Sorted multi-adjoint logic programs: termination results and applications

C.V. Damásio[1], J. Medina[2], and M. Ojeda-Aciego[2]

[1] Centro Inteligência Artificial. Universidade Nova de Lisboa. `cd@di.fct.unl.pt`
[2] Dept. Matemática Aplicada. Univ. de Málaga. {`jmedina,aciego`}`@ctima.uma.es`

Abstract. A general framework of logic programming allowing for the combination of several adjoint lattices of truth-values is presented. The main contribution is a new sufficient condition which guarantees termination of all queries for the fixpoint semantics for an interesting class of programs. Several extensions of these conditions are presented and related to some well-known formalisms for probabilistic logic programming.

1 Introduction

In the recent years there has been an increasing interest in models of reasoning under "imperfect" information. As a result, a number of approaches have been proposed for the so-called inexact or fuzzy or approximate reasoning, involving either fuzzy or annotated or similarity-based or probabilistic logic programming. Several proposals have appeared in the literature for dealing with probabilistic information, namely Hybrid Probabilistic Logic Programs [6], Probabilistic Deductive Databases [8], and Probabilistic Logic Programs with conditional constraints [9].

Residuated and monotonic logic programs [2] and multi-adjoint logic programs [10] were introduced as general frameworks which abstract the particular details of the different approaches cited above and focus only on the computational mechanism of inference. This higher level of abstraction makes possible the development of general results about the behaviour of several of the previously cited approaches.

The main aim of this paper is to focus on some termination properties of the fixed point semantics of a sorted version of multi-adjoint logic programming. In this sorted approach each sort identifies an underlying lattice of truth-values (weights) which must satisfy the adjoint conditions. Although we restrict to the ground case, we allow infinite programs, and thus there is not loss of generality.

The major contribution of this paper is the termination theorems for a general class of sorted multi-adjoint logic programs, complementing results in the literature and enhancing previous results in [1]. Then, we illustrate the application of the termination theorems to obtain known termination results for some of the previously stated approaches languages.

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 the basic results regarding the termination properties of our semantics, which are applied later in probabilistic settings in Section 5. The paper finishes with some conclusions and pointers to future work.

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 [3], 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. *A* signature *is a pair $\Sigma = \langle S, F\rangle$ where S is a set of elements, designated* sorts, *and F is a collection 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 $f\colon \tau$ to denote a pair $\langle f, \tau\rangle$ belonging to F.*

Definition 2. *Let $\Sigma = \langle S, F\rangle$ be a signature, a Σ*-algebra *is a pair $\langle \{A^s\}_{s\in S}, I\rangle$ satisfying the two following conditions:*

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 an element $I(c) \in A^s$ to each constant symbol $c\colon s$ 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. *Let $\langle P, \preceq\rangle$ be a partially ordered set and let $(\leftarrow, \&)$ be 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 and decreasing in the second argument.*
(a3) *For any $x, y, z \in P$, we have that $x \preceq (y \leftarrow z)$ iff $(x \,\&\, z) \preceq y$*

Then $(\leftarrow, \&)$ is said to form an adjoint pair *in $\langle P, \preceq\rangle$.*

Extending the results in [2,3,13] 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.

Definition 4. *A* multi-adjoint lattice *$\mathcal{L}$ is a tuple $(L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ satisfying the following conditions:*

(l1) *$\langle L, \preceq\rangle$ is a bounded lattice, i.e. it has bottom $(\bot)$ and top $(\top)$ elements;*
(l2) *$(\leftarrow_i, \&_i)$ is an adjoint pair in $\langle L, \preceq\rangle$ for all i;*
(l3) *$\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$ for all $\vartheta \in L$ for all i.*

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. *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 $\leftarrow_i^s\colon s \times s \to s$ and $\&_i^s\colon s \times s \to s$ for $i = 1, \dots, n^s$ (and possibly some extra operators) such that the tuple $\mathcal{L}^s$*

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

  is a multi-adjoint lattice.

Multi-adjoint Σ-algebras can be found underlying the probabilistic deductive databases framework of [8] where our sorts correspond to ways of combining belief and doubt probability intervals. 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).

3 Syntax and Semantics of Sorted Multi-Adjoint Logic Programs

Sorted multi-adjoint logic programs are constructed from the abstract syntax induced by a multi-adjoint Σ-algebra. Specifically, given an infinite set of sorted propositional symbols Π, we will consider the corresponding term Σ-algebra of formulas[1] $\mathfrak{F} = Terms(\Sigma, \Pi)$. In addition, we will consider a multi-adjoint Σ-algebra $\mathfrak{L}$, whose extra operators can be arbitrary monotone operators, to host the manipulation of the truth-values of the formulas in our programs.

Remark 2. As we are working with two Σ-algebras, in order to discharge the notation, we introduce a special notation to clarify which algebra a function symbol belongs to. 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 its interpretation under $\mathfrak{F}$ 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:

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

Definition 6. *A* sorted multi-adjoint logic program *is a set* $\mathbb{P}$ *of rules* $\langle A \leftarrow_i^s \mathcal{B}, \vartheta\rangle$ *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 $\mathcal{B}$ *is a formula of* $\mathfrak{F}$ *with sort* s*, built from sorted propositional symbols* $B_1, \dots, B_n$ *(*$n \geq 0$*) by the use of function symbols in* Σ.

Facts are rules with body $\top^s$, the top element of lattice $\mathcal{L}^s$. A *query* (or *goal*) is a propositional symbol intended as a question $?A$ prompting the system.

Sometimes, we will represent bodies of formulas as $@[B_1, \dots, B_n]$, where[2] the B_is are the propositional variables occurring in the body and $@$ is the aggregator obtained as a composition.

3.2 Semantics of Sorted Multi-Adjoint Logic Programs

Definition 7. *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 orderings $\preceq^s$ of the truth-values L^s can be easily extended to the set of interpretations as follows:

Definition 8. *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. *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 for an interpretation* I *and an arbitrary propositional symbol* A *of sort* s *is defined by*

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

where $\bigsqcup_s$ *is the least upper bound in the lattice* $\mathcal{L}^s$.

[2] Note the use of square brackets in this context.

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], 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 focus on the termination properties of the $T_{\mathbb{P}}$ operator. In what follows we assume that every function symbol is interpreted as a computable function. 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. It is not difficult to show examples in which exactly ω iterations may be necessary to reach the least fixpoint.

The termination property we investigate is stated in the following definition, and corresponds to the notion of fixpoint-reachability of Kifer and Subrahmanian [7]:

Definition 12. *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 ${T_{\mathbb{P}}}^n(\triangle)(A)$ is identical to $lfp(T_{\mathbb{P}})(A)$.*

In [1] several results were presented in order to provide sufficient conditions guaranteeing 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. Moreover, a general termination theorem for a wide class of sorted multi-adjoint logic programs, designated programs with finite dependencies, was anticipated.

The notion of dependency graph for sorted multi-adjoint logic programs captures (recursively) the propositional symbols which are necessary to compute the value of a given propositional symbol. 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 the nodes accessible from A and corresponding edges.

Definition 13. *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.*

The fact that a propositional symbol has finite dependencies gives us some guarantees that we can finitely evaluate its value. However, this is not sufficient since a propositional symbol may depend directly or indirectly on itself, and the $T_{\mathbb{P}}$ operator might after all produce infinite ascending chains of values for this symbol. The following definition identifies an important class of sorted multi-adjoint logic programs where we can show that these infinite ascending chains cannot occur, and thus ensuring termination.

Definition 14. *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 $c_{s_1 s_2} : s_2 \to s_1$ in Σ.*
- *All other function symbols have types of the form $f : s \times \cdots \times s \to s$, i.e. are closed operations in each sort, satisfying the following boundary conditions for every $v \in \mathcal{L}^s$:*
$$\begin{aligned} I(f)(v, 1^s, \ldots, 1^s) &\preceq^s v \\ I(f)(1^s, v, 1^s, \ldots, 1^s) &\preceq^s v \\ &\vdots \\ I(f)(1^s, \ldots, 1^s, v) &\preceq^s v \end{aligned}$$
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:*
$$(c_{ss_1} \circ c_{s_1 s_2} \circ \ldots \circ c_{s_n s})\,(v) \preceq^s v$$
for every $v \in \mathcal{L}^s$ and finite composition of casting functions with overall sort $s \to s$.

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. Furthermore, the connectives are not assumed to be continuous. Even in this case, we are able to state a main termination result about sorted multi-adjoint logic programs:

Theorem 1. *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.

The idea underlying the proof 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, whereas the non-maximal values are irrelevant for determining the new value for A by $T_{\mathbb{P}}$. This is formalized in the following definition:

Definition 15. *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 $\{\vartheta \mathrel{\dot{\&}^s_i} \hat{I}(\mathcal{B}) \mid \langle A \leftarrow^s_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}\}$*
- *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 $\vartheta \mathrel{\dot{\&}^s_i} \hat{I}(\mathcal{B})$ belongs to $R^I_{\mathbb{P}}(A)$. Rules in a culprit set are called* culprits.
- *The* culprit collection *for ${T_{\mathbb{P}}}^n(\triangle)(A)$ is defined as the set of culprits used in the tree of recursive calls of $T_{\mathbb{P}}$ in the computation.*

The proof of the theorem is based on the bounded growth of the culprit collection for ${T_{\mathbb{P}}}^n(\triangle)(A)$. By induction on n, it will be proved that if we assume $T^{n+1}_{\mathbb{P}}(\triangle)(A) \succ^s T^n_{\mathbb{P}}(\triangle)(A)$ 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 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.

As we shall see, Theorem 1 can be used to obtain the Probabilistic Deductive Databases termination theorem [8], since the connectives allowed in rule bodies obey to the boundary conditions. However, the theorem cannot be applied to show termination results of Hybrid Probabilistic Logic Programs (HPLPs) appearing in [5] because operators employed to capture disjunctive probabilistic strategies do not obey to the boundary conditions. For obtaining the termination theorem for HPLPs we require the notion of *range dependency graph*:

Definition 16. *The* range dependency graph *of a sorted multi-adjoint logic program $\mathbb{P}$ has a vertex for each propositional symbol in Π. 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 which does not appear in a sub-term with main function symbol having finite image.*

The rationale is to not include arcs of the dependency graph referring to propositional symbols which can only contribute directly or indirectly with finitely many values to the evaluation of the body. For instance, consider the rule $A \leftarrow f(g(A, B), B) \otimes g(f(C)) \otimes D \otimes g(E)$, where f is mapped to a function with infinite range and g corresponds to a function with finite range (i.e. g has finite image). According to the previous definition, we will introduce an arc from A to B and from A to D. The propositional symbol A occurs in the sub-term $g(A, B)$, with finite image, and the same happens with $g(f(C))$ and $g(E)$, and therefore they are excluded from the range dependency graph. The arc to B is introduced because of the second occurrence of B in $f(g(A, B), B)$. The notion of finite dependencies immediately extends to range dependency graphs, but one has to explicitly enforce that for each propositional symbol there are only finitely many rules for it in the program.

Theorem 2. *If $\mathbb{P}$ is a sorted multi-adjoint logic program with acyclic range dependency graph having finite dependencies, then $T_{\mathbb{P}}$ terminates for every query.*

Proof (Sketch). Consider an arbitrary propositional symbol A and the corresponding range dependency subgraph for A. We know that it is both finite and acyclic. It is possible to show that in these conditions only a finite number of values can be produced by the $T_{\mathbb{P}}$ operator, and therefore no infinite ascending chains for the values of A can be generated. This is enough to show the result (see for instance [1]). $\square$

Corollary 1. *If $\mathbb{P}$ is a sorted multi-adjoint logic program such that all function symbols in the underlying Σ-algebra have finite images, then $T_{\mathbb{P}}$ terminates for every query.*

The proof is immediate since in this case the range dependency graph is empty.

Mark that the conditions of the theorem do not imply that program $\mathbb{P}$ is acyclic. Cyclic dependencies through propositions in finitely ranged function symbols can occur, since these are discarded from the range dependency graph of $\mathbb{P}$. This is enough to show the results for Hybrid Probabilistic Logic Programs.

In order to remove the acyclicity condition from Theorem 2, boundary conditions are again necessary obtaining a new result combining Theorems 1 and 2. Specifically, the termination result can also be obtained if the local multi-adjoint Σ-algebra also contains function symbols $g\colon s_1 \times \cdots \times s_l \to s_k$ such that their interpretations are isotonic functions with finite range. We call this kind of algebra a *local multi-adjoint Σ-algebra with finite operators.*

Theorem 3. *Let $\mathbb{P}$ be a sorted multi-adjoint logic program with respect to a local multi-adjoint Σ-algebra with finite operators $\mathfrak{L}$ and the set of sorted propositional symbols Π, and having finite dependencies.*

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

The intuition underlying the proof of this theorem is simply to apply a cardinality argument. However, the formal presentation of the proof requires introducing some technicalities which offer enough control on the increase of the computation tree for a given query.

On the one hand, one needs to handle the number of applications of rules; this is done by using the concept of *culprit collection*, as in Theorem 1. On the other hand, one needs to consider the applications of the finite operators, which are not adequately considered by the culprit collections. With this aim, given a propositional symbol A, let us consider the subset of rules of the program associated to its dependency graph[3], and denote it by $\mathbb{P}^A$. This set is finite, for the program has finite dependencies, so we can write:

$$\mathbb{P}^A = \{\langle H_i \leftarrow \mathcal{B}_i, \vartheta_i\rangle \mid i \in \{1, \dots, s\}\}$$

In addition, let us write each body of the rules above as follows:

$$\mathcal{B}_i = @_i[g^i_1(\mathcal{D}^i_1), \dots, g^i_{k_i}(\mathcal{D}^i_{k_i}), C^i_1, \dots, C^i_{m_i}]$$

where $g^i_j(\mathcal{D}^i_j)$ represents the subtrees corresponding to the outermost occurrences of finite operators, the C^i_j are the propositional symbols which are not in the scope of finite operator, and $@_i$ is the operator obtained after composing all the operators in the body not in the scope of any finite operator.

Now, consider $G(\mathbb{P}^A) = \{g^1_1, \dots, g^1_{k_1}, \dots, g^s_1, \dots, g^s_{k_s}\}$, which is a finite multiset, and let us define the following counting sets for the contribution of the finite operators to the overall computation.

Definition 17. *The* counting sets for $\mathbb{P}$ and A *for all $n \in \mathbb{N}$, denoted Ξ^A_n, are defined as follows:*

$$\Xi^A_n = \{k < n \mid \text{ there is } g^i_j \in G(\mathbb{P}^A) \text{ s.t. } g^i_j({T_{\mathbb{P}}}^n(\triangle)(\mathcal{D}^i_j)) > g^i_j({T_{\mathbb{P}}}^{n-1}(\triangle)(\mathcal{D}^i_j))\}$$

With this definition we can state the main lemma needed in the proof of Thm 3.

Lemma 1. *Under the hypotheses of Theorem 3, if ${T_{\mathbb{P}}}^{n+1}(\triangle)(A) > {T_{\mathbb{P}}}^n(\triangle)(A)$ then either $|\Xi^A_{n+1}| > |\Xi^A_n|$ or the culprit collection for ${T_{\mathbb{P}}}^{n+1}(\triangle)(A)$ is greater than that for ${T_{\mathbb{P}}}^n(\triangle)(A)$.*

Proof (of Theorem 3). The previous lemma is the key to the proof:

[3] Mark we are using again the dependency graph, not the range dependency graph.

- Firstly, since the program has finite dependencies there cannot be infinitely many rules in the culprit collections for A.
- On the other hand, the sequence of cardinals $|\Xi_n^A|$ is upper bounded (since the range of each function g_j^i is finite and $G(\mathbb{P}(A))$ is also finite).

As a result we obtain that $T_{\mathbb{P}}$ terminates for every query. $\square$

In the next section we apply the above results to show the termination theorems for important probabilistic based logic programming frameworks.

5 Termination of Probabilistic Logic Programs

The representation of probabilistic information in rule-based systems has attracted a large interest of the logic programming community, fostered by knowledge representation problems in advanced applications, namely for deductive databases. Several proposals have appeared in the literature for dealing with probabilistic information, namely Hybrid Probabilistic Logic Programs [6], Probabilistic Deductive Databases [8], and Probabilistic Logic Programs with conditional constraints [9]. Both Hybrid Probabilistic Logic Programs, Probabilistic Deductive Databases, and Ordinary Probabilistic Logic Programs can be captured by Residuated Monotonic Logic Programs, as shown in [4]. We illustrate here the application of the theorems of the previous section to obtain known termination results for these languages. Notice that these results are obtained from the abstract properties of the underlying algebras and transformed programs. In this way we simplify and synthesize the techniques used to show these results, which can be applied in other settings as well.

5.1 Termination of Ordinary Probabilistic Logic Programs

Lukasiewicz [9] 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 [9] 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

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

Under positively correlated probabilistic interpretations (PCP-interpretations), reasoning becomes tractable and truth-functional. Ordinary conditional constraints (1) of ordinary probabilistic logic programs under PCP-interpretation can be immediately translated to a sorted multi-adjoint logic programming rule

$$A \xleftarrow{c} \min\left(B_1, \ldots, \min(B_{n-1}, B_n)\right)$$

over the multi-adjoint Σ-algebra containing a single sort u 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 $\min^u$) and the product of two reals (denoted by $\times^u$), and Goguen implication $\leftarrow^u$. The structure $< [0, 1], \leq, \leftarrow^u, \times^u >$ is a well-known adjoint lattice, where Goguen implication is the residuum of product t-norm. The function symbols $\leftarrow$, $\min$ are interpreted by $\leftarrow^u$ and $\min^u$, respectively. The previous rule can also be represented as:

$$A \overset{1}{\leftarrow} c \times \min\left(B_1, \ldots, \min(B_{n-1}, B_n)\right)$$

Clearly, as remarked in [9], the resulting rule is equivalent to a rule of van Emden's Quantitative Deduction [12]. It is pretty clear that in these circumstances all the conditions of Theorem 1 are fulfilled for ground programs of the above form having finite dependencies, and we can guarantee termination of $T_{\mathbb{P}}$ for every query. This is the case because we are using solely t-norms in the body, which by definition obey to the boundary condition, over the unit interval $[0, 1]$. Since the unit interval is totally ordered and we have a finite number of rules for every propositional symbol, we can guarantee that the set of relevant values for ${T_{\mathbb{P}}}^n(\Delta)$ is a singleton. Thus, we obtain a termination result for Ordinary Probabilistic Logic Programs and Quantitative Deduction, extending the one appearing in [12].

In general, if we have combinations of t-norms in the bodies of rules, over totally ordered domains, we can guarantee termination for programs with finite dependencies. This extends the previous results by Paulík [11]. The same applies if we reverse the ordering in the unit interval, and use t-conorms in the bodies. This is necessary to understand the termination result for Probabilistic Deductive Databases, presented in the next section.

5.2 Termination of Probabilistic Deductive Databases

A definition of a theory of probabilistic deductive databases is described in Lakshmanan and Sadri's work [8] 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. The authors present a termination result assuming that it is used solely positive correlation as disjunctive mode for combining several rules in the program, and arbitrary conjunctive modes. The truth-values of p-programs are confidence levels of the form $\langle[\alpha, \beta], [\gamma, \delta]\rangle$, where α, β, γ, and δ are real numbers in the unit interval[4]. The values α and β are, respectively, the expert's lower and upper bounds of belief, while γ and δ are the bounds for the expert's doubt. The fixpoint semantics of p-programs

[4] Even though the authors say that they usually assume that $\alpha \leq \beta$ and $\gamma \leq \delta$, this cannot be enforced otherwise they cannot specify properly the notion of trilattice. So, we also not assume these constraints.

relies on truth-ordering of confidence levels. Suppose $c_1 = \langle[\alpha_1, \beta_1], [\gamma_1, \delta_1]\rangle$ and $c_2 = \langle[\alpha_2, \beta_2], [\gamma_2, \delta_2]\rangle$ are confidence levels, then we say that:

$$c_1 \leq_t c_2 \text{ iff } \alpha_1 \leq \alpha_2, \beta_1 \leq \beta_2 \text{ and } \gamma_1 \geq \gamma_2, \delta_1 \geq \delta_2,$$

with corresponding least upper bound operation $c_1 \oplus_t c_2$ defined as

$$\langle[\max\{\alpha_1, \alpha_2\}, \max\{\beta_1, \beta_2\}], [\min\{\gamma_1, \gamma_2\}, \min\{\delta_1, \delta_2\}]\rangle$$

and greatest lower bound $c_1 \otimes_t c_2$ as:

$$\langle[\min\{\alpha_1, \alpha_2\}, \min\{\beta_1, \beta_2\}], [\max\{\gamma_1, \gamma_2\}, \max\{\delta_1, \delta_2\}]\rangle$$

The least upper bound of truth-ordering corresponds to the disjunctive mode designated “positive correlation”, which is used to combine the contributions from several rules for a given propositional symbol. We restrict attention to this disjunctive mode, since the termination results presented in [8] assume all the rules adopt this mode. Conjunctive modes are used to combine propositional symbols in the body, and $\otimes_t$ corresponds to the *positive correlation* conjunctive mode. Another conjunctive mode is *independence* with $c_1 \wedge_{ind} c_2$ defined as

$$\langle[\alpha_1 \times \alpha_2, \beta_1 \times \beta_2], [1 - (1 - \gamma_1) \times (1 - \gamma_2), 1 - (1 - \delta_1) \times (1 - \delta_2)]\rangle$$

The attentive reader will surely notice that all these operations work independently in each component of the confidence level. Furthermore, the *independence* conjunctive mode combines the α’s and β’s with a t-norm (product), and the γ and δ parts are combined with a t-conorm. This is a property enjoyed by all conjunctive modes specified in [8]. In order to show the termination result we require two sorts, both with carrier $[0, 1]$, the first one denoted by m and ordered by $\leq$, while the other is denoted by M and ordered by $\geq$ (this means that for this sort the bottom element is 1 and the top one is 0, least upper bound is $\min$). The program transformation translates each ground atom P in a p-program into four propositional symbols P^α, P^β, P^γ and P^δ, representing each component of the confidence level associated with P. The translation generates four rules, in the resulting sorted multi-adjoint logic programming, from each rule in the p-program. We illustrate this with an example, where the conjunctive mode use is independence (remember that the disjunctive mode is fixed). A p-program rule of the form

$$\left(A \xleftarrow{\langle[a,b],[c,d]\rangle} B_1, \ldots, B_n \;;\; ind, pc\right)$$

is encoded as the following four rules:

$$\begin{array}{ll} A^\alpha \overset{a}{\longleftarrow}{}^{m} B_1^\alpha \times \ldots \times B_n^\alpha & A^\beta \overset{b}{\longleftarrow}{}^{m} B_1^\beta \times \ldots \times B_n^\beta \\ A^\gamma \overset{c}{\longleftarrow}{}^{M} B_1^\gamma \oplus \ldots \oplus B_n^\gamma & A^\delta \overset{d}{\longleftarrow}{}^{M} B_1^\delta \oplus \ldots \oplus B_n^\delta \end{array}$$

The operation $\oplus$ denotes the t-conorm function defined by $v \oplus w = 1 - (1 - v) \times (1 - w)$. Other conjunctive modes can be encoded similarly. The termination of these programs is now immediate. First, the rules for α propositional symbols only involve α propositional symbols in the body. The same applies to the other β, γ and δ rules. The underlying carriers are totally ordered, and the functions symbols in the body obey to the boundary condition since they are either t-norms (for α and β rules) or t-conorms (for γ and δ rules). Thus, from the discussion on the previous section, Theorem 1 is applicable and the result immediately follows for programs with finite dependencies. This is a result shown based solely on general properties of the underlying lattices, not resorting to specific procedural concepts as in [8]. Furthermore, since the grounding of p-programs always results in a finite program, there is no lack of generality by assuming finite dependencies. The use of other disjunctive modes introduce operators in the bodies which no longer obey to the boundary condition. For this case, Lakshmanan and Sadri do not provide any termination result, which is not strange since this violates the general conditions of applicability of Theorem 1.

5.3 Termination of Hybrid Probabilistic Logic Programs

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. In order to obtain a residuated lattice, the carrier $\mathcal{INT}$ is the set of pairs $[a, b]$ where a and b are real numbers in the unit interval[5].

The termination results presented in [5] assume finite ground programs. From a difficult analysis of the complex fixpoint construction one can see that only a finite number of different intervals can be generated in the case of finite ground programs. We show how this result can be obtained from Theorem 2 almost directly, given the embedding of Hybrid Probabilistic Logic Programs into Residuated ones presented in [3]. This embedding generates rules of the following kind

1. $F \overset{[a,b]}{\leftarrowtail} s_{\mu_1}\left(\overline{\overline{F_1}}\right) \sqcap \ldots \sqcap s_{\mu_k}\left(\overline{\overline{F_k}}\right)$
2. $F \overset{[a,1]}{\leftarrowtail} s_{\mu_1}\left(\overline{\overline{E_1}}\right) \sqcap \ldots \sqcap s_{\mu_m}\left(\overline{\overline{E_m}}\right)$
3. $F \overset{[0,b]}{\leftarrowtail} s_{\mu_1}\left(\overline{\overline{E_1}}\right) \sqcap \ldots \sqcap s_{\mu_m}\left(\overline{\overline{E_m}}\right)$
4. $F \overset{[1,0]}{\leftarrowtail} c_\rho\left(\overline{\overline{G}}, \overline{\overline{H}}\right)$

resorting to the auxiliary double bar function $\overline{\overline{\cdot}}$ from $\mathcal{INT}$ to $\mathcal{INT}$ and the functions $s_\mu : \mathcal{INT} \to \mathcal{INT}$, with μ in $\mathcal{INT}$. For our analysis, it is only important to know that all these functions have finite image, and thus when constructing the range dependency graph no arc will be introduced for rules of the first three types.

The next important detail is that the rules of the fourth type, which use either conjunctive or disjunctive strategies c_ρ, do not introduce any cyclic dependencies and the dependencies are finite. This is the case, because F, G and H are propositional symbols which represent ground hybrid basic formulas (see [6,3] for details), such that $F = G \oplus_\rho H$, i.e. the propositional symbol F represents a more complex formula obtained from the conjunctive or disjunctive combination of the simpler formulas G and H. Therefore, it is not possible to have a dependency from a simpler formula to a more complex one. By application of Theorem 2 it immediately follows that $T_\mathbb{P}$ terminates for every finite ground program, as we intended to show. Just as a side remark, Theorem 3 can also be applied if in the program only occurs conjunctive basic formulas,

[5] We do not impose that $a \leq b$.

without requiring any reasoning about the shape of the transformed program and its dependencies.

6 Conclusions

A sorted version of multi-adjoint logic programming has been introduced, together with several general sufficient results about the termination of its fix-point semantics. Later, these results are instantiated in order to prove termination theorems for some probabilistic approaches to logic programming. Notice that these results are obtained solely from the abstract properties of the underlying algebras and transformed programs. In this way we simplify and synthesize the techniques used to show these results, which can be applied in other settings as well.

References

1. C.V. Damásio, J. Medina and M. Ojeda-Aciego. Termination results for sorted multi-adjoint logic programming. In *Information Processing and Management of Uncertainty for Knowledge-Based Systems*, IPMU'04. Accepted.
2. C. V. Damásio and L. M. Pereira. Monotonic and residuated logic programs. Lect. Notes in Artificial Intelligence 2143, pp. 748–759, 2001.
3. C. V. Damásio and L. M. Pereira. Hybrid probabilistic logic programs as residuated logic programs. *Studia Logica*, 72(1):113–138, 2002.
4. C. V. Damásio and L. M. Pereira. Sorted monotonic logic programs and their embeddings. In *Information Processing and Management of Uncertainty for Knowledge-Based Systems,* IPMU'04. Accepted.
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. M. Kifer and V. S. Subrahmanian, Theory of generalized annotated logic programming and its applications. *J. of Logic Programming* 12(4):335–367, 1992
8. L. Lakhsmanan and F. Sadri, On a theory of probabilistic deductive databases. *Theory and Practice of Logic Programming* 1(1):5–42, 2001
9. T. Lukasiewicz. Probabilistic logic programming with conditional constraints. *ACM Trans. Comput. Log.* 2(3): 289-339 (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. L. Paulík. Best possible answer is computable for fuzzy SLD-resolution. Lecture Notes on Logic 6, pp. 257–266, 1996.
12. M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 4(1):37–53, 1986.
13. P. Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124(3):361–370, 2001.