Termination Results for Sorted Multi-Adjoint Logic Programs

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

J. Medina
Dept. Matemática Aplicada
Univ. de Málaga
Spain
jmedina@ctima.uma.es

M. Ojeda-Aciego
Dept. Matemática Aplicada
Univ. de Málaga
Spain
aciego@ctima.uma.es

Abstract

In this paper we present a logic programming-based language allowing for the combination of several adjoint lattices of truth-values. A model and fixpoint theory are presented, but the main contribution of the paper is the study of general properties guaranteeing termination of all queries. New results are presented and related to other alternative formalisms.

Keywords: Fuzzy Logic Programming, Termination Results, Probabilistic Deductive Databases

1 Introduction

The interest in the development of logics for dealing with information which might be either vague or uncertain has increased in the recent years. Several different approaches to the so-called inexact or fuzzy or approximate reasoning have been proposed, involving either fuzzy or annotated or probabilistic or similarity-based logic programming, e.g. [1, 7, 8, 13, 6, 5, 15, 9].

Our proposal uses a sorted language, where each sort identifies an underlying lattice of truth-values (weights) which must satisfy adjoint conditions. This allows, for instance, the combination of arbitrary t-norms and t-conorms, with other operators. This seems very appropriate for performing and representing several reasoning tasks with imprecise and incomplete information, and is based on the proposal of Lakshmanan and Sadri [9] for probabilistic deductive databases. We restrict to the ground case but allow infinite programs, and thus do not loose generality.

The semantics of sorted multi-adjoint logic program is characterised, as usual, by the post-fixpoints of the immediate consequence operator $T_{\mathbb{P}}$, which is proved to be monotonic and continuous under very general hypotheses, see [10]. The current proposal is an important enhancement of our previous works [2, 3, 4, 10, 11].

The major contributions of this paper are the termination results for several classes of sorted multi-adjoint logic programs, extending or complementing existing results in the literature [8, 12, 6, 9, 5]. In particular, the case of programs obtained by arbitrary composition of operators obeying the boundary condition $\vartheta \otimes 1 = 1 \otimes \vartheta \leq \vartheta$ over the unit interval are shown to be terminating. As argued, the results extend to the first-order case.

The structure of the paper is as follows. In Section 2, we introduce the preliminary concepts necessary for the definition of the syntax and semantics of sorted multi-adjoint logic programs, presented in Section 3. In Section 4, we state several results regarding the termination properties of our semantics. Section 5 presents some comparisons to existing termination results for other proposals. The paper finishes with some conclusions and pointers to future work. The proof of our main theorem is annexed.

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 [4], is introduced below.

2.1 Some Definitions from Universal Algebra

The notions of signature and Σ-algebra will allow the interpretation of the function and constant symbols in the language, as well as for specifying the 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 different pairs. The number k is the arity of f. If k is 0 then f is a constant symbol.*

To simplify notation, we write $f{:}\tau$ 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 $f{:}\, s_1 \times \cdots \times s_k \to s \in F$, where $k > 0$, and an element $I(c) \in A^s$ to each constant symbol $c : s$ in F.

2.2 Multi-Adjoint Lattices and Multi-Adjoint Algebras

The main concept we will need in this section is that of *adjoint pair*.

Definition 3 (Adjoint pair) *Let $\langle P, \preceq\rangle$ be a partially ordered set and $(\leftarrow, \&)$ a pair of binary operations in P such that:*

(a1) Operation $\&$ is increasing in both arguments

(a2) Operation $\leftarrow$ is increasing in the first argument (the consequent) and decreasing in the second argument (the antecedent).

(a3) For any $x, y, z \in P$, we have that

$$x \preceq (y \leftarrow z) \quad \textit{iff} \quad (x \,\&\, z) \preceq y$$

Then we say that $(\leftarrow, \&)$ forms an adjoint pair *in $\langle P, \preceq\rangle$.*

Extending the results in [4, 3, 15] to a more general setting, in which different implications (Łukasiewicz, Gödel, product) and thus, several modus ponens-like inference rules are used, naturally leads to considering several *adjoint pairs* in the lattice. More formally,

Definition 4 (Multi-Adjoint Lattice) *Let $\langle L, \preceq\rangle$ be a lattice. A* multi-adjoint lattice $\mathcal{L}$ *is a tuple $(L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ satisfying the following items:*

(l1) $\langle L, \preceq\rangle$ is bounded, i.e. it has bottom ($\bot$) and top ($\top$) elements;

(l2) $(\leftarrow_i, \&_i)$ is an adjoint pair in $\langle L, \preceq\rangle$ for $i = 1, \ldots, n$;

(l3) $\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$ for all $\vartheta \in L$ for $i = 1, \ldots, n$.

Remark 1 *Note that residuated lattices are a special case of multi-adjoint lattice, in which the underlying poset has a lattice structure, has monoidal structure wrt $\&$ and $\top$, and only one adjoint pair is present.*

From the point of view of expressiveness, it is interesting to allow extra operators to be involved with the operators in the multi-adjoint lattice. The structure which captures this possibility is that of a multi-adjoint algebra.

Definition 5 (Multi-Adjoint Σ-Algebra) *A Σ-Algebra $\mathfrak{L}$ is a* Multi-Adjoint Σ-Algebra *whenever:*

- *The carrier L^s of each sort is a lattice under a partial order $\preceq^s$.*

- *Each sort s contains operators $\leftarrow_i^s$: $s \times s \to s$ and $\&_i^s : s \times s \to s$ for $i = 1, \ldots, n^s$ (and possibly some extra operators) such that the tuple $\mathcal{L}^s$*

  $$(L^s, \preceq^s, I(\leftarrow_1^s), I(\&_1^s), \ldots, I(\leftarrow_n^s), I(\&_n^s))$$

  is a multi-adjoint lattice.

A practical application of Multi-Adjoint Σ-Algebras can be found in the probabilistic deductive databases framework of Lakshmanan and Sadri [9] where our sorts correspond to disjunctive modes and the adjoint operators to different conjunctive modes for combining probabilistic knowledge. Our framework is richer since we do not restrain ourselves to a single and particular carrier set and allow more operators.

In practice, we will usually have to assume some properties on the extra operators considered. These extra operators will be assumed to be either aggregators, or conjunctors or disjunctors, all of which are monotone functions (the latter, in addition, are required to generalize their Boolean counterparts).

Note that the use of aggregators as weighted sums somehow covers the approach taken in [1] when considering the evidential support logic rules of combination.

3 Syntax and Semantics of Sorted Multi-Adjoint Logic Programs

Sorted multi-adjoint logic programs will be constructed from the abstract syntax induced by a multi-adjoint Σ-algebra on a set of sorted propositional symbols (or variables). Specifically, we will consider a multi-adjoint Σ-algebra $\mathfrak{L}$ whose extra operators can be arbitrary monotone operators. This algebra will host the manipulation the truth-values of the formulas 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 will often write $A \in \Pi^s$.

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

Remark 2 *As we are working with two Σ-algebras, and to discharge the notation, we introduce a special notation to clarify which algebra a function symbols belongs 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.*

3.1 Syntax of Sorted Multi-Adjoint Logic Programs

The definition of sorted multi-adjoint logic program is given, as usual, as a set of rules and facts. The particular syntax of these rules and facts is given below:

Definition 6 (Sorted MA Logic Programs) *A* sorted multi-adjoint logic program *is a set $\mathbb{P}$ of rules of the form $\langle A \leftarrow_i^s \mathcal{B}, \vartheta \rangle$ such that:*

1. *The* rule *$(A \leftarrow_i^s \mathcal{B})$ is a formula (an algebraic term) of $\mathfrak{F}$;*
2. *The* weight *ϑ is an element (a truth-value) of $\mathcal{L}^s$;*
3. *The* head *of the rule A is a propositional symbol of Π of sort s.*
4. *The* body *formula $\mathcal{B}$ 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.*
5. Facts *are rules with body $\top^s$, the top element of lattice $\mathcal{L}^s$.*
6. *A* query *(or* goal*) is a propositional symbol intended as a question $?A$ prompting the system.*

3.2 Semantics of Sorted Multi-Adjoint Logic Programs

Definition 7 (Interpretation) *An* interpretation *is a mapping $I : \Pi \to \bigcup_s L^s$ 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 orderings $\preceq^s$ of the truth-values L^s can be easily extended to the set of interpretations as follows:

Definition 8 (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 least element $\bot^s \in \mathcal{L}^s$.*

A rule of a sorted multi-adjoint logic program is satisfied whenever the truth-value of the rule is greater or equal than the weight associated with the rule. Formally:

Definition 9 (Satisfaction, Model) *Given an interpretation $I \in \mathcal{I}_{\mathfrak{L}}$, a weighted rule $\langle A \leftarrow_i^s \mathcal{B}, \vartheta \rangle$ is* satisfied *by I iff $\vartheta \preceq^s \hat{I}(A \leftarrow_i^s \mathcal{B})$. An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$ is a* model *of a sorted multi-adjoint logic program $\mathbb{P}$ iff all weighted rules in $\mathbb{P}$ are satisfied by I.*

Definition 10 *An element $\lambda \in \mathcal{L}^s$ is a* correct answer *for a program $\mathbb{P}$ and a query $?A$ of sort s if for an arbitrary interpretation I which is a model of $\mathbb{P}$ we have $\lambda \preceq^s I(A)$.*

The immediate consequences operator, given by van Emden and Kowalski, can be easily generalised to the framework of sorted multi-adjoint logic programs.

Definition 11 *Let $\mathbb{P}$ be a sorted multi-adjoint logic program. The* immediate consequences operator *$T_{\mathbb{P}}$ maps interpretations to interpretations, and is defined by*

$$T_{\mathbb{P}}(I)(A) = \bigsqcup_s \{\vartheta \,\dot{\&}_i^s\, \hat{I}(\mathcal{B}) \mid \langle A \leftarrow_i^s \mathcal{B}, \vartheta \rangle \in \mathbb{P}\}$$

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

The semantics of a sorted multi-adjoint logic program can be characterised, as usual, by the post-fixpoints of $T_{\mathbb{P}}$; that is, an interpretation I is a model of a sorted multi-adjoint logic program $\mathbb{P}$ iff $T_{\mathbb{P}}(I) \sqsubseteq I$. The single-sorted $T_{\mathbb{P}}$ operator is proved to be monotonic and continuous under very general hypotheses, see [10, 11], and it is remarkable that these results are true even for non-commutative and non-associative conjunctors. In particular, by continuity, the least model can be reached in at most countably many iterations of $T_{\mathbb{P}}$ on the least interpretation. These results immediately extend to the sorted case.

4 Termination Results

In this section we analyse the termination properties of the $T_{\mathbb{P}}$ operator, and show new results for some classes of sorted multi-adjoint logic programs. In what follows we assume that every operator is computable. If only monotone and continuous operators are present in the underlying sorted multi-adjoint Σ-algebra $\mathfrak{L}$ then the immediate consequences operator reaches the least fixpoint at most after ω iterations. The following adaptation of an example due to [8] shows that, even for finite programs, ω iterations may be necessary to reach the least fixpoint:

Example 1 *Consider the following single-sorted multi-adjoint logic program*

$$a \xleftarrow{1.0}{}^{unit} f(a)$$

over the lattice $\mathcal{L}^{unit} = ([0,1], \leq, \leftarrow_G, min)$ with Gödel's adjoint pair: minimum t-norm and corresponding residuum. The extra connective function symbol f has signature $f : unit \to unit$, where $[0,1]$ is the carrier of sort unit, and $I(f)$ is the function:

$$I(f): \begin{array}{ccc} [0,1] & \longrightarrow & [0,1] \\ x & \mapsto & \frac{1+x}{2} \end{array}$$

We present below several results in order to guarantee that every query can be answered after a finite number of iterations. In particular, this means that for finite programs the least fixpoint of $T_{\mathbb{P}}$ can also be reached after

a **finite** number of iterations, ensuring computability of the semantics.

Definition 12 (Termination) *Let $\mathbb{P}$ be a sorted multi-adjoint logic program with respect to a multi-adjoint Σ-algebra $\mathfrak{L}$ and a sorted set of propositional symbols Π. We say that $T_{\mathbb{P}}$* terminates for every query *iff for every propositional symbol A there is a finite n such that $T^n_{\mathbb{P}}(\triangle)(A)$ is identical to $lfp(T_{\mathbb{P}})(A)$.*

In order to not limit the discussion to finite programs, our results will be applicable to special classes of infinite sorted multi-adjoint logic programs, designated finitary.

Definition 13 (Finitary programs) *A sorted multi-adjoint logic program such that, for every propositional symbol A the number of rules with head A is finite, is said to be* finitary*.*

The *dependency graph* of $\mathbb{P}$ has a vertex for each propositional symbol in Π, and there is an arc from a propositional symbol A to a propositional symbol B iff A is the head of a rule with body containing an occurrence of B. The dependency graph for a propositional symbol A is the subgraph of the dependency graph containing all nodes accessible from A and corresponding edges. A special case of finitary programs is:

Definition 14 (Finite dependencies) *A sorted multi-adjoint logic program $\mathbb{P}$ has* finite dependencies *iff for every propositional symbol A the number of edges in the dependency graph for A is finite.*

A first immediate result is that all queries are computable for acyclic sorted multi-adjoint logic programs with finite dependencies:

Theorem 1 *Let $\mathbb{P}$ be a sorted multi-adjoint logic programs with respect a the multi-adjoint Σ-algebra $\mathfrak{L}$. If $\mathbb{P}$ has finite dependencies and the dependency graph does not contain cycles then $T_{\mathbb{P}}$ terminates for every query.*

Clearly, if the program is finite then the above theorem reduces to checking of cycles in the dependency graph of the program. An usual way of guaranteeing that the dependency subgraph, generated from a first-order program, is finite for every propositional symbol A is to assume the bounded term-size property [14], i.e. that the complexity of atoms in the body of programs is less than that of the head.

Another straightforward sufficient condition for termination arises when considering the cardinality of the set of computable values of the program:

Construct the signature Σ' from the weights of the rules in $\mathbb{P}$, the constant and function symbols in Σ occurring in the bodies of $\mathbb{P}$, all the adjoint operators $\&^s_i$, and a new function symbol $\sqcup_s$ for each sort s of type $s \times s \to s$ interpreted as the join of $\mathcal{L}^s$, and let:

$$\mathfrak{V}^s = \left\{ \hat{\triangle}(t) \text{ such that } t \in Terms^s(\Sigma') \right\}$$

where $Terms^s(\Sigma')$ are the algebraic terms of sort s.

Theorem 2 *Let $\mathbb{P}$ be a sorted multi-adjoint logic program with respect to a multi-adjoint Σ-algebra $\mathfrak{L}$, and the set of sorted propositional symbols Π.*

If $\mathfrak{V}^s$ does not have infinite ascending chains of values for all s, then $T_{\mathbb{P}}$ operator terminates for every query over program $\mathbb{P}$.

Corollary 1 *If the carriers of the multi-adjoint Σ-algebra $\mathfrak{L}$ are all finite then $T_{\mathbb{P}}$ terminates for every program $\mathbb{P}$.*

The intuition of the last corollary is that if all the combinations of operators in the program with least upper bound operators generate values in a finite subset of all possible truth-values, then it is impossible to generate infinite ascending chains for each propositional symbol A and thus the $T_{\mathbb{P}}$ must terminate foe every query. The following is an extension of previous results in [2] for finite programs:

Theorem 3 *Consider the single-sorted Σ-algebra $\mathfrak{L}$ over the unit interval $[0, 1]$ where the only operators are a t-norm and its residuum. Let $\mathbb{P}$ be a finitary sorted multi-adjoint logic program with respect to $\mathfrak{L}$, and the set of*

sorted propositional symbols Π*, such that the set of truth-values occurring in* $\mathbb{P}$ *is finite; then operator* $T_{\mathbb{P}}$ *terminates for every query.*

Proof. Under these conditions it is not possible to construct infinite ascending chains of values. $\Diamond$

We proceed by presenting our new major termination result valid for an important class of sorted multi-adjoint logic programs, where neither acyclicity nor finiteness properties are required:

Definition 15 *A multi-adjoint* Σ*-algebra is said to be* local *when the following conditions are satisfied:*

- *For every pair of sorts* s_1 *and* s_2 *there is a unary monotone casting function symbol* $c_{s_1 s_2} : s_2 \longrightarrow s_1$ *in* Σ*.*
- *All other function symbols have types of the form* $f : s \times \cdots \times s \longrightarrow s$*, i.e. are closed operations in each sort, satisfying the following boundary conditions for every* $v \in \mathcal{L}^s$*:*

$$\begin{array}{rcl} I(f)(v, 1^s, \ldots, 1^s) & \preceq^s & v \\ I(f)(1^s, v, 1^s, \ldots, 1^s) & \preceq^s & v \\ & \vdots & \\ I(f)(1^s, \ldots, 1^s, v) & \preceq^s & v \end{array}$$

  where 1^s *is the top element of* $\mathcal{L}^s$*. In particular, if* f *is a unary function symbol then* $I(f)(v) \preceq^s v$*.*

- *The following property is obeyed:*

$$\left(c_{ss_1} \circ c_{s_1 s_2} \circ \ldots \circ c_{s_n s}\right)(v) \preceq^s v$$

  for every $v \in \mathcal{L}^s$ *and finite composition of casting functions with overall sort* $s \longrightarrow s$*.*

In local sorted multi-adjoint Σ-algebras the non-casting function symbols are restricted to operations in a unique sort. In order to combine values from different sorts, one is deemed to use explicitly the casting functions in the appropriate places. This restriction simplifies the proof of our main result.

Definition 16 (Relevant values/Culprits) *Let* $\mathbb{P}$ *be a multi-adjoint program, and* $A \in \Pi^s$*. The set* $R^I_{\mathbb{P}}(A)$ *of* relevant values *for* A *with respect to interpretation* I *is the set of maximal values of the set*

$$\left\{ \vartheta \,\dot{\&}^s_i\, \hat{I}(\mathcal{B}) \;\mid\; \langle A \leftarrow^s_i \mathcal{B}, \vartheta \rangle \in \mathbb{P} \right\}$$

The culprit set *for* A *with respect to* I *is the set of rules* $\langle A \leftarrow^s_i \mathcal{B}, \vartheta \rangle$ *of* $\mathbb{P}$ *such that* $\vartheta \,\dot{\&}^s_i\, \hat{I}(\mathcal{B})$ *belongs to* $R^I_{\mathbb{P}}(A)$*. Rules in a culprit set are called* culprits*.*

The rationale is to use the set of relevant values for a propositional symbol A to collect the maximal values contributing to the computation of A in an iteration of the $T_{\mathbb{P}}$ operator. The non-maximal values are irrelevant for determining the new value for A by $T_{\mathbb{P}}$. Assuming the finitary condition, the set of rule values is always non-empty and finite, and thus infinite ascending chains of rule values cannot occur. The culprits are the contributing rules for relevant values. These concepts are used in the proof of the following major new result:

Theorem 4 *Let* $\mathbb{P}$ *be a sorted multi-adjoint logic program with respect to a local multi-adjoint* Σ*-algebra* $\mathfrak{L}$ *and the set of sorted propositional symbols* Π*, and having finite dependencies.*

If for every iteration n *and propositional symbol* A *of sort* s *the set of relevant values for* A *with respect to* $T^n_{\mathbb{P}}(\triangle)$ *is a singleton, then* $T_{\mathbb{P}}$ *terminates for every query .*

Proof. Included in the appendix. $\Diamond$

From the above result, the following corollaries are immediate, under the global assumption of being interpreted in a *local* multi-adjoint Σ-algebra:

Corollary 2 *If the conditions of Theorem 4 are fulfilled then at most* m *iterations of* $T_{\mathbb{P}}$ *are necessary to answer query* A*, where* m *is the number of rules in the dependency graph for* A*.*

Corollary 3 *If all the carrier lattices* $\mathcal{L}^s$ *are totally ordered then* $T_{\mathbb{P}}$ *terminates for every*

query over any program $\mathbb{P}$ *having finite dependencies.*

Proof. It is easy to see that $R^I_{\mathbb{P}}(A)$ contains at most one element, by finiteness of the rules for A and the fact that $\preceq^s$ is a total order. $\Diamond$

An important instance of the above is the case of the unit interval:

Corollary 4 *If the carrier of each sort* s *is the unit interval* $[0, 1]$ *then* $T_{\mathbb{P}}$ *terminates for every query over any program* $\mathbb{P}$ *having finite dependencies.*

Proof. Obvious from the fact that $[0, 1]$ is totally ordered and that there are no casting functions. $\Diamond$

Clearly, programs where only t-norms over the unit interval are used in weighted rules are catered by the previous result.

5 Comparisons

The seminal work by van Emden [13] presents a syntax and semantics for quantitative rules. These quantitative rules use product t-norm and corresponding residuum operation for defining the semantics of the $\leftarrow$ symbol and weights in the rules, and the Gödel t-norm (minimum operation) to combine atoms in the body. Thus, after grounding of quantitative programs, it is obtained a single-sorted multi-adjoint logic program over the unit interval. A termination result for arbitrary finite first-order quantitative programs is presented. The proof is based on the impossibility of constructing infinite ascending chains, as in Theorem 2. However, the proof procedure described in [13] assumes finite dependencies and thus our Corollary 4 generalizes these results.

Generalized Annotated Logic Programs (GAPs) are one of the most important formalisms for dealing with uncertainty in rule-based expert systems [8]. If all operators are continuous, then the semantics of single-sorted multi-adjoint logic programs can be captured by GAPs with only variable annotations in the bodies but with complex annotations in the heads. However, most of our results do not assume continuity conditions of the operators and thus our results complement the ones appearing in [8], namely Theorem 4 and its corollaries. The exact relations between the two semantics hinges upon the existence of translations from sorted multi-adjoint logic programs into GAPs, which we intend to fully explore.

More recently, Hybrid Probabilistic Logic Programs [6] have been proposed for constructing rule systems which allow the user to reason with and combine probabilistic information under different probabilistic strategies. 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. The termination results presented in [5] either only allow constant annotation or (finite) ground programs. From the analysis of the fixpoint construction one can see that only a finite number of different intervals can be generated. Thus by a simple cardinality argument the termination results follow.

The use of a sorted language is described in Lakshmanan and Sadri's work [9] for defining a theory of probabilistic deductive databases via p-programs. The syntax allows for variables but not arbitrary first-order terms, thus from a theoretical point of view all these programs can be considered finite. Several disjunctive and conjunctive modes for combining events are presented. The truth-values (confidence levels) are pairs of intervals. A disjunctive mode corresponds to a sort in our programs, while a conjunctive mode is related to our adjoint operators. The atoms occurring in a body can be combined with a conjunctive mode, but different rules for the same proposition may use different conjunctive modes. Ground p-programs can be immediately translated to our framework. The authors present a termination result for the case of p-program having rules combined solely with positive correlation mode, but arbitrary conjunctive modes. The positive correlation modes corresponds to maximum and minimum operations over the unit interval. The confidence levels are combined via t-norms

and t-conorms, independently in the several component of the confidence levels. These results carry over to our setting, extensions to our framework are under study, but are related to applications of Corollary 3.

Paulík proved a termination result for fuzzy SLD-resolution in a context which can now be seen as a particular case of our general approach. In [12], by using a result in the line of our Corollary 1, the following completeness theorem was proved:[2] Given a first-order fuzzy logic program $\mathbb{P}$ built from one adjoint pair whose conjunctor is a continuous t-norm (and no extra operators), then for all query A, the sequence $\{T^n_{\mathbb{P}}(\triangle)(A)\}$ is eventually constant.

Since all *t-norms* obey to the property $\vartheta \otimes_t 1 = 1 \otimes_t \vartheta = \vartheta$ Paulík's result can be generalized in order to allow arbitrary combinations of t-norms. Our results also apply to the recent framework of Fuzzy Logic Programming [15], which can be seen as single-sorted multi-adjoint logic programs.

6 Conclusions

We have presented a sorted multi-adjoint logic programming language, capable of capturing and combining several reasoning paradigms dealing with imprecision and uncertainty. Several important termination results are presented and compared with other ones in the literature. We intend to extend this work with tabling proof procedures for first-order sorted multi-adjoint logic programs, relying in the above results. The embedding of other proposals in the literature into our framework will be explored in subsequent work.

References

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

[2] C.V. Damásio and M. Ojeda-Aciego On termination of a tabulation procedure for residuated logic programming. 6th Intl Workshop on Termination, pp. 40-43, 2003

[3] C. V. Damásio and L. M. Pereira. Monotonic and residuated logic programs. Lect. Notes in Artificial Intelligence 2143, pp. 748–759, 2001.

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

[5] M. Dekhtyar, A. Dekhtyar and V.S. Subrahmanian. Hybrid Probabilistic Programs: Algorithms and Complexity. Proc. of Uncertainty in AI'99 conference, 1999

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

[7] D. Dubois, J. Lang and H. Prade. Towards Possibilistic Logic Programming. Proc. of International Conference on Logic Programming, pp. 581–598, MIT Press, 1991

[8] M. Kifer and V. S. Subrahmanian, Theory of generalized annotated logic programming and its applications. *J. of Logic Programming* 12(4):335–367, 1992

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

[10] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. Lect. Notes in Artificial Intelligence 2173, pp. 351–364, 2001.

[11] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. Lect. Notes in Artificial Intelligence 2258, pp. 290–297, 2001.

[12] L. Paulík. Best possible answer is computable for fuzzy SLD-resolution. Lecture Notes on Logic 6, pp. 257–266, 1996.

[13] M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 4(1):37–53, 1986.

[14] A. van Gelder Negation as failure using tight derivations for general logic programs. Foundations of deductive databases and logic programming, pp. 149–176, Morgan Kaufmann Publishers Inc., 1988

[15] P. Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124(3):361–370, 2001.

[2] Stated in a different terminology, we have written the statement with the notation a names used in this paper.

Proof of Theorem 4

Let the *culprit collection* for $T^n_{\mathbb{P}}(\triangle)(A)$ be the set of culprits used in the tree of recursive calls of $T_{\mathbb{P}}$ in the calculation. We proceed by induction on n, showing that if $T^{n+1}_{\mathbb{P}}(\triangle)(A) \succ^s T^n_{\mathbb{P}}(\triangle)(A)$ for $A \in \Pi$, then the culprit collection for $T^{n+1}_{\mathbb{P}}(\triangle)(A)$ has cardinality at least $n+1$. Since the number of rules in the dependency graph for for A is finite then the $T_{\mathbb{P}}$ operator must terminate after a finite number of steps, by using all the rules relevant for the computation of A.

Base case: For $n = 0$, consider $A \in \Pi^s$ and assume $T^1_{\mathbb{P}}(\triangle)(A) \succ^s T^0_{\mathbb{P}}(\triangle)(A) = \triangle(A)$ and then, by definition of $T_{\mathbb{P}}$, we must have used at least one rule, and thus the culprit collection contains at least one element.

Induction step: Now, we assume as the induction hypothesis that given $B \in \Pi^t$ such that $T^n_{\mathbb{P}}(\triangle)(B) \succ^t T^{n-1}_{\mathbb{P}}(\triangle)(B)$, then the culprit collection for $T^n_{\mathbb{P}}(\triangle)(B)$ has at least n different rules for all sorts t and $B \in \Pi$.

Let $A \in \Pi^s$ and assume $T^{n+1}_{\mathbb{P}}(\triangle)(A) \succ^s T^n_{\mathbb{P}}(\triangle)(A)$, then there is at least one rule in the program, $\langle A \leftarrow^s_i \mathcal{B}, \vartheta\rangle$, such that

$$T^{n+1}_{\mathbb{P}}(\triangle)(A) = \vartheta \,\dot{\&}^s_i\, \widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B})$$

Summing up, we have:

$$\begin{aligned} T^{n+1}_{\mathbb{P}}(\triangle)(A) &= \vartheta \,\dot{\&}^s_i\, \widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B}) \\ &\succ^s T^n_{\mathbb{P}}(\triangle)(A) \\ &\succ^s \vartheta \,\dot{\&}^s_i\, \widehat{T^{n-1}_{\mathbb{P}}(\triangle)}(\mathcal{B}) \end{aligned}$$

By monotonicity of the $T_{\mathbb{P}}$ operator and of $\dot{\&}^s_i$ then there must be at least one propositional symbol $C \in \Pi^u$ occurring in the body $\mathcal{B}$ which changed value from step $n-1$ to step n, i.e.

$$T^n_{\mathbb{P}}(\triangle)(C) \succ^u T^{n-1}_{\mathbb{P}}(\triangle)(C)$$

Applying the induction hypothesis, at least n different rules are in the culprit collection of $T^n_{\mathbb{P}}(\triangle)(C)$, and belong to the dependency graph for A since C occurs in the body of a rule for A. We will prove by contradiction that $\langle A \leftarrow^s_i \mathcal{B}, \vartheta\rangle$ is not in that culprit collection.

$$\begin{array}{ccc} & \&^s_i & \\ \vartheta & c_{ss_1} & \\ & f_1 & \\ \vdots & c_{s_1s_2} & \vdots \\ & f_2 & \\ \vdots & \vdots & \vdots \\ & f_k & \\ T_1 & c_{s_ks} & T_s \\ & T^m_{\mathbb{P}}(\triangle)(A) & \end{array}$$

Figure 1: Computation term for $T^{n+1}_{\mathbb{P}}(\triangle)(A)$

Assume that there exists $m < n+1$ such that $\langle A \leftarrow^s_i \mathcal{B}, \vartheta\rangle$ is also a culprit for $T^m_{\mathbb{P}}(\triangle)(A)$. In this case, we can view the computation performed by the $T_{\mathbb{P}}$ operator as the evaluation of the term in Fig 1, where each $c_{s_is_j}$ is either a casting function or the identity function on sort s c_{ss}, and T_i's are again terms. Furthermore, there are no occurrences of propositional symbols in the above term.

By the boundary condition one can easily conclude that

$$T^{n+1}_{\mathbb{P}}(\triangle)(A) \preceq^s \dot{c_{ss_1}}\left(\ldots...(\dot{c_{s_ks}}\left((T^m_{\mathbb{P}}(\triangle)A)\right)\right)$$

Now, by resorting to the properties of the casting functions we will obtain that:

$$T^{n+1}_{\mathbb{P}}(\triangle)(A) \preceq^s T^m_{\mathbb{P}}(\triangle)(A) \qquad (1)$$

obtaining a contradiction with the monotonicity of $T_{\mathbb{P}}$ since

$$T^{n+1}_{\mathbb{P}}(\triangle)(A) \succ^s T^n_{\mathbb{P}}(\triangle)(A) \succeq^s T^m_{\mathbb{P}}(\triangle)(A)$$

For the proof of inequality (1) recall that, for the function operator f_k in the above term we know that:

$$\dot{T_1} \preceq^{s_k} 1^{s_k} \quad \ldots \quad \dot{T_s} \preceq^{s_k} 1^{s_k}$$

By the boundary conditions we conclude immediately that

$$\dot{f}_k\left(\dot{T}_1,\ldots,\dot{c_{s_k s}}\left(T_{\mathbb{P}}^m(\triangle)(A)\right),\ldots,\dot{T}_s\right) \preceq^{s_k} \\ \preceq^{s_k} \quad \dot{c_{s_k s}}\left(T_{\mathbb{P}}^m(\triangle)(A)\right)$$

This argument can be applied to any function symbol in the computation tree.

As a result, we obtain that the culprit collection for $T_{\mathbb{P}}^{n+1}(\triangle)(A)$ has cardinality at least $n+1$, and the theorem is proved.