \in \{0, 1\}$ with 0 corresponding to elements and 1 to types. Such two sorted arities of the six operations of CC-S are, correspondingly, $((0,1),(1,1),1)$, (1), $((0,0),1)$, $((0,1),(1,0),0)$, $((0,1),(1,1),(0,0),(0,0),0)$ and $((0,1),(1,0),0)$.

Any restricted 2-sorted binding signature defines the usual, 1-sorted one, where the set of operations is the same and the arity function is the composition of the original arity function with the function that maps $((n_0, \epsilon_0), \dots, (n_{d-1}, \epsilon_{d-1}), \epsilon)$ to $(n_0, \dots, n_d)$.

Let Sig_2 be a (restricted) 2-sorted binding signature and Sig_1 the corresponding 1-sorted one. Let Z be a set such that the set of expressions with respect to Sig_1 with variables from Z is defined. Let us fix two (decidable) subsets $V, Y \subset Z$ such that $V \cap Y = \emptyset$ and both V and Y have the freshness property. Consider the subset $Exp_{Sig_2}[V, Y]$ of expressions that conform to the additional rules defined by the sequences $(\epsilon_0, \dots, \epsilon_{n-1}, \epsilon)$ of the 2-sorted arities of the operations of Sig_2 under the assumption that a variable can be used as an element variable if and only if it is in V and as a type variable if and only if it is in Y. This subset will be the disjoint union of two smaller subsets $ElExp_{Sig_2}[V, Y]$ and $TyExp_{Sig_2}[V, Y]$ where the first one

[2]The type theories whose syntax can be specified by an algebraic signature correspond to the "generalized algebraic theories" of John Cartmell [12], [11], [16].

consists of expressions of sort "element" and the second one of expressions of sort "type".

Next, for $X \in Fin(V)$ let $Exp^{\cdot}_{Sig_2}(X,Y)$ be the subset of $Exp_{Sig_2}[V,Y]$ that consists of expressions where an element variable is free if and only if it belongs to X with a similar notation for $ElExp$ and $TyExp$. Since V has the freshness property one can define the α-equivalence relation on $Exp^{\cdot}_{Sig_2}[V,Y]$ and therefore on $ElExp^{\cdot}_{Sig_2}(X,Y)$ and $TyExp^{\cdot}_{Sig_2}(X,Y)$. Let $ElExp^{\alpha}_{Sig_2}(X,Y)$ and $TyExp^{\alpha}_{Sig_2}(X,Y)$ be the corresponding sets of equivalence classes.

Let us fix a set $PrTy \in Fin(Y)$. This set will eventually play the role of the set of primitive types that we add to the base type theory. Consider X as a variable, writing $RR_V(X)$ and $LM_V(X)$ instead of $ElExp^{\alpha}_{Sig_2}(X, PrTy)$ and $TyExp^{\alpha}_{Sig_2}(X, PrTy)$.

The structures that we get on the families of sets $RR_V(-)$ and $LM_V(-)$ are slightly different. On RR we get the J_V-relative monad structure - for any $X \in Fin(V)$ we have a function $\eta_X : X \to RR_V(X)$ and for any $X, Y \in Fin(V)$ and a function $f : X \to RR_V(Y)$ we have a function

$$rr_{X,Y}(f) : RR_V(X) \to RR_V(Y)$$

On the other hand, on the LM_V we do not have η_X since variables from X are not type expressions and substitution defines for any $X, Y \in Fin(V)$ and a function $f : X \to RR_V(Y)$, a function

$$lm_{X,Y}(f) : LM_V(X) \to LM_V(Y)$$

This operation makes the family of sets $LM_V(X)$ into a left module $\mathbf{LM}_V = (LM_V, lm)$ over the J_V-relative monad $\mathbf{RR}_V = (RR_V, \eta, rr)$.

Precomposing $\mathbf{RR}_{\mathbb{N}}$ and $\mathbf{LM}_{\mathbb{N}}$ with the obvious functor $\Phi : \mathbb{F} \to \mathcal{F}in(\mathbb{N})$ we obtain a pair $(\mathbf{RR}, \mathbf{LM})$ of a Jf-relative monad and a left module over it.

In some type theories all types are elements of universes and moreover element expressions are not syntactically distinguishable from type expressions. For example, it is the case in the very important type theory MLTT79 - the Martin-Löf type theory from [22]. The inference rules related to the universes [22, p.172] make all type expressions also element expressions and an element expression of any form may be used as a type. In our notation it means that $LM(X) = RR(X)$.

The preceding discussion shows how pairs of a Jf-relative monad $\mathbf{RR}$ *and a left module* $\mathbf{LM}$ *over it correspond to the raw syntax of type theories because some expressions are type expressions and some are element expressions.*

To construct the pair $(\mathbf{RR}, \mathbf{LM})$ by methods of category theory without a reference to expressions one can proceed as follows.

A restricted 2-sorted binding signature defines a monad on the category $Sets(U) \times Sets(U)$. See [39] for a much more general case of multi sorted signatures. For the formalization of this construction in UniMath see [5].

Choosing an object $PrTy$ of $Sets(U)$ and applying to this monad on $Sets(U) \times Sets(U)$ two constructions from [33] one obtains a pair $(\mathbf{RR}, \mathbf{LM})$ of a Jf-relative monad and a module over it.

This is how the pairs $(\mathbf{RR}, \mathbf{LM})$ *can be obtained from a restricted 2-sorted binding signature by methods of category theory.*

Let us return to sentences of type theory that can be of the five kinds (2)-(6). The expressions in the sentences are considered modulo the α-equivalence. Moreover, the sentences themselves are also considered modulo the α-equivalence, that is, modulo the renaming of the variables $x_0, \dots, x_{n-1}$ introduced by the context. Using this α-equivalence we may assume that $(x_0, \dots, x_{n-1}) = (0, \dots, n-1)$. Then T_i has free variables from $stn(i)$ and T, T', t and t' free variables from $stn(n)$. When we are given all the necessary additional information for the construction of the pair $(\mathbf{RR}, \mathbf{LM})$ where $RR(n)$ and $LM(n)$ are the α-equivalence classes of type and element expressions with free variables from the sets $stn(n)$ we obtain the following description of the sets of all possible sentences of the five main kinds:

(1) a sentence of the form (2) is an element of

$$B(\mathbf{RR}, \mathbf{LM}) = \coprod_{n \geq 0} \prod_{i=0}^{n-1} LM(i)$$

(2) a sentence of the form (3) is an element of

$$Bt(\mathbf{RR}, \mathbf{LM}) = \coprod_{n \geq 0} (\prod_{i=0}^{n-1} LM(i)) \times LM(n)$$

(3) a sentence of the form (4) is an element of

$$\widetilde{B}(\mathbf{RR}, \mathbf{LM}) = \coprod_{n \geq 0} (\prod_{i=0}^{n-1} LM(i)) \times RR(n) \times LM(n)$$

(4) a sentence of the form (5) is an element of

$$Beq(\mathbf{RR}, \mathbf{LM}) = \coprod_{n \geq 0} (\prod_{i=0}^{n-1} LM(i)) \times LM(n) \times LM(n)$$

(5) a sentence of the form (6) is an element of

$$\widetilde{Beq}(\mathbf{RR}, \mathbf{LM}) = \coprod_{n \geq 0} (\prod_{i=0}^{n-1} LM(i)) \times RR(n) \times RR(n) \times LM(n)$$

In any approach to Martin-Löf type theory a sentence of the form $0 : T_0, \dots, n-1 : T_{n-1} \rhd T$ *type* is equivalent to the sentence $0 : T_0, \dots, n-1 : T_{n-1}, n : T \rhd ok$. This allows one not to consider sentences of the form (3).

This description of sentences immediately generalizes from the pairs $(\mathbf{RR}, \mathbf{LM})$ corresponding to the α-equivalence classes of expressions to all pairs $(\mathbf{RR}, \mathbf{LM})$ where $\mathbf{RR}$ is a Jf-relative monad and $\mathbf{LM}$ a left module over $\mathbf{RR}$.

The next question that we know the answer to is how subsets in the sets of sentences of the four (or five) kinds are related to C-systems.

Let B, $\widetilde{B}$, Beq and $\widetilde{Beq}$ be the subsets in $B(\mathbf{RR},\mathbf{LM})$, $\widetilde{B}(\mathbf{RR},\mathbf{LM})$, $Beq(\mathbf{RR},\mathbf{LM})$ and $\widetilde{Beq}(\mathbf{RR},\mathbf{LM})$ corresponding to $Cl_{\mathcal{S}}(\emptyset)$. We want, under the additional assumption that these subsets satisfy some conditions, to construct a C-system CC that corresponds to them.

This construction should be compatible with the constructions outlined in earlier papers, such as the construction of the category with families outlined in [19]. In particular, the set of objects of CC should be defined together with an isomorphism to the quotient set $B/\sim$ of the set B by the equivalence relation defined by the set Beq according to the rue that $(T_0,\dots,T_{n-1})$ is equivalent to $(T'_0,\dots,T'_{n'-1})$ if and only if $n'=n$ and the sequences defined by the table

$$\rhd \quad T_0 \equiv T'_0 \tag{7}$$

$$x_0 : T_0 \rhd \quad T_1 \equiv T'_1 \tag{8}$$

$$\dots \tag{9}$$

$$x_0 : T_0,\dots,x_{n-2} : T_{n-2} \rhd \quad T_{n-1} \equiv T'_{n-1} \tag{10}$$

are in Beq.

Hofmann and some other authors suggest to directly construct the set of morphisms and all the required structures using the subsets $\widetilde{B}$ and $\widetilde{Beq}$. Already the first step, the definition from $\widetilde{B}$ of a set that will later have to be factorized by an equivalence relation coming from $\widetilde{Beq}$ to produce the set of morphisms is non-trivial, c.f [19, Def. 2.11, p.97]. Constructing the composition and proving its properties such as the associativity represents additional difficulties.

In [33] and [37], we propose to proceed in a different manner. Instead of starting with $B/\sim$ and building the C-system structure on it, we will construct a C-system $C(\mathbf{RR},\mathbf{LM})$ and then use the results of [35] to show how any quadruple of subsets B,$\widetilde{B}$,Beq,$\widetilde{Beq}$, satisfying certain properties, defines a sub-quotient C-system of $C(\mathbf{RR},\mathbf{LM})$. This sub-quotient will be the carrier of the term model C-system of our type theory.

The properties that the B-subsets have to satisfy will be seen to be the ones that have long been known as the "structural properties" that the valid sentences of all type theories must satisfy. By approaching them from the direction of [35] we will see why these particular properties are necessary and sufficient for a subset of sentences to correspond to a C-system.

There is much more that one can add, but this is where I would like to end this review of what is known and what is not on the syntactic side of the mathematical theory of general dependent type theories.

Acknowledgements: I am very grateful for useful conversations to Benedikt Ahrens and Marcelo Fiore.

(1) This material is based on research sponsored by The United States Air Force Research Laboratory under agreement number FA9550-15-1-0053. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the United States Air Force Research Laboratory, the U.S. Government or Carnegie Mellon University.

(2) This publication was made possible through the support of a grant from the John Templeton Foundation. The opinions expressed in this publication are those of the author and do not necessarily reflect the views of the John Templeton Foundation.

References

[1] Peter Aczel. The type theoretic interpretation of constructive set theory. In *Logic Colloquium '77 (Proc. Conf., Wrocław, 1977)*, volume 96 of *Stud. Logic Foundations Math.*, pages 55–66. North-Holland, Amsterdam-New York, 1978.

[2] Jiří Adámek. Free algebras and automata realizations in the language of categories. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.

[3] Benedikt Ahrens and Ralph Matthes. Heterogeneous substitution systems revisited. *https://arxiv.org/abs/1601.04299*, 2016.

[4] Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. From signatures to monads in unimath. *https://arxiv.org/abs/1612.00693*, 2016.

[5] Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. Monads from multi-sorted binding signatures. *http://benedikt-ahrens.de/multisorted.pdf*, 2017.

[6] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. In *Foundations of software science and computational structures*, volume 6014 of *Lecture Notes in Comput. Sci.*, pages 297–311. Springer, Berlin, 2010.

[7] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. *Logical Methods in Computer Science*, 11(1:3):1–40, 2015.

[8] H. P. Barendregt. *The lambda calculus*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, revised edition, 1984. Its syntax and semantics.

[9] Michael Barr. Coequalizers and free triples. *Math. Z.*, 116:307–322, 1970.

[10] Garrett Birkhoff. On the structure of abstract algebras. *Mathematical Proceedings of the Cambridge Philosophical Society*, 31(4):433–454, 1935.

[11] John Cartmell. Generalised algebraic theories and contextual categories. *Ph.D. Thesis, Oxford University*, 1978. `http://www.cs.ru.nl/~spitters/Cartmell.pdf`.

[12] John Cartmell. Generalised algebraic theories and contextual categories. *Ann. Pure Appl. Logic*, 32(3):209–243, 1986.

[13] Alonzo Church. A set of postulates for the foundation of logic. *Ann. of Math. (2)*, 33(2):346–366, 1932.

[14] H. B. Curry. Grundlagen der Kombinatorischen Logik. *Amer. J. Math.*, 52(4):789–834, 1930.

[15] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding (extended abstract) (Warning: the paper uses the name "presheaves" for covariant functors to the category of sets). In *14th Symposium on Logic in Computer Science (Trento, 1999)*, pages 193–202. IEEE Computer Soc., Los Alamitos, CA, 1999.

[16] Richard Garner. Combinatorial structure of type dependency. *J. Pure Appl. Algebra*, 219(6):1885–1914, 2015.

[17] André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Inform. and Comput.*, 208(5):545–564, 2010.

[18] André Hirschowitz and Marco Maggesi. Nested abstract syntax in Coq. *J. Automat. Reason.*, 49(3):409–426, 2012.

[19] Martin Hofmann. Syntax and semantics of dependent types. In *Semantics and logics of computation (Cambridge, 1995)*, volume 14 of *Publ. Newton Inst.*, pages 79–130. Cambridge Univ. Press, Cambridge, 1997.

[20] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. *Available at* `http://arxiv.org/abs/1211.2851`, 2012, 2014.
[21] F. William Lawvere. Functorial semantics of algebraic theories and some algebraic problems in the context of functorial semantics of algebraic theories. *Repr. Theory Appl. Categ.*, (5):1–121, 2004. Reprinted from Proc. Nat. Acad. Sci. U.S.A. **50** (1963), 869–872 [MR0158921] and ıt Reports of the Midwest Category Seminar. II, 41–61, Springer, Berlin, 1968 [MR0231882].
[22] Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, methodology and philosophy of science, VI (Hannover, 1979)*, volume 104 of *Stud. Logic Found. Math.*, pages 153–175. North-Holland, Amsterdam, 1982.
[23] Ralph Matthes and Tarmo Uustalu. Substitution in non-wellfounded syntax with variable binding. *Theoret. Comput. Sci.*, 327(1-2):155–174, 2004.
[24] Andrew M. Pitts. *Nominal sets*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 2013. Names and symmetry in computer science.
[25] Moses Schönfinkel. Über die bausteine der mathematischen logik. *Math. Ann.*, 92:305 – 316; Jbuch 50, 23, 1924.
[26] Peter Selinger. The lambda calculus is algebraic. *J. Funct. Programming*, 12(6):549–566, 2002.
[27] Thomas Streicher. *Semantics of type theory*. Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1991. Correctness, completeness and independence results, With a foreword by Martin Wirsing.
[28] Jean van Heijenoort. *From Frege to Gödel. A source book in mathematical logic, 1879–1931*. Harvard University Press, Cambridge, Mass., 1967.
[29] Vladimir Voevodsky. The equivalence axiom and univalent models of type theory. `http://arxiv.org/abs/1402.5556`, pages 1–11, February 4, 2010. `http://arxiv.org/abs/1402.5556`.
[30] Vladimir Voevodsky. A C-system defined by a universe category. *Theory Appl. Categ.*, 30(37):1181–1215, 2015. `http://www.tac.mta.ca/tac/volumes/30/37/30-37.pdf`.
[31] Vladimir Voevodsky. An experimental library of formalized mathematics based on the univalent foundations. *Math. Structures Comput. Sci.*, 25(5):1278–1294, 2015.
[32] Vladimir Voevodsky. Martin-Lof identity types in the C-systems defined by a universe category. *arXiv 1505.06446*, pages 1–51, 2015. `http://arxiv.org/abs/1505.06446`, under review in Publication IHES.
[33] Vladimir Voevodsky. C-system of a module over a Jf-relative monad. *arXiv 1602.00352*, pages 1–32, February 1, 2016. `http://arxiv.org/abs/1602.00352`, under review in Pure and Applied Algebra.
[34] Vladimir Voevodsky. Products of families of types and (Π, λ)-structures on C-systems. *Theory Appl. Categ.*, 31:Paper No. 36, 1044–1094, 2016. `http://www.tac.mta.ca/tac/volumes/31/36/31-36.pdf`.
[35] Vladimir Voevodsky. Subsystems and regular quotients of C-systems. In *A panorama of mathematics: pure and applied*, volume 658 of *Contemp. Math.*, pages 127–137. Amer. Math. Soc., Providence, RI, 2016. prepublication version in `http://arxiv.org/abs/1406.7413`.
[36] Vladimir Voevodsky. The (Π, λ)-structures on the C-systems defined by universe categories. *Theory Appl. Categ.*, 32:Paper No. 4, 113–121, 2017. `http://www.tac.mta.ca/tac/volumes/32/4/32-04.pdf`.
[37] Vladimir Voevodsky. The regular sub-quotients of the C-systems $CC(\mathbf{RR}, \mathbf{LM})$. *In preparation*, 2017.
[38] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. *UniMath* - a library of formalized mathematics. Available at `https://github.com/UniMath`.
[39] Julianna Zsido. *Typed Abstract Syntax*. Theses, Université Nice Sophia Antipolis, June 2010.

School of Mathematics, Institute for Advanced Study, Princeton NJ, USA. e-mail: vladimir@ias.edu