Sorted Monotonic Logic Programs and their Embeddings

C. V. Damásio
Centro de Inteligência Artificial
Universidade Nova de Lisboa
Portugal
cd@di.fct.unl.pt

L. M. Pereira
Centro de Inteligência Artificial
Universidade Nova de Lisboa
Portugal
lmp@di.fct.unl.pt

Abstract

In this paper we present a logic programming-based language allowing for the combination of several lattices of truth-values under arbitrary monotonic operators. A model and fixpoint theory are presented, but the main contributions of the paper are the embedding results of a series of existing logic programming semantics dealing with uncertainty, vagueness, or probabilistic reasoning. A major benefit of this work is to provide a comparative overview of the several proposals, all of which are translate into a **single unified** general framework. This paves the way for the construction of integrated logic programming-based systems capturing several facets of human/formal uncertainty reasoning. We overview, and compare more than twenty different proposals in the extant literature.

Keywords: Fuzzy Logic Programming, Quantitative Deduction, Annotated Logic Programming, Probabilistic Logic Programming

1 Introduction

The interest in the development of logics for dealing with information which might be either vague or uncertain has increased in recent years. Several different approaches to so-called inexact, or fuzzy, or approximate reasoning have been proposed, involving either quantitative [49, 52, 43, 42], or possibilistic [22, 1, 34], or fuzzy [2, 4, 54, 5, 8, 53, 44, 12, 14, 27, 9], or signed and annotated [32, 37, 17, 8], or probabilistic [2, 4, 5, 18, 16, 17, 39, 35, 36], or similarity-based logic programming [1, 45].

Our proposal uses a sorted language, where each sort identifies an underlying lattice of truth-values (weights) which must obey a very general property for guaranteeing the validity of the usual classical logic programming results: model-theoretical declarative semantics, fixpoint semantics, and existence of least model. We designate this formal framework *Sorted Monotonic Logic Programming.* As this designation indicates, we do not address non-monotonic constructs in this setting though some works are already moving in that direction [19, 11, 20]. Disjunction is also not allowed in the consequents of rules, in order to guarantee the existence of the least model for every conceivable program. In fact, one can say that this framework is the most general one you can envisage without sacrificing, for truth-functional based logics, the existence of the least model for every program. We restrict our programs to the ground case but allow infinite programs, and thus do not preclude generality. The current proposal is an important enhancement of previous works [10, 12, 14, 44, 9]. We do not address any form of proof procedures, and in particular do not touch questions regarding termination and continuity of the fixpoint operators.

In our opinion, the major contributions of this paper are the overview and embeddings of a large number of semantics into our sorted monotonic logic programs. This can help us detect and explore similarities betwixt them.

We've tried to collect a representative number of semantics, but by no means do we pretend to state to have been exhaustive. Nevertheless, we are convinced that a good coverage of the extant literature has been accomplished.

The use of several sorts simultaneously enables the combination of several reasoning forms in the same program, aiming at the construction of new forthcoming advanced systems. From an analysis of the literature, we've concluded that this multi-sorted approach is already in use, for instance by Cao in [8]. However, almost all present semantics are defined for the single sorted case.

The paper is divided into two distinct parts. The first, comprising Sections 2 and 3, introduces the formal framework of sorted monotonic logic programs and sets forth the major theoretical results. The second consists of four lighter sections, where we compare our setting with quantitative deduction (Section 4), Fuzzy and Possibilistic Logic Programming (Section 5), Logic Programs with Annotations (Section 6), and finally Section 7 addresses Probabilistic-based Logic Programming Semantics. No formal proofs are given but the results are intuitive in most of the cases, so please bear with us. We finish with overall conclusions and pointers to future work. Of all the semantics analysed we were not able to compare just with one...

We request the reader to be patient since we were unable to shorten this work, or cut out parts of it: its value resides in the global perspective and similarities that we expect to transmit for you to gain on.

2 Preliminary Definitions

We will make extensive use of the constructions and terminology of universal algebra, in order to define formally the syntax and the semantics of the languages we will deal with. A minimal set of concepts from universal algebra, which will be used in the sequel in the style of [14], is introduced below.

The notions of signature and Σ-algebra will allow the interpretation of the function and constant symbols in the language, as well as the specifying of syntax.

Definition 1 (Signature) *A* signature *is a pair $\Sigma = \langle S, F\rangle$ where S is a set of elements, designated* sorts, *and F is a collection F of pairs $\langle f, s_1 \times \cdots \times s_k \to s\rangle$ denoting functions, such that $s, s_1, \ldots s_k$ are sorts and no symbol f occurs in two distinct pairs. The number k is the arity of f. If k is 0 then f is a constant symbol.*

Simplifying notation, we write $f\colon\tau$ to denote a pair $\langle f, \tau\rangle$ belonging to F.

Definition 2 (Σ-Algebra) *Given a signature $\Sigma = \langle S, F\rangle$, a* Σ-algebra $\mathfrak{A}$ *is a pair $\left\langle \{A^s\}_{s\in S}, I\right\rangle$ where:*

1. *Each A^s is a nonempty set called the carrier of sort s,*
2. *and I is a function which assigns a map*

$$I(f) : A^{s_1} \times \cdots \times A^{s_k} \to A^s$$

to each $f\colon s_1 \times \cdots \times s_k \to s \in F$, where $k > 0$, and assigns an element $I(c) \in A^s$ to each constant symbol $c : s$ in F.

The basic requirements for constructing and giving semantics for sorted monotonic logic programs are the existence of an "arrow" symbol for each sort, respecting the ordering of the underlying carrier lattice, and constant symbols representing the top elements of the sort carriers. From the point of view of expressiveness, it is fundamental to admit extra operators to be involved with the operators in the Σ-algebra. These provisos are captured in the following definition:

Definition 3 (Arrow Σ-Algebra) *A Σ-Algebra $\mathfrak{L}$ is an* Arrow Σ-Algebra *whenever:*

1. *The carrier L^s of each sort is a complete lattice under a partial order $\preceq^s$.*
2. *Each sort s contains a constant symbol $\top^s : s$ mapped to the top element of L^s and an operator $\leftarrow^s\colon s \times s \to s$ such that*

$$(x\ I(\leftarrow^s)\ y) = I(\top^s) \text{ iff } y \preceq^s x$$

for every pair of elements x, y of L^s.

Mark that it is always possible to construct an operator $\leftarrow^s$ as per Definition 3. Simply let $I(\leftarrow^s)$ be mapped to the top element of L^s whenever its right hand side is less or equal than its left hand side; otherwise, let the result be the bottom element of the lattice.

The existence of several sorts will enable us to combine in the same program several reasoning forms. The use of several sorts can be found in the annotated fuzzy logic programming framework of [8] where our sorts correspond to distinct universes of discourse with corresponding fuzzy sets. Our framework is richer since we do not restrain ourselves to complete fuzzy lattices, and allow for more operators.

In practice we shall usually have to assume properties bearing on the extra operators considered, namely monotonicity.

3 Syntax and Semantics of Sorted Monotonic Logic Programs

Sorted monotonic logic programs will be constructed from the abstract syntax induced by an arrow Σ-algebra on a set of sorted propositional symbols (or variables). Specifically, we will consider an arrow Σ-algebra $\mathfrak{L}$ whose extra operators can be arbitrary monotone operators. This algebra hosts the manipulation of truth-values in our programs.

In addition, let Π be an infinite set of sorted propositional symbols, disjoint from the set of function symbols in $\mathfrak{L}$, and the corresponding term Σ-algebra[1] of formulas $\mathfrak{F} = Terms(\Sigma, \Pi)$. To denote that a symbol $A \in \Pi$ has sort s we often write $A \in \Pi^s$.

Remark 1 *As we are working with two Σ-algebras, and to lighten the notation, we introduce a special one to clarify which algebra a function symbols belong to, instead of continuously using either $\sigma_{\mathfrak{L}}$ or $\sigma_{\mathfrak{F}}$. Let σ be a function symbol in Σ, its interpretation under $\mathfrak{L}$ is denoted $\dot{\sigma}$ (a dot on the operator), whereas σ itself will denote $\sigma_{\mathfrak{F}}$ when there is no risk of confusion arises.*

[1] In short, this corresponds to the algebra freely generated from Π and the set of function symbols in $\mathfrak{L}$, respecting sort assignments.

The definition of sorted monotonic logic program is provided, as usual, by a set of rules and facts, whose particular syntax is given below:

Definition 4 (Sorted Monotonic LPs) *A* sorted monotonic logic program *is a set $\mathbb{P}$ of rules of the form $A \leftarrow^s \mathcal{B}$ such that:*

1. *The* rule *$A \leftarrow^s \mathcal{B}$ is a formula (an algebraic term) of $\mathfrak{F}$;*
2. *The* head *A of the rule is a propositional symbol of Π of sort s.*
3. *The* body *$\mathcal{B}$ is a formula of $\mathfrak{F}$ with sort s, built from sorted propositional symbols $B_1, \ldots, B_n$ ($n \geq 0$) by the use of function symbols in Σ.*
4. Facts *are rules with body $\top^s$.*

Semantically, a propositional variable of sort s will be assigned an element of the carrier lattice of s. This is the extension of the classical notion of interpretation.

Definition 5 (Interpretation) *An* interpretation *is a mapping $I\colon \Pi \to \bigcup_s L^s$ such that for every propositional symbol p of sort s then $I(p) \in L^s$. The set of all interpretations of the sorted propositions defined by the Σ-algebra $\mathfrak{F}$ in the Σ-algebra $\mathfrak{L}$ is denoted $\mathcal{I}_{\mathfrak{L}}$.*

Note that, by the unique homomorphic extension theorem, each of these interpretations can be uniquely extended to the whole set of formulas $\mathfrak{F}$. The so induced valuation function by interpretation I will be denoted $\hat{I}$.

The orderings $\preceq^s$ of the truth-values L^s can be easily extended to the set of interpretations as follows:

Definition 6 (Lattice of interpretations) *Consider $I_1, I_2 \in \mathcal{I}_{\mathfrak{L}}$. Then $\langle \mathcal{I}_{\mathfrak{L}}, \sqsubseteq \rangle$ is a lattice where $I_1 \sqsubseteq I_2$ iff $I_1(p) \preceq^s I_2(p)$ for all $p \in \Pi^s$. The least interpretation $\triangle$ maps every propositional symbol of sort s to the bottom element of L^s.*

A rule of a sorted monotonic logic program is satisfied whenever the truth-value of the head is greater or equal than the truth-value of the body, i.e. if the value of the implication is the top element of the corresponding carrier of the sort of the rule. Formally:

Definition 7 (Satisfaction, Model) *Given an interpretation $I \in \mathcal{I}_{\mathfrak{L}}$, a rule $A \leftarrow^s \mathcal{B}$ is* satisfied *by I iff $\hat{I}(A \leftarrow^s \mathcal{B}) = \hat{I}(\top^s)$ iff $\hat{I}(\mathcal{B}) \preceq^s \hat{I}(A)$. An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$ is a* model *of a sorted monotonic logic program $\mathbb{P}$ iff all rules in $\mathbb{P}$ are satisfied by I.*

The immediate consequences operator, given by van Emden and Kowalski [51], can be easily generalised to the framework of sorted monotonic logic programs.

Definition 8 *Let $\mathbb{P}$ be a sorted monotonic logic program. The* immediate consequences operator *$T_{\mathbb{P}}$ maps interpretations to interpretations, and is defined by*

$$T_{\mathbb{P}}(I)(A) = \bigsqcup_s \{\hat{I}(\mathcal{B}) \mid \langle A \leftarrow^s \mathcal{B} \in \mathbb{P}\}$$

where A is an arbitrary propositional symbol of sort s, and $\sqcup_s$ is the least upper bound in the lattice L^s.

The immediate consequences operator evaluates the body of every rule for a propositional symbol A. The truth-value of A is simply the least upper bound of the truth-values of all the bodies of the rules for it.

Theorem 1 (Monotonicity of $T_{\mathbb{P}}$) *Let I_1 and I_2 be two interpretations in $\mathcal{I}_{\mathfrak{L}}$, and $\mathbb{P}$ be a sorted monotonic logic program. Operator $T_{\mathbb{P}}$ is monotonic: if $I_1 \sqsubseteq I_2$ then $T_{\mathbb{P}}(I_1) \sqsubseteq T_{\mathbb{P}}(I_2)$.*

As usual, the set of models of P is characterized by the post-fixpoints of $T_{\mathbb{P}}$:

Theorem 2 *An interpretation I of $\mathcal{I}_{\mathfrak{L}}$ is a model of a sorted monotonic logic program $\mathbb{P}$ iff $T_{\mathbb{P}}(I) \sqsubseteq I$.*

By the Knaster-Tarski fixpoint theorem, $T_{\mathbb{P}}$ has a least fixpoint. Thus:

Definition 9 (Declarative Semantics) *The semantics of a sorted monotonic logic program $\mathbb{P}$ is given by the least model $M_{\mathbb{P}}$ of $\mathbb{P}$, which always exists.*

Theorem 3 (Fixpoint Semantics) *Let $\mathbb{P}$ be a sorted monotonic logic program, and consider the transfinite sequence of interpretations of $\mathcal{I}_{\mathfrak{L}}$:*

$$\begin{aligned} T_{\mathbb{P}}\uparrow^0 &= \Delta \\ T_{\mathbb{P}}\uparrow^{n+1} &= T_{\mathbb{P}}(T_{\mathbb{P}}\uparrow^n) \\ T_{\mathbb{P}}\uparrow^\alpha &= \textstyle\bigsqcup_{\beta<\alpha} T_{\mathbb{P}}\uparrow^\beta, \ \alpha \text{ a limit ordinal} \end{aligned}$$

Then there is an ordinal λ such that $T_{\mathbb{P}}\uparrow^{\lambda+1} = T_{\mathbb{P}}\uparrow^\lambda$, the least fixpoint of $T_{\mathbb{P}}$. Moreover $M_{\mathbb{P}} = T_{\mathbb{P}}\uparrow^\lambda$.

The major difference from standard classical logic programming is that our $T_{\mathbb{P}}$ operator might not be continuous, and therefore more than ω iterations may be necessary to "reach" the least fixpoint. All the other important results carry over to our sorted monotonic logic programs. This possibility is unavoidable if one wants to retain generality. The sufficient conditions to guarantee the continuity of a similar immediate consequences operator in [44] are readily adaptable to the current setting.

In face of the previous definitions and results, the essence of logic programming is solving (possibly infinite) sets of inequations of the form

$$A \succeq^s \mathcal{B} \tag{1}$$

The previous theorems show that these sets of inequations always have the least solution in complete lattices, and the fixpoint semantics provides an iterative method for obtaining such single minimal solution. In some sense, one cannot be more general than sorted monotonic logic programs without sacrificing the existence of a single least model, as shown in the next example:

Example 1 *Reichenbach [47] devised a calculus for addressing the logical problems raised by quantum mechanics. He defined three implications, three negations, two equivalences, a conjunction, and a disjunction. For* *our example, the three implication operators and conjunction will suffice. The set of truth-values is* $\{0,1,2\}$ *with the usual ordering. The truth-tables*[2] *are:*

$\dot{\subset}$	0	1	2
0	2	1	0
1	2	2	1
2	2	2	2

$\dot{\Leftarrow}$	0	1	2
0	2	2	0
1	2	2	0
2	2	2	2

$\dot{\leftarrow}$	0	1	2
0	1	1	0
1	1	1	1
2	1	1	2

$\dot{\wedge}$	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

Consider the three programs below:

$a \subset b \wedge c$	$a \Leftarrow b \wedge c$	$a \leftarrow b \wedge c$
$b \subset 1$	$b \Leftarrow 1$	$b \leftarrow 1$
$c \subset 1$	$c \Leftarrow 1$	$c \leftarrow 1$

The least fixpoint of the immediate consequences operator when applied to the above programs results in the same interpretation, mapping propositions $a, b,$ *and* c *to truth-value* 1. *The first implication* ($\dot{\subset}$) *complies with the provisions of Definition 3. Therefore the interpretation so obtained is the least model of the program on the left, as can be checked easily.*

For the middle program, we know that $t_1 \geq t_2$ *implies* $(t_1 \dot{\Leftarrow} t_2) = 2$, *for* $t_1, t_2 \in \{0,1,2\}$ *but the converse does not hold. In general, the least fixpoint is a model of the program but might not be minimal, as in this case: the interpretation mapping every proposition to* 0 *is a model.*

Finally, for the program on the right, we have that $(t_1 \dot{\leftarrow} t_2) = 2$ *entails* $t_1 \geq t_2$ *but not conversely. This situation is more problematic, since the least fixpoint might not be a model of the program, as per the example. In fact, this program has no model because the implications will always be evaluated to a truth-value different from 2.*

By allowing the left hand side of equation (1) to be general monotonic functions on a set of propositional variables, $A_1, \ldots, A_m$ one can either obtain several minimal models or have no solutions. This is the realm of disjunctive logic programs, which we intend to explore in future work, in the style of [29].

[2] For the implication connectives, the consequent truth-value appears in rows and the antecedent one in columns.

4 Quantitative Deduction and its relatives

Our first set of results relates the quantitative deduction rules defined by van Emden in the seminal work [52] with our own sorted monotonic logic programs, and the logic programming with costs of Marek and Truszczyński [42]. Without loss of generality, we consider only the propositional versions of the languages. We do not provide any explicit theorems which are obvious from the translations presented and accompanying discussion. These general remarks do apply to the subsequent sections.

4.1 Quantitative Deduction

Van Emden's work on quantitative deduction [52] is one of the first attempts to develop a full-fledged declarative model-theory for quantitative rules, in the continuation of the work by Shapiro [49]. Quantitative rules are an extension of Horn clauses with an "attenuation factor", with the following syntax:

$$A \leftarrow \textcircled{f}\!\!- B_1 \& \ldots \& B_n, \quad n \geq 0 \qquad (2)$$

Without loss of generality, we assume that the atoms A and $B_1 \ldots B_n$ are ground. The attenuation factor f is a real number in the interval $(0,1]$, and truth-values of atoms are interpreted in the closed unit interval $[0,1]$.

The framework of van Emden can be easily translated into a single-sorted monotonic logic program. The underlying arrow Σ-algebra has a single sort *unit* signature with carrier $[0,1]$, with the usual ordering on real numbers. A constant symbol for every element of $[0,1]$ is necessary, as well as the minimum function (denoted by min) and the product of two reals (denoted by $\times$). The quantitative rule (2) corresponds to the following rule in our setting with respect to the previous Σ-algebra:

$$A \leftarrow \mathrm{f} \times \min(B_1, \min(\ldots, \min(B_{n-1}, B_n) \ldots))$$

Obviously, if the number of atoms in the body is 0 (resp. 1), then the rule will be translated simply into $A \leftarrow \mathrm{f}$ (resp., $A \leftarrow \mathrm{f} \times B_1$).

4.2 Logic Programming with costs

In [42], rules of definite logic programs are assigned non-negative real numbers, ascribing to every rule in the program the cost of applying it. The authors define three interpretations of cost. The non-reusability approach assumes that each individual conclusion about an atom in the program involves a cost. In the reusability approach, we only pay once for concluding about an atom, the first time around, and this will be ignored in the rest of the section. Costs can also be interpreted as (parallel) time and in this case the semantics proposed is isomorphic to van Emden's quantitative logic programs [52].

Logic programs with costs are sets of rules of the form

$$A \overset{c}{\leftarrow} B_1, \ldots, B_n \tag{3}$$

where c is a real number in $[0, +\infty[$ and A, $B_1 \ldots B_n$ are atoms. In the parallel time-approach the rule is translated into the following Sorted Monotonic Logic Programming rule:

$$A \leftarrow c + \max(B_1, \ldots, \max(B_{n-1}, B_n) \ldots)$$

In the no-reusability approach the resulting translated rule is:

$$A \leftarrow c + B_1 + \ldots + B_n$$

The underlying arrow Σ-algebra in both approaches has a single sort with carrier $[0, +\infty]$, where the least upper bound operation corresponds to infimum (not supremum!). Notice that in the least interpretation every atom is mapped to $+\infty$. The function symbol max is interpreted as maximum, and $+$ as addition of real numbers. These functions are extended in the usual way to deal with $+\infty$ arguments. We also assume the existence of constant symbols for every element in $[0, +\infty]$.

The interesting remark is that under the parallel time approach, it is shown in [42] that a cost rule of the form (3) is equivalent to a quantitative deduction rule with body $B_1 \& \ldots \& B_n$ and attenuation factor $-\log c$, and therefore an isomorphism holds between the least model of the logic program with costs and the least model of the corresponding quantitative program.

5 Fuzzy and Possibilistic Logic Programming

In this section we describe Fuzzy and Possibilistic Logic Programming, which formalizes in solid terms previous attempts at defining logic programming dealing with uncertainty and vagueness. We analyse several semantics and see how they have converged to the general setting of sorted monotonic logic programs proposed in this work.

5.1 Fuzzy Logic Programming

Vojtáš and Paulík [54] propose a language and corresponding model theory for a generalization of definite logic programming, based on Pavelka's and Hájek's works (see [31] for a summary). As Vojtáš states in a subsequent paper [53], this work can be classified as belonging to truth functional fuzzy logic in the narrow sense of Hájek [31]. Programs are constructed from an implication connective (say $\rightarrow_1$), with corresponding t-norm adjunction (resp. $\otimes_1$), and another t-norm operator denoted by $\otimes_2$. Programs are made of rules of this form, where q is a rational number in [0,1] expressing a confidence factor:

$$((B_1 \otimes_2 \ldots \otimes_2 B_n) \rightarrow_1 A) \text{ with-cf} = q.$$

As usual, A and $B_1, \ldots B_n$ are ground atoms with truth-value in the unit interval $[0, 1]$. These rules are translated into our setting as:

$$A \leftarrow_1 \otimes_1(q, \otimes_2(B_1, \otimes_2(\ldots, \otimes_2(B_{n-1}, B_n) \ldots)))$$

We assume that $\leftarrow_1$ is the same as $\rightarrow_1$, but with antecedent and consequent interchanged. The definition of the underlying single sorted arrow Σ-algebra with carrier $[0, 1]$ is left to the reader. Notice that the Quantitative Deduction framework of van Emden is an instance of the previous one, by letting $\leftarrow_1$ be Goguen implication, $\otimes_1$ its adjunction (product), and $\otimes_2$ be Gödel t-norm (minimum).

This work has been continued by Vojtáš in the article [53], still restricted to interpretations in the unit interval. Here the bodies of rules are built from arbitrary conjunctors, disjunctors, and aggregators (monotone operators). It is important to refer that conjunctors and disjunctors are assumed to be only monotone operators which extend the classical connectives, and thus might not be t-norms or t-conorms, respectively. The program might contain several implication connectives which obey the many-valued Modus Ponens [31]:

$$\frac{(B, x) \qquad (B \rightarrow A, y)}{(A, \mathcal{C}_{\rightarrow}(x, y))}$$

where x and y are the lower bounds of the truth-values of B and $B \rightarrow A$, and $\mathcal{C}_{\rightarrow}(x, y)$ is the resulting estimate for the lower bound of A. A rule of Fuzzy Logic Programs has the abstract form:

$$A \leftarrow_i Body[B_1, \ldots, B_n] \text{ with-cf} = q$$

The body of these programs are formulas constructed from the atoms $B_1, \ldots, B_n$ and the connectives available in the language. Again, q is a rational number. These are translated into our sorted Monotonic Logic Programs as expected:

$$A \leftarrow \&_i \, (Body[B_1, \ldots, B_n], q)$$

As before, the underlying arrow Σ-algebra is single-sorted with carrier $[0, 1]$. We assume that the conjunction $\&_i$ is made available in the language, and is interpreted as the conjunctor $\mathcal{C}_i$ evaluating modus ponens with respect to implication $\leftarrow_i$; the ordering of arguments in $\&_i$ is important since symmetry is not assumed by Vojtáš. The rest of connectives in the language are interpreted as in [53]. Mark that we use the same and arbitrary $\leftarrow$ symbol, irrespective of the original implication symbol, and interpreted in such way as to obey Definition 3, respecting the usual total ordering of real numbers in $[0, 1]$. A justification of this result can be found in the next section.

Finally, Vojtáš concludes that the fuzzy logic programming systems described in the seminal work [21] are either based on heuristic methods, or truth-functional based ones are instances of the framework presented in [53]. In this way, Vojtáš answers the proviso of Dubois, Lang, and Prade that a fuzzy logic programming language should be based on a real logical system. The FPROLOG interpreter of Martin, Baldwin and Pilsworth [43] describes an implementation of a language which can be seen as an instance of Vojtáš scheme, but do not specify exactly the properties of the "combination" operator of truth-values in the bodies, which they say that it is usually minimum. When using minimum, FPROLOG is an instance of Quantitative Deduction and coincides with Possibilistic Logic Programming, as we shall see later on.

5.2 Residuated and Multi-adjoint Logic Programming

The restriction to the carrier $[0, 1]$ is not essential and was lifted by ourselves in [12], starting from the initial work of Vojtáš and Paulík [54]. In [12] we have proposed single-sorted monotonic logic programs over arbitrary complete lattices and, as an instance, presented Residuated Logic Programs. Our Residuated Logic Programs are built over a single complete residuated lattice with residuum $\leftarrow$ and corresponding conjunctor $\&$. We assume that $\&$ is commutative, associative, non-decreasing in both arguments and the top element of the lattice is the unit element. Obviously, the pair $(\leftarrow, \&)$ must obey the adjunction property, guaranteeing that generalized modus ponens is a valid inference rule. The adjunction property allows the assignment of weights to every rule in the residuated logic program thus guaranteeing the existence of a least model. This idea was already present in the 1987 work by Fitting on Pseudo-Boolean Valued Prolog with an underlying Heyting Algebra truth-value structure [24]. However, this only conjunction is allowed in the body.

Residuated Logic Programs have been extended by Medina, Aciego and Vojtáš in [44].

Essentially, several implications are allowed in the language, and commutative and associativity of the conjunctors are dropped, in the spirit of [53]. However, the original Monotonic Logic Programs in [12] also subsume Multi-Adjoint Logic Programming. Just as a side remark, we have introduced anti-monotonic operators in the language of Monotonic Logic Programs in [11].

Quite recently, Multi-Adjoint Logic Programs have been extended to a sorted language in [9]. The translation of Sorted Multi-Adjoint Logic programs into Sorted Monotonic Logic programs is along the lines of the previous translation of Fuzzy Logic Programming, but now in a setting with several sorts. Rules in [9] have the general form $\langle A \leftarrow_i^s \mathcal{B}, \vartheta \rangle$. The implication connective $\leftarrow_i^s$ has sort s, A is a propositional variable and $\mathcal{B}$ a formula built from the propositional variables (atoms) in the underlying multi-adjoint Σ-algebra and propositional variables. The body formula $\mathcal{B}$ must have sort s and ϑ is an element (weight) of the carrier of sort s. In sorted monotonic logic Programs, the above rule is written as

$$A \leftarrow^s \vartheta \,\&_i^s\, \mathcal{B}$$

The arrow Σ-algebra is obtained directly from the Multi-adjoint Σ-algebra, extended with a new operator for each sort s, interpreting $\leftarrow^s$ such that it respects the underlying ordering of the complete lattice carrier of s. The least model of the sorted monotonic logic program corresponds exactly to the least model of the original sorted multi-adjoint logic program. This is immediate from the following equivalences, where I is an interpretation

$$\begin{array}{ll} & \hat{I}(A \leftarrow^s \vartheta \,\&_i^s\, \mathcal{B}) = \hat{I}(\top^s) \\ \text{iff} & \hat{I}(\vartheta \,\&_i^s\, \mathcal{B}) \preceq^s \hat{I}(A) \\ \text{iff} & \dot{\vartheta} \,\dot{\&}_i^s\, \hat{I}(\mathcal{B}) \preceq^s \hat{I}(A) \\ \text{iff} & \dot{\vartheta} \preceq^s \left(\hat{I}(A) \dot{\leftarrow}_i^s \hat{I}(\mathcal{B})\right) \\ \text{iff} & \dot{\vartheta} \preceq^s \left(\hat{I}(A \leftarrow_i^s \mathcal{B})\right) \end{array}$$

The first equivalence holds because we assumed that the operator interpreting $\leftarrow^s$ respects the ordering of s, and the second one by the adjunction property. The adjunction property and existence of the unit element also allow showing that $\leftarrow^s$ can be any of the implication connectives $\leftarrow_i^s$. This is immediate from the argument below:

$$\begin{array}{ll} & (y \dot{\leftarrow}_i^s z) = \dot{\top}^s \\ \text{iff} & \dot{\top}^s \preceq^s (y \dot{\leftarrow}_i^s z) \\ \text{iff} & \left(\dot{\top}^s \,\dot{\&}_i^s\, z\right) \preceq^s y \\ \text{iff} & z \preceq^s y \end{array}$$

Finally, it is worth referring that Guller [29] has proposed a semantics for fuzzy disjunctive programs with weak similarity. A single residuated lattice is assumed, and the bodies are restricted to applications of the underlying t-norm operator. The most notable feature is that disjunction is allowed in the heads of rules and a proof procedure was developed [28].

5.3 Possibilistic Logic Programming

As summarized in [23], possibilistic logic is a logic of uncertainty tailored for reasoning under incomplete evidence and partially inconsistent knowledge. Each formula in the language is assigned a weight in a totally ordered set, corresponding to a lower bound on the degree of necessity or possibility of the formula. The degree of necessity of a formula states to what extent the available evidence entails the truth of the formula, while the degree of possibility expresses to what extent the truth of a formula is not incompatible with the available evidence.

A Possibilistic Logic Program [22] is a finite set of (first-order) possibilistic Horn Clauses annotated only with necessity degrees, having the syntax:

$$(B_1 \wedge \ldots \wedge B_n \;\rightarrow A \;(N\alpha)) \qquad (4)$$

where $A, B_1, \ldots B_n$ are propositional symbols and $(N\alpha)$ is the necessity degree of the clause with $0 \leq \alpha \leq 1$.

The embedding of Possibilistic Logic Programming is straightforward and an instance of Vojta's Fuzzy Logic Programming, where the underlying implication and conjunctor are the corresponding Gödel's connectives. A possibilistic Horn clause of the form (4) is encoded in the usual unit interval single Sorted Monotonic Logic Programs :

$$A \leftarrow \min(\alpha, \min(B_1, \min(\ldots, \min(B_{n-1}, B_n)\ldots))$$

corresponding to the Fuzzy Logic Programming rule, where $\leftarrow_G$ is interpreted as Gödel's implication:

$$A \leftarrow_G \min(B_1, \min(\ldots, \min(B_{n-1}, B_n)\ldots)).\ \mathrm{cf} = \alpha$$

Mark that Possibilistic Logic Programming is also a particular instance of van Emden's Quantitative Deduction, by a simple program transformation. A possibilistic Horn clause is compiled into a quantitative rule and a quantitative fact:

$$A \leftarrow ①\!-\ r_\alpha \& B_1 \& \ldots \& B_n. \qquad r_\alpha \leftarrow @\!-$$

The atom r_α is a new atom not occurring in the program. The converse does not hold, i.e. possibilistic logic programming is not capable of capturing van Emden's framework. This is expected since the only connective allowed in PLP is minimum while in QD we have both product and minimum available.

In [12] we allow possibilistic Horn clauses annotated with possibility degrees, which is not addressed in [22], but our translation is sound but not complete with respect to possibilistic logic. When restricted to necessity valued clauses, our encoding of possibilistic logic programming is a complete one and is the same as above. The translation proposed in Kullman and Sandri [34] into annotated logic programming [32] is the composition of the above translation into van Emden's framework with the encoding of quantitative deduction in annotated logic programming described for the first-time in [32] (see Section 6.1), and is a particular case of our one one in [12].

Weighted logic programming with Fuzzy Constants (WLFC) is, according to its authors [1], a first-order logic programming language for fuzzy reasoning under uncertainty and ill-information. WLFC has a possibilistic semantics, and a sound automated deduction method is presented which resembles very much the ideas underlying possibilistic logic programming. However, no fixpoint operator nor completeness results are given. We conjecture that grounded WLFC programs correspond to ground possibilistic logic programs with extra rules of the form:

$$(\ p(s_1, \ldots s_m)\ \rightarrow\ p(t_1, \ldots, t_n)\ (N\beta))$$

with $\beta = \min\{\beta_1, \ldots, \beta_m\}$ where each β_i is the unification degree of ground term s_i with ground term t_i. Clearly, one of the major contributions of Alsinet and Godo, is the directional fuzzy unification which is orthogonal to the previous embedding result. The use of multi-adjoint logic programming to emulate the weak unification algorithm of Sessa [48] appears in [45], and resorts to axioms of fuzzy similarity encoded in rules. The above unification rule might be seen a particular instance of their more general technique, and apparently generalizing WLFC.

5.4 Fuzzy Prolog

Guadarrama, Muñoz, and Vaucheret [27] propose a Fuzzy Logic Programming Language over the Borel Algebra of the unit interval $[0, 1]$. Thus, atoms are interpreted as a countable union of sub-intervals of $[0, 1]$. This permits handling at the same time both uncertainty and fuzzyness. They employ an extension of t-norms and t-conorms to intervals, as well as interval-aggregation operators. The Borel Algebra can be given a structure of complete lattice (for more detail see [27]). Fuzzy clauses have the simple form $A \leftarrow_F B_1, \ldots, B_n$ where F is an interval aggregation operator; fuzzy facts are rules of the form $A \leftarrow v$ where v is an element in the Borel algebra. The translation into a sorted monotonic logic program is immediate, and we omit all the details. As a final note, this paper proposes an implementation based on Constraint Logic Programming, but it is not complete since there is not support for disjunctive constraints, and therefore being incapable of combining the contributions from the several rules. Further, the authors make available constructive negation in their implementation. Some interesting application examples are provided and the system is distributed as a module of Ciao Prolog [6].

6 Logic Programs with Annotations

In this section we overview Generalized Annotated Logic Programs [32] and the equivalent Signed Formula Logic Programs [37]. We provide embeddings of both semantics into sorted monotonic logic programs, and show that sorted monotonic logic programs can be "infinitely" more compact than Generalized Annotated Logic Programs. We conclude by analyzing the proposal of Annotated Fuzzy Logic Programs [8].

6.1 Generalized Annotated Logic Programs

Generalized Annotated Logic Programs (GAPs) are one of the most important and pervasive formalisms dealing with uncertainty in rule-based expert systems [32]. Two semantics are proposed, one based on ideals, and another designated restricted. Here we only compare with the restricted semantics of GAPs. The syntax of Generalized Annotated Logic Programs is elaborate and consists of annotated clauses:

$$A : \rho \leftarrow B_1 : \mu_1 \ \& \ldots \& \ B_n : \mu_n \qquad (5)$$

The intuitive reading of annotated clauses is if $B_1 \succeq \mu_1, \ldots, B_n \succeq \mu_n$ then $A \succeq \rho$. In the above clause $A, B_1, \ldots, B_n$ are usual atoms of first-order logic, while ρ is an annotation and $\mu_1 \ldots \mu_n$ are constant or variable annotations. Annotations are either variable annotations or annotation terms. An annotation term is inductively constructed from total and continuous annotation functions, annotation variables, and annotation constants. Note that annotation variables cannot be used as object variables in the atoms, and vice-versa. The underlying range and domain of annotation functions, variables, and constants is an upper semi-lattice. However, in the restricted semantics, it is assumed that this upper semi-lattice is complete, i.e. arbitrary least upper bounds exist. But, as is well-known [15], this entails that the underlying truth-value space must be an ordinary complete lattice.

Quantitative deduction rules can be encoded in annotated clauses as follows [32]:

$$A : f \times \min\{v_1, \ldots, v_n\} \leftarrow B_1 : v_1 \ \& \ldots \& \ B_n : v_n$$

where $v_1, \ldots, v_n$ are annotation variables. It is particularly remarkable that the quantitative rule does not require the extra annotation variables, i.e. for this particular case quantitative rules are a terser representation than annotated clauses. Mark that annotation variables must be instantiated by elements of the complete lattice in the definition of the restricted semantics fixpoint operator. We now argue that this is a disadvantage of GAPs, by showing how one can translate an annotated clause, with potentially infinite instantiations of annotation variables, into a single ground rule of a sorted monotonic logic program rule. The procedure is as follows:

- An annotated variable x in ρ is substituted by $glb\{B_{k_1}, \ldots, B_{k_j}\}$ where B_{k_i} $(1 \leq i \leq j)$ are all atoms annotated with x in the body[3]. Call the result ρ'.
- An atom annotated in the body with a variable which does not occur in the head is simply removed.
- A constant annotated atom B_{l_i} in the body is replaced by $B_{l_i} \leftarrow \mu_{l_i}$.
- A rule $A : \rho \leftarrow B_1 : \mu_1 \& \ldots \& B_n : \mu_n$ is translated into the sorted monotonic logic programming rule (infix notation):

  $$A \leftarrow \left(\rho' \wedge (B_{l_1} \leftarrow \mu_{l_1}) \wedge \ldots \wedge (B_{l_o} \leftarrow \mu_{l_o})\right)$$

  where $\wedge$ is interpreted as the greatest lower bound operator in the truth-value lattice. The $\leftarrow$ connective is assumed to be "two-valued", i.e. it either returns the bottom or top element of the lattice.

The necessary arrow Σ-algebra must contain all the annotated function symbols, interpreted as in GAPs, and the new connective $\wedge$. The translation was first proposed in [13] and is a generalization of the one in the recent work [33] (where the underlying lattice

[3]Notice that $glb = \top$, therefore an annotated variable in the head but not in the body is substituted by $\top$.

is assumed to be the unit interval). In [32], it is shown how GAPs can capture Fitting's bilattice logic programming [25, 26] and thus their results immediately carry over to sorted monotonic logic programs.

The XSB GAPs library by Swift [50], based on tabulation techniques, is planned to be used in the implementation of sorted monotonic logic programs.

6.2 Signed Formula Logic Programming

The logic of signed formulas [30] facilitates the examination of questions regarding many-valued logics through classical logic [7, 37]. Signed formula logic programming (SFLP) was introduced in [37] and starts from a given many-valued logic Λ and its associated set of truth values Δ. In its full generality, signed clauses have the form:

$$S_0 : A \leftarrow S_1 : \mathcal{F}_1 \ \& \ldots \& \ S_n : \mathcal{F}_n$$

where each S_i is a sign (an expressionn denoting a non-empty subset of Δ), $\mathcal{F}_i$ is a Λ-formula, and A is an atom. In order to guarantee the existence of a least model for every SFLP, a notion of extended interpretations and extended satisfaction is defined in order to obtain "good" approximations of the models of Signed Formula Logic Programs [7, 37]. In this case, the authors show that every SFLP is equivalent to a simpler program where every sign is a non-empty subset of Δ and every $\mathcal{F}_i$ is an atom, i.e.

$$s_0 : A \leftarrow s_1 : A_1 \ \& \ldots \& \ s_n : A_n \qquad (6)$$

Interpretations map ground atoms to subsets of Δ. The intuitive meaning of a Λ-atomic signed clause (6) is that whenever $I(A_1) \subseteq s_1, \ldots, I(A_n) \subseteq s_n$ then $I(A) \subseteq s_0$. From this discussion it is now clear how to capture signed formula logic programming in sorted monotonic logic programs.

Like until now, we have a single sort, say s, with the powerset of Δ as carrier. In this lattice, we say that $a \preceq^s b$ iff $a \supseteq b$, that is a contains b. This means that the least upper bound in this lattice corresponds to set intersection, and its least element is Δ. A Λ-atomic signed-clause is now written (in infix notation):

$$A \Subset (s_0 \cup (A_1 \Subset s_1) \cup \ \ldots \ \cup (A_n \Subset s_n))$$

The symbol $\cup$ is interpreted as usual set-union. The new $\Subset$ symbol maps to an operator which returns the empty set when its left hand side argument is contained in its right hand side one, and Δ otherwise. Thus the body of the above monotonic logic programming rule evaluates to s_0 whenever $I(A_1) \subseteq s_1, \ldots, I(A_n) \subseteq s_n$, and to Δ otherwise. This is the intended explained behaviour of SFLP signed clauses.

The signed logic programming framework is a very expressive one, and the "limitation" to carrier sets is not a formal one. Theoretically, both GAPs and SFLPs have the same expressive power [37]. It is a well-known result in lattice theory that every complete lattice is isomorphic to topped $\cap$-structures (see for instance [15]). Thus, signed logic programming is capable of emulating reasoning with any complete lattice. This is put to use for instance in [37] to show the embedding of annotated logics into signed formula logic programs. By looking at the similarities of the previous embeddings of GAPs and SFLPs into sorted monotonic logic program, one can easily see that SFLP can also be embedded into GAPs. In our perspective, and from a practical point of view, annotated logic programming is simpler for the user since one does not need the translation of the particular lattices to topped $\cap$-structures. We also believe that our framework might well be equivalent to SFLPs, but enjoying a shorter notation, as well as being more user-friendly, as sustained before.

6.3 Annotated Fuzzy Logic Programs

Cao's Annotated Fuzzy Logic Programs (AFLP) are a formal basis for fuzzy logic programming systems involving soft data [8]. Both an ideal and restricted semantics in the style of [32] is given, but the author adopts the restricted one. Syntactically, an AFLP clause is an expression:

$$O : A \leftarrow O_1 : B_1 \,\&\ldots\&\, O_n : B_n$$

Each of O, O_1, …, O_n are either atoms or terms of a first-order language, while A, B_1, …, B_n are annotations denoting fuzzy sets in a set of complete lattices of fuzzy sets. In terms of sorted monotonic logic programs these AFLP clauses are mapped to

$$\begin{aligned} Obj \leftarrow A{+}\max(&md_{i_1}(B_1/O_1), \ldots, \\ &\max(md_{i_{n-1}}(B_{n-1}/O_{n-1}), \\ &\quad md_{i_n}(B_n/O_n))\ldots) \end{aligned}$$

For capturing this framework we require several sorts u_i, one for each universe of discourse U_i in the AFLP, with carrier the corresponding complete lattice of fuzzy sets, and least upper bound the intersection of fuzzy sets. Each lattice must also be closed under constant addition, the operator represented here by '+'. Constant addition expects a fuzzy set in its first argument, and a real number belonging to $[0, 1]$ in its second. We thus require an extra sort for the unit interval lattice $[0, 1]$, with max denoting maximum in $[0, 1]$. The series of operators $md_i : U_i \times U_i \rightarrow [0, 1]$, one for each universe of discourse U_i, compute the mismatching degree of the first argument to the second one [8]. This allows implementing the fuzzy modus ponens model of [41]. Mark that in the above rule each B_j is a constant fuzzy set argument of sort u_{i_j}, the same sort of B_j. The author shows that all these operators are monotonic, and have a least fixpoint semantics, thus justifying the embedding. The theory is complemented by a SLD-like proof-procedure.

7 Probabilistic-based Logic Programs

This section covers some of the proposals for enjoining logic programming to probabilistic reasoning. Without any doubt, these are the most difficult semantics to grasp by sorted monotonic logic programming: either the translations are complex, or apply only to subsets of the language, or are not known.

7.1 Ordinary Probabilistic Logic Programs

Lukasiewicz [40] introduces a new approach to probabilistic logic programming in which probabilities are defined over a set of possible worlds and in which classical program clauses are extended by a subinterval of $[0, 1]$ that describes a range for the conditional probability of the head of a clause given its body. In its most general form, probabilistic logic programs of [40] are sets of conditional constraints $(H \mid B)[c_1, c_2]$ where H is a conjunction of atoms and B is either a conjunction of atoms or $\top$, and $c_1 \leq c_2$ are rational numbers in the interval $[0, 1]$. These conditional constraints express that the conditional probability of H given B is between c_1 and c_2 or that the probability of the antecedent is 0. A semantics and complexity of reasoning are exhaustively studied, and in most cases is intractable and not truth-functional. However, for a special kind of probabilistic logic programs the author provides relationships to "classical" logic programming. Ordinary probabilistic logic programs are probabilistic logic programs where the conditional constraints have the restricted form

$$(A \mid B_1 \wedge \ldots \wedge B_n)[c, 1] \text{ or } (A \mid \top)[c, 1] \quad (7)$$

Under positively correlated probabilistic interpretations (PCP-interpretations), reasoning becomes tractable and truth-functional. Ordinary conditional constraints (7) of ordinary probabilistic logic programs under PCP-interpretation can be immediately translated to a quantitative rule

$$A \leftarrow ⓒ\!- B_1\& \ldots \&B_n$$

The semantics of ordinary probabilistic logic programs and quantitative programs coincide under the previous assumptions, providing for the first time a probabilistic interpretation of quantitative deduction. Finally, it is shown in [40] how ordinary probabilistic logic programs under PCP-interpretation can be used to approximate reasoning in probabilistic logic programs with tractable algorithms. The embedding into our framework follows immediately from the results in Section 4.1. This work has been extended in [38] to consider both disjunction in the heads and default negation in the bodies, these being beyond the current expressive capabilities of sorted monotonic logic programs. Moreover, both default negation and disjunction are used to "unfold" many-valuedness into classical disjunctive logic programming.

7.2 Hybrid Probabilistic Logic Programs

The Hybrid Probabilistic Logic Programs of Dekhtyar and Subrahmanian [17] have been proposed for constructing rule systems which empower the user to reason with and combine probabilistic information under different probabilistic strategies, by improving on previous work by Ng and Subrahmanian [46].

A first important remark is that the probabilites of compound events may be closed intervals in $[0, 1]$, and not simply real-valued probability assignments. This is justified in that the conjunction and disjunction of events might not determine a single probability value, but may range over an interval of values. The authors define the necessary properties for reasonable conjunctive (disjunctive) probabilistic strategies. Through a simple analysis of the requirements, one concludes that the conjunctive (disjunctive) probabilistic strategies are pairwise combinations of t-norms (t-conorms, respectively) over pairs of real numbers in the unit interval $[0, 1]$, i.e. intervals. Many of the listed important strategies in [17] were already proposed by Baldwin and his colleagues in the middle 80s [2, 4, 5].

Syntactically, a hybrid probabilistic program over the set $\mathcal{S}$ of probabilistic-strategies is a finite set of formulas (called hp-clauses) of the form

$$B_0 : \mu_0 \leftarrow B_1 : \mu_1 \wedge \ldots \wedge B_k : \mu_k$$

where each $B_i : \mu_i$ is an hp-annotated basic formula over $\mathcal{S}$. Intuitively, an hp-clause means that "if the probability of B_1 falls in the interval μ_1 and $\ldots$ and the probability of B_k falls within the interval μ_k, then the probability of B_0 lies in the interval μ_0". The B_is are designated hybrid basic formulas and are either applications of conjunctive or disjunctive strategies to finite sets of distinct atoms, encoding complex events. The fixpoint semantics is given by a very complex monotonic operator, obtained by composition of other two operators. In [14] we have proposed an embedding of Hybrid Probabilistic Logic programs into Residuated Logic Programming, which is exponential in the size of the original program. This is expected since in most situations reasoning in HPLPs is exponential [18]. Due to complexity of the details involved, we refer the reader to our paper [14]. Hybrid Probabilistic Logic Programs have been further generalized to capture temporal aspects in real-world applications [16]. We have made no yet attempt to capture this...

7.3 Probabilistic Deductive Databases

A definition of a theory of probabilistic deductive databases is described in Lakshmanan and Sadri's work [35] where belief and doubt can both be expressed explicitly with equal status. Probabilistic programs (p-programs) are finite sets of triples of the form:

$$\left(A \overset{c}{\leftarrow} B_1, \ldots, B_n; \mu_r, \mu_p\right)$$

As usual, A, B_1, $\ldots$, B_n are atoms, which may not contain complex terms, c is a confidence level, and μ_r (μ_p) is the conjunctive (disjunctive) mode associated with the rule. For a given ground atom A, the disjunctive mode associated with all the rules for A must be the same. Confidence levels are pairs of intervals $\langle [\alpha, \beta], [\gamma, \delta] \rangle$, the first component denoting the belief and the second doubt. This terminology is borrowed from Fitting's bilattices [25]. The confidence levels form a "trilattice" structure, and the truth-ordering is selected to be the underlying one for defining the semantics of p-programs. Due to the limitations in the language, the grounding of a p-program is finite. The disjunctive combination mode *positive correlation* corresponds to taking the least upper bound in the truth-ordering. The translation into a sorted monotonic logic program is more complex because the other disjunction modes do not have the properties of a least upper bound operator in the truth-ordering. The shape of translated p-rules is the following:

$$A \leftarrow \bigvee_{\mu_p}^{i=1,\ldots,m} \left(c_i \wedge_{\mu_{r_i}} \bigwedge_{\mu_{r_i}}^{j=1,\ldots,n_i} B_j^i \right)$$

such that we have m p-rules for A of the form:

$$\left(A \stackrel{c_i}{\leftarrow} B_1^i, \ldots, B_{n_i}^i ; \mu_{r_i}, \mu_p \right) \quad 1 \leq i \leq m$$

Mark that the body of the translated rule is finite since the number of rules for A is finite, and there is solely one rule for each atom in the language. The necessary arrow Σ-algebra has a single sort with the carrier being the trilattice under truth-ordering. The signature should contain function symbols $\vee_{md}$ and $\wedge_{md}$ corresponding to the combination modes described in [35], i.e. $md \in \{ig, ind, pc, nc, me\}$, standing for ignorance, independence, positive correlation, negative correlation, and mutual exclusion. Of course, we also require the constants for denoting the confidence levels in the trilattice.

The probabilistic programs of Lakshmanan and Sadri are a particular instance of the parametric framework presented in [36], where truth values are elements of an arbitrary complete lattice and rules are constructed from conjunctive, disjunctive, and propagation combination functions. The authors postulate some axioms describing the natural intent/properties of each of those combination functions. The translation into sorted monotonic logic programs is similar to the above, except that it is necessary to introduce the propagation function which combines the confidence level of the rule with the result of the application of the conjunctive function to the atoms in the body. We must say that the ideas and motivations of this framework are very similar to ours.

7.4 FRIL

One of the most successful logic programming frameworks for dealing with uncertainty is programming language Fril [4, 5], resting on the theory of mass assignment [3]. Fril unites both fuzzy and probabilistic reasoning with sophisticated inference mechanisms, used by commercial applications. It is difficult to correctly place and compare Fril within the panorama of semantics studied in this work, since neither a fixpoint operator nor a declarative semantics is provided. However, a very high-level analysis is still possible. Fril rules have the aspect:

< *head* > if < *body* >:< *list of support pairs* >

The support pairs provide intervals for the conditional properties of the head given the body, and in this respect it is similar to the ideas of Lukasiewicz [39]. There are three types of rules: basic, extended, and evidential logic. Each form corresponds to a different inference mechanism, they may be combined together, and can involve fuzzy sets. The basic inference rule of Fril is a generalized Jeffrey's rule and corresponds to an optimization problem. In general, we do not know how to encode this rule in sorted monotonic logic programming, and this constitutes an open research issue. Furthermore, it is not clear also whether the rule is monotonic or not. An evidential logic rule is simpler, and can be immediately captured in our framework but we state this without proof.

If fuzzy sets are not used in rules, then the elements in the underlying lattice are again closed intervals of $[0, 1]$, and least upper bound is intersection of intervals as in [17]. Like in AFLP [8], if the head of a rule returns a fuzzy set then the method of combining information from multiple rules is fuzzy set intersection. We intend to further explore the relationships of this work with sorted (monotonic ?) logic programming.

8 Conclusions

We have presented a sorted monotonic logic programming language capable of capturing and combining several reasoning paradigms dealing with uncertainty and vagueness. Many of the (recent!) semantics [22, 42, 1, 39, 34] are actually instances of work in quantitative deduction from the middle eighties [49, 52]. Even though starting from diverse motivations their final result is the same. The work by van Emden was generalized and given a logical interpretation by the Fuzzy Logic Programming of Vojtáš in his several works [54, 53].

Multi-adjoint logic programming [44, 45, 9] and residuated logic programming [12, 14] are each a further generalization of Fuzzy Logic Programming, no longer restricted to the real unit interval lattice $[0,1]$. As we have seen, single-sorted monotonic logic programming [12] adumbrates them all.

We argue that Monotonic Logic Programming is preferable to Generalized Annotated Logic Programming [32] because our framework requires smaller programs to represent the same knowledge. This was independently discovered by Vojtáš and his colleagues in a recent work [33]. Signed Formula Logic Programming is equivalent to Generalized Annotated Logic Programming and the same caveat applies to it [32, 37].

The probabilistic-based frameworks are more difficult to handle by sorted monotonic logic programs due to the complexity of inference in these systems. The general probabilistic logic programs of Lukasiewickz [39, 38] are not covered, as well as the very sophisticated and prominent Fril [4, 5]. We intend to explore these in subsequent work. However, both Hybrid Probabilistic Logic Programs [17] and Probabilistic Deductive Databases [35, 36] are captured, but do require more complex translations.

The many-sorted language was only really necessary for capturing Annotated Fuzzy Logic Programming [8]. All the other semantics can be already embedded in Monotonic Logic Programming, defined in [12]. We intend to extend this work with tabling proof procedures for first-order sorted multi-adjoint logic programs, relying on previous results about termination obtained in [10, 9]. The issue of disjunction and non-monotonicity is an on-going subliminal problem.

Acknowledgments

Some of the results and content arises from previous works with Manuel Ojeda-Aciego and Jesús Medina, specially sections 2 and 3. Warm thanks go for them. This paper also benefitted from discussions and/or sharing of work between ourselves and Peter Vojtáš and Trevor Martin.

References

[1] T. Alsinet and L. Godo. Towards an automated deduction system for first-order possibilistic logic programming with fuzzy constants. *International Journal of Intelligent Systems*, 17(9):887–924, September 2002.

[2] J. F. Baldwin. Evidential support of logic programming. *Fuzzy Sets and Systems*, 24(1):1–26, 1987.

[3] J. F. Baldwin. A theory of mass assignments for artificial intelligence. *Lecture Notes in Computer Science*, 833:22–??, 1994.

[4] J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. *Fril - Fuzzy and Evidential Reasoning in Artificial Intelligence.* Research Studies Press Ltd, 1995.

[5] J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. Applications of fuzzy computation: Knowledge based systems: Knowledge representation. In E. H. Ruspini, P. Bonnissone, and W. Pedrycz, editors, *Handbook of Fuzzy Computing.* IOP Publishing, 1998.

[6] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The ciao prolog system. reference manual. Technical Report CLIPS3/97.1, School of Computer Science, Technical University of Madrid (UPM), 1997. Available at http://www.cliplab.org/Software/Ciao/.

[7] J. Calmet, J. Lu, M. Rodriguez, and J. Schü. Signed formula logic programming: operational semantics and applications. In Z. W. Rás and M. Michalewicz, editors, *Proceedings of the Ninth International Symposium on Foundations of Intelligent Systems*, volume 1079 of *LNAI*, pages 202–211, Berlin, June 9–13 1996. Springer.

[8] T. Cao. Annotated fuzzy logic programs. *Fuzzy Sets and Systems*, 113(2):277–298, 2000.

[9] C. V. Damásio, J. Medina, and M. Ojeda-Aciego. Termination results for sorted multi-adjoint logic programs. Technical report, 2004. Submitted. Available at http://centria.di.fct.unl.pt/∼cd/publicacoes.

[10] C. V. Damásio and M. Ojeda-Aciego. On termination of a tabulation procedure for residuated logic programming. In A. Rubio, editor, *Proc. of the 6th International Workshop on Termination*, pages 40–43, Valencia, Spain, 2003. Available electronically at http://www.dsic.upv.es/∼rdp03/procs.

[11] C. V. Damásio and L. M. Pereira. Antitonic logic programs. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, volume 2173, pages 379–392, 2001.

[12] C. V. Damásio and L. M. Pereira. Monotonic and residuated logic programs. In S. Benferhat and P. Besnard, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 6th European Conference, ECSQARU 2001, Toulouse, France, September 19-21, 2001, Proceedings*, volume 2143 of *Lecture Notes in Computer Science*, pages 748–759. Springer, 2001.

[13] C. V. Damásio and L. M. Pereira. Monotonic and residuated logic programs. In *Slides of Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 6th European Conference*, 2001. Available at http://www.irit.fr/ECSQARU-2001/Ecsqaru-2001.html.

[14] C. V. Damásio and L. M. Pereira. Hybrid probabilistic logic programs as residuated logic programs. *Studia Logica*, 72(1):113–138, 2002.

[15] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order: Second Edition*. Cambridge University Press, 2002.

[16] A. Dekhtyar, M. I. Dekhtyar, and V. S. Subrahmanian. Temporal probabilistic logic programs. In D. D. Schreye, editor, *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4*, pages 109–123, 1999.

[17] A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. *Journal of Logic Programming*, 43(3):187–250, 2000.

[18] M. I. Dekhtyar, A. Dekhtyar, and V. S. Subrahmanian. Hybrid probabilistic programs: Algorithms and complexity. In K. B. Laskey and H. Prade, editors, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 160–169, S.F., Cal., July 30–Aug. 1 1999. Morgan Kaufmann Publishers.

[19] M. Denecker, V. Marek, and M. Truszczyński. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In J. Minker, editor, *Logic-Based Artifical Intelligence*, pages 127–144. Kluwer Academic Publishers, 2000.

[20] M. Denecker, N. Pelov, and M. Bruynooghe. Ultimate well-founded and stable semantics for logic programs with aggregates. In P. Codognet, editor, *Logic Programming, 17th International Conference, ICLP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*, volume 2237 of *Lecture Notes in Computer Science*. Springer, 2001.

[21] D. Dubois, J. Lang, and H. Prade. Fuzzy sets in approximate reasoning, part 2: logical approaches. *Fuzzy Sets and Systems*, 40:203–244, 1991.

[22] D. Dubois, J. Lang, and H. Prade. Towards possibilistic logic programming. In K. Furukawa, editor, *Proceedings of the 8th International Conference on Logic Programming*, pages 581–598. MIT, June 1991.

[23] D. Dubois, J. Lang, and H. Prade. Possibilistic logic. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, pages 439–513. Oxford University Press, Oxford, 1994.

[24] M. Fitting. Pseudo-Boolean valued Prolog. *Studia Logica*, XLVII(2):85–91, 1987.

[25] M. Fitting. Bilattices and the semantics of logic programs. *Journal of Logic Programming*, 11:91–116, 1991.

[26] M. Fitting. Fixpoint semantics for logic programming a survey. *Theoretical Computer Science*, 278(1–2):25–51, Apr. 2002.

[27] S. Guadarrama, S. Muñoz, and C. Vaucheret. Fuzzy prolog: A new approach using soft constraints propagation. *Fuzzy Sets and Systems, FSS*, 2002.

[28] D. Guller. Procedural semantics for fuzzy disjunctive programs. In M. Baaz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002, Tbilisi, Georgia, October 14-18, 2002, Proceedings*, volume 2514 of *Lecture Notes in Computer Science*, pages 247–261. Springer, 2002.

[29] D. Guller. Semantics for fuzzy disjunctive programs with weak similarity. In A. Abraham and M. Köppen, editors, *Hybrid Information Systems, First International Workshop on Hybrid Intelligent Systems, Adelaide, Australia, December 11-12, 2001, Proceedings*, Advances in Soft Computing, pages 285–299. Physica-Verlag, 2002.

[30] R. Hähnle. Uniform notation of tableaux rules for multiple-valued logics. In *Proc. International Symposium on Multiple-Valued Logic, Victoria*, pages 238–245. IEEE Press, Los Alamitos, 1991.

[31] P. Hájek. *Metamathematics of Fuzzy Logic*. Trends in Logic. Studia Logica Library. Kluwer Academic Publishers, 1998.

[32] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *The Journal of Logic Programming*, 12(1, 2, 3 & 4):335–367, 1992.

[33] S. Krajči, R. Lencses, and P. Vojtáš. A comparison of fuzzy and annotated logic programming. *Fuzzy sets and systems*. To appear.

[34] P. Kullmann and S. Sandri. Possibilistic logic as an annotated logic. *Fuzzy sets and systems*. To appear.

[35] L. V. S. Lakshmanan and F. Sadri. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Programming*, 1(1):5–42, Jan. 2001.

[36] L. V. S. Lakshmanan and N. Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions Knowledge Data Engineering*, 13(4):554–570, 2001.

[37] J. J. Lu. Logic programming with signs and annotations. *Journal of Logic and Computation*, 6(6):755–778, Dec. 1996.

[38] T. Lukasiewicz. Logic programming and nonmonotonic reasoning, 6th international conference, lpnmr 2001, vienna, austria, september 17-19, 2001, proceedings. *Lecture Notes in Computer Science*, 2173:336–350, 2001.

[39] T. Lukasiewicz. Probabilistic logic programming with conditional constraints. *ACM Trans. Comput. Logic*, 2(3):289–339, 2001.

[40] T. Lukasiewicz. Probabilistic logic programming with conditional constraints. *ACM Transactions on Computational Logic*, 2(3):289–339, 2001.

[41] P. Magrez and P. Smets. Fuzzy modus ponens: a new model suitable for applications in knowledge-based systems. *International Journal of Intelligent Systems*, 4:181–200, 1989.

[42] V. W. Marek and M. Truszczyński. Logic programming with costs. Technical report, University of Kentucky, 2000. Available at ftp://al.cs.engr.uky.edu/cs/manuscripts/lp-costs.ps.

[43] T. P. Martin, J. F. Baldwin, and B. W. Pilsworth. The implementation of FPROLOG – A fuzzy prolog interpreter. *Fuzzy Sets and Systems*, 23(1):119–129, 1987.

[44] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, volume 2173, pages 351–364, 2001.

[45] J. Medina, M. Ojeda-Aciego, and P. Vojtás. A multi-adjoint approach to similarity-based unification. In P. Eklund and M. Ojeda-Aciego, editors, *Electronic Notes in Theoretical Computer Science*, volume 66. Elsevier, 2002.

[46] R. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, Dec. 1992.

[47] H. Reichenbach. Reply to Ernest Nagel's criticism of my views on quantum mechanics. *Journal of Philosophy*, 43:239–247, 1946.

[48] M. I. Sessa. Approximate reasoning by similarity-based sld resolution. *Theoretical Computer Science*, 275:389–426, 2002.

[49] E. Y. Shapiro. Logic programs with uncertainties: A tool for implementing rule-based

systems. In A. Bundy, editor, *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 529–532, Karlsruhe, FRG, Aug. 1983. William Kaufmann.

[50] T. Swift. Tabling for non-monotonic programming. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):201–240, 1999.

[51] M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of ACM*, 4(23):733–742, 1976.

[52] M. H. van Emden. Quantitative deduction and its fixpoint theory. *The Journal of Logic Programming*, 3(1):37–54, Apr. 1986.

[53] P. Vojtáš. Fuzzy logic programming. *Fuzzy sets and systems*, 124(3):361–370, 2001.

[54] P. Vojtáš and L. Paulík. Soundness and completeness of non-classical sld-resolution. In R. Dyckhoff, H. Herre, and P. Schroeder-Heister, editors, *Extensions of Logic Programming, 5th International Workshop, ELP'96, Leipzig, Germany, March 28-30, 1996, Proceedings*, volume 1050 of *Lecture Notes in Computer Science*, pages 289–301. Springer, 1996.