Available online at www.sciencedirect.com

SciVerse ScienceDirect

Fuzzy Sets and Systems 256 (2014) 211–235

FUZZY sets and systems

www.elsevier.com/locate/fss

Fuzzy terms

Patrik Eklund[a], M.Ángeles Galán[b,1], Robert Helgesson[a,*], Jari Kortelainen[c]

[a] *Department of Computing Science, Umeå University, Sweden*
[b] *Department of Applied Mathematics, University of Málaga, Spain*
[c] *Department of Electrical Engineering and Information Technology, Mikkeli University of Applied Sciences, Finland*

Available online 6 March 2013

Dedicated to Werner Gähler on the occasion of his 80th birthday.

Abstract

In this paper we will show how purely categorical constructions of terms are advantageous when investigating situations concerning uncertainty; more specifically where uncertainty comes from and how uncertainty is integrated when dealing with terms over selected signatures. There are basically two ways of invoking uncertainty for terms. On one hand, we may proceed by building composed monads where uncertainty is provided by some suitable monad composed with the traditional term monad. On the other hand, we can provide a strictly formal basis for term monads being created over categories themselves carrying uncertainty. This is the distinction between 'computing with fuzzy' and 'fuzzy computing' and the fundamental question raised by these constructions is where uncertainty resides in language constructions for logic. This paper also shows how the notion of signature often needs to be expanded to levels of signatures, in particular when dealing with type constructors. Such levels allow us to strictly delineate, e.g., primitive operations, type terms, and value level terms. Levels of signature will in this paper be exemplified by the construction of the signature of simply typed lambda calculus.

Keywords: Term monads; Quantale; Algebra; Category theory

1. Introduction

In foundations, we need to be careful about the distinction between 'logic for mathematics' and 'mathematics for logic', and we need to be respectful about not mixing metalanguages with object languages. In this paper we do 'mathematics for logic', or to be more precise, 'category theory for logic'. Category theory then is our metalanguage for creating sets of terms and term functors. We use Zermelo–Fraenkel set theory including the Axiom of Choice (ZFC) as part of the set-theoretic metalanguage for category theory.

In our view, terms are really the foundational cornerstones of logic, and we also have to be precise about the underlying signatures containing sorts and operators. Substitutions are morphisms in the Kleisli category for the particular term monad and therefore fuzziness in terms carry over canonically to fuzziness in substitutions.

* Corresponding author. Tel.: +46 739728830.
E-mail addresses: peklund@cs.umu.se (P. Eklund), magalan@ctima.uma.es (M.A. Galán), rah@cs.umu.se (R. Helgesson), jari.kortelainen@mamk.fi (J. Kortelainen).

[1] Partially supported by Spanish projects TIN2012-39353-C04-01, P09-FQM-5233, TIN2009-14562-C05-01.

0165-0114/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.fss.2013.02.012

Type constructors manipulate sorts, and create new sorts, and this presents a situation where we need to understand the notion of levels of signatures. The concept of λ-terms can be handled within this realm, and this in fact provides the underlying notions of terms when moving towards fuzzy λ-calculus. Fuzzy description logic comes as a special case in these considerations.

The distinction between one-sortedness and many-sortedness is important as one-sorted constructions do not *per se* carry over to many-sorted situations. Here is also where type constructors need special attention, where underlying categories are important carriers of uncertainty.

Fuzziness is traditionally seen mostly as a level of truth, comparable with and as distinguished from probability. This provides a rather narrow set theoretic view of fuzzy logic as terms and sentences are left as crisp. In this paper we therefore strongly argue in favor of allowing uncertainty not just to be considered for algebras of truth values but also in relation to uncertainty values of operators. This means we open up the avenue for considering fuzzy terms, and thereby influencing fuzziness in sentences, and so on throughout all constructions of logic languages.

The history of expressions as appearing in argumentation and logic is manyfold. Further, logic in a historical perspective can be viewed in terms of the distinction between argument and argumentation. Focusing on argument invites to thinking that there is an overall logic, i.e., one language common to everybody in dialogues involving arguments. Focusing on argumentation leads to thinking about individual logics, i.e., communication between people is really communication between the logics adopted by respective individual. Individualization then also calls for making a distinction between the observer and the observation, i.e., the function and the function value.

Function symbols and abstractions have been dealt with ever since the late 19th century. Lambda-like expressions were proposed already by Frege [1,2] and Peano [3], and Hilbert's lectures followed by *Grundlagen* is a culmination in its own right. Contemporary developments of set theory are today the standard metalanguage, e.g., for category theory. During the time of Hilbert's lectures, Schönfinkel went a step further with his untyped combinators, although his ideas emerged before 1920, the paper, *Bausteine*, was not published until 1924. During the 1930s, Curry and Church worked intensively on developing groundwork for type theory and λ-calculus, and by 1940, simply typed λ-calculus had matured to its final form. Gödel and his incompleteness results are not directly related to the issue of terms, but rather to sentences and provability of sentences. In these respects, 'provability of sentence' is itself considered as a sentence, and this self-referentiality has indeed been under debate for almost a century. Whereas self-referentiality related to sets are seen leading to 'paradoxes', e.g., as pointed out by Russell, self-referentiality related to sentences leads to 'incompleteness'. Kleene develops deep results, e.g., for recursion culminating in his *Metamathematics*, but Kleene was surprisingly invisible during the course of further developments of type theory. The 1960s then is the time for model theory and forcing, and by the 1970s, category theory and universal algebra are already well established.

The outline of this paper is as follows. Sections 2–4 introduce not only the objectives and backgrounds but also provide the notation used in the subsequent formal constructions, more specifically, the necessary algebraic and categorical notions are briefly presented. In Section 5 we introduce many-sorted signatures as well as their fuzzy enrichment. Section 6 then covers the construction of term functors and monads over the defined signatures. In Section 7 we consider the application of such term monads in, e.g., type theory, λ-calculus, and description logic. Section 8 includes some notes on foundational matters and, finally, Section 9 concludes the paper and provides some perspectives for future investigations.

2. The need for formal descriptions of the term sets

In defining terms and term sets, the historical and contemporary sources in computing science and mathematics almost universally rely on 'verbal' definitions. For example, in [4] we have a definition similar to the following:

> Let $\Sigma = (S, \Omega)$ be a many-sorted signature with Ω as set of operators and S a set of sorts. Assume for each $s \in S$ a set of variables X_s. We then form terms as follows: A variable $x \in X_s$ is a term of sort s, constants $\omega :\rightarrow s$ are terms of sort s, and then inductively, if $\omega : s_1 \times \cdots \times s_n \rightarrow s$ is an n-ary operator, and $t_1, \ldots, t_n$ are terms of sort $s_1, \ldots, s_n$, respectively, then $\omega(t_1, \ldots, t_n)$ is a term of sort s.

This is then followed by a seemingly strict definitions of algebras:

> A Σ-algebra A for a signature $\Sigma = (S, \Omega)$ consists of a carrier set $A(s)$, for each $s \in S$. Further, for each $\omega : \longrightarrow s \in \Omega$, there is an element $A(\omega) \in A(s)$, for each $\omega : s_1 \times \cdots \times s_n \longrightarrow s \in \Omega$ with $n \geq 1$, there is a mapping
>
> $$A(\omega : s_1 \times \cdots \times s_n \longrightarrow s) : A(s_1) \times \cdots \times A(s_n) \longrightarrow A(s).$$

These constructions are then typically completed with definitions of term valuations as follows:

Let $t \in T_\Sigma X$ be a term for the signature $\Sigma = (S, \Omega)$, and $X = \bigcup_{s\in S} X_s$ a set of variables. The value of t in a Σ-algebra A together with variable assignment $\mathfrak{v} : X \longrightarrow A$ is denoted $A(\mathfrak{v})(t)$ and is defined inductively as

$$A(\mathfrak{v})(t) = \begin{cases} \mathfrak{v}_s(x) & \text{if } t = x \in X_s \\ A(\omega) & \text{if } t = \omega,\ \omega : \longrightarrow s \in \Omega \\ A(\omega)(A(\mathfrak{v})(t_1), \ldots, A(\mathfrak{v})(t_n)) & \text{if } t = \omega(t_1, \ldots, t_n), n \geq 1, \\ & \quad t_i \in \mathsf{T}_{\Sigma,s_i} X, 1 \leq i \leq n, \text{ and} \\ & \quad \omega : s_1 \times \cdots \times s_n \longrightarrow s \in \Omega \end{cases}$$

for each $s \in S$.

Note here that the set $T_\Sigma X$ at this point is not yet defined functorially.

Clearly, these informal definitions are in many cases sufficient and easily understood by readers. Unfortunately, however, such definitions give us little insight into the nature of terms and term sets. In particular, they give no information on suitable generalizations of terms and term sets. Indeed, in this non-categorical definition it is not even clear what a 'set' is. The typical assumption, unless otherwise stated, is that it resides in the metalanguage for an underlying set theory, which typically is Zermelo–Fraenkel's set theory. In a categorical framework, sets can reside in the underlying set theory, and also as objects in the category $\mathtt{Set}$ of sets.

More strict functorial constructions are well known and Manes [5] proved that the one-sorted term functor can be extended to a monad. However, the term functor constructions of Manes are still treated relatively informally. Purely categorical constructions of the one-sorted term monad over $\mathtt{Set}$ appeared much later, e.g., in [6].

Such purely categorical constructions present a clear path towards the generalizations we find lacking in the less strict term constructions. In the particular case of fuzziness they allow us to be very precise as to where the 'fuzziness' actually reside and we may indeed consider several possible answers to this question. One such view is obtained by considering the many-valued powerset monad composed with the term monad as described in [7,8]. Another possibility is reached by generalizing the term monad itself from $\mathtt{Set}$ to some category with richer structure, in particular the category $\mathtt{Set}(\mathfrak{Q})$, as described in [9], which we here will extend to $\mathtt{Set}(\mathfrak{Q})$ and $\mathtt{Set}(\mathfrak{K})$ for quantales and Kleene algebras, respectively.

Thus, there is an established view of terms and term sets based on strict categorical definition that invites to interesting non-classical extensions. This view is, however, strictly relegated to one-sorted signatures. We therefore find ourselves trapped when considering the move to more advanced applications where many-sorted signatures are the norm. It is often, and perhaps reasonably, assumed that extending the untyped term monad to the many-sorted case is trivial or at least straightforward. As the construction given in this text will demonstrate, this is not the case. The many-sorted extension of the term-functor is in fact quite intricate and care must be taken to ensure that it remains a monad so as to retain the possibility of further non-classical generalization.

More specifically, the categorical construction of the many-sorted term monad $\mathbf{T}_\Sigma$ over the "sorted category of sets" $\mathtt{Set}_S$ as provided in this paper, gives substitutions as morphism in the Kleisli category $(\mathtt{Set}_S)_{\mathbf{T}_\Sigma}$, where $\Sigma = (S, \Omega)$ is the underlying many-sorted signature, and S is the corresponding set of sorts. This then opens up avenues for many-sorted extensions and applications over them. One such application is in frameworks for institutions and general logic as initiated in [10].

Clearly, the notion of algebra and term valuation must be adapted for this form of generalized terms and importantly, we may in this adaption consider more detailed definitions. For example, we view valuation as a two step operation with the first step performing a term transformation taking 'syntactic' operators to 'semantic' operators and the second step being the actual valuation over a 'semantic term'. In this form of valuation we are, for example, better able to handle variable assignment in a compositional manner.

3. Lattices, quantales and Kleene algebras

In this section we recall the algebraic structures needed in further sections. There are lots of textbooks and monographs which may be used as "standard" references for algebras in general, for example [11,12]. In this context we give a

traditional non-categorical and one-sorted view to algebras while in further sections many-sorted algebras are treated categorically, syntactic and semantic parts clearly distinguished, to serve categorical development of logic.

At first, a pair (A, F), where A is a set and F is a set of n-ary ($n \leq k$) operations on A, is called an *algebra*, and we write $\mathfrak{A} = (A, F)$. The elements of F are called *finitary operations on* A, because $k \in \mathbb{N}$ is a given maximum arity. The 0-ary (nullary) operations take no arguments and they are considered as constants with values in A. The set A is called base set or *underlying set of* $\mathfrak{A}$. If F is finite with m elements then it is convenient to just list the finitary operations, for example, $\mathfrak{A} = (A, f_0, f_1, \ldots, f_{m-1})$. If some operation, say f_i, is a binary operation then we usually write for all $a, b \in A$, af_ib instead of $f_i(a, b)$ or $(a, b)f_i$.

When exploring different kinds of algebras it is a habit to define a collection of certain properties for algebras by means of *equational laws*. In the following we define certain algebras, lattices, and some of its derivatives (see [13,14] for detailed discussion on lattices and order):

A *lattice* $\mathfrak{L}$ is an algebra $(L, \vee, \wedge)$, where $\vee$ and $\wedge$ are binary operations *join* and *meet*, respectively. These binary operations must satisfy for all $a, b, c \in L$,

idempotency : $a \vee a = a, a \wedge a = a$,
commutativity : $a \vee b = b \vee a, a \wedge b = b \wedge a$,
associativity : $(a \vee b) \vee c = a \vee (b \vee c), (a \wedge b) \wedge c = a \wedge (b \wedge c)$,
absorption : $a \wedge (a \vee b) = a, a \vee (a \wedge b) = a$.

Algebras $(L, \vee)$ and $(L, \wedge)$ are called *join semilattice* and *meet semilattice* if the operations satisfy applicative laws for lattices. A lattice $\mathfrak{L} = (L, \vee, \wedge)$ is *distributive* if for all $a, b, c \in L$, $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$. For any lattice $\mathfrak{L} = (L, \vee, \wedge)$ it is possible to define a binary relation $\leq$ called (*partial*) *order* by assigning for all $a, b \in L$, $a \vee b = b \iff a \leq b$. An element $u \in L$ is called an *upper bound of* $S \subseteq L$ if for all $a \in S$, $a \leq u$. An element $b \in L$ is called a least upper bound or *supremum of* $S \subseteq L$ (if it exists), denoted $\bigvee S$, if b is an upper bound of S and for each upper bound u of S we have $b \leq u$. It is well-known that $\bigvee\{a, b\} = a \vee b$ for all $a, b \in L$. Dually, *infimum of* $S \subseteq L$, denoted $\bigwedge S$, is given as the greatest lower bound of S (if it exists). A lattice $\mathfrak{L} = (L, \vee, \wedge)$ is *complete* if for all $S \subseteq L$, $\bigvee S \in L$ and $\bigwedge S \in L$. In a complete lattice $\mathfrak{L} = (L, \vee, \wedge)$ we have special elements, $\top = \bigvee L$ which is *the greatest element*, and $\bot = \bigwedge L$ which is *the smallest element* (if such an element exists). For a *complete join semi-lattice* $(L, \vee)$ we have for all $S \subseteq L$, $\bigvee S \in L$. Let us now enrich $\mathfrak{L}$ by an associative binary operation $\odot$ with the following distributive laws: for all $a \in L$ and $S \subseteq L$, $a \odot (\bigvee S) = \bigvee\{a \odot b | b \in S\}$, and $(\bigvee S) \odot a = \bigvee\{b \odot a | b \in S\}$. Given these enrichments we have now an algebraic structure $\mathfrak{Q} = (Q, \vee, \odot)$ called a *quantale* [15]. It is clear that we have for all $a \in Q$, $a \odot \bot = \bot \odot a = \bot$. A quantale $\mathfrak{Q} = (Q, \vee, \odot)$ is *unital* if there exists an element $1 \in Q$, called *unit* (or *identity*), such that for all $a \in Q$ we have $1 \odot a = a \odot 1 = a$. When such special element actually exists (by definitions), we may prefer to include them explicitly in the notation for the algebra, so that $(Q, \vee, \odot)$ would be explicitly written as $(Q, \vee, \odot, 1)$. Further, $1 \in Q$ is called *left* (*resp. right*) *unit* if for all $a \in Q$, $1 \odot a = a$ (resp. $a \odot 1 = a$). Clearly, if a quantale is not unital but there exists a left (or right) unit then the operation $\odot$ is not commutative. For any unital quantale $\mathfrak{Q}$ we may define a unary operation *, called *asterate*, such that for all $a \in Q$ we have

$$a^* = \bigvee\{a^n | n \in \mathbb{N}\}, \tag{AST}$$

where $a^n = \underbrace{a \odot a \odot \cdots \odot a}_{n \text{ times}}$ for $n \geq 1$ and $a^0 = 1$. At this point we mention that unital quantales were studied already in [16] under the name *standard Kleene algebra*. Because * may be defined on any unital quantale, we give now some interesting properties (see [16]) valid in standard Kleene algebras: Let $\mathfrak{K} = (K, \vee, \odot, ^*)$ be a standard Kleene algebra. For all $a, b \in K$,

$$a \leq a^*, \quad (a^*)^* = a^*.$$

and from $a \leq b$ it follows $a^* \leq b^*$. Because $\mathfrak{K}$ is also a complete lattice, we write $\mathfrak{K} = (K, \vee, \wedge, \odot, ^*)$, and we have

$$(a \wedge b)^* \leq a^* \wedge b^*, \quad a^* \vee b^* \leq (a \vee b)^*.$$

A unital quantale is called *integral* if the unit coincides with the top element. In this case the Kleene asterate is trivial as it always reaches the top element.

As mentioned earlier, if we give another collection of equational laws (than for lattices) to connect operations then we have another kind of algebra. Quite often algebras are defined by starting with one operation and then enriching the structure (see [12]). It is also important to notice that standard Kleene algebras could have been defined without the concept of order. Indeed, instead of defining the asterate operation by (AST) we might have started from an elementary algebraic structure called semiring and given certain laws for asterate, and thereby creating Kleene algebras and further standard Kleene algebras (see [16]). We also notice that the concept of Kleene algebra in this paper differs from the concept of Kleene algebra (or lattice) as De Morgan quantales [17] or algebras.

Finally, we give some historical notes: Kleene algebras as algebras of regular expressions in the context of automata can be found already in [18]. Further studies on algebras of regular expressions may be found, for example, in [19,20] and also in [16,21]. The concept of quantale was given in [15], and non-commutative quantales with quantale sets are studied in [17]. As an example of unital quantales (or standard Kleene algebras), it is well known that binary relations with reflexive–transitive closure (as the asterate) form a standard Kleene algebra. The study of relational algebras may be traced back to [22] and relation algebras with transitive closure can be found in [23].

4. Categories, functors and monads

In this section we aim not so much to provide a prerequisite for categorical notions as we aim at presenting the categorical notations adopted in this paper. Notation is important for the purpose of keeping concepts separated from each other, and also makes the reader clearly understand "what belongs to where".

4.1. Basic concepts and notations

In a category $\mathtt{C}$ with objects A and B, morphisms f from A to B are typically denoted by $f : A \longrightarrow B$ or $A \xrightarrow{f} B$. The (A-)identity morphism is denoted $A \xrightarrow{\mathrm{id}_A} A$ and morphism composition uses $\circ$. The set of $\mathtt{C}$-morphisms from A to B is written as $\mathrm{Hom}_{\mathtt{C}}(A, B)$ or $\mathrm{Hom}(A, B)$.

The category of sets, $\mathtt{Set}$, is the most typical example of a category, and consists of sets as objects and functions (in ZFC) as morphisms together with the ordinary composition and identity. Other categories may be defined, for example, using $\mathtt{Set}$ as a basis: a structure, defined by the given metalanguage, is added on $\mathtt{Set}$-objects, and then morphisms are defined as $\mathtt{Set}$-morphisms preserving these structures. A typical example is to add uncertainty, modelled by a quantale $\mathfrak{Q}$, on $\mathtt{Set}$-objects: The objects of the Goguen category $\mathtt{Set}(\mathfrak{Q})$ are pairs (X, α), where X is an object of $\mathtt{Set}$ and α: $X \longrightarrow Q$ is a function (in ZFC). The morphisms $(X, \alpha) \xrightarrow{f} (Y, \beta)$ are $\mathtt{Set}$-morphisms $X \xrightarrow{f} Y$ satisfying $\alpha \leq \beta \circ f$. The composition of morphisms is defined as composition of $\mathtt{Set}$-morphisms. Originally, Goguen considered a completely distributive lattice as the underlying lattice in [9] and further properties for Goguen categories can be found in [24].

A (covariant) *functor* $\mathsf{F} : \mathtt{C} \longrightarrow \mathtt{D}$ between categories is a mapping that assigns each $\mathtt{C}$-object A to a $\mathtt{D}$-object $\mathsf{F}(A)$ and each $\mathtt{C}$-morphism $A \xrightarrow{f} B$ to a $\mathtt{D}$-morphism $\mathsf{F}(A) \xrightarrow{\mathsf{F}(f)} \mathsf{F}(B)$, such that $\mathsf{F}(f \circ g) = \mathsf{F}(f) \circ \mathsf{F}(g)$ and $\mathsf{F}(\mathrm{id}_A) = \mathrm{id}_{\mathsf{F}(A)}$. Composition of functors is denoted $\mathsf{G} \circ \mathsf{F} : \mathtt{C} \longrightarrow \mathtt{E}$ and the identity functor is written as $\mathrm{id}_{\mathtt{C}} : \mathtt{C} \longrightarrow \mathtt{C}$. The (covariant) powerset functor $\mathsf{P} : \mathtt{Set} \longrightarrow \mathtt{Set}$ is the typical example of a functor, and is defined by $\mathsf{P}A$ being the powerset of A, i.e., the set of subsets of A, and $\mathsf{P}f(X)$, for $X \subseteq A$, being the image of X under f, i.e., $\mathsf{P}f(X) = \{f(x) | x \in X\}$. A contravariant functor $\mathsf{F} : \mathtt{C} \longrightarrow \mathtt{D}$ maps to each $\mathtt{C}$-morphism $A \xrightarrow{f} B$ and $\mathtt{D}$-morphism $\mathsf{F}(B) \xrightarrow{\mathsf{F}(f)} \mathsf{F}(A)$, and for the contravariant powerset functor $\overline{\mathsf{P}} : \mathtt{Set} \longrightarrow \mathtt{Set}$ we have $\overline{\mathsf{P}}A = \mathsf{P}A$ and $\overline{\mathsf{P}}f(Y) = \{x \in X | \exists y \in Y : f(x) = y\}$.

A *natural transformation* $\tau : \mathsf{F} \longrightarrow \mathsf{G}$ between functors assigns to each $\mathtt{C}$-object A and $\mathtt{D}$-morphism $\tau_A : \mathsf{F}A \longrightarrow \mathsf{G}A$ such that $\mathsf{G}f \circ \tau_A = \tau_B \circ \mathsf{F}f$, for any $f : A \longrightarrow B$. The identity natural transformation $\mathsf{F} \xrightarrow{\mathrm{id}_{\mathsf{F}}} \mathsf{F}$ is defined by $(\mathrm{id}_{\mathsf{F}})_A = \mathrm{id}_{\mathsf{F}A}$. If all τ_A are isomorphisms, τ is called a *natural isomorphism*, or *natural equivalence*. For functors F and natural transformations τ we often write $\mathsf{F}\tau$ and $\tau\mathsf{F}$ to mean $(\mathsf{F}\tau)_A = \mathsf{F}\tau_A$ and $(\tau\mathsf{F})_A = \tau_{\mathsf{F}A}$, respectively. It is easy to see that $\eta : \mathrm{id}_{\mathtt{Set}} \longrightarrow \mathsf{P}$ given by $\eta_X(x) = \{x\}$, and $\mu : \mathsf{P} \circ \mathsf{P} \longrightarrow \mathsf{P}$ given by $\mu_X(\mathcal{B}) = \bigcup \mathcal{B} (= \bigcup_{B \in \mathcal{B}} B)$ are natural transformations. The (vertical) composition $\sigma \circ \tau : \mathsf{F} \longrightarrow \mathsf{H}$ of natural transformations is defined by $(\sigma \circ \tau)_A = \sigma_A \circ \tau_A$, for all $\mathtt{D}$-objects A.

Whereas morphisms are typically seen as 'mappings' between objects in a category, functors are 'mappings' between categories, i.e., morphisms in (quasi-)categories of categories, and natural transformations are 'mappings' between functors, i.e., morphisms in functor categories. These notions clearly lead to views on hierarchies of sets, classes and conglomerates, where foundational issues enter the scene, and our approach roughly follows Grothendieck's [25] and Gähler's [26] views of set-theoretic foundations for category theory.

A *monad* (or triple, or algebraic theory) over a category $\mathtt{C}$ is written as $\mathbf{F} = (\mathsf{F}, \eta, \mu)$, where $\mathsf{F} : \mathtt{C} \longrightarrow \mathtt{C}$ is a (covariant) functor, and $\eta : \mathrm{id} \longrightarrow \mathsf{F}$ and $\mu : \mathsf{F} \circ \mathsf{F} \longrightarrow \mathsf{F}$ are natural transformations for which $\mu \circ \mathsf{F}\mu = \mu \circ \mu\mathsf{F}$ and $\mu \circ \mathsf{F}\eta = \mu \circ \eta\mathsf{F} = \mathrm{id}_{\mathsf{F}}$ hold. A Kleisli category $\mathtt{C}_{\mathbf{F}}$ for a monad $\mathbf{F}$ over a category $\mathtt{C}$ is defined as follows: Objects in $\mathtt{C}_{\mathbf{F}}$ are the same as in $\mathtt{C}$, and the morphisms are defined as $\mathrm{Hom}_{\mathtt{C}_{\mathbf{F}}}(X, Y) = \mathrm{Hom}_{\mathtt{C}}(X, \mathsf{F}Y)$, that is morphisms $f: X \longrightarrow Y$ in $\mathtt{C}_{\mathbf{F}}$ are simply morphisms $f: X \longrightarrow \mathsf{F}Y$ in $\mathtt{C}$, with $\eta_X: X \longrightarrow \mathsf{F}X$ being the identity morphism on X. Composition of morphisms is defined as

$$(X \xrightarrow{f} Y) \diamond (Y \xrightarrow{g} Z) = X \xrightarrow{\mu_Z \circ \mathsf{F}g \circ f} \mathsf{F}Z.$$

The category $\mathtt{Rel}$ with sets as objects and binary relations as morphisms is isomorphic with the Kleisli category of the powerset monad over $\mathtt{Set}$. This invites to viewing Kleisli morphisms as a general notion for relations in the sense of intuitively being "substitutions".

Monads were formally recognized and represented in [27,28]. Monads, substitutions as morphisms in Kleisli categories and F-algebras with further specifications for Eilenberg–Moore algebras for monads appear in [29–31].

Powerset monads and their many-valued extensions are in close connection to fuzzification and are good candidates to represent situations with incomplete or imprecise information. The many-valued covariant powerset functor L for a completely distributive lattice $\mathfrak{L} = (L, \vee, \wedge)$ is obtained by $\mathsf{L}X = L^X$, i.e., the set of functions (or $\mathfrak{L}$-sets) $\alpha : X \longrightarrow L$, and following [9], for a morphism $f: X \longrightarrow Y$ in $\mathtt{Set}$, by defining $\mathsf{L}f(\alpha)(y) = \bigvee_{f(x)=y} \alpha(x)$. Further, if we define $\eta_X : X \longrightarrow \mathsf{L}X$ by

$$\eta_X(x)(x') = \begin{cases} \top & \text{if } x = x' \\ \bot & \text{otherwise} \end{cases}$$

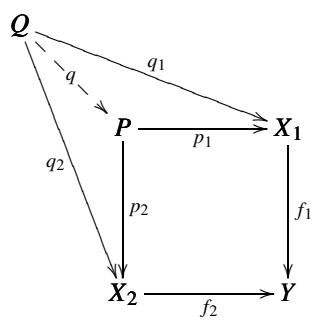
and $\mu : \mathsf{L} \circ \mathsf{L} \longrightarrow \mathsf{L}$ by

$$\mu_X(\mathcal{M})(x) = \bigvee_{\alpha \in \mathsf{L}X} A(x) \wedge \mathcal{M}(\alpha)$$

then $\mathbf{L} = (\mathsf{L}, \eta, \mu)$ is a monad.

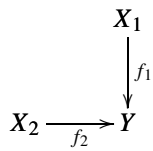
As a further categorical tool we define, for categories $\mathtt{C}$ and $\mathtt{D}$ and a given $\mathtt{D}$-object $A \in \mathrm{Ob}(\mathtt{D})$, the *constant object functor* $A_{\mathtt{C}} : \mathtt{C} \longrightarrow \mathtt{D}$ that assigns all $\mathtt{C}$-objects to A, and morphisms in $\mathtt{C}$ to the identity morphism id_A.

The universal constructions like limits, e.g., products, pullbacks, equalizers and inverse limits, and their duals, colimits, e.g., coproducts, pushouts, coequalizers and direct limits, are important instruments in category theory, and specialize in respective categories, whenever they exist. For illumination, the pullback (also called fibre product) diagram



says that, given the morphisms $f_1 : X_1 \longrightarrow Y$ and $f_2 : X_2 \longrightarrow Y$, P, with the morphisms $p_1 : P \longrightarrow X_1$ and $p_2 : P \longrightarrow X_2$, is pullback, with respect to f_1 and f_2, if, whenever $f_1 \circ q_1 = f_2 \circ q_2$, there exists a unique $q : Q \longrightarrow P$, such that the

pullback diagram commutes. This is the same as saying that the pullback is the limit to the diagram



which e.g., in [32] is seen as a candidate for a general notion of a co-relation in a category. Such relations are closer to relational algebraic views as appearing e.g., in database theory for queries. Note that this is intuitively an entirely different notion of relations as compared to the view of Kleisli morphisms being general relations.

In the case of $\mathtt{Set}$, the set $P = \{(x_1, x_2) \in X_1 \times X_2 | f_1(x_1) = f_2(x_2)\}$, with $p_1(x_1, x_2) = x_1$ and $p_2(x_1, x_2) = x_2$, is a pullback. The category $\mathtt{Set}(\mathfrak{Q})$, where $\mathfrak{Q} = (Q, \vee, \odot)$, also has pullbacks since $\mathfrak{Q}$ is in fact a complete lattice. When replacing in the pullback diagram $\mathtt{Set}$-objects by $\mathtt{Set}(\mathfrak{Q})$-objects it is enough to see that the morphisms are also $\mathtt{Set}(\mathfrak{Q})$-morphisms. Clearly, this is found by replacing P, X_1, X_2 by $(P, \beta), (X_1, \alpha_1), (X_2, \alpha_2)$, respectively, such that $\beta(x_1, x_2) = \alpha_1(x_1) \wedge \alpha_2(x_2), (x_1, x_2) \in P$. Especially, if we replace Y by $(Y,\ 1)$, X_1 by $(X_1,\ 1)$, and X_2 by $(X_2,\ 1)$ then we have for all $(x_1, x_2) \in P$, $\beta(x_1, x_2) = \alpha_1(x_1) = \alpha_2(x_2)$.

4.2. Sorted categories

In the one-sorted (and crisp) case for signatures we typically work in $\mathtt{Set}$, but in the many-sorted (and crisp) case we need the "sorted category of sets" for the many-sorted term functor. We start this section by a more general view by considering "a sorted category of objects".

Let S be an index set (in ZFC), the indices are called *sorts* (or types), and we do not assume any order on S. For a category $\mathtt{C}$, we write $\mathtt{C}_S$ for the product category $\prod_S \mathtt{C}$. The objects of $\mathtt{C}_S$ are tuples $(X_\mathtt{s})_{\mathtt{s}\in S}$ such that $X_\mathtt{s} \in \mathrm{Ob}(\mathtt{C})$ for all $\mathtt{s} \in S$. We will also use X_S as a shorthand notation for these tuples. The morphisms between objects $(X_\mathtt{s})_{\mathtt{s}\in S}$ and $(Y_\mathtt{s})_{\mathtt{s}\in S}$ are tuples $(f_\mathtt{s})_{\mathtt{s}\in S}$ such that $f_\mathtt{s} \in \mathrm{Hom}_\mathtt{C}(X_\mathtt{s}, Y_\mathtt{s})$ for all $\mathtt{s} \in S$, and similarly we will use f_S as a shorthand notation. The composition of morphisms is defined sortwise (componentwise), i.e., $(g_\mathtt{s})_{\mathtt{s}\in S} \circ (f_\mathtt{s})_{\mathtt{s}\in S} = (g_\mathtt{s} \circ f_\mathtt{s})_{\mathtt{s}\in S}$.

Functors $\mathsf{F}_\mathtt{s}\colon \mathtt{C} \longrightarrow \mathtt{D}$ are lifted to functors $\mathsf{F} = (\mathsf{F}_\mathtt{s})_{\mathtt{s}\in S}$ from $\mathtt{C}_S$ to $\mathtt{D}_S$, so that e.g., the regular powerset functor $\mathsf{P}_S = (\mathsf{P})_{\mathtt{s}\in S}$ and the regular many-valued powerset functor $\mathsf{L}_S = (\mathsf{L})_{\mathtt{s}\in S}$, both are lifted to functors on $\mathtt{Set}_S$.

Products and coproducts, $\prod$ and $\coprod$, are handled sortwise. We also have a "subobject relation", thus, $(X_\mathtt{s})_{\mathtt{s}\in S} \subseteq (Y_\mathtt{s})_{\mathtt{s}\in S}$ if and only if $X_\mathtt{s} \subseteq Y_\mathtt{s}$ for all $\mathtt{s} \in S$. It is clear that all limits and colimits exist in $\mathtt{Set}_S$, because operations on $\mathtt{Set}_S$-objects are defined sortwise for sets. Further, the product $\prod_{i\in I} \mathsf{F}_i$ and coproduct $\coprod_{i\in I} \mathsf{F}_i$ of covariant functors F_i over $\mathtt{Set}_S$ are defined as

$$\left(\prod_{i\in I} \mathsf{F}_i\right)(X_\mathtt{s})_{\mathtt{s}\in S} = \prod_{i\in I} \mathsf{F}_i(X_\mathtt{s})_{\mathtt{s}\in S}$$

and

$$\left(\coprod_{i\in I} \mathsf{F}_i\right)(X_\mathtt{s})_{\mathtt{s}\in S} = \coprod_{i\in I} \mathsf{F}_i(X_\mathtt{s})_{\mathtt{s}\in S}$$

with morphisms being handled accordingly.

The category $\mathtt{Set}(\mathfrak{Q})_S$ is called the *many-sorted Goguen category*. Objects in this category are tuples of pairs $((X_\mathtt{s}, \alpha_\mathtt{s}))_{\mathtt{s}\in S}$ as objects, where for each $\mathtt{s} \in S$, $\alpha_\mathtt{s}\colon X_\mathtt{s} \longrightarrow Q$ is a function (in ZFC). So, fixing $\mathtt{s} \in S$ we consider pairs $(X_\mathtt{s}, \alpha_\mathtt{s})$ as objects in $\mathtt{Set}(\mathfrak{Q})$. Now, the $\mathtt{Set}(\mathfrak{Q})$-morphisms $(X_\mathtt{s}, \alpha_\mathtt{s}) \xrightarrow{f_\mathtt{s}} (Y_\mathtt{s}, \beta_\mathtt{s})$ form morphisms $((X_\mathtt{s}, \alpha_\mathtt{s}))_{\mathtt{s}\in S} \xrightarrow{(f_\mathtt{s})_{\mathtt{s}\in S}} ((Y_\mathtt{s}, \beta_\mathtt{s}))_{\mathtt{s}\in S}$. It is known that all limits and colimits exist in $\mathtt{Set}(\mathfrak{Q})$. For example, we have the inductive limit, i.e., the direct colimit, of the inductive system $(((X_\mathtt{s}, \alpha_\mathtt{s}))^{(i)}_{\mathtt{s}\in S})_{i\in\mathbb{N}} = (((X^{(i)}_\mathtt{s}, \alpha^{(i)}_\mathtt{s}))_{\mathtt{s}\in S})_{i\in\mathbb{N}}$, written as $\mathrm{ind}\lim_{\rightarrow}((X^{(i)}_\mathtt{s}, \alpha^{(i)}_\mathtt{s}))_{\mathtt{s}\in S}$ being an object $((Y_\mathtt{s}, \beta_\mathtt{s}))_{\mathtt{s}\in S}$ in $\mathtt{Set}(\mathfrak{Q})_S$. Here, $Y_\mathtt{s}$ is the set $\bigcup_{i\in\mathbb{N}} X^{(i)}_\mathtt{s}$, and $\beta_\mathtt{s}$ is defined as follows. For $x \in Y_\mathtt{s}$, we have $A_x = \{i \in \mathbb{N} | x \in X^{(i)}_\mathtt{s}\}$, and we can define $\beta_\mathtt{s}(x) = \bigvee\{\alpha^{(i)}_\mathtt{s}(x) | i \in A_x\}$.

It may be sometimes reasonable to handle a quantale $\mathfrak{Q}$ as a complete lattice, since $\wedge$ may be needed. Moreover, a standard Kleene algebra $\mathfrak{K} = (K, \vee, \wedge, \odot, ^*)$ is also of interest since application of the asterate in *term construction* is possible. In further sections we study, for example, term monads over sorted Goguen categories $\mathtt{Set}(\mathfrak{K})_S$.

4.3. Monoidal biclosed categories

Ever since the beginning of category theory, the notion of a product, and its relation to corresponding exponential objects, has been of great importance. Furthermore, the Hom-set $\mathrm{Hom}(X, Y)$ in a category, with X and Y as objects, is indeed an object itself of that category, and therefore a candidate for being an *exponential object*.

The underlying fundamental situation is related to the natural equivalence

$$\mathrm{Hom}(A \times B, C) \cong \mathrm{Hom}(A, \mathrm{Hom}(B, C))$$

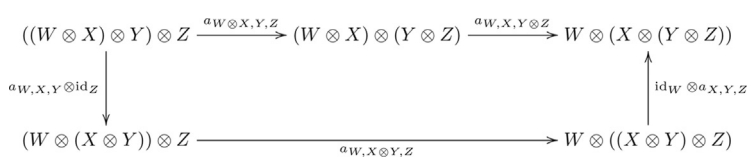
In $\mathtt{Set}$, $\mathrm{Hom}(B, C)$ is indeed the exponential objects required, and the natural equivalence means that $\mathtt{Set}$ is *Cartesian closed*. The Cartesian product is, however, quite strong, and therefore, given a category $\mathtt{C}$, the notion of product can be weakened using bifunctors $\otimes : \mathtt{C} \times \mathtt{C} \longrightarrow \mathtt{C}$, where it is, similar to Cartesian products, a custom to write $A \otimes B$ instead of $\otimes(A, B)$), for objects A and B in $\mathrm{Ob}(\mathtt{C})$. This branch of category theory is usually called *categorical algebra*. For this bifunctor we then come to the issue of so-called *monoidal closedness* and *monoidal closed categories*. Note here also how the natural equivalence is a situation of *currying*, as known in type theory. There is indeed the intuition that the monoidal product acts like a composition, i.e., strengthening the currying view of this natural equivalence. Further, the natural equivalence also invites to thinking about morphisms carrying uncertainty, and this will open up considerations even for sorts being uncertain, as we more clearly distinguish the name of the sort from how it appears as a morphism. As morphisms appear in fibres, this in turn invites to intuitively thinking about uncertainty in fibres as well.

Adding further structure to $\mathrm{Hom}(X, Y)$ is sometimes desirable, e.g., by having operations on morphisms satisfying certain properties, bringing us e.g., to 2-categories, and from there onwards. Morphisms carrying uncertainties would also mean adding structure to the Hom-set. In specific situations, the Hom-set may indeed already carry some algebraic or topological structure, and this is then explored further, as required by the context. The name *enriched category theory* indeed comes from this enrichment of Hom-sets, or replacing them with "monoidal objects" constructed using the bifunctor.

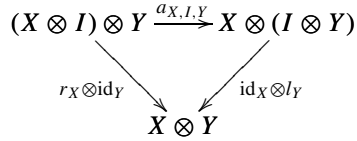
For readers not so well-versed in category theory, all these notions may appear as confusing, and basically we do not have to answer the question: *Is a monoidal category a category or an algebra*?, but we may answer it by saying *It's neither*, *and it's both*!

The history of monoidal closed categories, and *categorical algebra* more broadly, goes back to the study of such natural equivalences [33], in turn inspired e.g., by theories of linear operators [34] and homology theory. Incidently, from fuzzy arithmetic point of view it is very much unknown how natural equivalences relate to Birkhoff's generalized arithmetic [35], on bridging the gap between cardinal and ordinal arithmetic using a partially ordered set view of numbers, as the construction of "large numbers of functors" including natural equivalences and transformations between these functors [33]. Thereafter, the notion of *adjoint functors* [36] came to play an important role for the bifunctor, and for monoidal closed categories to be more formally defined [37] and analyzed with respect to the *coherence condition* [38]. At the beginning, monoidal categories were called *categories with multiplication* in [39–41]. The name *monoidal closed category* emerges more or less in [42], and attains its simple and clean formulation in [43].

In order to make this paper more self-contained, we include here the definition for monoidal closed category, following the notational style as appearing in [44]. For this purpose, let $\mathtt{C}$ be a category, $\otimes : \mathtt{C} \times \mathtt{C} \longrightarrow \mathtt{C}$ a bifunctor, and I a *unit object* in $\mathtt{C}$. If there are natural isomorphisms $a_{X,Y,Z} : (X \otimes Y) \otimes Z \longrightarrow X \otimes (Y \otimes Z)$, $l_X : I \otimes X \longrightarrow X$ and $r_X : X \otimes I \longrightarrow X$ making the diagrams



and



commute, we say that $\mathtt{C}$, equipped with the bifunctor, a unit object, and these natural isomorphisms are a monoidal category.

For a terminal object I we always have $r_I = l_I$. Additionally, a monoidal category becomes a *monoidal* (*left*) *closed category*, if the functor $\_ \otimes B : \mathtt{C} \longrightarrow \mathtt{C}$ has a right adjoint, denoted $[B, \_]$, for all objects B. It is right closed, if $A \otimes \_ : \mathtt{C} \longrightarrow \mathtt{C}$, for all objects A, has a right adjoint. A monoidal closed category is *biclosed*, if it is both left and right closed, and a *symmetric monoidal category*, whenever the tensor product is commutative. Note that, in this context, $\mathrm{Hom}(I, \_)$ plays the role of the *basic functor* [42] up to natural isomorphism, because of

$$\mathrm{Hom}(A, B) \cong \mathrm{Hom}(I, [A, B]).$$

A symmetric monoidal category is not necessarily monoidal closed. If $\mathtt{C}$ is monoidal biclosed, then the tensor product preserves colimits in both variables separately.

The tensor product for vector spaces is monoidal closed, but not Cartesian closed. Given that the underlying quantale is commutative and unital, the Goguen category $\mathtt{Set}(\mathfrak{Q})$ is, by Theorem 5.3.2 in [45], a symmetric monoidal closed category and it is therefore also biclosed. Note also that any topos is a symmetric monoidal closed category, but $\mathtt{Set}(\mathfrak{Q})$ is not a topos.

Generally speaking it is important to say that uncertainty and fuzziness is enrichness to, not generalization of, the crisp world. In this context the basic functor V removes the enrichment from hom-objects and plays therefore the important role of a defuzzifer.

5. Signatures

Traditional universal algebra recognizes signatures as consisting of sorts and operators. Computer scientists frequently speak of types instead of sorts. Mathematicians, on the other hand, have been influenced, e.g., by [46] where the sequence of the argument types is seen as an element in the free monoid over the set of sorts.

Further, the shift from one-sorted signatures to many-sorted signatures is usually seen as a mere technicality, but very few actually bother to check detail. It is, however, fairly well-known that the shift to many-sortedness is not always that evident, and in our case when dealing with term functor constructions over various underlying categories, the many-sorted cases are far from trivial, as we will see.

In the computer science notation, a many-sorted signature $\Sigma = (S, \Omega)$ consists of a set S of sorts (or types), and a set Ω of operators. The question as to which scope and what extend a signature is a categorical object is non-trivial, and the categorization can be arranged in different ways, as seen below. The computer science intuitive view is that S as an index set, and as residing in ZFC, whereas Ω may be an object in $\mathtt{Set}_S$. Thus, $\Omega = (\Omega_{\mathtt{s}})_{\mathtt{s} \in S}$, where $\Omega_{\mathtt{s}}$ is an object in $\mathtt{Set}$, and we would say informally that $\Sigma = (S, \Omega)$ is a *signature over* $\mathtt{Set}$. Operators in $\Omega_{\mathtt{s}}$ are then syntactically written as $\omega : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$, and we say that the arity of $\omega : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ is n, or that it is an n-ary operator. The 0-ary operators $\omega :\to \mathtt{s}$ are called constant operators, or constants. For simplicity, when we will emphasize the sorts or arities in syntactic expressions we write $\omega : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$, and in the case of sorts and arities are known we may write only ω for the operators also in syntactic expressions. Note that each set $\Omega_{\mathtt{s}}$ does not necessarily make preferences for arities, but the "end sort" of the operation, or the sort of "the result of the operation", is always $\mathtt{s}$. Notice also that $\times$ and $\to$ in the syntactic notation for operators at this point come without any meaning, but algebras will eventually provide $\times$ and $\to$ with meaning in the underlying category.

In the many-sorted term construction it will be convenient to use the notation $\Omega^{\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}}$ for the set, as an object in $\mathtt{Set}$, of operators $\omega : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s} \in \Omega_{\mathtt{s}}$ with n given, and $\Omega^{\to \mathtt{s}}$ for the set of constants $\omega :\to \mathtt{s}$. With these

notations we keep explicit track of operator sorts as well as their arities and we consider

$$\Omega_{\mathtt{s}} = \coprod_{\substack{\mathtt{s}_1,\ldots,\mathtt{s}_n \\ n\leq k}} \Omega^{\mathtt{s}_1\times\cdots\times\mathtt{s}_n\to\mathtt{s}}.$$

For example, if $S = \{\mathtt{s}, \mathtt{t}\}$ then a set $\Omega^{\mathtt{s}\times\mathtt{t}\to\mathtt{s}}$ may differ from $\Omega^{\mathtt{t}\times\mathtt{s}\to\mathtt{s}}$. Clearly, if S is one-pointed, then Σ is a one-sorted signature.

If $\Sigma_1 = (S_1, \Omega_1)$ and $\Sigma_2 = (S_2, \Omega_2)$ are signatures, then a signature mapping $\varsigma : \Sigma_1 \longrightarrow \Sigma_2$ is a pair of mappings $(\mathfrak{s} : S_1 \longrightarrow S_2, \mathfrak{o} : \Omega_1 \longrightarrow \Omega_2)$ such that for each $\omega_1 \in \Omega_1^{\mathtt{s}_1\times\cdots\times\mathtt{s}_n\to\mathtt{s}}$, there exists a $\omega_2 \in \Omega_2^{\mathfrak{s}(\mathtt{s}_1)\times\cdots\times\mathfrak{s}(\mathtt{s}_n)\to\mathfrak{s}(\mathtt{s})}$ such that $\mathfrak{o}(\omega_1) = \omega_2$.

For signature mappings $\varsigma = (\mathfrak{s}, \mathfrak{o}) : \Sigma_1 \longrightarrow \Sigma_2$ and $\varrho = (\mathfrak{r}, \mathfrak{n}) : \Sigma_2 \longrightarrow \Sigma_3$, the composed mapping $\varrho \circ \varsigma : \Sigma_1 \longrightarrow \Sigma_3$ is defined by $(\mathfrak{r} \circ \mathfrak{s}, \mathfrak{n} \circ \mathfrak{o})$. Composition of signature mappings is associative, and we may then anticipate the formal definition of a category of signatures.

Following a strictly mathematical view, in [46] signatures are called *S-schéma* over set of sorts, and consists of a triples (Ω, τ, S) where Ω is seen only as a set of operator symbols, $\tau : \Omega \longrightarrow \hat{S} \times S$ attaches sorts to operator symbols, where the set of sorts S is extended to the free monoid $\mathfrak{S} = (\hat{S}, \cdot, e)$ so that elements in $\hat{S}$ reflect sequences of types. Using the computer science notation, an operator $\omega \in \Omega^{\mathtt{s}_1\times\cdots\times\mathtt{s}_n\to\mathtt{s}}$ would have $\tau(\omega) = (\mathtt{s}_1 \cdots \mathtt{s}_n, \mathtt{s})$. Note also how the monoidal extension, $\hat{\mathfrak{s}}$, of the signature mapping $\mathfrak{s}$ will have the monoid homomorphism property.

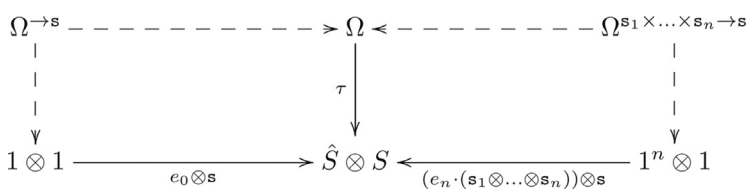
From computer science point of view we indeed view S simply as a set of sorts, i.e., mostly as an index set, so that S is not even seen as an object in the category $\mathtt{Set}$. However, in the monoidal view of sequences of sorts, S is an object of $\mathtt{Set}$ and $\mathfrak{S}$ an object on $\mathtt{Mon}$, the category of monoids, so that we can formally speak of *signatures over* $\mathtt{Set}$.

The category $\mathtt{Set}$ is monoidal biclosed with respect to the Cartesian product, and signatures can more generally be arranged over any monoidal biclosed category $\mathtt{C}$ with tensor product $\bigotimes$. In fact, let S be an object of $\mathtt{C}$, i.e., S represents the "sort set" in a generalized sense. If we by S^0 denote the unit object 1 in $\mathtt{C}$, we can write $S^{n+1} = S \bigotimes S^n$, $n \in \mathbb{N}$, and define $\hat{S} = \coprod_{n\in\mathbb{N}} S^n$. Sorts $\mathtt{s}$ can now be "recovered" in S as arrows $1 \xrightarrow{\mathtt{s}} S$, and *one-sortedness* would mean that S is a unit object. In $\mathtt{C}$, we need to require that there is only one arrow $1 \longrightarrow 1$.

With $\mathtt{Set}$ as the underlying category for the monoidal biclosed category with the Cartesian product as the tensor product, the unit object in the monoidal biclosed category is a terminal object in $\mathtt{Set}$. Note that every one-pointed set is a terminal object in $\mathtt{Set}$, so the unit object is a selected terminal object, e.g., represented by the one-pointed set $\{\emptyset\}$. Note also that the arrow $1 \xrightarrow{\mathtt{s}} S$ can be identified with an element in S, and vice versa, any element in S generates such an arrow. This means that there is a bijection between S and $\mathrm{Hom}_{\mathtt{Set}}(1, S)$. Now note that in the more general cases, the sets S and $\mathrm{Hom}_{\mathtt{C}}(1, S)$ are not necessarily one-to-one, and then $\mathrm{Hom}_{\mathtt{Set}}(1, S)$ represents the sorts, whereas S acts only as a "base" for that construction of that set of sorts. For these situations it is necessary to introduce the notation $\mathtt{S}$ for the set $\mathrm{Hom}_{\mathtt{Set}}(1, S)$ of sorts, so that we can write $\mathtt{s} \in \mathtt{S}$ which then intuitively means $\mathtt{s}$ *is a sort in* S in the traditional sense on computer science oriented reading and informal notation.

If $\mathfrak{Q}$ is a commutative unital quantale, then $\mathtt{Set}(\mathfrak{Q})$ is a monoidal biclosed category, where again there are many terminal objects, but the unit object need not necessarily be selected among the terminal objects. In these respects, we may select as unit object $1 = (\{\emptyset\}, \mathfrak{1})$, where $\mathfrak{1}$ denotes the constant mapping for which $\mathfrak{1}(\emptyset) = \mathfrak{1}$. This unit object is not a terminal object, but there is a unique arrow $1 \longrightarrow 1$. In this case again we have that there is a one-to-one correspondence between S and $\mathrm{Hom}_{\mathtt{Set}(\mathfrak{Q})}(1, (S, \top))$. Note that it is intuitively desirable indeed to have $(S, \top)$ corresponding to "crisp sorts", but technically we may view S as being one-to-one with $\mathrm{Hom}_{\mathtt{Set}(\mathfrak{Q})}(1, (S, \mathfrak{1}))$. In the case of integral quantales it makes no difference. The membership functions are multiplied according to the monoid operation in $\mathfrak{Q}$ (see [45, Theorem 5.3.2]). Both monoidal structures are symmetric and therefore biclosedness is equivalent to closedness.

Given the sorts, we can now construct the "operator sets" using the following pullback squares:



where e_n are the canonical arrows of the coproduct, with e_0 being the unit in $\hat{S}$. We then say that (Ω, τ, S) is a *signature over* $\mathtt{C}$, or a $\mathtt{C}$-*signature*.

This intuitively also relates to the situation where the inverse image of a set with respect to a function can be viewed as a pullback for the corresponding inclusion map and that the function. Indeed, in this intuition, $\Omega^{\to \mathtt{s}}$ is a kind of the inverse image of $1 \otimes 1$, and similarly $\Omega^{\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}}$ of $1^n \otimes 1$ with respect to τ.

In order to verify that $\Omega^{\to \mathtt{s}}$ and $\Omega^{\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}}$ really correspond to our intended objects of operators, let us look at $\Omega^{\to \mathtt{s}}$ in the situation with $\mathtt{C} = \mathtt{Set}$. By definition of pullback, $\Omega^{\to \mathtt{s}}$ is isomorphic with the set $\{(\omega, (\varnothing, \varnothing)) \in \Omega \times (1 \times 1) | \tau(\omega) = (e_0 \times \mathtt{s})((\varnothing, \varnothing))\}$, where $p_1(\omega, (\varnothing, \varnothing)) = \omega$ and $p_2(\omega, (\varnothing, \varnothing)) = (\varnothing, \varnothing)$. Note that e_0 is the "empty string", and how then the requirement $\tau \circ p_1 = (e_0 \times \mathtt{s}) \circ p_2$ means that $\tau(\omega) = (e_0, \mathtt{s})$, i.e., ω is of the form $\omega :\to \mathtt{s}$, as expected.

For $\mathtt{Set}(\mathfrak{Q})$ the situation concerning the choice between $(S, \top)$ and $(S, 1)$ would be delicate, indeed in the non-integral case, as then choosing one over the other will bring us to two different situations concerning the uncertainties for operators in $\Omega^{\to \mathtt{s}}$ and $\Omega^{\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}}$, but in the integral case this makes no difference.

With $\Sigma_1 = (\Omega_1, \tau_1, S_1)$ and $\Sigma_2 = (\Omega_2, \tau_2, S_2)$ being $\mathtt{C}$-signatures, we can now define a signature morphism $\varsigma : \Sigma_1 \longrightarrow \Sigma_2$ as a pair of morphisms $(\mathfrak{s} : S_1 \longrightarrow S_2, \mathfrak{o} : \Omega_1 \longrightarrow \Omega_2)$ such that

$$\begin{array}{ccc} \Omega_1 & \xrightarrow{\mathfrak{o}} & \Omega_2 \\ {\scriptstyle \tau_1}\downarrow & & \downarrow{\scriptstyle \tau_2} \\ \hat{S}_1 \otimes S_1 & \xrightarrow[\hat{\mathfrak{s}} \otimes \mathfrak{s}]{} & \hat{S}_2 \otimes S_2 \end{array}$$

commutes. Here $\hat{\mathfrak{s}}$ is the monoidal extension of $\mathfrak{s}$. Thus we arrive at a category $\mathtt{Sign}_{\mathtt{C}}$ of signatures over $\mathtt{C}$. The category $\mathtt{Sign}_{\mathtt{Set}}$ corresponds intuitively with informally defined and anticipated category of signatures.

In the following we provide some typical examples of signatures in $\mathtt{Sign}_{\mathtt{Set}}$. We use now the computer science notation rather than the mathematical notation.

Example 5.1. A signature for natural numbers could be given by $\mathtt{NAT} = (\{\mathtt{nat}\}, \{\mathtt{0} :\to \mathtt{nat}, \mathtt{succ} : \mathtt{nat} \to \mathtt{nat}\})$, which indeed is a fundamental example syntactically producing the natural numbers. This signature could be extended e.g., with an operator for addition.

Example 5.2. The very basic signature for Booleans could be $\mathtt{BOOL} = (\{\mathtt{bool}\}, \{\mathtt{false} :\to \mathtt{bool}, \mathtt{true} :\to \mathtt{bool}\})$, and additional operators could be introduced for Boolean operators.

Example 5.3. A signature $\mathtt{NATORD} = (\{\mathtt{nat}, \mathtt{bool}\}, \{\mathtt{0} :\to \mathtt{nat}, \mathtt{succ} : \mathtt{nat} \to \mathtt{nat}, \mathtt{false} :\to \mathtt{bool}, \mathtt{true} :\to \mathtt{bool}, \leq : \mathtt{nat} \times \mathtt{nat} \to \mathtt{bool}\})$ could be introduced as a many-sorted situation where the operator $\leq$ needs both $\mathtt{nat}$ and $\mathtt{bool}$. Again, this signature can be extended e.g., to $\mathtt{NATADDORD}$ by adding the operator $+ : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}$ (reserved for addition, but still semantically or equationally undefined) to the set of operators in $\mathtt{NATORD}$. Similarly, logical operators could be included, and so on.

Example 5.4. The set of terms for the signature $\Sigma = (S, \varnothing)$, $S \neq \varnothing$, consists only of variables.

6. Term functors

In this section we continue to be two-fold with respect to computer science and mathematical notation. Given the fact that the literature in both areas is surprisingly sparse with respect to the formal construction of the inductive nature of the term functor, we start off by providing that formal construction firstly in the many-sorted and the crisp case. We then shift to the more general term functor construction being over underlying categories for uncertainty. One-sorted considerations of the term functor was given in [5], however, without a formal inductive construction of the term functor. The more informal term set construction for one-sorted signatures was originally given in [6], and in [47] it was used to exemplify convergence structures. In [7], the term construction was used as a basis for providing distributive laws in connection with the many-valued powerset monad in the one-sorted case.

6.1. The general term functor construction over monoidal biclosed categories

In the following we provide the general term functor construction that works for any underlying monoidal biclosed category $\mathtt{C}$ with tensor product $\bigotimes$. However, to allow for a more intuitive reading, we present the constructions specifically for the monoidal biclosed category $\mathtt{Set}$, where the tensor product is the Cartesian product.

We therefore start off with having a signature $\Sigma = (S, \tau, \Omega)$ over $\mathtt{Set}$, where $\mathtt{S} = \mathrm{Hom}_{\mathtt{Set}}(1, S)$. One of the cornerstones for the construction will be providing a functor $\mathsf{T}_{\Sigma,\mathtt{s}} : \mathtt{Set}_{\mathtt{S}} \longrightarrow \mathtt{Set}$, appearing in

$$\mathsf{T}_{\Sigma} X_{\mathtt{S}} = (\mathsf{T}_{\Sigma,\mathtt{s}} X_{\mathtt{S}})_{\mathtt{s}\in\mathtt{S}}$$

so that the term functor $\mathsf{T}_{\Sigma} : \mathtt{Set}_{\mathtt{S}} \longrightarrow \mathtt{Set}_{\mathtt{S}}$ can be extended to a monad over $\mathtt{Set}_{\mathtt{S}}$.

As before, we have $\Omega^{\mathtt{s}}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ for the set of operators $\omega : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ in Ω, i.e., n-ary operators where we are explicit about the sorts. The many-sorted term functor $\mathsf{T}_{\Sigma} : \mathtt{Set}_{\mathtt{S}} \longrightarrow \mathtt{Set}_{\mathtt{S}}$ can now be defined by induction. Firstly, for $\mathsf{T}^1_{\Sigma,\mathtt{s}}$, we need to introduce some further notation. For any $\mathtt{m} \in \hat{\mathtt{S}}$ and any $\mathtt{s} \in \mathtt{S}$ we define a functor $\Psi_{\mathtt{m},\mathtt{s}} : \mathtt{Set}_{\mathtt{S}} \longrightarrow \mathtt{Set}$ as follows. The special case $\Psi_{e,\mathtt{s}}$, with e being the unit element in the monoid, is the constant object functor $((\Omega^{\to \mathtt{s}})_{\mathtt{s}\in\mathtt{S}})_{\mathtt{Set}_{\mathtt{S}}}$. Further, for $\mathtt{m} = \mathtt{s}_1 \cdot \ldots \cdot \mathtt{s}_n$, we define

$$\Psi_{\mathtt{m},\mathtt{s}}((X_{\mathtt{t}})_{\mathtt{t}\in\mathtt{S}}) = \Omega^{\mathtt{s}_1\times\cdots\times\mathtt{s}_n\to\mathtt{s}} \times \prod_{i=1,\ldots,n} X_{\mathtt{s}_i},$$

for objects, and

$$\Psi_{\mathtt{m},\mathtt{s}}((f_{\mathtt{s}})_{\mathtt{s}\in\mathtt{S}}) = \mathrm{id}_{\Omega^{\mathtt{s}_1\times\cdots\times\mathtt{s}_n\to\mathtt{s}}} \times \prod_{i=1,\ldots,n} f_{\mathtt{s}_i}$$

for morphisms.

Note indeed how everything above still works with the Cartesian product for $\mathtt{Set}$ generalized to the tensor product for $\mathtt{C}$.

We can now set

$$\mathsf{T}^1_{\Sigma,\mathtt{s}} = \coprod_{\mathtt{m}\in\hat{\mathtt{S}}} \Psi_{\mathtt{m},\mathtt{s}},$$

and, for $\iota > 1$, we can then recursively proceed to define

$$\mathsf{T}^{\iota}_{\Sigma,\mathtt{s}} X_{\mathtt{S}} = \coprod_{\mathtt{m}\in\hat{\mathtt{S}}} \Psi_{\mathtt{m},\mathtt{s}}(\mathsf{T}^{\iota-1}_{\Sigma,\mathtt{t}} X_{\mathtt{S}} \sqcup X_{\mathtt{t}})_{\mathtt{t}\in\mathtt{S}}),$$

and

$$\mathsf{T}^{\iota}_{\Sigma,\mathtt{s}} f_{\mathtt{S}} = \coprod_{\mathtt{m}\in\hat{\mathtt{S}}} \Psi_{\mathtt{m},\mathtt{s}}(\mathsf{T}^{\iota-1}_{\Sigma,\mathtt{t}} f_{\mathtt{S}} \sqcup f_{\mathtt{t}})_{\mathtt{t}\in\mathtt{S}}).$$

This then allows us to define the functors $\mathsf{T}^{\iota}_{\Sigma}$ by

$$\mathsf{T}^{\iota}_{\Sigma} X_{\mathtt{S}} = (\mathsf{T}^{\iota}_{\Sigma,\mathtt{s}} X_{\mathtt{S}})_{\mathtt{s}\in\mathtt{S}},$$

and

$$\mathsf{T}^{\iota}_{\Sigma} f_{\mathtt{S}} = (\mathsf{T}^{\iota}_{\Sigma,\mathtt{s}} f_{\mathtt{S}})_{\mathtt{s}\in\mathtt{S}}.$$

We can also show that there is a natural transformation

$$\Xi^{\iota+1}_{\iota} : \mathsf{T}^{\iota}_{\Sigma} \longrightarrow \mathsf{T}^{\iota+1}_{\Sigma}$$

such that $(\mathsf{T}^{\iota}_{\Sigma})_{\iota>0}$ is an inductive system of endofunctors with $\Xi^{\iota+1}_{\iota}$ as its connecting maps. This natural transformation builds upon the canonical "embeddings" $j_{\mathtt{s}} : X_{\mathtt{s}} \longrightarrow \mathsf{T}^1_{\Sigma,\mathtt{s}} X_{\mathtt{S}} \sqcup X_{\mathtt{s}}$ that define $(\xi^{\iota+1}_{\iota})_{\mathtt{S}}$ according to

$$(\xi^2_1)_{\mathtt{s}} = \coprod_{\mathtt{m}\in\hat{\mathtt{S}}} \Psi_{\mathtt{m},\mathtt{s}}((j)_{\mathtt{S}})$$

and

$$(\xi_\iota^{\iota+1})_{\mathtt{s}} = \coprod_{\mathtt{m}\in\hat{\mathtt{S}}} \Psi_{\mathtt{m},\mathtt{s}}(((\xi_{\iota-1}^{\iota})_{\mathtt{t}} \sqcup \mathsf{id}_{X_{\mathtt{t}}})_{\mathtt{t}\in\mathtt{S}})$$

for $\iota > 2$, and then with $(\Xi_\iota^{\iota+1})_{X_{\mathtt{S}}} = ((\xi_\iota^{\iota+1})_{\mathtt{s}})_{\mathtt{s}\in\mathtt{S}}$.

Further, since $\mathtt{Set}_{\mathtt{S}}$ is cocomplete, the inductive limit $\mathsf{F} = \operatorname{ind}\lim_{\rightarrow} \mathsf{T}_\Sigma^\iota$ (in this case with ι starting from 1, not 0!) exists, and the final term functor T_Σ is then given by

$$\mathsf{T}_\Sigma = \mathsf{F} \sqcup \mathsf{id}_{\mathtt{Set}_{\mathtt{S}}}.$$

In the case of NAT, we have

$$\mathsf{T}^1_{\mathtt{NAT}} X = \{\mathtt{0}\} \cup \{\mathtt{succ}(x) | x \in X\}$$

and

$$\mathsf{T}^2_{\mathtt{NAT}} X = \{\mathtt{0}\} \cup \{\mathtt{succ}(x) | x \in X\} \cup \{\mathtt{succ}(\mathtt{0})\} \cup \{\mathtt{succ}(\mathtt{succ}(x)) | x \in X\}.$$

In the case of NATORD, with $S = \{\mathtt{nat}, \mathtt{bool}\}$, we have

$$\mathsf{T}^1_{\mathtt{NATORD},\mathtt{bool}} X_{\mathtt{S}} = \{\mathtt{false}, \mathtt{true}\} \cup \{\leq (x_1, x_2) | x_1, x_2 \in X_{\mathtt{nat}}\}$$

and

$$\mathsf{T}^1_{\mathtt{NATORD},\mathtt{nat}} X_{\mathtt{S}} = \{\mathtt{0}\} \cup \{\mathtt{succ}(x) | x \in X_{\mathtt{nat}}\},$$

and so on.

Note that e.g., for $\leq (\mathtt{succ}(\mathtt{0}), \mathtt{0})$ we have not provided any sentences or semantics giving preference to it being identified with false or true. This is done at a later stage when sentences enter the scene, and when the axioms representing properties of $\leq$ are introduced.

The construction above also implies that

$$\mathsf{T}_\Sigma X_{\mathtt{S}} = (\mathsf{T}_{\Sigma,\mathtt{s}} X_{\mathtt{S}})_{\mathtt{s}\in\mathtt{S}}$$

and that T_Σ is idempotent, or to be more precise, T_Σ is idempotent up to isomorphism, as there is a natural isomorphism between $\mathsf{T}_\Sigma \mathsf{T}_\Sigma$ and T_Σ.

The extension of the functor T_Σ to a monad over $\mathtt{Set}_S$ is enabled by $(\eta_{\mathtt{s}}^{\mathsf{T}_\Sigma})_{X_{\mathtt{s}}} : X_{\mathtt{s}} \longrightarrow \mathsf{T}_{\Sigma,\mathtt{s}} X_{\mathtt{S}}$, such that for all $x \in X_{\mathtt{s}}$,

$$(\eta_{\mathtt{s}}^{\mathsf{T}_\Sigma})_{X_{\mathtt{s}}}(x) = x,$$

and also $((\mu_{\mathtt{s}}^{\mathsf{T}_\Sigma})_{X_{\mathtt{s}}})_{\mathtt{s}\in\mathtt{S}} : \mathsf{T}_\Sigma \mathsf{T}_\Sigma X_{\mathtt{S}} \longrightarrow (\mathsf{T}_{\Sigma,\mathtt{s}} X_{\mathtt{S}})_{\mathtt{s}\in\mathtt{S}}$ as the identity, and again with 'identity' with respect to the natural isomorphism. Therefore we can say that we have natural transformations $\eta^{\mathsf{T}_\Sigma}: \mathrm{id}_{\mathtt{Set}_S} \longrightarrow \mathsf{T}_\Sigma$ and $\mu^{\mathsf{T}_\Sigma}: \mathsf{T}_\Sigma \circ \mathsf{T}_\Sigma \longrightarrow \mathsf{T}_\Sigma$ which gives $\mathbf{T}_\Sigma = (\mathsf{T}_\Sigma, \eta^{\mathsf{T}_\Sigma}, \mu^{\mathsf{T}_\Sigma})$ as a monad.

In the general case of monoidal biclosed categories, with underlying category C, T_Σ can also be completed to a monad over C, because the tensor product preserves colimits, in particular inductive limits. The only open question is here whether this monad is idempotent. Also note that in the case of idempotent monads the Kleisli category is always equivalent to the Eilenberg–Moore category [48].

Monadic extensions are important for computer science applications, since composition of substitutions, morphisms in the corresponding Kleisli category to these monads, become enabled. The notation culture in computer science, as oriented more towards using Set, expresses a term t as being within the term set $\mathsf{T}_{\Sigma,\mathtt{s}} X_{\mathtt{S}}$, and this is frequently written briefly as $t :: \mathtt{s}$, intuitively saying "the term t is of type s". Similarly, for a substitution $\sigma : X_{\mathtt{s}} \longrightarrow \mathsf{T}_{\Sigma,\mathtt{s}} X_{\mathtt{S}}$, in programming languages we often write $x := t$, or even just $x = t$, instead of $\sigma(x) = t$, where $x \in X_{\mathtt{s}}$ and $t :: \mathtt{s}$.

6.2. Many-sorted term functors over underlying categories for uncertainty

We now shift from $\mathtt{Set}$ as the underlying category for signatures to $\mathtt{Set}(\mathfrak{Q})$, where $\mathfrak{Q} = (Q, \vee, \odot)$ is a quantale. The first idea for constructing term monads over $\mathtt{Set}(\mathfrak{L})$, where $\mathfrak{L}$ is a complete lattice, was given in [49] and further developed in [50,51]. In this section we discuss the situation concerning the term functor, generally constructed over any monoidal biclosed category, and how it now specializes in the specific case of $\mathtt{Set}(\mathfrak{Q})$. Note that $(A, \alpha) \otimes (B, \beta) = (A \times B, \alpha \odot \beta)$.

Invoking uncertainty in signatures can now be performed e.g., using $\mathtt{C} = \mathtt{Set}(\mathfrak{Q})$. However, from computer science point of view, whereas uncertainty attached to operators can be intuitively justified, the interpretation of uncertain sorts is less clear. *Polymorphism of types*, on the other hand, means variables can appear under several type settings, in particular in situations where subtypes occur. Generalizations of many-sortedness to order-sortedness describes such subtypings, allowing the sort set to be partially ordered, and has been developed e.g., in [52].

For a more conventional interpretation concerning computations and programming languages, a signature $\Sigma = ((\Omega, \alpha), \tau, (S, \beta))$ in $\mathtt{Sign}_{\mathtt{Set}(\mathfrak{Q})}$ would typically come with the restriction $\beta = \top$. The computer science notation for such restricted signatures over $\mathtt{Set}(\mathfrak{Q})$ would be pairs $(S, (\Omega, \alpha))$, where S is a crisp set, and $\alpha : \Omega \longrightarrow Q$ assigns uncertain values to operators. For the term monad construction we need objects $(\Omega^{\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}}, \alpha^{\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}})$ for the operators $\omega : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$ with n given, and $(\Omega^{\to \mathtt{s}}, \alpha^{\to \mathtt{s}})$ for the constants $\omega :\to \mathtt{s}$. These objects are provided by respective pullbacks using (Ω, α).

In our general term functor construction we have

$$\Psi_{\mathtt{m},\mathtt{s}}((X_{\mathtt{t}})_{\mathtt{t}\in\mathtt{S}}) = \Omega^{\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}} \otimes \bigotimes_{i=1,\ldots,n} X_{\mathtt{s}_i},$$

and this now specializes to

$$\Psi_{\mathtt{m},\mathtt{s}}(((X_{\mathtt{t}}, \delta_{\mathtt{t}}))_{\mathtt{t}\in\mathtt{S}}) = (\Omega^{\mathtt{s}_1 \times \ldots \times \mathtt{s}_n \to \mathtt{s}}, \alpha^{\mathtt{s}_1 \times \ldots \times \mathtt{s}_n \to \mathtt{s}}) \otimes \bigotimes_{i=1,\ldots,n} (X_{\mathtt{s}_i}, \delta_{\mathtt{s}_i})$$

$$= \left(\Omega^{\mathtt{s}_1 \times \ldots \times \mathtt{s}_n \to \mathtt{s}} \times \prod_{i=1,\ldots,n} X_{\mathtt{s}_i}, \alpha^{\mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}} \odot \bigodot_{i=1,\cdots,n} \delta_{\mathtt{s}_i} \right).$$

The extension of $\mathsf{T}_\Sigma : \mathtt{Set}(\mathfrak{Q})_{\mathtt{S}} \longrightarrow \mathtt{Set}(\mathfrak{Q})_{\mathtt{S}}$ follows the same type of construction as compared to T_Σ over $\mathtt{Set}_{\mathtt{S}}$, and we arrive at $\mathbf{T}_\Sigma = (\mathsf{T}_\Sigma, \eta^{\mathsf{T}_\Sigma}, \mu^{\mathsf{T}_\Sigma})$ as a monad over $\mathtt{Set}(\mathfrak{Q})_S$.

Example 6.1. The uncertainty related to operators is more intuitive, and clearly makes the distinction between operator and value of operation. In [53] we provided examples for signatures representing assessment scales in old age psychiatry, where the distinction between observer and observation is important e.g., when using assessment scales for depression.

Example 6.2. Uncertainty related to variables is less intuitive. Let us consider the following situation concerning the perception and use of "100 m". A golfer A receives this information, into a variable x_A, about a distance to flag from ball position on the fairway. The green is surrounded by bunkers both behind and in front of the green, so hitting the ball 90 or 110 m may lead to the ball landing in a bunker. This distance information together with visual identification of the shape of green is a basis for club selection. The choice may be the pitching or the sand wedge. A non-golfer B and a person not so well aware of distances in the range of 80–200 m with a ± 5 m accuracy receives the same information into a variable x_B, and uses that information to suggest carrying some heavy items by hand instead of waiting for a car to provide that transportation of the items. Clearly, x_A and x_B are differently typed so that "100" is also differently typed for A and B. This in effect means that the variables have a different capacity to carry and embrace that information.

6.3. Substitution and assignment

In this subsection we again follow the computer science notations. Variables are syntactic and their values are semantic. In variable assignments we further need to distinguish between syntactic and semantic operators. Note how

ω in the framework of its Σ-algebra $\mathfrak{A} = (A, a)$ is hosted by Σ. However, the interpretation $a(\omega)$ of ω is also an operator but not formally hosted by a signature as compared to ω. This usually leads to confusion. In order to overcome this confusion we have to be more clear about the semantic operators to which syntactic operators are mapped. Now consider Σ as the syntactic hosting signature and Σ' the corresponding semantic hosting signature. Further, let $\varsigma = (\mathfrak{s}, \mathfrak{o}) : \Sigma \longrightarrow \Sigma'$ be a signature mapping that binds syntactic operators over to operators on the algebraic side, and let $\widehat{\varsigma} : \mathsf{T}_\Sigma \longrightarrow \mathsf{T}_{\Sigma'}$ be the corresponding natural transformation.

For X_S being a $\mathtt{Set}_S$-object of variables, $\mathfrak{A}_\Sigma = (A_S, a_S)$ being a Σ-algebra, and $\mathfrak{A}_{\mathsf{T}_\Sigma} = (A_S, h_S)$ a T_Σ-algebra, by a *variable assignment* with respect to X and $\mathfrak{A}_\Sigma$, i.e., an assignment for variables in X to values in $\mathfrak{A}_\Sigma$, we mean a morphism $(\mathfrak{v}^{X_S, \mathfrak{A}_\Sigma})_S : X_S \longrightarrow A_S$, where the term $\mathfrak{A}_\Sigma$-evaluation is $\mathsf{T}_\Sigma(\mathfrak{v}^{X_S, \mathfrak{A}_\Sigma})_S : \mathsf{T}_\Sigma X_S \longrightarrow \mathsf{T}_\Sigma A_S$. A variable assignment with respect to X_S and $\mathfrak{A}_{\mathsf{T}_\Sigma}$, i.e., an assignment for variables in X_S to values in $\mathfrak{A}_{\mathsf{T}_\Sigma}$, is a morphism $(\mathfrak{v}^{X_S, \mathfrak{A}_{\mathsf{T}_\Sigma}})_S : X_S \longrightarrow A_S$, where the term $\mathfrak{A}_{\mathsf{T}_\Sigma}$-evaluation is $h_S \circ \widehat{\varsigma}_{A_S} \circ \mathsf{T}_\Sigma(\mathfrak{v}^{X_S, \mathfrak{A}_{\mathsf{T}_\Sigma}})_S : \mathsf{T}_\Sigma X_S \longrightarrow A_S$.

For a term $t_S \in \mathsf{T}_\Sigma X_S$, note how $\mathsf{T}_\Sigma(\mathfrak{v}^{X_S, \mathfrak{A}_{\mathsf{T}_\Sigma}})_S(t_S)$ is the syntactic term in the appearance as its semantic counterpart, but still using its syntactical operators, and $\widehat{\varsigma}_{A_S} \circ \mathsf{T}_\Sigma \mathfrak{v}^{X, \mathfrak{A}_{\mathsf{T}_\Sigma}}(t_S)$ is correspondingly the term in its semantic appearance using its semantic operators. Finally, $h_S \circ \widehat{\varsigma}_{A_S} \circ \mathsf{T}_\Sigma(\mathfrak{v}^{X_S, \mathfrak{A}_{\mathsf{T}_\Sigma}})_S(t_S)$ represents 'the value of $t_\mathfrak{s}$ in $A_\mathfrak{s}$', as the computed value given the semantics of the operators.

7. Terms in type theory

Similar to the informal definition of the set of terms, also the set of λ-terms is defined informally. Whereas the informal definition of the set of terms corresponds well with the formal definition, in the case of the set of λ-terms the situation is different. The symbol λ is frequently seen as an "abstractor", but it is not all that clear if λ really has a general purpose capacity to abstract, or if operators in the underlying signature "owns" their abstractions, i.e., is λ in an abstraction really to be indexed by the operator it abstract. Note also that it is actually the operator that is abstracted to another operator, not the term that is abstracted to an operator. Traditional presentations of λ-calculus is not always specific on that point. Church [54] called "λ" an *improper symbol*, together with "(" and ")" also being improper symbols. The *proper symbols* are then those residing in the signature, or being symbols for variables. Church's *simple typing* [54] has as a motivation the statement that *a complete incorporation of the calculus of λ-conversion into the theory of types is impossible if we require that λ and juxtaposition shall retain their respective meanings as an abstraction operator and as denoting the application of function to argument*. Therefore λ is not to be seen as an operator in any signature. Church's types ι and o, and their algebras, are not at all obvious, and it is indeed unclear in which signature they actually reside. Church introduces the type constructor so that $(\beta\alpha)$ is a type whenever β and α are types.

The informal definition of untyped λ-terms is as follows:

- a variable is a λ-term;
- if M is a λ-term, then $\lambda x.M$ is a λ-term, where x is a variable (abstraction);
- if M and N are λ-terms, then also MN is a λ-term (application).

This is seen as an elegant definition of λ-term, even if it is unspecific, e.g., about possible underlying 'basic' terms formed out of an underlying signature. Variables are such basic terms, but it is not explicitly said that all underlying basic terms are λ-terms. If the underlying signature is $\Sigma = (S, \varnothing)$, then variables are the only basic terms. Usually basic terms are seen as λ-terms also in the case where $\Omega \neq \varnothing$. For example, for the basic term $x + 1$, $\lambda x.x + 1$ is the abstraction.

The situation concerning syntactically constructed natural numbers is interesting. The natural numbers can be introduced as λ-terms either as basic terms given the NAT signature, or, as Church did, fixing a function variable $f_{\alpha\alpha}$ as kind of a "fixsymbol" for natural numbers, and then proceed to define $0_{\alpha'}$ as $\lambda f_{\alpha\alpha}.\lambda x_\alpha.x_\alpha$, $1_{\alpha'}$ as $\lambda f_{\alpha\alpha}.\lambda x_\alpha.(f_{\alpha\alpha} x_\alpha)$, $2_{\alpha'}$ as $\lambda f_{\alpha\alpha}.\lambda x_\alpha.(f_{\alpha\alpha}(f_{\alpha\alpha} x_\alpha))$, and so on.

7.1. Levels of signatures

The underlying basic signatures do not include function types and a categorical handling, e.g., of λ-term functors therefore requires an arrangement into levels of signatures. Let $\mathtt{s}_1$ and $\mathtt{s}_2$ be two sorts in S. The function sort involving $\mathtt{s}_1$ and $\mathtt{s}_2$ can be denoted $\mathtt{s}_1 \Rrightarrow \mathtt{s}_2$. Even if we want to view $\mathtt{s}_1 \Rrightarrow \mathtt{s}_2$ as a (constructed) sort, it is not part of S. The

question is then how to expand the signature $\Sigma = (S, \Omega)$ to a signature $\Sigma' = (S', \Omega')$ so that $\mathtt{s}_1 \Rightarrow \mathtt{s}_2 \in S'$ whenever $\mathtt{s}_1, \mathtt{s}_2 \in S$. Such an arrangement also enables to keep λ-abstractions, as members of Ω', clearly apart from λ-terms, residing in the set of λ-terms as defined by the λ-term functor.

We propose a three-level arrangement of signatures, where the basic signature Σ is on level one, and Σ' is on level three. On level two we have the (Σ-)*superseding type signature* as a one-sorted signature $\mathtt{S}_\Sigma = (\{\mathtt{type}\}, \mathtt{Q})$, where $\mathtt{Q}$ is a set of *type constructors* satisfying

(i) $\mathtt{s} :\rightarrow \mathtt{type}$ is in $\mathtt{Q}$ for all $\mathtt{s} \in S$;
(ii) there is a $\Rightarrow : \mathtt{type} \times \mathtt{type} \rightarrow \mathtt{type}$ in $\mathtt{Q}$.

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

For any Σ-superseding type signature $\mathtt{S}_\Sigma$ we obviously have the term monad, called the *type term monad*, $\mathbf{T}_{\mathtt{S}_\Sigma}$, over a given category whose objects then intuitively represent the object of variables, and on the superceding level indeed intuitively as type variables. Then $\mathsf{T}_{\mathtt{S}_\Sigma} X$, where X is the tuple of objects representing (type) variables, contains all terms which we call *type terms*. We may write $\mathtt{s} \Rightarrow \mathtt{t}$ for the type term $\Rightarrow(\mathtt{s}, \mathtt{t})$, which we call an *arrow type term*.

The signature $\Sigma' = (S', \Omega')$ on level three then is based on $S' = \mathsf{T}_{\mathtt{S}_\Sigma} \emptyset$, i.e., the sorts on level three are those from level one together with the constructed sorts, on level two appearing as terms (the type terms), added to those basic sorts coming from level one.

For the operators in Ω' it may sound natural to include all operators from Ω into Ω' so that $\Omega \subseteq \Omega'$, but it is not always desirable. If we consider the NAT signature on level one we obviously may have both $\mathtt{0} :\rightarrow \mathtt{nat}$ and $\mathtt{succ} : \mathtt{nat} \rightarrow \mathtt{nat}$ included in the operators for NAT′. However, the unary operator succ, i.e., unary both on level one and level three, can alternatively be (λ-)abstracted to become a constant (0-ary) operator $\lambda_1^{\mathtt{succ}} :\rightarrow (\mathtt{nat} \Rightarrow \mathtt{nat})$ on level three. Clearly, the constant $\mathtt{0} :\rightarrow \mathtt{nat}$ converts to $\lambda_0^0 :\rightarrow \mathtt{nat}$, i.e., a constant on level remains as a constant on level three. Note also that nat on level one is not the same as nat on level three. If we need to be strict, we should use e.g., $\mathtt{nat}'$ for the corresponding sort on level three.

We usually want to have $S \subseteq S'$, i.e., an S is embedded into S', where nat maps to $\mathtt{nat}'$, but not necessarily $\Omega \subseteq \Omega'$. The situation involving the construction of λ-terms, where $\Omega \nsubseteq \Omega'$, is discussed more in detail in the subsection below.

Remark. For $\Sigma = (S, \Omega)$ on level one, $\mathtt{S}_\Sigma$ can obviously be extended with further operators beyond just $\Rightarrow$. We could e.g., include $\mathtt{F} : \mathtt{type} \rightarrow \mathtt{type}$ semantically corresponding to a functor, as we do for description logic (see below) when using the powerset functor to represent relations.

Remark. Note, in the mathematical notation for signatures according to [46], that the monoidal arrangement does not embrace any structure related to the Σ-superceding level. That is, in [46] there is no clearly expressed anticipation of applications in type theory.

Remark. Whereas the algebras $\mathfrak{A}(\Sigma)$ of signatures Σ, involving assignments of sorts s to domains $\mathfrak{A}(\mathtt{s})$ of $\mathfrak{A}$, are standard according to universal algebra, the 'algebra' of the (Σ-)superseding type signature $\mathtt{S}_\Sigma$ is not immediate since the domain assigned to the sort type clearly cannot be just a set. There are several options for this, and these considerations may go beyond traditional universal algebra. These discussions are outside the scope of this paper.

Church's type constructor is in effect our $\Rightarrow$, so that $(\beta \Rightarrow \alpha)$ is his $(\beta\alpha)$. An interpretation of Church's ι to be our type is clearly less controversial, but for the interpretation of o there are a number of alternative intuitions. Church says, *o is the type of propositions*, but at that point nothing is said about *proposition*. In fact, Church states the following: *We purposely refrain from making more definite the nature of the types o and ι, the formal theory admitting of a variety of interpretations in this regard. Of course the matter of interpretation is in any case irrelevant to the abstract construction of the theory, and indeed other and quite different interpretations are possible* (*formal consistency assumed*).

We will conclude this section with a discussion on natural language expressions involving modifiers and quantifiers, and the possibility to understand and even encode these expressions more formally. Lotfi Zadeh frequently uses expressions like "most Swedes are tall" and "there are more small balls than large balls in the box". The difficulty to handle these expressions in fuzzy logic is well-known, and unique solutions have not been presented in the literature. We believe that unique solutions in fact do not exist, and here we will provide a brief view about possible encodings

of such expression, or related subexpressions, in our three level exposition of signatures involving type constructors. Intuitively it seems as modifiers are closely related to type constructors, or at least that modifiers are operators on level three being specified using constructed types on level two. Quantifiers are more like abstractors of sentences, and it may be anticipated that the formalization of quantifiers in this sense is similar to the formalization of the way λ acts expressions.

Let us have a closer look at modifiers. When we say "most a are b" we usually see a is specified by types of non-Boolean character, whereas b is more seen as specified by a Boolean-like type. In this case "most" is about counting something we loosely accept to be a's that satisfy the condition that we semantically and individually understand to be b. This counting is more like a process or procedure, or almost like an algorithm producing a degree of confidence that indeed "most a are b". It is, however, unclear whether or not "most" can be seen as a quantifier in some general sense. If we say "there are more a's than b's in c", or concretely, "there are more apples than pears in the fruit basket", this may be informative in the kitchen. Note also that "there are more apples than pears, but there are less Ingrid Marie apples than Anjou pears in the fruit basket" is maybe more informative for advanced cooking. An expert on fruit might also say "the Comice pear on the fruit plate looks more fresh than the Granny Smith apple". The *fruit plate* and *fruit basket* clearly shows we are dealing with sets and even hierarchies of sets, in a sense to be semantically defined by and within respective context. A basket can be seen as having more structure than a plate, and a plate is something more that just a set.

Before proceeding with concretization of these examples, we need some prerequisites about type constructors and the superceding signatures. For any unary type constructors $\phi, \psi : \mathtt{type} \to \mathtt{type}$, we define the *composed type constructor* $\psi \circ \phi : \mathtt{type} \to \mathtt{type}$ by $(\psi \circ \phi)\mathtt{s} = \psi(\phi\mathtt{s})$. For unary type constructors $\phi, \psi : \mathtt{type} \to \mathtt{type}$, a *type transformation* τ *from* ϕ *to* ψ, denoted $\tau : \phi \Rightarrow \psi$, if it exists, is assumed, for all $\mathtt{s} \in \mathsf{T}_{\mathrm{S}_\Sigma}\varnothing$, to be given by a unique (constant operator) $\tau_{\mathtt{s}} :\to (\phi\mathtt{s} \Rrightarrow \psi\mathtt{s})$.

It is indeed tempting to view the type constructor ϕ itself as a functor, but syntactically it obviously not a functor. Further, we may want to assume that any $\mathtt{f} :\to (\mathtt{s} \Rrightarrow \mathtt{t})$ gives rise to a unique $\phi\mathtt{f} :\to (\phi\mathtt{s} \Rrightarrow \phi\mathtt{t})$, and this intuitively opens up the question e.g., preservation of composition, as we syntactically have not defined any composition of constants, otherwise than using the application operator on level three.

Example 7.1. Hierarchies of sets, or sets of sets, sets of sets of sets, and so on, can be modelled by the 'powerset' type constructor $\mathsf{P} : \mathtt{type} \to \mathtt{type}$ on level two, i.e., intuitively thinking that P indeed is a powerset functor, not necessarily the ordinary powerset functor, over `Set`. We have to pay attention to 'double powerset' type constructors, since in the case of composing powerset functor into a so-called double powerset functor we have two choices, namely, composing the covariant powerset functor with itself or the contravariant powerset functor with itself. Note that composing contravariant functors produces a covariant functor. We may denote the 'contravariant powerset' type constructor by $\overline{\mathsf{P}} : \mathtt{type} \to \mathtt{type}$. For the composition of the contravariant powerset type constructor with itself we would expect $\mathfrak{U}(\overline{\mathsf{P}} \circ \overline{\mathsf{P}}) = \mathfrak{U}(\overline{\mathsf{P}}) \circ \mathfrak{U}(\overline{\mathsf{P}})$.

We are now able to show how encoding a fruit basket and its content needs to make clear distinctions about what are types on level two and which are the constants or operators in general on level three. It seems natural to say that *Ingrid Marie* and *Anjou* are types of apples and pears, respectively, so we should then have $\mathtt{IngridMarie}, \mathtt{Anjou} :\to \mathtt{type}$ on level two. This enables to have a specific apple $apple_0$ and a pear $pear_0$ as constants on level three if we include $apple_0 :\to \mathtt{IngridMarie}$ and $pear_0 :\to \mathtt{Anjou}$, or we may declare variables x and y according to $x :: \mathtt{IngridMarie}$ and $y :: \mathtt{Anjou}$. Then, both substitutions $x := apple_0$ and $y := pear_0$ make sense.

We should clearly include $\mathtt{Fruit}, \mathtt{Apple}, \mathtt{Pear} :\to \mathtt{type}$ on level two, and then we can include $Apple, Pear :\to \mathtt{Fruit}$ on level three, but *Apple* and *Pear* as (constant) terms on level three must not be identified or confused with `Apple` and `Pear` as terms on level two. Similarly, we can include $IngridMarie :\to \mathtt{Apple}$ and $Anjou :\to \mathtt{Pear}$ on level three, where again *IngridMarie* on level three and `IngridMarie` on level two, and *Anjou* on level three and `Anjou` on level two, must not be identified. On level three we are then allowed to include also that *Apple* and *Pear* are of type `Fruit`.

For the typing of the fruit basket we have a number of options. We may see fruit baskets as always allowing any fruits, or we may want to have different baskets for fruits in general, but also baskets specifically for apples, pears, and so on. For the latter, we would first specify $\mathtt{FruitBasket} : \mathtt{type} \to \mathtt{type}$ on level two as a type intuitively being a powerset constructor, so that $\mathtt{FruitBasket}(\mathtt{Fruit}), \mathtt{FruitBasket}(\mathtt{Apple}), \mathtt{FruitBasket}(\mathtt{Pear}) :: \mathtt{type}$, as terms on level two, and then include $\mathtt{AllFruitsBasket} :\to \mathsf{P}(\mathtt{Fruit}), \mathtt{AppleFruitBasket} :\to \mathsf{P}(\mathtt{Apple})$ and

$\mathtt{PearFruitBasket} :\rightarrow \mathsf{P}(\mathtt{Pear})$ as a constant on level three. This captures the idea that a fruit basket is a set of fruits, i.e., $FruitBasket :: \mathsf{P}(\mathtt{Fruit})$.

When we start to use the basic constants and terms in connection with other operators providing counting we must decide to have one or the other, and be very careful about not moving freely between respective definitions. In natural language, this in fact happens frequently, and one person communicating with another is not always aware of which typing is adopted. Even worse, if nobody is aware of distinct typing, each and everyone may shift to adopt definitions from one typing to another. This is the main problem of natural language.

Counting fruits, apples or pears in the fruit basket is now something close to establishing cardinality of subsets of apples and pears in a set of fruits, i.e., we might have operators $card^{\mathtt{Apple}} : \mathtt{FruitBasket(Fruit)} \rightarrow \mathtt{nat}$ and $card^{\mathtt{Pear}} : \mathtt{FruitBasket(Fruit)} \rightarrow \mathtt{nat}$ on level three. Here we must assume to have nat on level one so that $\mathtt{nat} :\rightarrow \mathtt{type}$ is on level two. Similarly we may have $card^{\mathtt{IngridMarie}} : \mathtt{FruitBasket(Apple)} \rightarrow \mathtt{nat}$ and $card^{\mathtt{Anjou}} : \mathtt{FruitBasket(Pear)} \rightarrow \mathtt{nat}$ for the cardinality of the subset of Ingrid Marie apples in a set of apples, and the cardinality of Anjou pears in a set of pears.

Finally we need to include conversion operators so that an Ingrid Marie really is an apple, an Anjou really a pear, and that apples and pears are fruits.

In summary, the typing on level two is

$$\begin{gathered}\mathsf{P}, \mathtt{FruitBasket}, \mathtt{FruitPlate} : \mathtt{type} \rightarrow \mathtt{type} \\ \mathtt{Fruit}, \mathtt{Apple}, \mathtt{Pear} :\rightarrow \mathtt{type} \\ \mathtt{IngridMarie}, \mathtt{Anjou} :\rightarrow \mathtt{type} \\ \mathtt{nat} :\rightarrow \mathtt{type}\end{gathered}$$

and on level three

$$\begin{gathered}\mathtt{AllFruitsBasket} :\rightarrow \mathtt{FruitBasket(Fruit)} \\ \mathtt{AppleFruitBasket} :\rightarrow \mathtt{FruitBasket(Apple)} \\ \mathtt{PearFruitBasket} :\rightarrow \mathtt{FruitBasket(Pear)} \\ Apple, Pear :\rightarrow \mathtt{Fruit} \\ IngridMarie :\rightarrow \mathtt{Apple} \\ Anjou :\rightarrow \mathtt{Pear} \\ apple_0, apple_1, \ldots :\rightarrow \mathtt{IngridMarie} \\ pear_0, pear_1, \ldots :\rightarrow \mathtt{Anjou} \\ card^{\mathtt{Apple}} : \mathtt{FruitBasket(Fruit)} \rightarrow \mathtt{nat} \\ card^{\mathtt{Pear}} : \mathtt{FruitBasket(Fruit)} \rightarrow \mathtt{nat} \\ card^{\mathtt{IngridMarie}} : \mathsf{P}(\mathtt{Apple}) \rightarrow \mathtt{nat} \\ card^{\mathtt{Anjou}} : \mathsf{P}(\mathtt{Pear}) \rightarrow \mathtt{nat} \\ \phi^{\mathtt{Apple}\rightarrow\mathtt{Fruit}} : \mathtt{Apple} \rightarrow \mathtt{Fruit} \\ \phi^{\mathtt{Pear}\rightarrow\mathtt{Fruit}} : \mathtt{Pear} \rightarrow \mathtt{Fruit} \\ \phi^{\mathtt{IngridMarie}\rightarrow\mathtt{Apple}} : \mathtt{IngridMarie} \rightarrow \mathtt{Apple} \\ \phi^{\mathtt{Anjou}\rightarrow\mathtt{Pear}} : \mathtt{Anjou} \rightarrow \mathtt{Pear}\end{gathered}$$

We now leave it to the reader to extend the example with various definitions for "there are more apples than pears, but there are less Ingrid Marie apples than Anjou pears in the fruit basket", where at least types and operators from NATORD need to be included at level one. This may expectedly be done in a number of ways.

An additional subtlety arises from the intuitively different semantics of 'set', 'plate', and 'basket', which then also calls for using type transformations between FruitBasket and P, and between FruitPlate and P, semantically acting as a forgetful and/or flattening transformation:

$$\begin{gathered}\tau^{BasketToSet}_{\mathtt{Fruit}} :\rightarrow (\mathtt{FruitBasket(Fruit)} \Rrightarrow \mathsf{P}(\mathtt{Fruit})) \\ \tau^{PlateToSet}_{\mathtt{Fruit}} :\rightarrow (\mathtt{FruitPlate(Fruit)} \Rrightarrow \mathsf{P}(\mathtt{Fruit}))\end{gathered}$$

In this context, note how a corresponding type transformation from FruitBasket to FruitPlate, or from FruitPlate to FruitBasket, for that matter, is much less obvious, as the former may be represented simply by a action pouring fruit from a plate into a basket, and the latter allow fruit from a basket to fall randomly out onto a plate.

This example can then be extended further to work over selections of underlying categories with $\mathtt{Set}(\mathfrak{Q})$ as the prime example for such an underlying category.

7.2. λ-Terms and fuzzy λ-calculus

The three signature levels underlying the production of λ-terms are the following.

1. the level of primitive underlying operations, with a usual many-sorted signature $\Sigma = (S, \Omega)$;
2. the level of type constructors, with a single-sorted signature $\mathtt{S}_\Sigma = (\{\mathtt{type}\}, \{\mathtt{s} :\to \mathtt{type} | \mathtt{s} \in S\} \cup \{\Rightarrow : \mathtt{type} \times \mathtt{type} \to \mathtt{type}\})$;
3. the level including λ-terms based on the signature $\Sigma' = (S', \Omega')$ where $S' = \mathsf{T}_{\mathtt{S}_\Sigma}\emptyset$, $\Omega' = \{\lambda^{\omega}_{i_1,\ldots,i_n} :\to (\mathtt{s}_{i_1} \Rightarrow \cdots \Rightarrow (\mathtt{s}_{i_{n-1}} \Rightarrow (\mathtt{s}_{i_n} \Rightarrow \mathtt{s})) | \omega : \mathtt{s}_1 \times \ldots \times \mathtt{s}_n \to \mathtt{s} \in \Omega\} \cup \{\mathtt{app}_{\mathtt{s},\mathtt{t}} : (\mathtt{s} \Rightarrow \mathtt{t}) \times \mathtt{s} \to \mathtt{t}\}$.

Here $(i_1, \ldots, i_n)$ is a permutation of $(1, \ldots, n)$. Note also that level one operators are always transformed to constants on level three. In traditional notation in λ-calculus, substituting x by $\mathtt{succ}(y)$ in $\lambda y.\mathtt{succ}(x)$ requires a renaming of the bound variable y, e.g., $\lambda z.\mathtt{succ}(\mathtt{succ}(y))$. In our approach we avoid the need for renaming. On level one, and in the case of NAT, we have the substitution (Kleisli morphism) $\sigma_{\mathtt{nat}} : X_{\mathtt{nat}} \longrightarrow \mathsf{T}_{\mathtt{NAT},\mathtt{nat}}(X_{\mathtt{t}})_{\mathtt{t}\in\{\mathtt{nat}\}}$, where $\sigma_{\mathtt{nat}}(x) = \mathtt{succ}(y)$, x being a variable on level one, and the extension of $\sigma_{\mathtt{nat}}$ is $\mu_{X_{\mathtt{nat}}} \circ \mathsf{T}_{\mathtt{NAT},\mathtt{nat}}(\sigma_{\mathtt{t}})_{\mathtt{t}\in\{\mathtt{nat}\}} : \mathsf{T}_{\mathtt{NAT},\mathtt{nat}}(X_{\mathtt{t}})_{\mathtt{t}\in\{\mathtt{nat}\}} \longrightarrow \mathsf{T}_{\mathtt{NAT},\mathtt{nat}}(X_{\mathtt{t}})_{\mathtt{t}\in\{\mathtt{nat}\}}$. On level three we have $\sigma_{\mathtt{nat}'} : X_{\mathtt{nat}'} \longrightarrow \mathsf{T}_{\mathtt{NAT}',\mathtt{nat}'}(X_{\mathtt{t}})_{\mathtt{t}\in S'}$, with $\sigma_{\mathtt{nat}'}(x) = \mathtt{app}_{\mathtt{nat}',\mathtt{nat}'}(\lambda_1^{\mathtt{succ}}, x)$, x being a variable on level three, and $\mu_{\mathtt{nat}'} \circ \mathsf{T}_{\mathtt{NAT}',\mathtt{nat}'}\,\sigma_{\mathtt{nat}'}(\mathtt{app}_{\mathtt{nat}',\mathtt{nat}'}(\lambda_1^{\mathtt{succ}}, x))$ requiring no renaming.

At this point we have the crisp set of λ-terms, given the term functor $\mathsf{T}_{\Sigma'} : \mathtt{Set}_{S'} \longrightarrow \mathtt{Set}_{S'}$. The sets of λ-terms with respect to each end sort $\mathtt{s}' \in S'$ are then represented by respective sets $\mathsf{T}_{\Sigma',\mathtt{s}'}(X_{\mathtt{s}})_{\mathtt{s}\in S'}$.

Fuzzy λ-terms can now be introduced either using monad compositions, or allowing $\mathsf{T}_{\Sigma'}$ to be a functor over an underlying category $\mathtt{Set}(\mathfrak{Q})$, where $\mathfrak{Q}$ typically is a quantale, but can also be a lattice or a Kleene algebra.

In [55], fuzzy λ-calculus is treated differently as λ-terms remain the classical ones, and as informally defined. The underlying signatures are also crisp. Fuzzy aspects appear not until λ-calculus is used as a basis for logic. This is presented similar to the approach in [54], i.e., levels of signatures are not considered. The underlying basic terms are crisp, i.e., arise from the many-sorted term functor over Set, even if the functorial construction of terms is not explicitly mentioned. Fuzziness in [55] is therefore an ad hoc construction applied on the crisp and traditional view of λ-terms.

7.3. Description logic

Widening the scope of relational structures begins with the observation that the category Rel of sets and relations is isomorphic with the Kleisli category $\mathtt{Set}_{\mathbf{P}}$ of the powerset monad $\mathbf{P}$ over the category Set of sets and functions. Every relation $R \subseteq X \times X$ is representable by a mapping $\sigma_R : X \longrightarrow \mathsf{P}X$, and every mapping $\sigma : X \longrightarrow PX$ is representable by a relation $R_\sigma \subseteq X \times X$. It is easily seen that $\sigma_{R_\sigma} = \sigma$ and $R_{\sigma_R} = R$.

Considering other monads $\mathbf{\Phi}$ over Set, and over other categories C, for that matter, widens the concept of "relation" significantly, viewing it more like a "substitution", and moreover, going beyond just Set enables to analyze various attributions, like uncertainty and stochasticity, to be canonically integrated into terms and sentences under considerations in chosen syntax for a logic.

The history of modal logic syntax goes back to Clarence Lewis and his work [56] dating back to the time after publication of *Principia Mathematica* and at the footstep of *Hilbert's Lectures*. The history of modal logic semantics started with work by Rudolph Carnap in the 1940s and culminates with Saul Kripke's semantical analysis of modal logic [57]. Thereafter we witness a wide range of extensions to modal logic, e.g., in dynamic logic that was initiated by Pratt [58] based on his lecture notes at MIT in 1974, again inspired by Floyd–Hoare's logic for program description, also adopted by Dijkstra in his predicate transformer semantics. Dynamic logic also mirrors to concepts within languages in the sense of "automata and languages", related, e.g., to Chomsky's hierarchies, and also spilling over to Kleene algebras and analysis of program construction in that fashion.

Another historical branch leading to description logic starts with Ross Quillian's developments of his word concepts [59] in the realm of memory models, and based on his PhD thesis in 1966 [60]. This restricts it to "hold only denotative, factual information", and "not a person's plans for doing things", in the latter exclusion referring to Jean Piaget's "schemata" [61]. Quillian is not explicitly saying whether "denotative" is syntax or semantics, but when taking also

Quillian work as a historical background to description logic, "denotative" may be seen as embracing both "syntactic (de)notation", i.e., signature, and semantic denotation, either as such without syntax or as a semantic assignment for syntactic elements. Piaget is weaker on the distinction between syntax and semantics. In his *The Psychology of Intelligence* [61], Piaget writes about "thought psychology" and the psychological nature of logical operations, saying that *How far a psychological explanation of intelligence is possible depends on the way in which logical operations are interpreted: Are the reflection of an already formed reality or the expression of a genuine activity*? It may seem that Piaget on "already formed reality" explicitly refers to logical semantics, and by "expression of a genuine activity" to syntax, but this is not so. In fact, Piaget continues *the logician then proceeds as does the geometer with the space that he constructs deductively*, *while the psychologist can be likened to the physicist*, *who measures space in the real world.* Piaget almost wants to exclude logic from psychology, not include it.

Marvin Minsky introduced a knowledge representation schema with rule-based and logic-based formalisms [62]. In order to represent these concepts he introduced 'frames', including descriptive information about how to understand and use frames. Collections of such frames are organized in systems in which the frames are interconnected.

Developments then pass through description languages, and eventually 'language' is changed to 'logic'. *Description logic* (DL) is not a specific logic but rather seen as a family of logics being variants of the *attributive concept description language* ($\mathcal{ALC}$) [63], in turn based on the underlying *formal frame description language* ($\mathcal{FL}$) [64] for KL-ONE [65]. In [64], *structured types* are mentioned in connection with frames [62], but typing here is not explained in more formal type theoretic fashion. Types become connected with concepts, and are treated similarly and intuitively as Quillian's (*nodes that can be reached by an exhaustive tracing process, originating at its initial*) *patriarchical type nodes* (*together with the total sum of relationships among these nodes specified by within-plane, token-to-token links*) that are also interesting from typing point of view, even if Quillian's work does not touch upon typing or type constructors more formally, so there are no historical remarks on distinctions to be found for concepts and concept types. Roles and relations were at that time intuitively seen as relations between concepts, and not relations between their respective types.

We can now show how to connect modal operators, having relations as their semantics, to type constructors having powerset functors as their semantics. We will make these situations explicit in particular for description logics and its "existential roles" as modal operators. Typing the operators and quantifiers in this way also emphasizes the propositional nature of description logic.

For the purpose of this section we will refer to notations in [63], and transforms that machinery into our categorical framework for superceding signatures and related typing, eventually arriving at our enriched notions and properties for DL.

We will start with providing some required formalism to $\mathcal{ALC}$, which exists implicitly, and is mostly seen as folklore for the DL community, but needs to be stated formally, before we can proceed with our categorical and term monadic formalism. The need for this formalism, on the other hand, is not folklore, and is in fact, a major reason for this paper.

Firstly, we will examine the *interpretation* $\mathcal{I} = (D^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\cdot^{\mathcal{I}}$ *maps every concept description* to a subset of $D^{\mathcal{I}}$. The use of the letter D for that universe is a bit unfortunate, since D is also used for concept descriptions, e.g., in expressions like $C \bigsqcup D$, where D is not to be understood as the "D in $D^{\mathcal{I}}$".

If C is a "concept", the interpretation $C^{\mathcal{I}}$ of that concept is a subset of $D^{\mathcal{I}}$, i.e., an element of the powerset $\mathsf{P}D^{\mathcal{I}}$. This means that $\mathsf{P}D^{\mathcal{I}}$ is the actual domain in the classical sense of domains as part of universal algebras. "Roles" R as relations $R^{\mathcal{I}}$ on the semantic side are then seen as subsets of $D^{\mathcal{I}} \times D^{\mathcal{I}}$. This is equivalent to saying that $R^{\mathcal{I}}$ is a mapping from $D^{\mathcal{I}}$ to $\mathsf{P}D^{\mathcal{I}}$, which categorically is a morphism in the corresponding Kleisli category of the powerset monad $\mathbf{P} = (\mathsf{P}, \eta, \mu)$ over the category $\mathtt{Set}$ of sets and functions. Here η and μ are natural transformations defined by $\mu_X(\mathcal{B}) = \bigcup_{B \in \mathcal{B}} B$, and $\eta_X(x) = \{x\}$. The morphism $R^{\mathcal{I}} : D^{\mathcal{I}} \longrightarrow \mathsf{P}D^{\mathcal{I}}$ can be lifted to the morphism $\mathsf{P}R^{\mathcal{I}} : \mathsf{P}D^{\mathcal{I}} \longrightarrow \mathsf{PP}D^{\mathcal{I}}$. The inverse relation R^{-1} is important in these respects as it is the actual relation used on the semantic side. It is easy to see that in fact

$$(\exists R : C)^{\mathcal{I}} = \{a \in D^{\mathcal{I}} | \exists (a, b) \in R^{\mathcal{I}} : b \in C^{\mathcal{I}}\} = \mu_{D^{\mathcal{I}}}(\mathsf{P}R^{-1}C)$$

Now we should be aware of related typing and variables selected as being of expected types. In [63], this situation is unclear, as they *assume the existence of two further disjoint alphabets of symbols*, which are called *individual* and *concept variables*. In signatures and terms over signatures, variables are not part of the alphabet, in the sense of "alphabet" being the signature of sorts and operators. If $\Sigma = (S, \Omega)$ is a (many-sorted) signature, then variables are specific terms and are always typed. Categorically, we have the category $\mathtt{Set}_S$, and a set of variables is an object $(X_{\mathtt{s}})_{\mathtt{s} \in S}$ in that category, and $\mathsf{T}_\Sigma : \mathtt{Set}_S \to \mathtt{Set}_S$ is the many-sorted term monad [50].

If we speak of "individual concept" rather than "individual variable" we may use x, y, z as variables for individual concepts, and X, Y, Z as variables for concepts. It then remains to decide how to type "concept" and "individual concept", and here we will need type constructors. We will again change language adopted in [63] by saying "concept" instead of "individual concept", and "powerconcept" instead of "concept".

The key solution in this typing is that `concept` is a sort in the underlying given signature (on level one), and then we need to construct a *superceding* signature, so that `concept` becomes a constant operator in that signature. A type constructor P is then used to produce a new type $\mathsf{P}\texttt{concept}$, which in its algebra will be assigned according to $\mathfrak{A}(\texttt{concept}) = D^{\mathcal{I}}$ and $\mathfrak{A}(\mathsf{P}(\texttt{concept})) = \mathsf{P}D^{\mathcal{I}}$.

We are now in position to describe the *simply typed description logic*. Note that the "existential quantifier" in the syntactic expression $\exists R : C$ is actually more like an "R-modality" applied to the powerconcept C, and that the semantic expression $(\exists R : C)^{\mathcal{I}}$ invokes the existential quantifier residing in the first-order logic for ZFC. In the following we will show this more explicitly, i.e., that this "existential quantifier" is more of a modal operator, and that such modal operators need not necessarily be equipped with semantics consisting only of relations a la Tarski.

Now let $\Sigma = (S, \Omega)$ be a signature, where $S = \{\texttt{concept}\}$. We may add constants like $\texttt{c}_1, \ldots, \texttt{c}_n :\rightarrow \texttt{concept}$ already at this level, and it would be tempting to call these "syntactic concepts" with $\mathfrak{A}_\Sigma(\texttt{c}_i) \in \mathfrak{A}_\Sigma(\texttt{concept})$, where $\mathfrak{A}_\Sigma(\texttt{concept})$ could be equal to $D^{\mathcal{I}}$. However, we do not want to do that, as concepts and powerconcepts must reside in the same signature. We therefore go to the superceding signature $\mathtt{S}_\Sigma$, so that $\texttt{concept} :\rightarrow \texttt{type}$ becomes a constant in $\mathtt{S}_\Sigma$. We now propose to include into $\mathtt{S}_\Sigma$ a type constructor $\mathsf{P} : \texttt{type} \rightarrow \texttt{type}$, with an intuitive semantics of being the powerset functor. Clearly, $\mathsf{P}(\texttt{concept})$ is now the constructed type for "powerconcept". Note that $\mathsf{P}(\texttt{concept})$ is a term on the Σ-superceding level, and a sort on S', so as a candidate for its algebra we could consider $\mathfrak{A}_{\Sigma'}(\mathsf{P}(\texttt{concept})) = \mathsf{P}\mathfrak{A}_{\Sigma'}(\texttt{concept}) = \mathsf{P}D^{\mathcal{I}}$. Note how we refrain from making the distinction between P as a syntactic and semantic object, and indeed we see this as a fundamental weakness of description logic, namely this intertwining of syntax and semantics has not been properly resolved within the description logic community. A variable $x \in X_{\mathsf{P}(\texttt{concept})}$ is then a "concept variable" in the sense of [63], and is also a 'term' in the sense of being an element of $\mathsf{T}_{\Sigma',\mathsf{P}(\texttt{concept})}(X_{\mathtt{s}})_{\mathtt{s}\in S'}$.

Now it is natural to include "constant concepts" $\texttt{c}_1, \ldots, \texttt{c}_n :\rightarrow \texttt{concept}$, with `concept` having been shifted, from being a term on the Σ-superceding level over to becoming a sort in S'. Thus $\texttt{c}_1, \ldots, \texttt{c}_n$ are (constant) terms as elements of $\mathsf{T}_{\Sigma',\texttt{concept}}(X_{\mathtt{s}})_{\mathtt{s}\in S'}$, i.e., $\texttt{c}_1, \ldots, \texttt{c}_n \in \mathsf{T}_{\Sigma',\texttt{concept}}(X_{\mathtt{s}})_{\mathtt{s}\in S'}$, and $\mathfrak{A}_{\Sigma'}(\texttt{c}_i) \in \mathfrak{A}_{\Sigma'}(\texttt{concept}) = D^{\mathcal{I}}$.

Now "roles" are of the form $\texttt{r} :\rightarrow (\mathsf{P}(\texttt{concept}) \Rrightarrow \mathsf{P}(\mathsf{P}(\texttt{concept})))$, and we need further to include operators $\eta :\rightarrow (\texttt{concept} \Rrightarrow \mathsf{P}(\texttt{concept}))$ and $\mu :\rightarrow (\mathsf{P}(\mathsf{P}(\texttt{concept})) \Rrightarrow \mathsf{P}(\texttt{concept}))$ into Ω'.

Note how a concept $\texttt{c} :\rightarrow \texttt{concept}$ is shifted to becoming a "one-point powerconcept" as represented by

$$\mathtt{app}_{\texttt{concept},\mathsf{P}(\texttt{concept})}(\eta, \texttt{c})$$

Now the syntactic expression like "$\exists r.x$", i.e., a term of type $\mathsf{P}(\texttt{concept})$ can be defined according to

$$\exists r.x = \mathtt{app}_{\mathsf{P}(\mathsf{P}(\texttt{concept})),\mathsf{P}(\texttt{concept})}(\mu, \mathtt{app}_{\mathsf{P}(\texttt{concept}),\mathsf{P}(\mathsf{P}(\texttt{concept}))}(\texttt{r}, x))$$

and with the obvious algebras for η and μ, $\mathfrak{A}_{\Sigma'}(\exists r.x)$ will then be the expected one.

"Disjunction" and "negation" are then added as expected, i.e., $\sqcup : \mathsf{P}(\texttt{concept}) \times \mathsf{P}(\texttt{concept}) \rightarrow \mathsf{P}(\texttt{concept})$ and $\neg : \mathsf{P}(\texttt{concept}) \rightarrow \mathsf{P}(\texttt{concept})$, with the obvious algebras, and "universal quantification" can be added accordingly.

It is now explicit how P as a type constructor at the Σ-superceding level becomes the main building block for the "monadic modality operator" $\exists r$. With the semantics of P intuitively being the powerset functor (on the Σ-superceding level), we are more traditional, but nothing stops us from considering other set functors, or other endofunctors over more elaborate categories than just `Set`.

Modeling uncertainty in this context provides an excellent example. The simply typed framework opens up many possibilities to define *fuzzy description logic*. In [66,67] fuzzy DL is basically simply typed DL with the semantics of P intuitively being extended to the many-valued powerset monad. It is really not "fuzzy logic for description" but rather something like "fuzzy description logic". A more canonic way to invoke uncertainty modeling is either to compose the many-valued powerset monad with the term monad $\mathbf{T}_{\Sigma'}$ or to allow the term functor $\mathsf{T}_{\Sigma'}$ to, e.g., go over $\mathtt{Set}(\mathfrak{Q})$, where $\mathfrak{Q}$ is a quantale.

We have shown how simply typed description logic, on one hand, reveals the modal nature of DL more clearly, and, on the other hand, shows that the justification of speaking about the "existential quantifier" in DL is somewhat doubtful. A more precise meaning of DL being a "sublanguage of first-order logic" and expressions being "variable-free" is now

also given in this typing framework. Signatures are very explicit and therefore syntax and semantics are kept apart more rigorously.

In these notations it is, given Example 7.1, also obvious how syntactic aspects of description logic can be extended to involve more elaborate relations than just the one represented by powerset functors. Indeed, double powerset functors, both covariant and contravariant can be used, as well as set functors for filters and ideals. Many other functors, e.g., representing relations in a generalized and broad sense can be used in this context, and their extension over various underlying categories can be investigated.

8. Some notes on foundations and first-order logic

In this paper we adopt the view that set theory is the foundation for category theory, and not vice versa like in efforts to show how category theory can be a foundation for mathematics [68]. The set-theoretic foundation for category theory (see e.g., [69–71]) is mostly based on the Zermelo–Fraenkel set theory including the Axiom of Choice (ZFC), or the von Neumann–Bernays–Gödel (NBG) set theory. We lean a bit more on the setup for ZFC, but this is of no major importance for the main purpose of this paper.

Furthermore, category theory goes beyond using just sets and proper classes e.g., when accepting the use of the set-theoretic universe. Such universes are clearly not unique as they can be axiomatized in various ways. So-called *Grothendieck universes* were axiomatized within the Bourbaki group [25]. Any notion of a universe inevitably leads to considerations for subsets of that universe, i.e., the powerset of the universe, thus leading to notions of conglomerates and even higher-level conglomerates, thereby extending ZFC. The need for using conglomerates appears e.g., in formulation of categories of categories, functor categories, and when using functor power series [26].

In this context we also need some clarifications concerning the distinction between meta languages and object languages, in particular in discussions on first-order logic. It is important to note that the first-order logic which is seen as a basis for foundations of mathematics does not have a formal metalanguage, but is a *fons et origo* formal language where e.g., formal set theory does not exist at start. Therefore signatures and terms do not appear formally at this level, nor do sentences. Hilbert often pointed out that first-order logic and set theory was developed in parallel, and paradoxes were removed one by one as the theories evolved. Being at the end-point of ZFC developments, first-order logic has matured to be the formal logic language of set theory, and set theory strengthens the formalism of first-order theories. The boarder between object language and metalanguage was not always so clear throughout the course of these developments that had as starting point Frege's *Begriffsschrift* [72] and was culminating in Hilbert's and Bernays' *Grundlagen* [73,74]. Even worse, 'provability of a sentence' was by Gödel used again as a sentence like any other sentence, so that everything about logical constructions related to 'sentences' basically come 'into one bag'. Some developments in modern type theory are also less careful in these respects.

For many-valued logic and non-classical logic extensions these observations and standpoints are then very important, namely, the question about viewing first-order logic as the foundation of mathematics or as a formalism constructed in a clear interaction between a metalanguage and a object language. There is a fundamental question about where uncertainty in the end resides, and does it reside only in aspects of truth and truth values [75], or also more deeply e.g., within the signature and its operators, i.e., with terms as we strongly propose in this paper? Is uncertainty defined and modelled in the object language, where the metalanguage is assumed to be crisp, or do we need to integrate uncertainty modelling into the metalanguage? Our standpoint is that both are needed, namely monad compositions representing uncertainty as modelled within the object language, and underlying categories capturing uncertainty aspects residing in the metalanguage. In the end, application developments decide which type of uncertainty modelling is most appropriate. Needless to say, all these aspect turn up in one form or another also in discussions on quantifiers and modifiers, and in efforts to study the interaction between natural languages and formal languages.

9. Conclusions and future work

Our formal production of fuzzy terms is a subgoal of an overall ambition to describe a fuzzy logic framework more generally as was done in [76] concerning institutions and in [77] concerning general logic. The underlying view is that logics communicate and logic is individually adopted. Argumentation therefore happens in dialogue rather than in a search for universal or canonic truth. First steps were presented in [10], and work is on-going concerning developments of other building blocks of logic based on functors and monads.

In these respects, it is important to define what is allowed and not allowed in logic constructions. Our view is the following. A logic has its signature with sorts (types) and operators, and algebras providing the meaning of the signature. Terms are constructed (syntactically and functorially) using operators in the signature, and the resulting algebraic interpretations (semantics) of these terms can be provided accordingly. Substitutions and assignments have to be handled with care, as we have shown in this paper. Sentences have terms as ingredients, and also need to be equipped with corresponding meaning. Conglomerates of sentences can be formally treated. The 'set' of sentences is like the 'set' of terms, once it has been fixed, we *cannot go back* and throw in new sentences even if we find some method to define what appears to be new sentences. Entailment is the relation between these conglomerates representing what we already know, and sentences representing knowledge we are trying to arrive at. Satisfaction as the semantic counterpart to entailment provides the notion of valid conclusions. Axioms saying what we take for granted at start, and inference rules say how we can jump to conclusions in a chain of entailments. 'Soundness and completeness' between the entailment and satisfaction relations is desirable but sometimes difficult to reach. We take here is that if S is a sentence, $\vdash$ is entailment, then 'provability of S', expressed as $\vdash S$, is not a sentence. By logic we then make a distinction between logic as in 'logic as foundation for mathematics' and logic as in 'mathematics as foundation for logic'. Our future work on logic is then focused on 'category theory as foundations for generalized general logic', and this generalized general logic we call *substitution logic*, as a continuation of developments as done in [10].

Real applications involving all the subtleties for uncertainty management are important and we find health and social care as an application area that has always been aware of needs to be more precise about terminologies. In recent years, formal logic is being recognized as valuable for providing enrichments to ontology. Processes and workflow have also been in focus over decades, but languages for processes have basically not been of much concern. This is seen among several national authorities for social and health care, and in their on-going efforts to provide various canonic and generic process views.

Acknowledgments

The authors are grateful to the anonymous reviewer that significantly has improved the generality of our signatures and term constructions, drawing our attention to the use of monoidal biclosed categories.

References

[1] G. Frege, Grundgesetze der Arithmetik, vol. I, Verlag Hermann Pohle, Jena, 1893.
[2] G. Frege, Grundgesetze der Arithmetik, vol. II, Verlag Hermann Pohle, Jena, 1903.
[3] G. Peano, Arithmetices Principia, Novo Methodo Exposita, Fratres Bocca, Turin, 1889.
[4] J. Loeckx, H.-D. Ehrich, M. Wolf, Specification of Abstract Data Types, Wiley-Teubner, 1996.
[5] E.G. Manes, Algebraic Theories, Graduate Texts in Mathematics, vol. 26, Springer-Verlag, 1976.
[6] W. Gähler, Monads and convergence, in: Generalized Functions, Convergences Structures, and Their Applications, Plenum Press, New York, 1988, pp. 29–46.
[7] P. Eklund, M.A. Galán, M. Ojeda-Aciego, A. Valverde, Set functors and generalised terms, in: Proceedings of the IPMU 2000, 8th Information Processing and Management of Uncertainty in Knowledge-Based Systems Conference III, 2000, pp. 1595–1599.
[8] P. Eklund, M. Galán, J. Medina, M. Ojeda-Aciego, A. Valverde, Powersets of terms and composite monads, Fuzzy Sets and Systems 158 (23) (2007) 2552–2574.
[9] J. Goguen, L-fuzzy sets, J. Math. Anal. Appl. 18 (1967) 145–174.
[10] P. Eklund, R. Helgesson, Monadic extensions of institutions, Fuzzy Sets and Systems 161 (18) (2010) 2354–2368 (A tribute to Ulrich Höhle on the occasion of his 65th birthday).
[11] G. Grätzer, Universal Algebra, Van Nostrand, 1968.
[12] S. Lang, Algebra, Graduate Texts in Mathematics, vol. 211, revised third edition, Springer, 2002.
[13] G. Birkhoff, Lattice Theory, vol. 25, 3rd edition, American Mathematical Society, 1995.
[14] B. Davey, H. Priestley, Introduction to Lattices and Order, 2nd edition, Cambridge University Press, 2002.
[15] C. Mulvey, Rend. Circ. Mat. Palermo 12 (1986) 99–104.
[16] J.H. Conway, Regular Algebra and Finite Machines, William Clowes and Sons, 1971.
[17] U. Höhle, T. Kubiak, Non-commutative and non-idempotent theory of quantale sets, Fuzzy Sets and Systems 166 (2011) 1–43.
[18] S. Kleene, Representation of events in nerve nets and finite automata, in: C. Shannon, J. Mccarthy (Eds.), Automata Studies, Princeton University Press, Princeton, NJ, 1956, pp. 3–42.
[19] J. Brzozowski, Canonical regular expressions and minimal state graphs for definite events, in: Proceedings of Symposium on Mathematical Theory of Automata, Polytechnic Press of the Polytechnic Institute of Brooklyn, 1963, pp. 529–561.
[20] A. Salomaa, Two complete axiom systems for the algebra of regular events, J. ACM 13 (1) (1966) 158–169.

[21] D. Kozen, Kleene algebra with tests, ACM Trans. Program. Lang. Syst. 19 (3) (1997) 427–443.
[22] A. Tarski, On the calculus of relations, J. Symb. Log. 6 (3) (1941) 73–89.
[23] K. Ng, A. Tarski, Relation algebras with transitive closure, in: Notices of American Mathematical Society, vol. 24, 1977, pp. A29–A30, abstract 742-02-09.
[24] L.N. Stout, The logic of unbalanced subobjects in a category with two closed structures, in: S. Rodabaugh, E. Klement, U. Höhle (Eds.), Applications of Category Theory to Fuzzy Subsets, Kluwer Academic Publishers, 1992, pp. 73–106.
[25] N. Bourbaki, Séminaire de Géométrie Algébrique du Bois Marie - 1963–64 - Théorie des topos et cohomologie étale des schémas - (SGA 4) - vol. 1, Lecture Notes in Mathematics, vol. 269, Springer-Verlag, 1972. http://dx.doi.org/10.1007/BFb0081551.
[26] W. Gähler, Axioms of structures and functor power series, in: J. Novák, W. Gähler, H. Herrlich, J. Mikisiński (Eds.), Proceedings of the Conference on Convergence Structures, Bechyně, Akademie-Verlag, Berlin, 1984, pp. 138–152.
[27] R. Godement, Topologie algébrique et théorie des faisceaux, Hermann, Paris, 1958.
[28] P. Huber, Homotopy theory in general categories, Math. Ann. 144 (5) (1961) 361–385.
[29] F.W. Lawvere, Functorial semantics of algebraic theories, Proc. Natl. Acad. Sci. USA 50 (5) (1963) 869–872.
[30] S. Eilenberg, J. Moore, Adjoint functors and triples, Illinois J. Math. 9 (3) (1965) 381–398.
[31] H. Kleisli, Every standard construction is induced by a pair of adjoint functors, Proc. Amer. Math. Soc. 16 (3) (1965) 544–546.
[32] J. Goguen, Information integration in institutions, in: L. Moss (Ed.), Thinking Logically: a Memorial Volume for Jon Barwise, Indiana University Press, pp. 1–48.
[33] S. Eilenberg, S. MacLane, General theory of natural equivalences, Trans. Amer. Math. Soc. 58 (2) (1945) 231–294.
[34] S. Banach, Théorie des opérations linéares, Warsaw, 1932.
[35] G. Birkhoff, Generalized arithmetic, Duke Math. J. 9 (2) (1942) 283–302.
[36] D.M. Kan, Adjoint functors, Trans. Amer. Math. Soc. 87 (2) (1958) 294–329.
[37] S. MacLane, Natural associativity and commutativity, Rice Univ. Stud. 49 (1963) 28–46.
[38] G.M. Kelly, On Mac Lane's conditions for coherence of natural associativities, commutativities, etc., J. Algebra 1 (1964) 397–402.
[39] J. Bénabou, Catégories avec multiplication, C. R. Acad. Sci. Paris 256 (1963) 1887–1890.
[40] J. Bénabou, Algébre élémentaire dans les catégories avec multiplication, C. R. Acad. Sci. Paris 258 (1964) 771–774.
[41] S. MacLane, Categorical algebra, Bull. Amer. Math. Soc. 71 (1) (1965) 40–106.
[42] S. Eilenberg, G.M. Kelly, Closed categories, in: S. Eilenberg, D.K. Harrison, S. MacLane, H. Röhrl (Eds.), Proceedings of the Conference on Categorical Algebra, La Jolla 1965, Springer-Verlag, 1966, pp. 421–562.
[43] S. MacLane, Categories for the Working Mathematician, Springer-Verlag, 1971.
[44] G.M. Kelly, Basic Concepts of Enriched Category Theory, London Mathematical Society Lecture Notes Series, vol. 64, Cambridge University Press, 1982.
[45] U. Höhle, L.N. Stout, Foundations of fuzzy sets, Fuzzy Sets and Systems 40 (2) (1991) 257–296.
[46] J. Bénabou, Structures algébriques dans les catégories, Cahiers Topologie Gèom. Différentielle Catég. 10 (1) (1968) 1–126.
[47] P. Eklund, W. Gähler, Fuzzy filter functors and convergence, in: S.E. Rodabaugh, E.P. Klement, U. Höhle (Eds.), Applications of Category Theory to Fuzzy Subsets, Kluwer Academic Publishers, 1992, pp. 109–136.
[48] H. Schubert, Categories, Springer-Verlag, 1972.
[49] P. Eklund, J. Kortelainen, L.N. Stout, Introducing fuzziness in monads: cases of the term monad and the powerobject monad, in: Proceedings of the LINZ2009, 30th Linz Seminar on Fuzzy Set Theory, Linz, Austria, 2009.
[50] P. Eklund, M. Galán, R. Helgesson, J. Kortelainen, Paradigms for many-sorted non-classical substitutions, in: 41st IEEE International Symposium on Multiple-Valued Logic (ISMVL), 2011, pp. 318–321.
[51] P. Eklund, J. Kortelainen, L.N. Stout, Adding fuzziness to terms and powerobjects using a monadic approach, Fuzzy Sets and Systems 192 (2012) 104–122.
[52] J.A. Goguen, J. Meseguer, Order-sorted algebra I: equational deduction for multiple inheritance, exceptions and partial operations, Theor. Comput. Sci. 105 (2) (1992) 217–273.
[53] P. Eklund, Signatures for assessment, in: E. Hüllermeier, R. Kruse, F. Hoffmann (Eds.), IPMU (2), Communications in Computer and Information Science, vol. 81, Springer, 2010, pp. 271–279.
[54] A. Church, A formulation of the simple theory of types, Journal of Symbolic Logic 5 (2) (1940) 56–68.
[55] V. Novák, On fuzzy type theory, Fuzzy Sets and Systems 149 (2005) 235–273.
[56] C.I. Lewis, A Survey of Symbolic Logic, University of California Press, Berkeley, CA, 1918.
[57] S. Kripke, Semantical analysis of modal logic in normal modal propositional calculi, Math. Logic Quart. 9 (5–6) (1963) 67–96.
[58] V. Pratt, Semantical consideration on Floyd–Hoare logic, in: Proceedings of the 17th Annual IEEE Symposium on Foundations of Computer Science, IEEE, 1976, pp. 109–121.
[59] M.R. Quillian, Word concepts: a theory and simulation of some basic semantic capabilities, Behavioral Sci. 12 (5) (1967) 410–430.
[60] M.R. Quillian, Semantic Memory, Ph.D. Thesis, Carnegie Institute of Technology, Pittsburgh, Pennsylvania, February 1966.
[61] J. Piaget, The Psychology of Intelligence, International Library of Psychology, Philosophy, and Scientific Method, Routledge & Paul, 1950.
[62] M. Minsky, A Framework for Representing Knowledge, Technical Report, Cambridge, MA, USA, 1974.
[63] M. Schmidt-Schauß, G. Smolka, Attributive concept descriptions with complements, Artif. Intell. 48 (1) (1991) 1–26.
[64] R.J. Brachman, H.J. Levesque, The tractability of subsumption in frame-based description languages, in: R.J. Brachman (Ed.), AAAI, AAAI Press, 1984, pp. 34–37.
[65] R.J. Brachman, J.G. Schmolze, An overview of the KL-ONE knowledge representation system, Cogn. Sci. 9 (2) (1985) 171–216.
[66] J. Yen, Generalizing term subsumption languages to fuzzy logic, in: IJCAI, 1991, pp. 472–477.
[67] U. Straccia, A fuzzy description logic, in: J. Mostow, C. Rich (Eds.), AAAI/IAAI, AAAI Press, The MIT Press, 1998, pp. 594–599.

[68] F.W. Lawvere, The category of categories as a foundation for mathematics, in: Proceedings of the Conference on Categorical Algebra, La Jolla, CA, 1965, Springer, New York, 1966, pp. 1–20.
[69] S. Mac Lane, Categories for the Working Mathematician, Graduate Texts in Mathematics, Springer, 1998.
[70] H. Herrlich, G. Strecker, Category Theory: An Introduction, Sigma Series in Pure Mathematics, Heldermann, 1979.
[71] J. Adámek, H. Herrlich, G. Strecker, Abstract and Concrete Categories, Wiley-Interscience, New York, NY, USA, 1990.
[72] G. Frege, Begriffsschrift, eine der Arithmetischen Nachgebildete Formelsprache des Reinen Denkens, Halle, 1879. English translation in From Frege to Gödel, a Source Book, in: J. van Heijenoort (Ed.), Mathematical Logic, Harvard University Press, Cambridge, 1967, pp. 1–82.
[73] D. Hilbert, P. Bernays, Grundlagen der Mathematik. I, Die Grundlehren der mathematischen Wissenschaften, vol. 40, Springer-Verlag, 1934.
[74] D. Hilbert, P. Bernays, Grundlagen der Mathematik. II, Die Grundlehren der mathematischen Wissenschaften, vol. 50, Springer-Verlag, 1939.
[75] P. Hájek, The Metamathematics of Fuzzy Logic, Kluwer, 1998.
[76] J.A. Goguen, R.M. Burstall, Institutions: abstract model theory for specification and programming, J. ACM 39 (1) (1992) 95–146.
[77] J. Meseguer, General logics, in: H.-D. Ebbinghaus (Ed.), Logic Colloquium '87, North-Holland, Granada, Spain, 1989, pp. 275–329.