C-SYSTEM OF A MODULE OVER A Jf-RELATIVE MONAD

VLADIMIR VOEVODSKY

Abstract. This is the second paper in a series started in [41]. Let $\mathbb{F}$ be the category with the set of objects $\mathbf{N}$ and morphisms being the functions between the standard finite sets of the corresponding cardinalities. Let $Jf : \mathbb{F} \to Sets$ be the obvious functor from this category to the category of sets. In this paper we construct, for any relative monad $\mathbf{RR}$ on Jf and a left module $\mathbf{LM}$ over $\mathbf{RR}$, a C-system $C(\mathbf{RR}, \mathbf{LM})$ and explicitly compute the action of the four B-system operations on its B-sets. In the following paper it is used to provide a rigorous mathematical approach to the construction of the C-systems underlying the term models of a wide class of dependent type theories.

Contents

1. Introduction 1
2. Monads and relative monads 19
3. Modules over monads and modules over relative monads 25
References 29

1. Introduction

The first few steps in all approaches to the 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 [33]. 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 [41]. 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.

In the approach of [33], in order to provide a mathematical representation (semantics) for a type theory one 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, in particular, the main construction of the present paper. The second C-system is constructed from the category of abstract mathematical objects using the results of [34]. 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 the inference rules the term model is an initial object in the category of models. This is known as the Initiality Conjecture. In the case of the pure Calculus of Constructions with a "decorated" application operation this conjecture was proved in 1988 by Thomas Streicher [30]. 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.

For such inference rules, then, 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 homomorphisms 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 (see [42], [35]).

Only if we know that the initiality 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 ([42],[35]) is sketched in [21]. At the time when that paper was written it was unfortunately assumed that a proof of the initiality result can be found in the existing body of work on type theory which is reflected in [21, Theorem 1.2.9] (cf. also [21, Example 1.2.3] that claims as obvious everything that is done in both the present paper and in [41]). Since then it became clear that this is not the case and that a mathematical theory leading to the initiality 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 [21] 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 [40], [39] and [37]. 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 this paper provides the first step in the construction of the "concrete" term C-system model. The result of our construction is equivalent to the results of constructions sketched by earlier authors [20]. The main innovation, other than the first careful mathematical proofs of all the required assertions, is the observations that one can take *all* raw judgements as the source for the construction and build from them a C-system. The term C-system of a type theory with a given raw syntax is then a sub-quotient of the raw syntax C-system. The raw syntax C-system can be either defined directly in a way that allows for a straightforward rigorous verification of all the axioms, see Remark **??**, or understood from the perspective of the abstract mathematical theory of C-systems as a particular case of a more general construction of the presheaf extensions of C-systems. In this paper we follow the second path that also allows us to connect our construction to the main constructions of [38] and [36].

The description of a type theory in a modern paper is usually given in the form of a list of "inference rules" that may look like, for example, this one:

$$[\mathbf{2017.03.02.eq1}]\frac{\Gamma, x : A \vdash B\,type}{\Gamma \vdash \prod x : A, B\,type} \tag{1}$$

These inference rules are formulated in terms of five kinds of "sequents" originally introduced by Per Martin-Löf in [24, p.161][1] . These sequents are sequences of expressions of the form

$$[\mathbf{2017.02.06.eq1}]x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd ok \tag{2}$$

$$[\mathbf{2017.02.06.eq2}]x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd T\ type \tag{3}$$

$$[\mathbf{2017.02.06.eq3}]x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd t : T \tag{4}$$

$$[\mathbf{2017.02.06.eq4}]x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd T \equiv T' \tag{5}$$

$$[\mathbf{2017.02.06.eq5}]x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd t \equiv t' : T \tag{6}$$

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 most 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 sequent 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 [20], considers six kinds of judgements adding the equality of contexts, $\rhd \Gamma \equiv \Delta$, as a separate kind.

We, as will be seen below, will consider only four kinds of sentences deducing the $\Gamma \rhd T\ type$ kind from the $\Gamma \rhd ok$ kind.

The meaning of sentences of various kinds is as follows. The first two kinds are closely related. The sentence $\rhd ok$ is always valid, the sentence (2) for $n \geq 1$, is valid if and only if the sentence

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

is valid and x_{n-1} is a name of free variable that is not an element of the set $\{x_0, \dots, x_{n-2}\}$. From this rule we conclude that (2) is valid if and only if all x_i are names of free variables, $x_i \neq x_j$ for $i \neq j$, and the sentences in the following sequence are valid:

$$\begin{aligned} \rhd\ & T_0\ type \\ x_0 : T_0 \rhd\ & T_1\ type \\ & \dots \\ x_0 : T_0, \dots, x_{n-2} : T_{n-2} \rhd\ & T_{n-1}\ type \end{aligned}$$

[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.

A sentence of the form $\rhd T_0$ *type* asserts that T_0, which must be a closed expression according to the rules stated above, is a valid expression that describes *a type* in the system. For example, in any of the Martin-Löf type theories, there is a type $\mathbf{N}$, which is called the type of natural numbers. The formal equivalent of this assertion is that $\mathbf{N}$ is an expression and the sentence $\rhd \mathbf{N}$ *type* is valid.

One often uses the words "type", "type expression" and "expression that describes a type" interchangeably. The same applies to "element", "element expression" and "expression that describes an element".

A sentence of the form $x_0 : T_0 \rhd T$ *type* asserts that T_0 is a valid, closed, type expression and T is an expression with the only possible free variable being x_0 that describes a family of types parametrized by T_0. For example, in any of the Martin-Löf type theories, given a type T and two elements $t, t' : T$ of T, there is a type $Id\,T\,t\,t'$ whose elements are to be thought of as constructions of equalities between t and t' in T. Correspondingly, the sentence $x_0 : T \rhd Id\,T\,x_0\,x_0$ *type* is valid if and only if the sentence $\rhd T$ *type* is.

Sentences of the form (3) with $n > 1$ describe "iterated type families". For example, for $n = 2$, T_0 is a type, T_1 is a type family parametrized by T_0 and T_2, which is an expression that may contain x_0 and x_1 as free variables, is a type family with two parameters $x_0 : T_0$ and $x_1 : T_1(x_0)$.

Let $\Gamma = (x_0 : T_0, \dots, x_{n-1} : T_{n-1})$. If the sentence (4) is valid then so is the sentence $\Gamma \rhd T$ *type*. A sentence (4) with $n = 0$, that is a sentence of the form $\rhd t : T$, asserts that T is a valid (closed) type expression and t is a valid (closed) expression that describes an element of type T. For example, the element 0 of $\mathbf{N}$ in the Martin-Löf type theories is denoted by O so that the sentence $\rhd O : \mathbf{N}$ is valid in all these theories. A sentence of the form (4) with $n = 1$ describes a family T of types parametrized by T_0 together with a "section" of this family, that is, a family of elements $t(x_0)$ of types $T(x_0)$ for all x_0. If T does not contain x_0 then the family of types is constant and the sentence (4) is a syntactic representation of a function from T_0 to T.

If the sentence (5) is valid than so are the sentences $\Gamma \rhd T$ *type* and $\Gamma \rhd T'$ *type*. The validity of (5) asserts that the type expressions T and T' are *definitionally equal* in the context Γ. The similar meaning is assigned to sentences of the form (6). Definitional equality of type expressions can be used to define definitional equality of contexts. Namely, one defines two contexts $x_0 : T_0, \dots, x_{n-1} : T_{n-1}$ and $x_0 : T'_0, \dots, x_{n-1} : T'_{n-1}$ to be definitionally equal if the sentences in the following sequence are valid

$$[\mathbf{2017.04.07.eq1}] \rhd \quad T_0 \equiv T'_0 \tag{7}$$

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

$$\dots \tag{9}$$

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

This provides us with the concept of definitional equality of sentences of the form (2). Two such sentences are called definitionally equal if their contexts are.

Two sentences of the form (3) $\Gamma \rhd T$ *type* and $\Gamma' \rhd T'$ *type* are definitionally equal if $\Gamma \equiv \Gamma'$ and $\Gamma \rhd T \equiv T'$.

Similarly, one can use definitional equality of type expressions and definitional equality of element expressions defined by sentences of the form (5), and (6) to define definitional equality of sentences of the form (4). Two such sentences $\Gamma \rhd t : T$ and $\Gamma' \rhd t' : T'$ are called definitionally equal if $\Gamma \equiv \Gamma'$ and the following two sentences are valid:

$$\Gamma \rhd \ \ T \equiv T'$$
$$\Gamma \rhd \ \ t \equiv t' : T$$

It should be immediately clear from the asymmetry of these rules that in order for the definitional equality relations on sentences of the form (2), (3) and (4) to be equivalence relations the sets of sentences of various kinds should satisfy more conditions than the ones that we have mentioned so far. These conditions are among the conditions **??** whose mathematical meaning is established in this paper.

To speak about the mathematical meaning of the conditions that the valid sentences should satisfy, we need to find a way to view the structure formed by the five kinds of sentences as a mathematical object and to describe, and analyze, a construction that generates a C-system from such an object.

The first fundamental step in solving this problem 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 [18]. The two approaches were shown to be equivalent 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 monoids of [15] are particular cases of *relative monads*.

These results would have closed the issue and we would have probably used in this paper the Hirschowitz and Maggesi concept of an abstract expression based on the

concept of a monad if not for the fact that the proof of the equivalence uses the axiom of choice.

This may be the place to say a few words about the intended meta-theories of the present paper. By an intended meta-theory of a paper we mean a foundation of mathematics in which its results can be stated and proved. It has become customary not to specify intended meta-theories for papers in pure mathematics due to the tacit assumption that this meta-theory is the Zermelo-Fraenkel set theory with Axiom of Choice (ZFC). Here ZFC, or any other foundation that we may mention, is considered as a body of mathematical knowledge that includes both the underlying formal theory and the dictionary that is needed to translate informal mathematics into the statements about this formal system. In the case of the ZFC, the formal deduction system is classical predicate logic with a distinguished theory and statements of informal mathematics are translated into the provability statement applied to various formulas of this theory.

We intend two meta-theories for the present paper. The first one is the usual ZFC with a Grothendieck universe U. The second one is a univalent foundation called UniMath. We consider ZFC to be the primary meta-theory and UniMath a secondary one. That is, we take care to adapt our definitions and proofs precisely for the ZFC while accepting that UniMath formalization will require some small degree of modification.

Having UniMath as a secondary meta-theory imposes a number of strong restrictions on the features of the ZFC that we can and can not use. Most notably, we can not use the law of excluded middle and we can not use the axiom of choice.

Let us go back to the concept of an abstract expression. Let us consider the case of algebraic expressions first. Modern mathematical theory of algebraic expressions has been developed at least as far back as the 1963 Ph.D. thesis of Bill Lawvere. However, our interest in the actual syntactic expressions 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 $Ar : Op \to \mathbf{N}$, called arity. Given a signature Sig and a set V such that $V \cap Op = \emptyset$ one can define, for any $X \subset V$, the set $Exp(X)$ of expressions relative to Sig with (free) variables from X. Note that there are many different families of sets $Exp(X)$ whose elements may be called expressions. For example, one can use a subset of sequences of elements of $Op \cup X \cup \{l, r\}$ where l and r are symbols for the left and the right parenthesis. One has to impose the condition that $l \neq r$ and neither l nor r is an element of $Op \cup V$. Alternatively, one can use some axiomatization of planar trees with labels from $Op \cup V$ on the nodes. Note also that when we write $Exp(X)$ we assume that the signature Sig and the set V have been fixed.

In the chosen representation of expressions one can construct the substitution operation that, for any $X, Y \subset V$ and $f : X \to Exp(Y)$ defines a function $f^* : Exp(X) \to Exp(Y)$. In addition, for any $X \subset V$ one can define a function $\eta_X : X \to Exp(X)$.

If U is a Grothendieck universe that contains Op, V, and $Exp(X)$ for all $X \subset V$ then we may consider Exp as a function $2^V \to U$. If $Sets(2^V)$ is the category of sets whose set of objects is 2^V, $Sets(U)$ is the category of sets whose set of objects is U, and $J_V : Sets(2^V) \to Sets(U)$ is the inclusion functor then the structure $\mathbf{Exp}^V = (Exp, \eta, -^*)$ is an example of a J_V-relative monad ([6],[7] and Definition 2.5).

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 $\mathbf{N}$ and the set of morphisms

$$[\mathbf{2017.02.24.eq1}] Mor(\mathbb{F}) = \cup_{m,n} Fun(stn(m), stn(n)) \tag{11}$$

where $stn(m) = \{i \in \mathbf{N} \,|\, i < m\}$ 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 defined as in [10, p.81] such that each function has a well defined domain and codomain. Then the union in (11) is disjoint and

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

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

Assume that $\mathbf{N} \cap Op = \emptyset$. Then our previous construction applies to $V = \mathbf{N}$. Consider the functor $\Phi : \mathbb{F} \to Sets(2^{\mathbf{N}})$ that takes n to $stn(n)$ and that is again the inclusion of the sets of morphisms.

Relative monads on a functor $\mathcal{C}_1 \to \mathcal{C}_2$ can be precomposed with functors $\mathcal{C}_0 \to \mathcal{C}_1$. See Construction 2.15. Precomposing the monad of expressions $\mathbf{Exp}^{\mathbf{N}}$ with Φ and observing that $\Phi \circ J_{\mathbf{N}} = Jf$ we obtain, for any algebraic signature Sig such that $Op \cap \mathbf{N} = \emptyset$, 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 the function $Exp : 2^V \to U$ 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 $X \subset V$. This requires developing a constructive theory of filtered colimits and functors that commute with such colimits and it is not an obvious task.

Alternatively, one can build a 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 [25, Th.3, p.161]. One is then left with the task of establishing a family of bijections between $Exp''(X)$ and $Exp(X)$ for $X \subset V$ 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 appear in (1) 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 $Ar : Op \to Fseq(\mathbf{N})$. Here $Fseq(\mathbf{N})$ is the set of finite sequences of elements of $\mathbf{N}$. The set $\mathbf{N}$ is considered as a subset of $Fseq(\mathbf{N})$ 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

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

Here $x_n, \ldots, x_0$ 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 of 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 [27], 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 $X \subset V$, 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 $Sets(2^V)$ to $Sets(U)$. Performing the same construction as the one described above

in the case of algebraic expressions one obtains from a $J_{\mathbf{N}}$-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 structures equivalent to the Jf-relative monads are introduced under the name of abstract clones.

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

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

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

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

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 $X \subset V$ we have

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

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-monad that we obtain in this case is isomorphic to the Jf-monad defined by the algebraic signature with operations a_k, $k \in \mathbf{N}$ 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-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-monad structure after passing to the equivalence classes under the equivalence relation generated by these "reductions" requires a proof.

It appears that the Jf-monad, corresponding to the $\lambda_{\eta\beta}$-calculus has an algebraic presentation closely related to the combinatory logic of Schönfinkel [28] (translated in [31]) and Curry [14]. However many subtle difficulties arise in making it precise (cf. [29]) 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 see 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 λx 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 $Id + H_{Sig}$ where H_{Sig} is a *functor of the second order* - a functor from functors to functors. 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 $Id + H_{Sig}$ can be constructed as the colimit of the sequence of functors $(Id + H_{Sig})^n(\underline{\emptyset})$ 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 set 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 paper.

Next one needs to construct a monad structure on Exp''. The corresponding theory is developed in [25], [4] and [5]. An outline of the theory that allows one to give a universal characterization to the monad structure itself can be found in [19]. 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 $Id+H$. The main idea was introduced in [25]. In [4] a functor with this structure is called an (abstract) signature. As became understood later in [5], 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 [25, Th. 15, p.170] leading to [5, 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 [19].

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 [30, 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 this paper we use pairs of a Jf-relative monad $\mathbf{RR}$ and a left module $\mathbf{LM}$ (see Definition 3.6) 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 $((0,1),(1,0),0)$.

[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].

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 subsets $V, Y \subset Z$ such that $V \cap Y = \emptyset$. 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 consists of expressions of sort "element" and the second one of expressions of sort "type".

Next, for a subset X of 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 $TyExp$. Let us assume in addition that V has the freshness property. Then 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 $PrTy \subset Z$ such that $V \cap PrTy = \emptyset$. 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^V we get the usual J_V-relative monad structure - for any $X \subset V$ we have a function $\eta_X : X \to RR^V(X)$ and for any $X, Y \subset V$ and a function $f : X \to RR^V(Y)$ we have a function

$$rr^V_{X,Y}(f) : RR^V(X) \to RR^V(Y)$$

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 $X, Y \subset V$ and a function $f : X \to RR^V(Y)$, a function

$$lm^V_{X,Y}(f) : LM^V(X) \to LM^V(Y)$$

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

Precomposing $\mathbf{RR}^{\mathbf{N}}$ and $\mathbf{LM}^{\mathbf{N}}$ with the obvious functor $\Phi : \mathbb{F} \to Sets(2^{\mathbf{N}})$ using Constructions 2.15 and 3.10 we obtain a pair $(\mathbf{RR}, \mathbf{LM})$ of a J_f-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 [24]. The inference rules related to the universes [24, 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 [43] for a much more general case of multi sorted signatures. This construction is being formalized in the UniMath and will appear in the forthcoming paper [].

Choosing an object $PrTy$ of $Sets(U)$ and applying Construction **??**, one obtains from this monad a pair $(\mathbf{RR}^*, \mathbf{LM}^*)$ of a monad on $Sets(U)$ and a module over this monad.

Constructions **??** and **??** associate to this pair a pair of a an Id-relative monad and a left module over it. Precomposing with the functor $Jf : \mathbb{F} \to Sets(U)$ using Constructions 2.15 and 3.10 one obtains a pair $(\mathbf{RR}, \mathbf{LM})$ of a Jf-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.*

To make it easier to compare the constructions that follow with the earlier constructions let us recall that the substitution structure on elements of $RR(n)$ and $LM(n)$ can be easily expressed in terms of the structures of the relative monad and a module of a relative monad. Namely, if $E_1, \dots, E_k \in RR(m)$, $E \in RR(n)$ and $0 \leq i_1, \dots, i_k \leq n-1$ then $E[E_1/i_1, \dots, E_k/i_k] \in RR(m)$ is the element $rr(f)(E)$ where $f : stn(n) \to RR(m)$ is given by $f(i) = i$ for $i \neq i_1, \dots, i_k$ and $f(i_j) = E_j$. Exactly the same formula with the replacement of $RR(n)$ by $LM(n)$ describes $T[E_1/i_1, \dots, E_k/i_k]$ for $T \in LM(n)$.

This completes a large section of our introduction. We can now view the α-equivalence classes of the type and element expressions with the given sets of free variables as mathematical objects.

Next, we need to find a way to view as a mathematical object the structure formed by the five kinds of the Martin-Löf sentences (2)-(6).

Given a sentence of the form (2) we can define a sentence of the form (3) by forgetting the name of the last variable in the context and moving the last type expression to the right hand side of the $\triangleright$ symbol. To perform a construction in the opposite direction one considers not only expressions in the sentences, but also the sentences themselves up to the α-equivalence, i.e., up to the renaming of the variables $x_0, \dots, x_{n-1}$. This allows us to assume that these variables are always chosen to be $0, \dots, n-1 \in \mathbf{N}$. Then we have a canonical choice for the next variable which will be non-equal to any of the preceding ones. Therefore, given a sentence of the form (3) we can define a sentence of the form (2) by adding the type expression T to the

context with the variable d attached to it. Together with the first construction, it defines a bijection between sentences of the first kind and sentences of the second and allows us to restrict our attention to sentences of four kinds:

$$[\mathbf{2017.03.30.eq1}]x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd ok \tag{12}$$

$$[\mathbf{2017.03.30.eq2}]x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd t : T \tag{13}$$

$$[\mathbf{2017.03.30.eq3}]x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd T \equiv T' \tag{14}$$

$$[\mathbf{2017.03.30.eq4}]x_0 : T_0, \dots, x_{n-1} : T_{n-1} \rhd t \equiv t' : T \tag{15}$$

Generalizing and abstracting element expressions to elements of the sets $RR(n)$ for a Jf-relative monad $\mathbf{RR}$ and type expression to elements of the sets $LM(n)$ for a module $\mathbf{LM}$ over this monad, we see that

1. the set of sentences of kind (12) defines a subset B in the set
$$B(\mathbf{RR}, \mathbf{LM}) = \coprod_{n\geq 0} \prod_{i=0}^{n-1} LM(i)$$
2. the set of sentences of kind (13) defines a subset $\widetilde{B}$ in the set
$$\widetilde{B}(\mathbf{RR}, \mathbf{LM}) = \coprod_{n\geq 0} (\prod_{i=0}^{n-1} LM(i)) \times RR(n) \times LM(n)$$
3. the set of sentences of kind (14) defines a subset Beq in the set
$$Beq(\mathbf{RR}, \mathbf{LM}) = \coprod_{n\geq 0} (\prod_{i=0}^{n-1} LM(i)) \times LM(n) \times LM(n)$$
4. the set of sentences of kind (15) defines a subset $\widetilde{Beq}$ in the set
$$\widetilde{Beq}(\mathbf{RR}, \mathbf{LM}) = \coprod_{n\geq 0} (\prod_{i=0}^{n-1} LM(i)) \times RR(n) \times RR(n) \times LM(n)$$

Next, we need to construct from $(\mathbf{RR}, \mathbf{LM})$, B, $\widetilde{B}$, Beq and $\widetilde{Beq}$ a C-system CC. This construction should be compatible with the constructions outlined in earlier papers, such as the construction of the category with families outlined in [20]. 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 (7) 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 [20, Def. 2.11, p.97]. Constructing the composition and proving its properties such as the associativity represents additional difficulties.

We 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 $CC(\mathbf{RR},\mathbf{LM})$, which knows nothing about the subsets B,$\widetilde{B}$,Beq,$\widetilde{Beq}$, and then use the results of [41] to show how any quadruple of subsets B,$\widetilde{B}$,Beq,$\widetilde{Beq}$, satisfying certain properties, defines a sub-quotient C-system of $CC(\mathbf{RR},\mathbf{LM})$. This sub-quotient will be the term model C-system of our type theory.

The conditions on the B-subsets will be seen to be the conditions many of which have been long known as the "structural properties" that the valid sentences of all type theories must satisfy. By approaching them from the direction of [41] we will see why these particular conditions are necessary and sufficient for a type theory to "make sense".

For $p : X \to Y$ in a category, let $sec(p) = \{s : Y \to X \mid s \circ p = Id_X\}$. Elements of $sec(p)$ are called sections of p. To any C-system CC one associates a pair of sets $(Ob(CC), \widetilde{Ob}(CC))$ where $Ob(CC)$ is the set of objects of the category underlying CC and where

(16)

$$[\mathbf{2017.02.04.eq1}]\widetilde{Ob}(CC) = \{s \in Mor(CC) \mid s \in sec(p_{codom(s)}), l(codom(s)) > 0\}$$

or, in words, where $\widetilde{Ob}(CC)$ is the set of sections of the non-trivial p-morphisms of CC. When CC is clear from the context we will abbreviate $Ob(CC)$ and $\widetilde{Ob}(CC)$ to Ob and $\widetilde{Ob}$ respectively.

The sets $(Ob, \widetilde{Ob})$ are equipped with a system of operations $(l, ft, \partial, T, S, \widetilde{T}, \widetilde{S}, \delta)$. Pairs of sets equipped with operations of such form are called (unital) pre-B-systems (c.f. [**?**]). The first main result of [41], Proposition 4.2, shows how to construct from a sub-pre-B-system $(B, \widetilde{B})$ of $(Ob, \widetilde{Ob})$ a sub-C-system of CC whose associated pre-B-system is $(B, \widetilde{B})$. The second main result of that paper, Proposition 5.4, shows how to construct from a pair of equivalence relations $(\sim, \approx)$ on Ob and $\widetilde{Ob}$ respectively that satisfies certain conditions, a quotient C-system of CC whose associated pre-B-system is $(Ob/\sim, \widetilde{Ob}/approx)$. These conditions are shows in Proposition **??** of the present paper to be compatible with isomorphisms of pre-B-systems.

After defining $C(\mathbf{RR},\mathbf{LM})$ in Sections **??** and **??** we define, in Construction **??**, a pre-B-system structure on the sets $(B(\mathbf{RR},\mathbf{LM}), \widetilde{B}(\mathbf{RR},\mathbf{LM}))$ and, in Constrcution **??**, an isomorphism between this pre-B-system and the pre-B-system of $C(\mathbf{RR},\mathbf{LM})$.

Using this isomorphism we formulate the conditions on subsets $B \subset B(\mathbf{RR},\mathbf{LM})$ and $\widetilde{B} \in \widetilde{B}(\mathbf{RR},\mathbf{LM})$ that are necessary and sufficient for such a pair to correspond to a sub-C-system of $C(\mathbf{RR},\mathbf{LM})$.

Next, we construct, for any subsets B, $\widetilde{B}$ and Beq, $\widetilde{Beq}$ a pair of relations $\sim$ and $\approx$ on B and $\widetilde{B}$ respectively and show under which condition on Beq, $\widetilde{Beq}$ the transport of these relations to the subsets of $(Ob(C(\mathbf{RR},\mathbf{LM})), \widetilde{Ob}(C(\mathbf{RR}, LM)))$ corresponding to B, $\widetilde{B}$, satisfy the conditions of [41, Prop. 5.4] and therefore define a (regular) quotient of the C-system corresponding to $(B, \widetilde{B})$.

Summing it up in Construction **??** we obtain a construction, for any four subsets B,$\widetilde{B}$,Beq,$\widetilde{Beq}$ satisfying certain condition, of a C-system. This is the final construction of the paper.

It should be possible to approach this construction in a more direct way. First, one would define the concept of a B-system as a pre-B-system whose operations satisfy some axioms. Next one would show that for a C-system CC the pre-B-system $(Ob(CC), \widetilde{Ob}(CC))$ is a B-system and that the mapping

$$CC \mapsto (Ob(CC), \widetilde{Ob}(CC))$$

is the object component of a functor from the category of C-systems to the category of B-systems with obviously defined homomorphisms as morphisms. Next, one would construct, using in particular an abstract analog of [20, Def. 2.11], a functor in the opposite direction, from B-systems to C-systems. Finally, one would construct two functor isomorphisms extending this pair of functors to an equivalence between the categories of C-systems and B-systems.

Then one would prove that for any **RR**, **LM** the pre-B-system structure that we define on $(B(\mathbf{RR}, \mathbf{LM}), \widetilde{B}(\mathbf{RR}, \mathbf{LM}))$ is a B-system structure. A sub-pre-B-system of a B-system is a B-system. Therefore the subsets $(B, \widetilde{B})$ should define a B-system and the relations $(\sim, \approx)$ constructed from the subsets $(Beq, \widetilde{Beq})$ should be a congruence relation such that the quotients $(B/\sim, \widetilde{B}/\approx)$ again carry a structure of a B-system. Finally, applying the inverse functor to this B-system we would obtain a C-system that will be the C-system corresponding to the quadruple of sets B,$\widetilde{B}$,Beq and $\widetilde{Beq}$.

Such a direct construction would probably be more satisfying than the one that we provide. However, it would require a lot of non-trivial work and, as far as the goal of constructing a C-system from the subsets B,$\widetilde{B}$,Beq and $\widetilde{Beq}$, will give the same result as our less direct, but more simple approach. Still, defining B-systems and constructing an equivalence between the categories of C-systems and B-systems is important and we plan to address it in future papers. The approach that we take here, using the results of [41] instead, gives us a rigorous construction that can be completed today.

We use neither the axiom of excluded middle nor the axiom of choice. The paper is written in the formalization-ready style and should be easily formalizable both in the UniMath and in the ZF.

We use the diagrammatic order of composition, i.e., for morphisms $f : X \to Y$ and $g : Y \to Z$ we write their composition as $f \circ g$.

A category $\mathcal{C}$ is always understood as a pair of sets $Ob(\mathcal{C})$, $Mor(\mathcal{C})$ connected by the operations of domain, codomain, identity and composition where composition is a partially defined operation. A functor $F : \mathcal{C} \to \mathcal{D}$ is a pair of functions $F_{Ob} : Ob(\mathcal{C}) \to Ob(\mathcal{D})$, $F_{Mor} : Mor(\mathcal{C}) \to Mor(\mathcal{D})$ satisfying the well known conditions. We emphasize it here because it is also possible to define a category starting with a set $Ob(\mathcal{C})$ and a *family* of sets $Mor_{\mathcal{C}}(X, Y)$ parametrized by $X, Y \in Ob(\mathcal{C})$ where a

family is understood in the sense of [40, Remark 3.9]. These two concepts are a little different from each other. For example there exists a category with $Ob(\mathcal{C}) = \{0, 1\}$ and $Mor_{\mathcal{C}}(a, b) = \{0\}$ for all $a, b \in Ob(\mathcal{C})$, in the sense of the second definition, but not in the sense of the first. Indeed, any category in the sense of the first definition has the property that

$$Mor(X, Y) \cap Mor(X', Y') = \emptyset$$

if $X \neq X'$ or $Y \neq Y'$.

Note that the expression "for all x in A, a $y(x)$ in $B(x)$" is a form of saying "a family y, parametrized by x in A, such that $y(x)$ is in $B(x)$". For a way to define a family of sets in the ZFC without a universe see [40, Remark 3.9]. Families of sets do not form a set. However, families of sets with a given parameter set and such that all the sets of the family are subsets of given set do. A formulation in terms of iterated families can be used for "for all" expressions where there are several parameters such as in Definition 2.5(3). Similarly, a "collection of data", is understood as an n-tuple, which is understood as an iterated pair $(\dots(-,-),-)\dots,-)$.

We fix a universe U without making precise what conditions on the set U we require. It is clear that it is sufficient for all constructions of this paper to require U to be a Grothendieck universe. However, it is likely that a much weaker set of conditions on U is sufficient for our purposes. In all that follows we write $Sets$ instead of $Sets(U)$.

This is one the papers extending the material which I started to work on in [32]. I would like to thank the Institute Henri Poincare in Paris and the organizers of the "Proofs" trimester for their hospitality during the preparation of the first version of this paper. The work on this paper was facilitated by discussions with Benedikt Ahrens, Richard Garner and Egbert Rijke.

2. Monads and relative monads

Let us start by reminding the definition of a monad in the form that became standard after MacLane's textbook [23, p.133] and that we will call the *monoidal* form. For a category $\mathcal{C}$ we often write $X \in \mathcal{C}$ instead of $X \in Ob(C)$. For a functor $F = (F_{Ob}, F_{Mor})$ from $\mathcal{C}$ to $\mathcal{D}$ we often write $F(X)$ instead of $F_{Ob}(X)$ for $X \in Ob(\mathcal{C})$ and $F(f)$ instead of $F_{Mor}(f)$ for $f \in Mor(\mathcal{C})$. We emphasize these standard conventions here because below we will sometimes work with the object and morphism components of a functor separately in which case the subscripts Ob and Mor will need to be enforced.

Definition 2.1. *[2017.04.01.def1] A monad in the monoidal form on a category $\mathcal{C}$ is a triple $\mathbf{R} = (R, \eta, \mu)$ where $R : \mathcal{C} \to \mathcal{C}$ is a functor and $\eta : Id_{\mathcal{C}} \to R$, $\mu : R \circ R \to R$ are natural transformations such that for any $X \in \mathcal{C}$ one has*

1. *$R(\mu_X) \circ \mu_X = \mu_{R(X)} \circ \mu_X$ ("associativity"),*
2. *$\eta_{R(X)} \circ \mu_X = Id_{R(X)}$ and $R(\eta_X) \circ \mu_X = Id_{R(X)}$ (two "unity axioms").*

We will omit the qualification "in the monoidal form" when it is clear from the context.

The following definition specifies objects that we, following [26], will call "monads in the Kleisli form".

Definition 2.2. */2017.04.13.def1/ A monad in the Kleisli form on a category $\mathcal{C}$ is a triple $\mathbf{RR} = (RR_{Ob}, \eta, rr)$ where*

1. *$RR_{Ob} : Ob(\mathcal{C}) \to Ob(\mathcal{C})$ is a function,*
2. *η is a family, parametrized by $X \in Ob(\mathcal{C})$, of morphisms $\eta_X : X \to RR_{Ob}(X)$,*
3. *rr is a family, parametrized by pairs $X, Y \in Ob(\mathcal{C})$, of functions*

$$rr_{X,Y} : Mor_{\mathcal{C}}(X, RR_{Ob}(Y)) \to Mor_{\mathcal{C}}(RR_{Ob}(X), RR_{Ob}(Y))$$

such that

4. *for all $X \in \mathcal{C}$, $rr_{X,X}(\eta_X) = Id_{RR_{Ob}(X)}$,*
5. *for all X, Y, $f : X \to RR_{Ob}(Y)$, $\eta_X \circ rr_{X,Y}(f) = f$,*
6. *for all X, Y, Z, $f : X \to RR_{Ob}(Y)$, $g : Y \to RR_{Ob}(Z)$,*

$$rr_{X,Y}(f) \circ rr_{Y,Z}(g) = rr_{X,Z}(f \circ rr_{Y,Z}(g))$$

We will omit the qualification "in the Kleisli form" when it is clear from the context. Definition 2.1 is equivalent to the Definition 2.2 in the precise sense that is specified by Problem 2.3 and following it Construction 2.4.

Problem 2.3. */2017.01.04.prob1/ Given a function $RR_{Ob} : Ob(\mathcal{C}) \to Ob(\mathcal{C})$ and a family η, parametrized by $X \in Ob(\mathcal{C})$, of morphisms $\eta_X : X \to RR_{Ob}(X)$, to construct a bijection between the following two sets:*

1. *the set of pairs of the form*
   - *1.1. a function $RR_{Mor} : Mor(\mathcal{C}) \to Mor(\mathcal{C})$,*
   - *1.2. a family, parametrized by $X \in Ob(\mathcal{C})$, of morphisms*

$$\mu_X : RR_{Ob}(RR_{Ob}(X)) \to RR_{Ob}(X)$$

   such that $((RR_{Ob}, RR_{Mor}), \eta, \mu)$ is a monad on $\mathcal{C}$ in the monoidal form, that is,
   - *1.3. $RR = (RR_{Ob}, RR_{Mor})$ is a functor,*
   - *1.4. (RR_{Ob}, μ) satisfies condition (1) of Definition 2.1,*
   - *1.5. (RR_{Ob}, η, μ) satisfies condition (2) of Definition 2.1,*
   - *1.6. η is a natural transformation $Id_{\mathcal{C}} \to RR$,*
   - *1.7. μ is a natural transformation $RR \circ RR \to RR$,*
2. *the set of families, parametrized by $X, Y \in Ob(\mathcal{C})$, of functions*

$$rr_{X,Y} : Mor_{\mathcal{C}}(X, RR_{Ob}(Y)) \to Mor_{\mathcal{C}}(RR_{Ob}(X), RR_{Ob}(Y))$$

   such that (RR_{Ob}, η, rr) is a monad on $\mathcal{C}$ in the Kleisli form, that is,
   - *2.1. (RR_{Ob}, η, rr) satisfies condition (1) of Definition 2.2,*
   - *2.2. (RR_{Ob}, η, rr) satisfies condition (2) of Definition 2.2,*
   - *2.3. (RR_{Ob}, rr) satisfies condition (3) of Definition 2.2.*

We have intentionally expanded the definitions of monads in the monoidal form and monads in the Kleisli form to show that that the expanded definition of the latter is much shorter than that of the former. Monads are used extensively in computer

science, but almost always in the Kleisli form and the fact that Kleisli form is much more concise than the monoidal form may be one of the reasons.

On the other hand, it is likely that monads in the Kleisli form have not been widely known among mathematicians because they are defined not as functors with a structure, but as functions between sets of objects with a structure. In particular, it is not obvious from their definition that monads in the Kleisli form can be transported along equivalences of categories.

We devote so much attention to these two definitions because they provide one of the clearest examples of how the same objects can have different and often mutually incomprehensible definitions in the parallel realities of mathematics and that part of theoretical computer science which is known as Theoretical Computer Science B.

We now outline the construction for Problem 2.3

Construction 2.4. **[2017.01.04.constr1]** To go from the monoidal form to the Kleisli form, one defines, for $X, Y \in Ob(\mathcal{C})$ and $f : X \to RR_{Ob}(Y)$

$$[\mathbf{2017.04.17.eq1}] rr_{X,Y}(f) = RR_{Mor}(f) \circ \mu_Y \tag{17}$$

To go from the Kleisli form to the monoidal form one defines

1. for $f : X \to Y$ in $Mor(\mathcal{C})$

$$[\mathbf{2017.04.17.eq2}] RR_{Mor}(f) = rr_{X,Y}(f \circ \eta_Y) \tag{18}$$

2. for $X \in Ob(\mathcal{C})$

$$[\mathbf{2017.04.17.eq2}] \mu_X = rr_{RR_{Ob}(X),X}(Id_{RR_{Ob}(X)}) \tag{19}$$

We leave the verification of the conditions and the proof that these functions are mutually inverse to the formally verified version of the paper. □

For a monad $\mathbf{RR}$ in the Kleisli form we let $\mathbf{RR}^M$ denote the corresponding monad in the monoidal form and a monad $\mathbf{R}$ in the monoidal form we let $\mathbf{R}^K$ denote the corresponding monad in Kleisli form.

The notion of a relative monad arises very naturally for monads in the Kleisli form. It was introduced in [6, Def.1, p.299] and considered in more detail in [7]. Let us remind it here.

Definition 2.5. *[2015.12.22.def1] Let $J : \mathcal{C} \to \mathcal{D}$ be a functor. A relative monad $\mathbf{RR}$ on J or a J-relative monad or a J-monad is a triple (RR_{Ob}, η, rr) where*

1. *$RR_{Ob} : Ob(\mathcal{C}) \to Ob(\mathcal{D})$ is a function,*
2. *for all X in $\mathcal{C}$, a morphism $\eta_X : J(X) \to RR_{Ob}(X)$ in $\mathcal{D}$,*
3. *for all X, Y in $\mathcal{C}$ and $f : J(X) \to RR_{Ob}(Y)$ in $\mathcal{D}$, a morphism*

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

in $\mathcal{D}$,

such that the following conditions hold:

4. *for all $X \in \mathcal{C}$, $rr_{X,X}(\eta_X) = Id_{RR_{Ob}(X)}$,*
5. *for all $X, Y \in \mathcal{C}$ and $f : J(X) \to RR_{Ob}(Y)$, $\eta_X \circ rr_{X,Y}(f) = f$,*

6. for all $X, Y, Z \in \mathcal{C}$, $f : J(X) \to RR_{Ob}(Y)$ and $g : J(Y) \to RR_{Ob}(Z)$,

$$rr_{X,Y}(f) \circ rr_{Y,Z}(g) = rr_{X,Z}(f \circ rr_{Y,Z}(g))$$

Sometimes one writes f^* instead of $rr_{X,Y}(f)$. It makes long computations look nicer, but one should remember that the notation f^* is under-specified because f^* depends not only on f but also on X and Y and it is possible that for example $RR_{Ob}(Y_1) = RR_{Ob}(Y_2)$ while $Y_1 \neq Y_2$.

Problem 2.6. */2016.01.15.prob1/* *Given a J-relative monad $\mathbf{RR} = (RR_{Ob}, \eta, rr)$ to construct a function $RR_{Mor} : Mor(\mathcal{C}) \to Mor(\mathcal{D})$ such that $\mathbf{RR}^f = (RR_{Ob}, RR_{Mor})$ is a functor.*

Construction 2.7. **[2016.01.15.constr1]**For $f : X \to Y$ in $\mathcal{C}$ set

$$[\mathbf{2017.04.05.eq3}]RR_{Mor}(f) = rr_{X,Y}(J(f) \circ \eta_Y) \tag{20}$$

The proofs of the composition and the identity axioms of a functor are straightforward. □

A relative monad on the identity functor of a category is precisely a monad on the corresponding category given in the Kleisli form. As the explicit form of functions $(-)^K$ and $(-)^M$ defined in Construction 2.4 shows that for $\mathbf{R} = (R, \eta, \mu)$ we have $\mathbf{R}^K = (R_{Ob}, \eta, rr)$ and for $\mathbf{RR} = (RR_{Ob}, \eta, rr)$ we have $\mathbf{RR}^M = (\mathbf{RR}^f, \eta, \mu)$. In particular, since $(\mathbf{R}^K)^M = \mathbf{R}$, we have

$$[\mathbf{2017.04.05.eq2}](\mathbf{R}^K)^f = R \tag{21}$$

The construction of the Kleisli category of a monad[3] was extended to the case of relative monads in [7, p.8] (see also [38, Constr. 2.9]). Since it plays an important role in what follows let us remind the definition of the corresponding category data here without proving that it actually defines a category.

Problem 2.8. */2017.03.12.prob3/* *Given a functor $J : \mathcal{C} \to \mathcal{D}$ and a J-monad $\mathbf{RR} = (RR, \eta, rr)$ to construct a category $K(\mathbf{RR})$ that will be called the Kleisli category of $\mathbf{RR}$.*

Construction 2.9. **[2017.03.12.constr3]** We set $Ob(K(\mathbf{RR})) = Ob(\mathcal{C})$ and

$$Mor(K(\mathbf{RR})) = \amalg_{X,Y \in K(\mathbf{RR})} Mor(J(X), RR(Y))$$

For $X, Y \in K(\mathbf{RR})$, we will identify the set of morphisms in $K(\mathbf{RR})$ from X to Y with the set $Mor(J(X), RR(Y))$ by means of the obvious bijections.

For $X \in Ob(\mathcal{C})$ we set $Id_{X,K(\mathbf{RR})} = \eta_X$.

For $f \in Mor_{\mathcal{D}}(J(X), RR(Y))$ and $g \in Mor_{\mathcal{D}}(J(Y), RR(Z))$ we set

$$[\mathbf{2017.04.05.eq1}]f \circ_{K(\mathbf{RR})} g := f \circ_{\mathcal{D}} rr_{Y,Z}(g) \tag{22}$$

□

[3]Actually Kleisli, in [22], introduced the corresponding category for what we would today call a comonad.

Problem 2.10. *[2017.04.05.prob2] In the context of Problem 2.8, to construct a functor*

$$Et_{\mathbf{RR}} : \mathcal{C} \to K(\mathbf{RR})$$

Construction 2.11. **[2017.04.05.constr2]** We omit the index $\mathbf{RR}$ at Et. We set $Et_{Ob} = Id$. For a morphism $f : X \to Y$ in $\mathcal{C}$ we set

$$[\mathbf{2017.04.09.eq2}] Et_{Mor}(f) = J(f) \circ \eta_Y \tag{23}$$

For $f = Id_X$ we have

$$Et(Id_X) = J(Id_X) \circ \eta_X = Id_{J(X)} \circ \eta_X = \eta_X = Id_{X,K(\mathbf{RR})}$$

This proves the identity axiom. For $f : X \to Y$, $g : Y \to Z$ we have

$$Et(f \circ g) = J(f \circ g) \circ \eta_Z = J(f) \circ J(g) \circ \eta_Z$$

and

$$Et(f) \circ Et(g) = (J(f) \circ \eta_Y) \circ_{K(\mathbf{RR})} (J(g) \circ \eta_Z) =$$
$$J(f) \circ \eta_Y \circ rr_{Y,Z}(J(g) \circ \eta_Z) = J(f) \circ J(g) \circ \eta_Z$$

where the second equality is by (22) and the third by condition (2) of Definition 2.5. This proves the composition axiom. □

Problem 2.12. *[2017.04.11.prob1] Let $J : \mathcal{C} \to \mathcal{D}$ be a functor and $\mathbf{RR} = (RR_{Ob}, \eta, rr)$ a J-relative monad. To construct a functor $\mathbf{RR}^{lm} : K(\mathbf{RR}) \to \mathcal{D}$.*

Construction 2.13. **[2017.04.09.constr1]** We set $\mathbf{RR}^{lm}_{Ob} = RR_{Ob}$. For

$$f \in Mor_{K(\mathbf{RR})}(X, Y) = Mor_{\mathcal{D}}(J(X), RR_{Ob}(Y))$$

where the equality is actually the bijection mentioned in the definition of $K(\mathbf{RR})$, we set $\mathbf{RR}^{lm}_{Mor}(f) = rr_{X,Y}(f)$.

The verification of the composition and identity axioms for $(\mathbf{RR}^{lm}_{Ob}, \mathbf{RR}^{lm}_{Mor})$ is straightforward and is left for the formalized version of the paper. □

So far our only examples of relative monad were provided by the "usual" monads in the Kleisli form, in particular, they all were endo-monads. The following construction allows one to obtain a large class of relative monads that are not endo-monads.

Problem 2.14. *[2017.02.24.prob1] Given functors $F : \mathcal{C}_0 \to \mathcal{C}_1$, $J : \mathcal{C}_1 \to \mathcal{D}$ and a J-relative monad $\mathbf{RR}$ to construct a $(F \circ J)$-relative monad $F^\circ(\mathbf{RR})$.*

Construction 2.15. *[2017.02.24.constr1]* We omit the indexes Ob and Mor at F and J. Let $\mathbf{RR} = (RR_{Ob}, \eta, rr)$. We set

$$[\mathbf{2017.04.11.eq3}] F^\circ(\mathbf{RR}) = (F^\circ(RR_{Ob}), F^\circ(\eta), F^\circ(rr)) \tag{24}$$

where, for for $X \in \mathcal{C}_0$

$$[\mathbf{2017.04.11.eq1}] F^\circ(RR_{Ob})(X) = RR_{Ob}(F(X)) \qquad F^\circ(\eta)_X = \eta_{F(X)} \tag{25}$$

and for $X, Y \in \mathcal{C}_0$ and $f : J(F(X)) \to RR_{Ob}(F(Y))$,

$$[\mathbf{2017.04.11.eq2}] F^\circ(rr)_{X,Y}(f) = rr_{F(X),F(Y)}(f) \tag{26}$$

Let us verify conditions (1)-(3) of Definition 2.5. We write RR instead of RR_{Ob} :

1. Let $X \in \mathcal{C}_0$. Then
$$F^\circ(rr)_{X,X}(F^\circ(\eta)_X) = rr_{F(X),F(X)}(\eta_{F(X)}) = Id_{RR(F(X))} = Id_{F^\circ(RR)(X)}$$
2. Let $X, Y \in \mathcal{C}_0$ and $f : J(F(X)) \to RR(F(Y))$. Then
$$F^\circ(\eta)_X \circ F^\circ(rr)_{X,Y}(f) = \eta_{F(X)} \circ rr_{F(X),F(Y)}(f) = f$$
3. Let $X, Y, Z \in \mathcal{C}_0$ and
$$f : J(F(X)) \to RR(F(Y)) \qquad g : J(F(Y)) \to RR(F(Z))$$
are morphisms in $\mathcal{D}$. Then
$$F^\circ(rr)_{X,Y}(f) \circ F^\circ(rr)_{Y,Z}(g) = rr_{F(X),F(Y)}(f) \circ rr_{F(Y),F(Z)}(g) =$$
$$rr_{F(X),F(Z)}(f \circ rr_{F(Y),F(Z)}(g)) = F^\circ(rr)_{X,Z}(f \circ F^\circ(rr)_{Y,Z}(g))$$

This completes Construction 2.15. □

Lemma 2.16. */2017.04.09.l1/* *In the context of Problem 2.14 one has*

$$[\mathbf{2017.04.17.eq6}](F^\circ(\mathbf{RR}))^f = F \circ RR^f \tag{27}$$

Proof. In what follows we omit the indexes Ob and Mor at F and J. Both the left and right hand side of (27) are functors $\mathcal{C}_0 \to \mathcal{D}$. Let $\mathbf{RR} = (RR_{Ob}, \eta, rr)$. Then both of these functors on objects are given, by construction, by $F \circ RR_{Ob}$. It remains to show that for $f :\to Y$ in $\mathcal{C}_0$ we have

$$[\mathbf{2017.04.09.eq5}]F^\circ(\mathbf{RR})_{Mor}(f) = (F_{Mor} \circ RR_{Mor})(f) \tag{28}$$

We have

$$F^\circ(\mathbf{RR})_{Mor}(f) = F^\circ(rr)((F \circ J)(f) \circ F^\circ(\eta)_Y) = rr_{F(X),F(Y)}(J(F(f)) \circ \eta_{F(Y)})$$

where the first equality is by (20) and (24) and the second by (25) and (26). On the other hand

$$(F_{Mor} \circ RR_{Mor})(f) = RR_{Mor}(F(f)) = rr_{F(X),F(Y)}(J(F(f)) \circ \eta_{F(Y)})$$

where the second equality is by (20). This completes the proof of the lemma. □

Remark 2.17. **[2017.04.09.rem1]** Note that in the construction of $F^\circ(\mathbf{RR})$ there participate only F_{Ob}, but not F_{Mor}. On the other hand we have (28). The explanation for this seeming contradiction is that $F^\circ(\mathbf{RR})$ is a $(F \circ J)$-relative monad and $(F \circ J)_{Mor}$, and, therefore F_{Mor}, participates in the definition of $F^\circ(\mathbf{RR})_{Mor}$.

Problem 2.18. */2017.03.12.prob1/* *Given functors $F : \mathcal{C}_0 \to \mathcal{C}_1$ and $J : \mathcal{C}_1 \to \mathcal{D}$, and a J-monad $\mathbf{RR}$ to construct a functor $F_{\mathbf{RR}} : K(F^\circ(\mathbf{RR})) \to K(\mathbf{RR})$.*

Construction 2.19. **[2017.03.12.constr2]** We set $F_{\mathbf{RR},Ob} = F_{Ob}$. For $X, Y \in \mathcal{C}_0$ and a morphism $f : J(F(X)) \to RR(F(Y))$ in $K(F^\circ(\mathbf{RR}))$ from X to Y, we set

$$[\mathbf{2017.03.12.eq2}]F_{\mathbf{RR},X,Y}(f) = f \tag{29}$$

that is, $F_{\mathbf{RR},Mor}$ is given by the function

$$\amalg_{X_0,Y_0 \in \mathcal{C}_0} Mor_{\mathcal{D}}(J(F(X_0)), RR(F(Y_0))) \to \amalg_{X_1,Y_1 \in \mathcal{C}_1} Mor_{\mathcal{D}}(J(X_1), RR(Y_1))$$

of the form $((X_0, Y_0), f) \mapsto ((F(X_0), F(Y_0)), f)$.

Let us show that $(F_{\mathbf{RR},Ob}, F_{\mathbf{RR},Mor})$ is a functor.

Note first that for $X \in \mathcal{C}_0$ one has

$$[\mathbf{2017.03.12.eq1}] Id_{K(F^\circ(\mathbf{RR})),X} = F^\circ(\eta)_X = \eta_{F(X)} \tag{30}$$

where the first equality is by Construction 2.9 and the second by Construction 2.15. The data $(F_{\mathbf{RR},Ob}, F_{\mathbf{RR},Mor})$ satisfies the identity axiom because of the equalities

$$F_{\mathbf{RR},X,X}(Id_{K(F^\circ(\mathbf{RR})),X}) = F_{\mathbf{RR},X,X}(\eta_{F(X)}) = \eta_{F(X)} = Id_{K(\mathbf{RR}),F(X)}$$

where the first equality is by (30), the second by (29) and the third by Construction 2.9.

Next, for $X, Y, Z \in \mathcal{C}_0$ and $f : J(F(X)) \to RR(F(Y))$, $g : J(F(Y)) \to RR(F(Z))$ one has

$$[\mathbf{2017.03.12.eq3}] f \circ_{KK(F^\circ(\mathbf{RR}))} g = f \circ_{\mathcal{D}} F^\circ(rr)_{Y,Z}(g) = f \circ_{\mathcal{D}} rr_{F(Y),F(Z)}(g) \tag{31}$$

where, again, the first equality is by Construction 2.9 and the second by Construction 2.15. The data $(F_{\mathbf{RR},Ob}, F_{\mathbf{RR},Mor})$ satisfies the composition axiom because of the equalities

$$F_{\mathbf{RR},X,Z}(f \circ_{K(F^\circ(\mathbf{RR}))} g) = F_{\mathbf{RR},X,Z}(f \circ_{\mathcal{D}} rr_{F(Y),F(Z)}(g)) =$$
$$f \circ_{\mathcal{D}} rr_{F(Y),F(Z)}(g) = f \circ_{K(\mathbf{RR})} g = F_{\mathbf{RR},X,Y}(f) \circ_{K(RR)} F_{\mathbf{RR},Y,Z}(g)$$

where the first equality is by (31), the second by (29), the third by Construction 2.9 and the fourth again by (29).

This completes Construction 2.19. □

3. Modules over monads and modules over relative monads

A left module in the monoidal form over a monad in the monoidal form is defined as follows (cf. [17, p.222]).

Definition 3.1. */2017.04.01.def2/ Let $\mathcal{C}$,$\mathcal{E}$ be categories, $\mathbf{R} = (R, \eta, \mu)$ be a monad on $\mathcal{C}$, and $L : \mathcal{C} \to \mathcal{E}$ a functor. A (left) $\mathbf{R}$-module structure on L is a natural transformation $\rho : R \circ L \to L$ such that for all $X \in \mathcal{C}$ one has:*

1. $L(\mu_X) \circ \rho_X = \rho_{R(X)} \circ \rho_X$,
2. $L(\eta_X) \circ \rho_X = Id_{L(X)}$.

A left $\mathbf{R}$-module in the monoidal form with values in $\mathcal{E}$ is a pair $\mathbf{L} = (L, \rho)$ where $L : \mathcal{C} \to \mathcal{E}$ is a functor and ρ an $\mathbf{R}$-module structure on L.

Example 3.2. */2017.04.15.ex1/*For a monad $\mathbf{R} = (R, \eta, \rho)$ the pair $\mathbf{R}^{lm} = (R, \rho)$ is a left module over $\mathbf{R}$.

Left modules can also be defined in Kleisli form.

Definition 3.3. */2017.04.15.def1/ Let $\mathcal{C}$,$\mathcal{E}$ be categories and $\mathbf{RR} = (RR_{Ob}, \eta, rr)$ a monad on $\mathcal{C}$. A (left) $\mathbf{RR}$-module with values in $\mathcal{E}$ in the Kleisli form is a pair (LM_{Ob}, lm) where $LM_{Ob} : Ob(\mathcal{C}) \to Ob(\mathcal{E})$ is a function and lm is a family, parametrized by $X, Y \in Ob(\mathcal{C})$ of functions*

$$lm_{X,Y} : Mor_{\mathcal{C}}(X, RR_{Ob}(Y)) \to Mor_{\mathcal{E}}(LM_{Ob}(X), LM_{Ob}(Y))$$

such that

1. *for all* $X \in \mathcal{C}$, $lm_{X,X}(\eta_X) = Id_{LM_{Ob}(X)}$,
2. *for all* $X, Y, Z \in \mathcal{C}$, $f : X \to RR_{Ob}(Y)$, $g : Y \to RR_{Ob}(Z)$,

$$lm_{X,Y}(f) \circ lm_{Y,Z}(g) = lm_{X,Z}(f \circ rr_{Y,Z}(g))$$

As in the case of monads the monoidal and Kleisli forms of left modules are equivalent in the following sense.

Problem 3.4. */2017.04.03.prob1/* *Given categories* $\mathcal{C}$, $\mathcal{E}$, *a monad* $\mathbf{RR} = (RR_{Ob}, \eta, rr)$ *on* $\mathcal{C}$, *and a function* $LM_{Ob} : Ob(\mathcal{C}) \to Ob(\mathcal{E})$ *to construct a bijection between the following two sets:*

1. *the set of pairs* (LM_{Mor}, ρ) *where* $LM_{Mor} : Mor(\mathcal{C}) \to Mor(\mathcal{E})$ *is a function such that* $LM = (LM_{Ob}, LM_{Mor})$ *is a functor and* (LM, ρ) *is a left* $\mathbf{RR}^M$*-module in the monoidal form,*
2. *the set of families* lm, *parametrized by* $X, Y \in Ob(\mathcal{C})$, *of functions*

$$lm_{X,Y} : Mor_{\mathcal{C}}(X, RR_{Ob}(Y)) \to Mor_{\mathcal{C}}(LM_{Ob}(X), LM_{Ob}(Y))$$

such that (LM_{Ob}, lm) *is a left* $\mathbf{RR}$*-module in the Kleisli form.*

Construction 3.5. **[2017.04.03.constr1]** In one direction, given LM_{Mor}, a family $\rho_X : LM(RR_{Ob}(X)) \to LM(X)$ parametrized by $X \in Ob(\mathcal{C})$, and $f : X \to R(Y)$, one defines

$$[\mathbf{2017.04.09.eq4}] lm_{X,Y}(f) = LM(f) \circ \rho_Y \tag{32}$$

In the other direction, given LM_{Ob} and lm, one defines, for $f : X \to Y$,

$$[\mathbf{2017.04.09.eq3}] LM_{Mor}(f) = lm_{X,Y}(f \circ \eta_Y) \tag{33}$$

and for X,

$$\rho_X = lm_{RR_{Ob}(X),X}(Id_{RR_{Ob}(X)})$$

We leave the verification of the conditions and the proof that these functions are mutually inverse to the formally verified version of the paper. □

Left $\mathbf{RR}$-modules in the Kleisli form with values in $\mathcal{E}$ are precisely the (covariant) functors from the Kleisli category of $\mathbf{RR}$ to $\mathcal{E}$, see below.

Left modules over relative monads were introduced in [3, Definition 9]. One can observe by direct comparison of unfolded definitions that there is a bijection between the set of modules over a relative monad $\mathbf{RR}$ with values in a category $\mathcal{E}$ and the set of functors from the Kleisli category $K(\mathbf{RR})$ of $\mathbf{RR}$ to $\mathcal{E}$. Whether this bijection is the identity bijection or not depends on how the expressions such as "collection of data" or "family of functions" are translated into the formal constructions of set theory. We assume that in this particular case they have been translated in a such a way that this bijection is the identity and left modules over $\mathbf{RR}$ with values in $\mathcal{E}$ are actually and precisely the same as (covariant) functors from $K(\mathbf{RR})$ to $\mathcal{E}$.

Definition 3.6. */2017.03.16.def1/* *Let* $J : \mathcal{C} \to \mathcal{D}$ *be a functor and* $\mathbf{RR} = (RR_{Ob}, \eta, rr)$ *a* J*-monad. A left module over* $\mathbf{RR}$ *with values in a category* $\mathcal{E}$ *is a functor* $\mathbf{LM}$:

$K(\mathbf{RR}) \to \mathcal{E}$, that is, a pair (LM_{Ob}, lm) where $LM_{Ob} : Ob(\mathcal{C}) \to Ob(\mathcal{E})$ is a function and lm is a family, parametrized by $X, Y \in Ob(\mathcal{C})$, of functions

$$lm_{X,Y} : Mor_{\mathcal{D}}(J(X), RR_{Ob}(Y)) \to Mor_{\mathcal{E}}(LM_{Ob}(X), LM_{Ob}(Y))$$

such that

1. *for all $X \in \mathcal{C}$, $lm_{X,X}(\eta_X) = Id_{LM_{Ob}(X)}$,*
2. *for all $X, Y, Z \in \mathcal{C}$, $f : X \to RR_{Ob}(Y)$, $g : Y \to RR_{Ob}(Z)$,*

$$lm_{X,Y}(f) \circ lm_{Y,Z}(g) = lm_{X,Z}(f \circ rr_{Y,Z}(g))$$

We will say that $\mathbf{LM}$ is an $\mathbf{RR}$-module if it is a left $\mathbf{RR}$-module. We will say that $\mathbf{LM}$ is a module over $\mathbf{RR}$ without specifying $\mathcal{E}$ if $\mathcal{E} = \mathcal{D}$.

From the unfolded definition we see that the left modules over an Id-relative monad are exactly the same as the left modules in the Kleisli form over the corresponding monad. Following the notation $\mathbf{R}^K$ for the monad in the Kleisli form corresponding to a monad $\mathbf{R}$ in the monoidal form, we let $\mathbf{L}^K$ denote the module over $\mathbf{R}^K$ in the Kleisli form corresponding to a module $\mathbf{L}$ over $\mathbf{R}$ in the monoidal form.

Definition 3.7. */2017.04.05.def1/ Let $J : \mathcal{C} \to \mathcal{D}$ be a functor, $\mathbf{RR}$ a J-monad and $\mathbf{LM}$ a left module over $\mathbf{RR}$ with values in a category $\mathcal{E}$. We define the functor $\mathbf{LM}^f : \mathcal{C} \to \mathcal{E}$ corresponding to $\mathbf{LM}$ as the composition $Et_{\mathbf{RR}} \circ \mathbf{LM}$.*

Explicitly, for $\mathbf{LM} = (LM_{Ob}, lm)$, we have

$$\mathbf{LM}^f_{Ob} = LM_{Ob}$$

which follows from $Et_{\mathbf{RR},Ob} = Id_{Ob(\mathcal{C})}$, and for $f : X \to Y$

$$[\mathbf{2017.04.11.eq5}]\mathbf{LM}^f_{Mor}(f) = lm(J(f) \circ \eta_Y) \tag{34}$$

which follows from (23).

As in the case of $\mathbf{RR}^f$ we will use the notation $\mathbf{LM}^f_{Ob}$ and $\mathbf{LM}^f_{Mor}$, with or without the subscripts Ob and Mor as our preferential notation for the corresponding objects.

If $\mathbf{R}$ and $\mathbf{L} = (L, \rho)$ are a monad on $\mathcal{C}$ and a left module over it with values in $\mathcal{E}$ given in the monoidal form then we have

$$[\mathbf{2017.04.17.eq4}](\mathbf{L}^K)^f = L \tag{35}$$

On objects we have $(\mathbf{L}^K)^f_{Ob} = L_{Ob}$ by construction. It remains to show that

$$[\mathbf{2017.04.09.eq1}](\mathbf{L}^K)^f_{Mor} = L_{Mor} \tag{36}$$

Indeed, for $f : X \to Y$ in $Mor(\mathcal{C})$ we have

$$(\mathbf{L}^K)^f_{Mor}(f) = \mathbf{L}^K_{Mor}(Et_{\mathbf{R}^K,Mor}(f)) =$$

$$\mathbf{L}^K_{Mor}(f \circ \eta_Y) = L_{Mor}(f \circ \eta_Y) \circ \rho_Y = L_{Mor}(f) \circ (L_{Mor}(\eta_Y) \circ \rho_Y) =$$

$$L_{Mor}(f) \circ Id_{L(Y)} = L_{Mor}(f)$$

where the first equality is by Definition 3.7, the second by (23), the third by (32), the fourth by the composition axiom for L and associativity of composition of $\mathcal{E}$, the fifth by Definition 3.1(1) and the sixth by the right unity axiom of $\mathcal{E}$.

Example 3.8. **[2017.04.15.ex2]** Construction 2.13 gives us, for any $J : \mathcal{C} \to \mathcal{D}$ and any J-monad $\mathbf{RR}$ a left module $\mathbf{RR}^{lm}$ over $\mathbf{RR}$ with values in $\mathcal{D}$. If $\mathbf{RR} = (RR_{Ob}, \eta, rr)$ then $\mathbf{RR}^{lm} = (RR_{Ob}, rr)$. This is the same relationship as in the case of monads in the monoidal form where for $\mathbf{R} = (R, \eta, \mu)$ we have $\mathbf{R}^{lm} = (R, \mu)$.

We have

$$(\mathbf{RR}^{lm})^f = \mathbf{RR}^f$$

Indeed, for $\mathbf{RR} = (RR_{Ob}, \eta, \mu)$ both functors are given by RR_{Ob} on objects and on morphisms they also coincide by construction because (34) becomes (20) when $lm = rr$.

When $J = Id_{\mathcal{C}}$ we also have

$$(\mathbf{R}^K)^{lm} = (\mathbf{R}^{lm})^K$$

Indeed, for $\mathbf{R} = (R, \eta, \mu)$ we have

$$(\mathbf{R}^K)^{lm} = (R_{Ob}, \eta, rr(R_{Mor}, \mu))^{lm} = (R_{Ob}, rr(R_{Mor}, \mu))$$

and

$$(\mathbf{R}^{lm})^K = (R, \mu)^K = (R_{Ob}, lm(R_{Mor}, \mu))$$

and $rr(R_{Mor}, \mu)$ and $lm(R_{Mor}, \mu)$ coincide by construction because formulas (17) and (32) become the same when $RR_{Mor} = LM_{Mor}$ and $\mu = \rho$.

Problem 3.9. *[2017.03.12.prob2] Given functors $F : \mathcal{C}_0 \to \mathcal{C}_1$ and $J : \mathcal{C}_1 \to \mathcal{D}$, a J-monad $\mathbf{RR}$ and an $\mathbf{RR}$-module $\mathbf{LM}$ with values in $\mathcal{E}$ to construct an $F^\circ(\mathbf{RR})$-module $F^\circ(\mathbf{LM})$ with values in $\mathcal{E}$.*

Construction 3.10. **[2017.03.12.constr1]** We need to construct a functor $K(F^\circ(\mathbf{RR})) \to \mathcal{E}$. We define this functor as the composition $F_{\mathbf{RR}} \circ \mathbf{LM}$, where $F_{\mathbf{RR}}$ is defined in Construction 2.19. Explicitly, for $\mathbf{RR} = (RR_{Ob}, \eta, rr)$ and $\mathbf{LM} = (LM_{Ob}, lm)$, we let $F^\circ(\mathbf{LM}) = (F^\circ(LM_{Ob}), F^\circ(lm))$. In this notation we have

$$[\mathbf{2017.04.17.eq7}] F^\circ(LM_{Ob}) = F \circ LM_{Ob} \tag{37}$$

and

$$[\mathbf{2017.04.17.eq8}] F^\circ(lm)_{X,Y} = lm_{F(X),F(Y)} \tag{38}$$

□

Lemma 3.11. *[2017.04.17.l1] In the context of Problem 3.9 we have*

$$[\mathbf{2017.04.13.eq2}] (F^\circ(\mathbf{LM}))^f = F \circ \mathbf{LM}^f \tag{39}$$

Proof. The equality

$$(F^\circ(\mathbf{LM}))^f_{Ob} = (F \circ \mathbf{LM}^f)_{Ob}$$

is by construction and to prove the equality

$$[\mathbf{2017.04.13.eq1}] (F^\circ(\mathbf{LM}))^f_{Mor} = (F \circ \mathbf{LM}^f)_{Mor} \tag{40}$$

we have, for $f : X \to Y$ in $\mathcal{C}_0$,

$$(F^\circ(\mathbf{LM}))^f_{Mor}(f) = F^\circ(lm)_{X,Y}((F \circ J)(f) \circ F^\circ(\eta)_Y) =$$
$$F^\circ(lm)_{X,Y}(J(F(f)) \circ \eta_{F(Y)}) = lm_{F(X),F(Y)}(J(F(f)) \circ \eta_{F(Y)}) =$$
$$\mathbf{LM}^f_{Mor}(F(f)) = (F \circ \mathbf{LM}^f_{Mor})(f)$$

where the first equality is by (34), the second by definition of $F \circ J$ and (25), the third by (38), the fourth by (34) and the fifth by the definition of $F \circ \mathbf{LM}^f_{Mor}$. This completes the proof of Lemma 3.11. □

Combining the previous results we obtain a solution to the following problem that we find convenient to formulate for the future reference.

Problem 3.12. */2017.04.05.prob1/* *Let $\mathcal{C}_0, \mathcal{C}_1, \mathcal{E}$ be categories. Let $\mathbf{R} = (R, \eta, \mu)$ be a monad on $\mathcal{C}_1$ and $\mathbf{L} = (L, \rho)$ a left $\mathbf{R}$-module with values in $\mathcal{E}$. Let further $J : \mathcal{C}_0 \to \mathcal{C}_1$ be a functor. To construct a pair of a J-relative monad module with values in $\mathcal{E}$ over it.*

Construction 3.13. **[2017.04.05.constr1]** We take $(J^\circ(\mathbf{R}^K), J^\circ(\mathbf{L}^K))$. □

Note that in the notation of Problem 3.12, we have

$$[\mathbf{2017.04.17.eq9}] J^\circ(\mathbf{R}^K)^f = J \circ (\mathbf{R}^K)^f = J \circ R \tag{41}$$

where the first equality is by (27) and the second by (21). Similarly,

$$[\mathbf{2017.04.17.eq10}] J^\circ({}^K)^f = J \circ (\mathbf{L}^K)^f = J \circ L \tag{42}$$

where the second equality is by (39) and the third by (35).

Constructions 2.15 and 3.9 show that both relative monads and left modules over them can be "precomposed" with any functor. The left modules can also be "post-composed" with any functor. It is done by literal post-composition. Since a left module is a functor $\mathbf{LM} : K(\mathbf{RR}) \to \mathcal{E}$ we can post-compose it with any functor $F : \mathcal{E} \to \mathcal{E}'$ and obtain a new module that we will denote $\mathbf{LM} \circ F$. Explicitly, for $\mathbf{LM} = (LM_{Ob}, lm)$ one has

$$\mathbf{LM} \circ F = (LM_{Ob} \circ F, F(lm))$$

where, for $X, Y \in \mathcal{C}$ and $f : J(X) \to RR_{Ob}(Y)$, one has

$$F(lm)_{X,Y}(f) = F(lm_{X,Y}(f))$$

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] Ji˘ ríAdámek. Free algebras and automata realizations in the language of categories. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.

[3] Benedikt Ahrens. Modules over relative monads for syntax and semantics. *Math. Structures Comput. Sci.*, 26(1):3–37, 2016.

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

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

[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] Nicolas Bourbaki. *Theory of sets*. Elements of Mathematics (Berlin). Springer-Verlag, Berlin, 2004. Reprint of the 1968 English translation [Hermann, Paris; MR0237342].
[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 linearity. In *Logic, language, information and computation*, volume 4576 of *Lecture Notes in Comput. Sci.*, pages 218–237. Springer, Berlin, 2007.
[18] André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Inform. and Comput.*, 208(5):545–564, 2010.
[19] André Hirschowitz and Marco Maggesi. Nested abstract syntax in Coq. *J. Automat. Reason.*, 49(3):409–426, 2012.
[20] 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.
[21] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. *Available at* `http://arxiv.org/abs/1211.2851`, 2012, 2014.
[22] H. Kleisli. Every standard construction is induced by a pair of adjoint functors. *Proc. Amer. Math. Soc.*, 16:544–546, 1965.
[23] S. MacLane. *Categories for the working mathematician*, volume 5 of *Graduate texts in Mathematics*. Springer-Verlag, 1971.
[24] 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.
[25] Ralph Matthes and Tarmo Uustalu. Substitution in non-wellfounded syntax with variable binding. *Theoret. Comput. Sci.*, 327(1-2):155–174, 2004.
[26] Eugenio Moggi. Notions of computation and monads. *Inform. and Comput.*, 93(1):55–92, 1991. Selections from the 1989 IEEE Symposium on Logic in Computer Science.
[27] 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.
[28] Moses Schönfinkel. Über die bausteine der mathematischen logik. *Math. Ann.*, 92:305 – 316; Jbuch 50, 23, 1924.
[29] Peter Selinger. The lambda calculus is algebraic. *J. Funct. Programming*, 12(6):549–566, 2002.
[30] 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.
[31] Jean van Heijenoort. *From Frege to Gödel. A source book in mathematical logic, 1879–1931*. Harvard University Press, Cambridge, Mass., 1967.
[32] Vladimir Voevodsky. Notes on type systems. 2009–2012. `https://github.com/vladimirias/old_notes_on_type_systems`.

[33] Vladimir Voevodsky. The equivalence axiom and univalent models of type theory. *arXiv 1402.5556*, pages 1–11, 2010. http://arxiv.org/abs/1402.5556.
[34] 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.
[35] Vladimir Voevodsky. An experimental library of formalized mathematics based on the univalent foundations. *Math. Structures Comput. Sci.*, 25(5):1278–1294, 2015.
[36] Vladimir Voevodsky. Lawvere theories and C-systems. *arXiv 1512.08104*, pages 1–15, 2015.
[37] Vladimir Voevodsky. Martin-Lof identity types in the C-systems defined by a universe category. *arXiv 1505.06446, under review in Publication IHES*, pages 1–51, 2015.
[38] Vladimir Voevodsky. Lawvere theories and Jf-relative monads. *arXiv 1601.02158*, pages 1–21, 2016.
[39] Vladimir Voevodsky. The (Π, λ)-structures on the C-systems defined by universe categories. *to appear in Theory and Applications of Categories*, pages 1–35, 2016.
[40] Vladimir Voevodsky. Products of families of types and (Π, λ)-structures on C-systems. *Theory Appl. Categ.*, 31(36):1044–1094, 2016. http://www.tac.mta.ca/tac/volumes/31/36/31-36.pdf.
[41] Vladimir Voevodsky. Subsystems and regular quotients of C-systems. In *Conference on Mathematics and its Applications, (Kuwait City, 2014)*, number 658 in Contemporary Mathematics, pages 127–137, 2016.
[42] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. *UniMath*: Univalent Mathematics. Available at https://github.com/UniMath.
[43] 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