Towards Categorical Fuzzy Logic Programming

Patrik Eklund[1], M.Ángeles Galán[2], Robert Helgesson[1], Jari Kortelainen[3], Ginés Moreno[4], Carlos Vázquez[4]

[1] Department of Computing Science
Umeå University, SE-90187 Umeå, Sweden
peklund@cs.umu.se, rah@cs.umu.se
[2] Department of Applied Mathematics
University of Málaga, E-29071, Málaga, Spain
magalan@ctima.uma.es
[3] Department of Electrical Engineering and Information Technology
Mikkeli University of Applied Sciences, FIN-50100, Mikkeli, Finland
jari.kortelainen@mamk.fi
[4] Department of Computing Systems**
University of Castilla-La Mancha, E-02071, Albacete, Spain
gines.moreno@uclm.es, carlos.vazquez@alu.uclm.es

Abstract. In this paper we investigate the shift from two-valued to many-valued logic programming, including extensions involving functorial and monadic constructions for sentences building upon terms. We will show that assigning uncertainty is far from trivial, and the place where uncertainty should be used is also not always clear. There are a number of options, including the use of composed monads, and replacing the underlying category for monads with categories capturing uncertainty in a more canonic way. This is indeed important concerning terms and sentences, as classic logic programming, and also predicate logic for that matter, is not all that clear about the distinctive characters of terms and sentences. Classically, they are sets, and in our approach they are categorical objects subject to being transformed e.g. by transformations between functors. Naive set-theoretic approaches, when dealing e.g. with 'sets of sentences' and 'sets of ground atoms', may easily lead to confusion and undesirable constructions if generalizations are performed only as a shift from 'set' to 'fuzzy set'. We present some basic results on how adaptation of a strictly categorical framework enables us to be very precise about the distinction between terms and sentences, where predicates symbols become part of a signature which is kept apart from the signature for terms. Implication will not be included in signatures, but appears integrated into our sentence functors. Doing so we are able to relate propositional logic to predicate logic in a more natural way. Integration of uncertainty then becomes much more transparent.

Keywords: Fuzzy logic programming, fuzzy terms, fuzzy sentences

** C. Vázquez and G. Moreno received grants for International mobility from the University of Castilla-La Mancha (CYTEMA project, Vicerrectorado de Profesorado).

1 Introduction

Intuitively speaking, terms are produced by signatures such that variables and constants are terms, and if $t_1, \ldots, t_n$ are terms then also $\omega(t_1, \ldots, t_n)$ is a term, where ω resides in the given signature. Categorically, it is well known that terms may be produced by functors that are extendible to monads (see e.g. [7, 14]), whereas sentences are produced by functors. Indeed, variables may be substituted by terms, but sentence variables are dubious. For example, we may have terms $P(x)$ and $Q(y)$, where P and Q are predicate symbols residing in the signature, with x and y as variables. We might now produce a sentence in some abstract form like as a pair $(P(x), Q(x))$, intuitively then corresponding to "$P(x)$ is inferred by $Q(y)$" to check whether that sentence is valid or not. Here, the 'pairing operation' is not given in the underlying signature, but clearly appears within a sentence constructor. This indeed reveals that substitution with sentences makes no sense. The distinction between monads for representing terms and functors only to represent sentences makes the situation concerning substitution very transparent.

The overall scope of logic in this paper is that of generalized general logic [14], extending the frameworks of institutions [13] and general logic [24]. Morphisms between logics play an important role, and such morphisms are built up of morphisms respectively between underlying signatures, terms, sentences, and so on, all the way through all building blocks of logic. This means that the 'set of terms' and 'set of sentences' cannot be simple sets, so that we would have straightforward mappings between them. Categorically, they are based on functors and monads, which provides a richer algebraic structure and constraints morphisms between logics in a more canonic way. In logic programming, informal production of sets of terms and well-formed formulas in fact leads to confusion concerning the borderline between terms and sentences. In [21], notation and concepts mention 'signature', 'functions' (operators of formal universal algebra based signatures), and 'predicates'. In this conventional view, predicates are typically seen as different from operators in some underlying signature, and such treatments are also 'unsorted', or in fact one-sorted concerning the underlying set of terms.

In first-order logic based logic programming we are immediately confronted with the issue of the underlying signature. Informal treatments of first-order logic are not always clear about predicates being operator symbols or simply relations or functions, the latter confusing semantics with syntax in a way where the 'semantic jacket' acts as a 'dress code' for syntactic treatment of predicate symbols. Even more confusing is the adoption in [21] to say that operator symbols in signatures are 'function symbols', and the Boolean like operators representing predicates are called a 'predicate symbols'. Indeed, in [21], first-order logic is called a 'first order theory' consisting of alphabet, language, and so on, but these notion are not in harmony with the necessity to keep terms apart from sentences. Such verbose notations and names as used in [21] are not clear about the distinction between terms and sentences, and an alphabet is simply assumed to contain both operator as a well as predicate symbols. This means that terms

and sentences are lumped together within that informal language description. The use of 'function' in this context is obviously misleading as ω is only a syntactic symbol, but its semantic counterpart is a function (in the sense of ZFC), which in [21] is called a 'mapping'.

In our categorical approach, 'alphabet' is the underlying signature of sorts and operators, and we are always many-sorted. In [21] the treatment is basically one-sorted, and operators are called 'constants' and 'functions'. The confusion concerning terms and sentences also leads to technicalities involving interpretations and models. The classical treatment of models is using sets rather than algebras, which in turns invites or even enforces [21] to say that "the identification of a Herbrand interpretation as a subset of the Herbrand base will be made throughout". Strictly speaking, we do not have subsets in this case.

In this paper we will have predicates as operators, so atoms are terms, but program clauses become sentences. Basically this means that conjunction of predicates are still terms, but clauses involving implication is not a term, since implication is not included as an operator in the underlying signature. We will categorically aim at being precise so that notions like 'ground terms', 'well-formed formulas' in predicate logic, 'predicates' or 'predicate symbols', and 'atoms' can be explained more strictly in the categorical machinery.

Preliminary notions used in this paper appear in a working version [10], and the categorical framework of our monad constructions appear in [7].

These monads make use of constructions in *categorical algebra* more broadly, which goes back to the study of natural equivalences [5]. Monoidal closed categories emerges more or less in [4], and attained its simple and clean formulation in [22]. This is the categorical realm of this paper, and the categorical notation adopted in this paper is the same as in [7].

2 Traditional extensions from two-valued to many-valued logic programming

In traditional two-valued logic, and once negation is given, implication and conjunction are defined by one another. In the intuitionistic tradition, negation as a basic building block is avoided, and then implication and conjunction needs to be otherwise related, and this is done by the residuation property, which categorically is an adjoint situation, given as a Galois connection. This enables to define negation, if negation is desired. Residuated lattices have been extensively studied and are appealing algebraic structures for semantics of logic, and indeed because of the tight bound between implication and conjuction.

In many-valued logic, this 'semantic jacket' has been adopted in several approaches. E.g. in [2], this residuated situation appears in what is called 'implication algebras', and later on, e.g. in [23], where the name 'multi-adjoint' is used in this context. 'Multi-adjointness' in logic programming then refers to the use of residuated lattices that provides the desired semantic jacket that prescribes the behaviour of the truth values.

All this is, from the viewpoint of that semantic jacket, basically an extension of two-valued logic to many-valued logic using algebras of truth values. The underlying language involving its terms and sentences remain traditional. It also follows the tradition of extending propositional calculus to predicate calculus, where the implication operator receives a similar semantic. Traditional predicate logic is set-theoretic, not functorial, about its 'set of sentences'. Many-valued logic programming has followed that same tradition, and then the semantics, which restricts to management of truth values, is adopted using this semantic jacket provided by that residuated situation. The acronym 'MALP' for multi-adjoint logic programming has then been quite widely adopted, and as an acronym is seen as a specific version of fuzzy logic programming. 'Adjointness' refers to the residuated situation, and 'multi' to the allowance of using particular lattices for each separate logic program. Much of this work basically keeps the classical constructions of logic programming as they are e.g. in [21], and many-valued extensions indeed focus on the many-valued extensions of the truth values. It does deal with uncertainty issues, but restricted to consequences of algebraic manipulation of truth values. It is also seen to represent 'approximate reasoning', which it certainly does, but as restricted to that focus on truth values, leaving all the other bits and pieces of logic as they are in a two-valued setting.

As mentioned before, our scope is logic as categorical object, that is constructed functorially and monadically, with morphisms between respective substructures in logic. Thus we do not propose to have a 'universal logic', and further, logic programs in our setting is the axiom system in a particular logic. This means e.g. that logic programs can have different inference rules, and morphisms between logic programs makes no sense unless we would have morphisms between their underlying logics, which in turn include appropriate transformations between their respective inference systems.

Resolution in these approaches eventually enters the scene, and theory developments are confronted e.g. with fixpoint issues and inference rules. This then is mostly ad hoc as typically seen e.g. in [1, 3, 15, 19, 20, 25, 28, 34]. Essentially, they differ in the underlying notion of uncertainty theory and vagueness theory (probability theory, possibilistic logic, fuzzy logic and multi-valued logic) and how uncertainty/vagueness values, associated to rules, are managed. Annotated logic programming [17] also falls within adoption of such jackets.

Fixpoint considerations [33] are interesting in these settings, even if it cannot be expected that the relation between fixpoints and least Herbrand models appears as it is in a many-valued extension. Nevertheless, analyzing this fixpoint situation [30] reveals some crucial underlying structures that are important to consider in dealing with soundness and completeness issues. Operational and fix-point semantics are provided also in [23], and these considerations has been extended with a declarative semantics based on model-theory [16].

Further, there are a number of independently developed more general language based approaches to fuzzy logic programming, where there are less considerations involving first-order aspects, and more papers covering truth value

considerations only. For the first-order aspects see [18, 27] for some historically important contributions. Categories for logic programming enters the scene in [29] with co-equalizers seen to correspond precisely to most general unifiers. A word of caution, however, is that co-equalizers as such do not suffice as categorical constructions when we move over to the many-valued setting. This then affects the resolution principle as an algorithm that has been subject of fuzzification e.g. in [1, 19, 23]. This mostly focuses on truth values more than extending the underlying language. Similarly for fixpoint considerations, interpretations are considered mostly as points in sets, and uncertainties are added [34]. The fixpoint semantics framework has been enriched with a declarative semantics based on model-theory as described in [16].

3 Signatures, terms and sentences

Throughout this paper we assume the readers are familiar with categorical concepts. However, this section starts with introducing some categorical concepts needed in the paper. Then, signatures and term monads are recalled and we introduce sentences in a logic programming context. Finally, we show how fixpoints can be considered.

3.1 Some Categorical Concepts and Notations

Let $\mathtt{C}$ be a category and S a set of sorts. Then, $\mathtt{C}_S$ is a category with objects $X_S = (X_\mathtt{s})_{\mathtt{s}\in S}$ where each $X_\mathtt{s} \in \mathrm{Ob}(\mathtt{C})$. The morphisms between X_S and Y_S are $f_S \colon X_S \to Y_S$ where $f_S = (f_\mathtt{s})_{\mathtt{s}\in S}$ and each $f_\mathtt{s} \in \hom_\mathtt{C}(X_\mathtt{s}, Y_\mathtt{s})$. The composition of morphisms is defined sortwise, thus, $f_S \circ g_S = (f_\mathtt{s} \circ g_\mathtt{s})_{\mathtt{s}\in S}$.

We may sometimes need to pick an object $X_\mathtt{s}$ in $\mathrm{Ob}(\mathtt{C})$ when X_S is given in a form or another. For this purpose we define a functor $\mathsf{arg}^\mathtt{s} \colon \mathtt{C}_S \to \mathtt{C}$ such that $\mathsf{arg}^\mathtt{s} X_S = X_\mathtt{s}$ and $\mathsf{arg}^\mathtt{s} f_S = f_\mathtt{s}$. Especially, when working in $\mathtt{Set}_S$, $\mathrm{card}(S) > 1$, we may define two emptifying functors: $\phi^{S\setminus\mathtt{s}} : \mathtt{Set}_S \to \mathtt{Set}_S$ such that $\phi^{S\setminus\mathtt{s}} X_S = X'_S$, where for all $\mathtt{t} \in S\setminus\{\mathtt{s}\}$ we have $X'_\mathtt{t} = X_\mathtt{t}$, and $X'_\mathtt{s} = \varnothing$. Similarly we define the functor $\phi^\mathtt{s} : \mathtt{Set}_S \to \mathtt{Set}_S$ as $\phi^\mathtt{s} X_S = X'_S$, where for all $\mathtt{t} \in S\setminus\{\mathtt{s}\}$ we have $X'_\mathtt{t} = \varnothing$, and $X'_\mathtt{s} = X_\mathtt{s}$. Actions on morphisms are defined in obvious way.

Clearly, a functor $\mathsf{F} \colon \mathtt{C} \to \mathtt{D}$ may be extended to a functor $\mathsf{F}_S = (\mathsf{F})_{\mathtt{s}\in S} \colon \mathtt{C}_S \to \mathtt{D}_S$ (for all $\mathtt{s} \in S$, the functor remains the same). For example, the powerset functor $\mathsf{P} \colon \mathtt{Set} \to \mathtt{Set}$ as well as the many-valued powerset functor $\mathsf{L} \colon \mathtt{Set} \to \mathtt{Set}$ both determine functors on $\mathtt{Set}_S$, we write $\mathsf{P}_S = (\mathsf{P})_{\mathtt{s}\in S}$ and $\mathsf{L}_S = (\mathsf{L})_{\mathtt{s}\in S}$. Also functors $\mathsf{G}_\mathtt{s} \colon \mathtt{C}_S \to \mathtt{D}$, $\mathtt{s} \in S$, are of interest, because we can determine a functor $\mathsf{G} \colon \mathtt{C}_S \to \mathtt{D}_S$ such that for all $X_S \in \mathrm{Ob}(\mathtt{C}_S)$ we have $\mathsf{G}X_S = (\mathsf{G}_\mathtt{s} X_S)_{\mathtt{s}\in S}$. Notice that we have now $\mathsf{G}_\mathtt{s} = \mathsf{arg}^\mathtt{s} \circ \mathsf{G}$.

Now, assume any two functors $\mathsf{F}, \mathsf{G} \colon \mathtt{C} \to \mathtt{D}$. A natural transformation τ between F and G, denoted by $\tau \colon \mathsf{F} \to \mathsf{G}$, assigns for each $\mathtt{C}$-object X a $\mathtt{D}$-morphism $\tau_X \colon \mathsf{F}X \to \mathsf{G}X$ satisfying $\mathsf{G}f \circ \tau_X = \tau_Y \circ \mathsf{F}f$ for all $f \in \hom_\mathtt{C}(X, Y)$. Notice that $\mathtt{C}_S$ is also a category, thus we may have natural transformations, between functors on $\mathtt{C}_S$, for example.

Finally, we recall a monad $\mathbf{F}$ over a category $\mathtt{C}$, which is a triple (F, η, μ), where $\mathsf{F}\colon \mathtt{C} \to \mathtt{C}$ is a (covariant) functor, and $\eta\colon \mathrm{id} \to \mathsf{F}$ and $\mu\colon \mathsf{F} \circ \mathsf{F} \to \mathsf{F}$ 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.

3.2 Signatures and the Term Monad Construction

A many-sorted signature $\Sigma = (S, \Omega)$ consists of a set S of sorts (or types), and a set Ω of operators. Here S as an index set, whereas Ω may be an object in $\mathtt{Set}_S$. Operators in $\Omega_{\mathtt{s}}$ are written as $\omega : \mathtt{s}_1 \times \cdots \times \mathtt{s}_n \to \mathtt{s}$.

It is 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

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

On algebraic structures for truth values, we mostly prefer to use quantales as they play an important role when invoking the use of monoidal closed categories for the formal construction of signatures. Quantales fulfill the properties of residuated lattices, and complete residuated lattices are quantales. We further restrict to quantales $\mathfrak{Q}$ that are commutative and unital, as this makes the Goguen category[5] $\mathtt{Set}(\mathfrak{Q})$ to be a symmetric monoidal closed category and therefore also biclosed. This Goguen category carries all structure needed for modelling uncertainty using underlying categories for fuzzy terms over appropriate signatures, and as constructed by their term monads [7]. Note indeed that the signature, as a categorical object itself, also carries uncertainty, which is brought up partly to represent the overall uncertainties attached to fuzzy terms. Recall that $(A, \alpha) \otimes (B, \beta) = (A \times B, \alpha \odot \beta)$ provides the monoidal operation on objects in the Goguen category. If $\odot$ is the meet operator, then $\otimes$ is the categorical product.

A signature $(S, (\Omega, \alpha))$ over $\mathtt{Set}(\mathfrak{Q})$ then typically has S as a crisp set, and $\alpha : \Omega \to Q$ then 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

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

and this specializes, in the case of the Goguen category, to

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

[5] Objects in the Goguen category are pairs (A, α), where $\alpha : A \to Q$ is a mapping.

The inductive steps start with $\mathsf{T}^1_{\Sigma,\mathtt{s}} = \coprod_{\mathtt{m}\in\hat{\mathtt{S}}} \Psi_{\mathtt{m},\mathtt{s}}$, and, for $\iota > 1$, proceeds with $\mathsf{T}^\iota_{\Sigma,\mathtt{s}} X_{\mathtt{S}} = \coprod_{\mathtt{m}\in\hat{\mathtt{S}}} \Psi_{\mathtt{m},\mathtt{s}}(\mathsf{T}^{\iota-1}_{\Sigma,\mathtt{t}} X_{\mathtt{S}} \sqcup X_{\mathtt{t}})_{\mathtt{t}\in\mathtt{S}})$, 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 T^ι_Σ by $\mathsf{T}^\iota_\Sigma X_{\mathtt{S}} = (\mathsf{T}^\iota_{\Sigma,\mathtt{s}} X_{\mathtt{S}})_{\mathtt{s}\in\mathtt{S}}$, and $\mathsf{T}^\iota_\Sigma f_{\mathtt{S}} = (\mathsf{T}^\iota_{\Sigma,\mathtt{s}} f_{\mathtt{S}})_{\mathtt{s}\in\mathtt{S}}$. There is a natural transformation $\Xi^{\iota+1}_\iota : \mathsf{T}^\iota_\Sigma \to \mathsf{T}^{\iota+1}_\Sigma$ such that $(\mathsf{T}^\iota_\Sigma)_{\iota>0}$ is an inductive system of endofunctors with $\Xi^{\iota+1}_\iota$ as its connecting maps. The inductive limit $\mathsf{F} = \mathrm{ind}\varinjlim \mathsf{T}^\iota_\Sigma$ exists, and the final term functor T_Σ is $\mathsf{T}_\Sigma = \mathsf{F} \sqcup \mathsf{id}_{\mathtt{Set}_{\mathtt{S}}}$. We also have $\mathsf{T}_\Sigma X_{\mathtt{S}} = (\mathsf{T}_{\Sigma,\mathtt{s}} X_{\mathtt{S}})_{\mathtt{s}\in\mathtt{S}}$, and T_Σ is strictly not idempotent, but only "idempotent like", as there is a natural isomorphism between $\mathsf{T}_\Sigma \mathsf{T}_\Sigma$ and T_Σ.

For more detail concerning this term construction, see [7].

3.3 Sentences in a Logic Programming Context

Let then $\Sigma_0 = (S_0, \Omega_0)$ be a signature over $\mathtt{Set}$, and $\mathbf{T}_{\Sigma_0}$ be the term monad over $\mathtt{Set}_{S_0}$. For the variables in X_{S_0}, the set of terms $\mathsf{T}_{\Sigma_0} X_{S_0}$, as an object of $\mathtt{Set}_{S_0}$, then correspond to the 'terms', and similarly $\mathsf{T}_{\Sigma_0} \varnothing_{S_0}$ will be the set of so called 'ground terms' in the sense of [21].

In order to introduce predicates as operators in a separate signature, and then composing that resulting 'predicate' functor with the term functor, we assume that Σ contains a sort $\mathtt{bool}$, which does not appear in connection with any operator in Ω, i.e., we set $S = S_0 \cup \{\mathtt{bool}\}$, $\mathtt{bool} \notin S_0$, and $\Omega = \Omega_0$. This means that $\mathsf{T}_{\Sigma,\mathtt{bool}} X_S = X_{\mathtt{bool}}$, and for any substitution $\sigma_S : X_S \to \mathsf{T}_\Sigma X_S$, we have $\sigma_{\mathtt{bool}}(x) = x$ for all $x \in X_{\mathtt{bool}}$. The composition of the 'predicate' functor with T_Σ is intuitively expected to be the desired 'predicates as terms' functor.

We can now also separate propositional logic from predicate logic, and also decide whether or not to include negation. The key effect in doing this arrangement is that implication becomes 'sentential' where as conjunction (and negation, if included) produces terms from terms.

To proceed towards this goal, let $\Sigma_{PL} = (S_{PL}, \Omega_{PL})$ be the underlying *two-valued propositional logic* signature, where $S_{PL} = S$, and $\Omega_{PL} = \{\mathtt{F}, \mathtt{T} :\to \mathtt{bool}, \& : \mathtt{bool} \times \mathtt{bool} \to \mathtt{bool}, \neg : \mathtt{bool} \to \mathtt{bool}\} \cup \{\mathtt{P}_i : \mathtt{s}_{i_1} \times \cdots \times \mathtt{s}_{i_n} \to \mathtt{bool} \mid i \in I, \mathtt{s}_{i_j} \in S\}$. Similarly as $\mathtt{bool}$ leading to no additional terms, except for additional variables being terms when using Σ, the sorts in S_{PL}, other than $\mathtt{bool}$, will lead to no additional terms except variables. Adding predicates as operators even if they produce no terms seems superfluous at first sight, but the justification is seen when we compose these term functors with T_Σ.

In the many-valued case we would have some sort $\mathtt{lat}$, so that $\mathfrak{A}(\mathtt{lat}) = L$, the underlying set of a complete lattice $\mathfrak{L}$. Now, $\mathfrak{L}$ could indeed more specifically be a residuated lattice, when conjunction is desired to be residuated with the implication operator (in the lattice), or a quantale, justifying the use of monoidal closed subcategories. The choice of the lattice or quantale is typically justified by the application context.

It is important also to notice indeed that it is possible to include the both sorts $\mathtt{bool}$ and $\mathtt{lat}$ in the same signature, if one needs to distinguish the two-valued case from the many-valued case also on the syntactic level.

In the *many-valued propositional logic* signature $\Sigma^{mv}_{PL} = (S^{mv}_{PL}, \Omega^{mv}_{PL})$ constants clearly map algebraically to uncertainty values. In what follows we will not explicitly distinguish between Σ^{mv} and Σ, so whenever we write Σ, the underlying lattice representing the algebra may be two-valued or many-valued. We now introduce the notation $\Sigma_{PL\setminus\neg}$ for the signature where the operator $\neg$ is removed, and $\Sigma_{PL\setminus\neg,\&}$ for the signature where both $\neg$ and $\&$ are removed.

The ZFC-set of 'terms' over Σ may now be given by

$$\bigcup_{\mathtt{s}\in S}(\mathsf{T}_{\Sigma,\mathtt{s}} \circ \phi^{S\setminus\mathtt{bool}})X_S,$$

and now the ZFC-set of propositional logic formulas are

$$\bigcup_{\mathtt{s}\in S}(\mathsf{arg}^{\mathtt{s}} \circ \mathsf{T}_{\Sigma_{PL}} \circ \phi^{\mathtt{bool}})X_S = (\mathsf{arg}^{\mathtt{bool}} \circ \mathsf{T}_{\Sigma_{PL}} \circ \phi^{\mathtt{bool}})X_S.$$

We use the expression $\mathsf{arg}^{\mathtt{s}} \circ \mathsf{T}_{\Sigma_{PL}}$ instead of $\mathsf{T}_{\Sigma_{PL},\mathtt{s}}$ for convenience. Note how

$$(\mathsf{arg}^{\mathtt{bool}} \circ \mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \circ \phi^{\mathtt{bool}})X_S = \{\mathtt{F}, \mathtt{T}\}.$$

Sentences, i.e., formulas in propositional logic are now obviously given by the functor

$$\mathsf{Sen}_{PL} = \mathsf{arg}^{\mathtt{bool}} \circ \mathsf{T}_{\Sigma_{PL}} \circ \phi^{\mathtt{bool}},$$

and sentences in 'Horn clause logic' can now be given by the functor

$$\begin{aligned}\mathsf{Sen}_{HCL} &= (\mathsf{arg}^{\mathtt{bool}})^2 \circ (((\mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \circ \mathsf{T}_{\Sigma}) \times (\mathsf{T}_{\Sigma_{PL\setminus\neg}} \circ \mathsf{T}_{\Sigma})) \circ \phi^{S\setminus\mathtt{bool}}) \\ &= (\mathsf{arg}^{\mathtt{bool}})^2 \circ ((\mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \times \mathsf{T}_{\Sigma_{PL\setminus\neg}}) \circ \mathsf{T}_{\Sigma} \circ \phi^{S\setminus\mathtt{bool}})\end{aligned}$$

Note that $\mathsf{Sen}_{HCL}X_S$ is an object in $\mathtt{Set}$, and therefore the pair $(h, b) \in \mathsf{Sen}_{HCL}X_S$, as a sentence representing the 'Horn clause', means that h is an 'atom' and b is a conjunction of 'atoms'. Further, $(h, \mathtt{T})$ is a 'fact', $(\mathtt{F}, b)$ is a 'goal clause', and $(\mathtt{F}, \mathtt{T})$ is a 'failure'.

This obviously relates to similar approaches for using sentence functors in other logics. Intuitively, the identity functor is the sentence functor for lambda terms as 'sentences' in λ-calculus, and id^2 is the sentence functor for equations as 'sentences' in equational logic [9].

Before proceeding, now note a fundamental appearance of the residuated situation. The quantale, as a residuated lattice, uses the residuation at least for the underlying signature to work properly in the setting of monoidal biclosed categories, but is in no way at that point necessarily related to 'implication' as appearing in Horn clauses. In our treatment we therefore clearly show where and how residuation can be introduced. Indeed, residuation as possibly used for uncertainty consideration in terms has nothing to do with residuation related properties as possible used for uncertainty on sentence level.

We are now in a position to introduce *variable substitutions*. Indeed, because we have a monad $\mathbf{T}_{\Sigma} = (\mathsf{T}_{\Sigma}, \eta, \mu)$, we may now perform a variable substitution

$\sigma_S \colon \phi^{S\setminus \mathtt{bool}} X_S \to \mathsf{T}_\Sigma \phi^{S\setminus \mathtt{bool}} Y_S$, that is, variables $\phi^{S\setminus \mathtt{bool}} X_S$ are subsituted by terms $\mathsf{T}_\Sigma \phi^{S\setminus \mathtt{bool}} Y_S$. The substitution is defined sortwise $\sigma_S = (\sigma_{\mathtt{s}})_{\mathtt{s}\in S}$ such that $\sigma_{\mathtt{s}} \colon \mathsf{arg}^{\mathtt{s}}(\phi^{S\setminus \mathtt{bool}} X_S) \to \mathsf{T}_{\Sigma,\mathtt{s}} \phi^{S\setminus \mathtt{bool}} Y_S$. We have the following:

$$\mu \circ \mathsf{T}_\Sigma \sigma_S : \mathsf{T}_\Sigma \phi^{S\setminus \mathtt{bool}} X_S \to \mathsf{T}_\Sigma \phi^{S\setminus \mathtt{bool}} Y_S$$

$$\sigma_S^{head} = \mathsf{T}_{\Sigma_{PL\setminus\neg,\&}}(\mu \circ \mathsf{T}_\Sigma \sigma_S) : (\mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \circ \mathsf{T}_\Sigma)\phi^{S\setminus \mathtt{bool}} X_S \to (\mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \circ \mathsf{T}_\Sigma)\phi^{S\setminus \mathtt{bool}} Y_S$$

$$\sigma_S^{body} = \mathsf{T}_{\Sigma_{PL\setminus\neg}}(\mu \circ \mathsf{T}_\Sigma \sigma_S) : (\mathsf{T}_{\Sigma_{PL\setminus\neg}} \circ \mathsf{T}_\Sigma)\phi^{S\setminus \mathtt{bool}} X_S \to (\mathsf{T}_{\Sigma_{PL\setminus\neg}} \circ \mathsf{T}_\Sigma)\phi^{S\setminus \mathtt{bool}} Y_S$$

$$(\sigma_S^{head}, \sigma_S^{body}) = (\mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \times \mathsf{T}_{\Sigma_{PL\setminus\neg}})(\mu \circ \mathsf{T}_\Sigma \sigma_S) :$$
$$((\mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \times \mathsf{T}_{\Sigma_{PL\setminus\neg}}) \circ \mathsf{T}_\Sigma)\phi^{S\setminus \mathtt{bool}} X_S \to ((\mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \times \mathsf{T}_{\Sigma_{PL\setminus\neg}}) \circ \mathsf{T}_\Sigma)\phi^{S\setminus \mathtt{bool}} Y_S$$

Finally,

$$\sigma^{HC} = (\sigma_{\mathtt{bool}}^{head}, \sigma_{\mathtt{bool}}^{body}) : \mathsf{Sen}_{HCL} X_S \to \mathsf{Sen}_{HCL} Y_S$$

Notice that σ_S^{head}, σ_S^{body} and $(\sigma_S^{head}$, $\sigma_S^{body})$ are morphisms in $\mathtt{Set}_S$ but σ^{HC} is a morphism in $\mathtt{Set}$.

It is now clear that a candidate for the underlying category can be the Goguen category $\mathtt{Set}(\mathfrak{Q})$. Further, and as will be explored in subsequent papers, replacement of T_Σ with the composed functor $\mathsf{Q} \circ \mathsf{T}_\Sigma$ [12], provides another style of fuzzy extension.

3.4 Algebras, models and fixpoints

In the two-valued case, $\mathfrak{A}(\mathtt{bool})$ is often $\{false, true\}$, so that $\mathfrak{A}(\mathtt{F}) = false$ and $\mathfrak{A}(\mathtt{T}) = true$. Further, $\mathfrak{A}(\&) : \mathfrak{A}(\mathtt{bool}) \times \mathfrak{A}(\mathtt{bool}) \to \mathfrak{A}(\mathtt{bool})$, is expected to be defined by the usual 'truth table'. Further $\mathfrak{A}(\mathtt{s}_0)$ is usually denoted by D so that the semantics for a (syntactic) n-ary operator $\omega : \mathtt{s}_0 \times \cdots \times \mathtt{s}_0 \to \mathtt{s}_0$ is an n-ary operation (function) $\mathfrak{A}(\omega) : D^n \to D$. Generally speaking, a many-sorted algebra is not a traditional algebra, not even a tuple of traditional algebras, since an operator ω may touch many sorts and then the semantics of ω is not an n-ary function on some set. For example, we may assign for a signature $\Sigma_{PL} = (S_{PL}, \Omega_{PL})$ a pair, the 'many-sorted algebra', $(\mathsf{T}_{\Sigma_{PL}} X_S, (\mathfrak{A}(\omega))_{\omega\in\Omega_{PL}})$, where $X_{\mathtt{s}} = \varnothing$ if $\mathtt{s} \neq \mathtt{bool}$. Then, $(\bigcup_{\mathtt{s}\in S}(\mathsf{arg}^{\mathtt{s}} \circ \mathsf{T}_{\Sigma_{PL}})X_S, (\mathtt{F}, \mathtt{T}, \&, \neg))$ serves as a traditional Boolean algebra, when certain equational laws are given.

For a finite set of program clauses $\Gamma = \{(h_1, b_1), \ldots, (h_n, b_n)\} \subseteq \mathsf{Sen}_{HCL} X_S$, based on Σ and Σ_{PL}, we assign a $\mathtt{Set}_S$ object

$$(U_\Gamma)_S = \mathsf{T}_\Sigma \varnothing_S = (\mathsf{T}_{\Sigma,\mathtt{s}} \varnothing_S)_{\mathtt{s} \in S}$$

where $\mathsf{T}_{\Sigma,\mathtt{s}} \varnothing_S$ is the set of all ground terms of type $\mathtt{s}$, and indeed $\mathsf{T}_{\Sigma,\mathtt{bool}} \varnothing_S = \varnothing$. Note how $\bigcup_{\mathtt{s} \in S} (U_\Gamma)_\mathtt{s}$ corresponds to the traditional and unsorted view of the *Herbrand universe* as a ZFC-set.

We are also interested in the $\mathtt{Set}$-object

$$B_\Gamma = (\mathsf{arg}^{\mathtt{bool}} \circ \mathsf{T}_{\Sigma_{PL} \setminus \neg, \&} \circ \mathsf{T}_\Sigma)\ \varnothing_S$$

corresponding to the *Herbrand base* in the traditional sense [21].

Herbrand interpretations of a program Γ are subsets $\mathcal{I} \subseteq B_\Gamma$, that is, $\mathcal{I} \in \mathsf{P} B_\Gamma$.

For sake of convenience, when dealing with the immediate consequences operator for the fixpoint considerations, we will need the *Herbrand expression base*

$$B_\Gamma^\& = (\mathsf{arg}^{\mathtt{bool}} \circ \mathsf{T}_{\Sigma_{PL} \setminus \neg} \circ \mathsf{T}_\Sigma)\ \varnothing_S.$$

Note that a Herbrand interpretation $\mathcal{I}$ canonically extends to a *Herbrand expression interpretation* $\mathcal{I}^\& \subseteq B_\Gamma^\&$. Similarly, when $\mathcal{I} \in \mathsf{L} B_\Gamma$, one might extend $\mathcal{I}$ to *Herbrand fuzzy expression interpretation* $\mathcal{I}^\&$ (semantically) as follows: for an element $b \in B_\Gamma^\&$ of the form $b = b_1 \& \cdots \& b_n$ we have $\mathcal{I}^\&(b) = \bigwedge\{\mathcal{I}(b_1), \ldots, \mathcal{I}(b_n)\}$ and for an atom element $b \in B_\Gamma^\&$, $\mathcal{I}^\&(b) = \mathcal{I}(b)$. However, it is questionable to call $\mathcal{I} \in \mathsf{L} B_\Gamma$ to an interpretation.

Note that in this paper we avoid describing the informal passage [21] from 'interpretation' to 'Herbrand interpretation', which categorically means describing the shift from algebras to term algebras. The Herbrand interpretation is the 'ground term algebra' [21] in the universal algebra sense. This is the T_Σ-algebra, rather than the Σ-algebra which corresponds to 'interpretation', and in all case we are 'ground' in the sense of the variable sets in the tuples being empty sets.

The extension to the many-valued case is now a question about composing with the many-valued powerset functor L with term functors, producing a style of "logic with fuzzy" or having the term functors work over the Goguen category, producing a style of "fuzzy logic". It should therefore not be looked at simply from the viewpoint of replacing the functor P to L with $\mathfrak{L}$ as the underlying complete lattice, and extending the Herbrand interpretations to *Herbrand fuzzy interpretations* of a program Γ by $\mathcal{I} \in \mathsf{L} B_\Gamma$. We will here look more into the first situation, as the "squeezing in" of L can indeed be done in two ways. Either we annotate it "outside", as mentioned above, with sentences in such a 'annotated fuzzy Horn clause logic' can be given by the sentence functor $\mathsf{L} \circ \mathsf{Sen}_{HCL}$ and then proceed to produce interpretations for fuzzy sets of predicates

$$\mathsf{L} B_\Gamma = (\mathsf{L} \circ \mathsf{arg}^{\mathtt{bool}} \circ \mathsf{T}_{\Sigma_{PL} \setminus \neg, \&} \circ \mathsf{T}_\Sigma)\ \varnothing_S.$$

A fuzzy interpretation in this case is then just a mapping $\mathcal{I} : B_\Gamma \to L$, and uncertainties arising from terms and substitutions remain unaffected. On the

other hand, we may go "inside" to produce the *substitution fuzzy Horn clause logic* with the sentence functor

$$\mathsf{Sen}_{SFHCL} = (\mathsf{arg}^{\mathtt{bool}})^2 \circ ((\mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \times \mathsf{T}_{\Sigma_{PL\setminus\neg}}) \circ \mathsf{L}_S \circ \mathsf{T}_\Sigma \circ \phi^{S\setminus\mathtt{bool}})$$

so that ground predicates over fuzzy sets of terms is the set

$$B_\Gamma^{\mathsf{L}} = (\mathsf{arg}^{\mathtt{bool}} \circ \mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \circ \mathsf{L}_S \circ \mathsf{T}_\Sigma)\ \varnothing_S$$

with the corresponding extension $B_\Gamma^{\mathsf{L},\&}$ being defined in the obvious way. The resulting fuzzy sets of ground predicates then comes about from considering the swapper

$$\varsigma : \mathsf{T}_{\Sigma_{PL\setminus\neg,\&}} \circ \mathsf{L}_S \to \mathsf{L}_S \circ \mathsf{T}_{\Sigma_{PL\setminus\neg,\&}}$$

which is given in [6] for the many-sorted case, and in [11] for the one-sorted case. Indeed we can use $\mathsf{arg}^{\mathtt{bool}}\varsigma_{\mathsf{T}_\Sigma\varnothing_S} : B_\Gamma^{\mathsf{L}} \to \mathsf{L}B_\Gamma$. Note also how $\mathsf{L}B_\Gamma^{\mathsf{L}}$ would correspond to a Herbrand base like the set with uncertainty considerations both for the sets of clauses, as well as sets of terms.

Moving to fixpoints, we first consider crisp *ground term substitution*, that is, a $\mathtt{Set}_S$-morphism $\sigma_S\colon X_S \to \mathsf{T}_\Sigma\varnothing_S$. By the previous discussion, this induces a morphism $\sigma^{HC}\colon \mathsf{Sen}_{HCL}X_S \to \mathsf{Sen}_{HCL}\varnothing_S$. We can now define a mapping $\varpi : \mathsf{L}B_\Gamma \to \mathsf{L}B_\Gamma$, where the underlying lattice $\mathfrak{L}$ for the many-valued powerset functor L is a complete lattice, by

$$\varpi(\mathcal{I})(\sigma^{head}_{\mathtt{bool}}(h)) = \bigvee_{(h,b)\in\Gamma} \mathcal{I}^{\&}(\sigma^{body}_{\mathtt{bool}}(b)).$$

When $(h,b) \in B_\Gamma \times B_\Gamma$ is such that $(h,b) \notin R_{\sigma^{HC}}$ (the range of σ^{HC}), then $\varpi(\mathcal{I})(h) = \mathcal{I}(h)$ and $\varpi(\mathcal{I})(b) = \mathcal{I}(b)$.

Clearly, ϖ is monotonic, and it is now well-known that ϖ has the least and greatest fixpoints.

This, however, is a simpler approach to fuzzy models, as substitutions remain crisp. For fuzzy ground term substitution, that is, a $\mathtt{Set}_S$-morphism of the form $\sigma_S^{\mathsf{L}}\colon X_S \to \mathsf{L}_S\mathsf{T}_\Sigma\varnothing_S$, corresponding $\sigma_S^{\mathsf{L},head}$ and $\sigma_S^{\mathsf{L},body}$ mappings can be provided with L_S "inside".

A mapping $\varpi^{\mathsf{L}} : \mathsf{L}B_\Gamma^{\mathsf{L}} \to \mathsf{L}B_\Gamma^{\mathsf{L}}$, considering the effect of substitutions with fuzzy sets of terms, can now, using $\mathsf{arg}^{\mathtt{bool}}\varsigma_{\mathsf{T}_\Sigma\varnothing_S} : B_\Gamma^{\mathsf{L}} \to \mathsf{L}B_\Gamma$, be considered in various forms, e.g., as

$$\varpi^{\mathsf{L}}(\mathcal{I})(\sigma^{\mathsf{L},head}_{\mathtt{bool}}(h)) = (\bigvee_{t\in B_\Gamma} (\mathsf{arg}^{\mathtt{bool}}\varsigma_{\mathsf{T}_\Sigma\varnothing_S}(h))(t)) \wedge \mathcal{I}^{\mathsf{L},\&}(\sigma^{\mathsf{L},body}_{\mathtt{bool}}(b)).$$

In this case, ϖ^{L} is also monotonic.

4 Conclusions

What is Logic? Logic is a structure containing signatures, terms, sentences, structured sets of sentences, entailments, algebras, satisfactions, axioms, theories and proof calculi. Signature have sorts (types) and operators, and algebras provide the meaning of the signature. All terms are constructed (syntactically) using operators in the signature, and sentences have terms as building blocks. Entailment is the relation between the structured sets of sentences, representing what we already know, and sentences representing knowledge we are trying to arrive at. Satisfaction is the semantic counterpart to entailment providing the notion of valid conclusions. Axioms prescribe what we take for granted at start, and act as the logic program. Inference rules say how we can jump to conclusions in a chain of entailments.

Further, unsortedness and many-sortedness are clearly different, and so are crisp and fuzzy cases. Moreover, we should distinguish between "fuzzy logic programming", which requires considerations of underlying categories [8], from "logic programming with fuzzy", which is more about composing with term monads using `Set` as the underlying category [12].

Fuzzy considerations in logic are then indeed similarly related to structures which contain fuzzy signatures, fuzzy terms, fuzzy sentences, fuzzy structured sets of sentences, fuzzy entailments, fuzzy algebras, fuzzy satisfactions, fuzzy axioms, fuzzy theories and fuzzy proof calculi.

Details related to generalized general logic appear in [14], and further developments in particular related to sentence constructions will appear in [9].

References

1. J. F. Baldwin, T. P. Martin, B. W. Pilsworth. *Fril - Fuzzy and Evidential Reasoning in Artificial Intelligence.* John Wiley & Sons, Inc., 1995.
2. C. V. Damásio, L. Moniz-Pereira. Monotonic and Residuated Logic Programs. In *Proceedings of Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (Eds. S. Benferhat, P. Besnard), 6th European Conference (ECSQARU'01), Lecture Notes in Computer Science (LNAI) 2143:748–759, 2001.
3. R. Ebrahim. Fuzzy logic programming. *Fuzzy Sets and Systems*, 117(2):215–230, 2001.
4. S. Eilenberg, G. M. Kelly. Closed categories. In Samuel Eilenberg et al., editor, *Proceedings of the Conference on Categorical Algebra, La Jolla 1965*, pages 421–562. Springer-Verlag, 1966.
5. S. Eilenberg, S. MacLane. General theory of natural equivalences. *Transactions of the American Mathematical Society*, 58(2):231–294, 1945.
6. P. Eklund, M.A. Galán, R. Helgesson, J. Kortelainen. Paradigms for many-sorted non-classical substitutions. In *2011 41st IEEE International Symposium on Multiple-Valued Logic (ISMVL 2011)*, pages 318–321, 2011.
7. P. Eklund, M.A. Galán, R. Helgesson, J. Kortelainen. Fuzzy terms. *Fuzzy Sets and Systems.* in press.
8. P. Eklund, M.A. Galán, R. Helgesson, J. Kortelainen. From Aristotle to Lotfi. In R. Seising, E. Trillas, C. Moraga, S. Termini (eds.), *On Fuzziness: A Homage to*

Lotfi A. Zadeh – Volume 1. Studies in Fuzziness and Soft Computing, vol. 298, Springer-Verlag, 2013, pages 147–152.
9. P. Eklund, M.A. Galán, R. Helgesson, J. Kortelainen. Fuzzy sentences. in progress.
10. P. Eklund, M.A. Galán, R. Helgesson, J. Kortelainen, G. Moreno, C. Vázquez. Towards a Categorical Description of Fuzzy Logic Programming. Working paper accepted for presentation in PROLE'13, 2013.
11. P. Eklund, M. Ángeles Galán, M. Ojeda-Aciego, A. Valverde. Set functors and generalised terms. *Proc. IPMU 2000, 8th Information Processing and Management of Uncertainty in Knowledge-Based Systems Conference*, III:1595–1599, 2000.
12. M.A. Galán. *Categorical Unification.* PhD thesis, Umeå University, Department of Computing Science, 2004.
13. J. A. Goguen, R. M. Burstall, *INSTITUTIONS: Abstract Model Theory for Specification and Programming*, J. ACM 39(1):95–146, 1992.
14. R. Helgesson. *Generalized General Logics.* PhD thesis, Umeå University, Department of Computing Science, 2013.
15. M. Ishizuka, N. Kanai. Prolog-ELF Incorporating Fuzzy Logic. In Aravind K. Joshi, editor, *Proc. of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85). Los Angeles, CA, August 1985.*, pages 701–703. Morgan Kaufmann, 1985.
16. P. Julián, G. Moreno, J. Penabad. On the declarative semantics of multi-adjoint logic programs. In *Proc. of the 10th International Work-Conference on Artificial Neural Networks, IWANN'09*, Lecture Notes in Computer Science (LNCS) 5517:253–260, 2009.
17. M. Kifer, V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
18. F. Klawonn, R. Kruse. A Łukasiewicz logic based prolog. *Mathware and Soft Computing*, 1:5–29, 1994.
19. R. C. T. Lee. Fuzzy Logic and the Resolution Principle. *Journal of the ACM*, 19(1):119–129, 1972.
20. D. Li, D. Liu. *A fuzzy Prolog database system.* John Wiley & Sons, Inc., 1990.
21. J. W. Lloyd. *Foundations of logic programming.* Springer-Verlag, 1984.
22. S. MacLane. *Categories for the working mathematician.* Springer-Verlag, 1971.
23. J. Medina, M. Ojeda-Aciego, P. Vojtáš. Similarity-based Unification: a multi-adjoint approach. *Fuzzy Sets and Systems*, 146:43–62, 2004.
24. J. Meseguer, General logics, in: H.-D. Ebbinghaus (Ed.), Logic Colloquium '87, pages 275–329. North-Holland, Granada, Spain, 1989.
25. S. Muñoz-Hernández, V. P. Ceruelo, H. Strass. RFuzzy: Syntax, semantics and implementation details of a simple and expressive fuzzy tool over Prolog. *Information Sciences*, 181(10):1951–1970, 2011.
26. R. Ng, V. S. Subrahmanian. Stable semantics for probabilistic deductive databases. *Information and Computation*, 110(1):42–83, 1994.
27. J. Pavelka. On fuzzy logic I, II, III. *Zeitschrift für Math. Logik und Grundlagen der Math*, 25:45–52, 119–134, 447–464, 1979.
28. M. Rodríguez-Artalejo, C. A. Romero-Díaz. Quantitative logic programming revisited. In J. Garrigue, M. Hermenegildo (Eds.), *Functional and Logic Programming (FLOPS'08)*, Lecture Notes in Computer Science (LNCS) 4989:272–288, 2008.
29. D. E. Rydeheard, R. M. Burstall. A categorical unification algorithm. In D. H. Pitt, S. Abramsky, A. Poigné, D. E. Rydeheard (Eds.), *CTCS*, Lecture Notes in Computer Science (LNCS) 240:493–505, 1985.
30. M. I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Fuzzy Sets and Systems*, 275:389–426, 2002.

31. E. Y. Shapiro. Logic programs with uncertainties: A tool for implementing rule-based systems. In *Proc. of the 8th International Joint Conference on Artificial Intelligence, IJCAI'83, Karlsruhe*, pages 529–532, 1983.
32. V. S. Subrahmanian. On the Semantics of Quantitative Logic Programs. In *Proc. of International Symposium on Logic Programming*, pages 173–182, 1987.
33. M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.
34. P. Vojtáš. Fuzzy Logic Programming. *Fuzzy Sets and Systems*, 124(1):361–370, 2001.