A termination theorem for sorted multi-adjoint logic programming

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

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

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

Resumen

In this paper we present a general framework of logic programming allowing for the combination of several adjoint lattices of truth-values. The main contribution is a new sufficient condition which guarantees termination of all queries for the fixpoint semantics for an interesting class of programs.

Keywords: Fuzzy Logic Programming, Termination Results.

1 Introduction

A number of approaches have been developed for 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. [7, 11, 6, 5, 12, 8] as a result of the increasing interest in models of reasoning under "imperfect" information.

In this work we propose a sorted version of the multi-adjoint paradigm of logic programming, where each sort identifies an underlying lattice of truth-values (weights) which must satisfy adjoint conditions. 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 [9]. The current proposal is an important enhancement of our previous works [1, 4, 9].

The major contribution of this paper is a termination result for a classes of sorted multi-adjoint logic programs complementing results in [6, 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.

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 extended later 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 [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 *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 $f\colon \tau$ to denote a pair $\langle f, \tau\rangle$ belonging to F.

Definition 2 *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 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

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

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

Extending the results in [4, 3, 12] 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 *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 all i;

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

Remark: 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 $\&_i^s : s \times s \to s$ 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.

Multi-Adjoint Σ-Algebras can be found underlying the probabilistic deductive databases framework of [8] 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).

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 $A \in \Pi^s$.

Remark: 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.

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

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 *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 *$\mathcal{B}$ 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.*

Sometimes, we will represent bodies of formulas as $@[B_1, \ldots, B_n]$, where[2] $B_1, \ldots, B_n$ 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:

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

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 is defined by*

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

where A is an arbitrary propositional symbol of sort s, and $\bigsqcup_s$ 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 [9], 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 recall the termination properties of the $T_{\mathbb{P}}$ operator. 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. 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:

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* $lfp(T_{\mathbb{P}})(A)$.

We presented in [2] 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. Nevertheless, the results are applicable to a special class of infinite sorted multi-adjoint logic programs, designated programs with finite dependencies.

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.

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.*

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. To begin with, the following definition is needed:

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* $c_{s_1 s_2} \colon s_2 \to s_1$ *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* $v \in \mathcal{L}^s$*:*

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

  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* $s \to s$.

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.

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* $\{\vartheta \,\&^s_i \hat{I}(\mathcal{B}) \mid \langle A \leftarrow^s_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}\}$
- *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* $\vartheta \,\&^s_i \hat{I}(\mathcal{B})$ *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 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 $T_{\mathbb{P}}$.

Theorem 1 ([2]) *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 proof is based on the bounded growth of the culprit collection for ${T_{\mathbb{P}}}^n(\triangle)(A)$. By induction on n, it can be proved 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 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.

5 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. 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.

Theorem 1 above can be generalized to a case which will allow us to prove the termination result for hybrid probabilistic logic programs in an abstract way. 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 (that is, the image of g is a finite set). We call this kind of algebra a *local multi-adjoint Σ-algebra with finite operators.*

The major result in this paper is stated below, as an extension of Theorem 1.

Theorem 2 *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 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 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 dependence graph, and denote it by $\mathbb{P}^A$. This set is finite, for the program has finite dependencies, so we can write:

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

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

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

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 $G(\mathbb{P}^A) = \{g^1_1,\ldots,g^1_{k_1},\ldots,g^s_1,\ldots,g^s_{k_s}\}$, 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 16 *The* counting sets for $\mathbb{P}$ and A *are defined for all $n \in \mathbb{N}$ as follows:*

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

With these definitions we can state the main lemma needed in the proof of Theorem 2.

Lemma 1 *Under the hypotheses of Theorem 2, 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: We will proceed by induction on n.

Base case $n = 0$: for any A it is straightforward that if $T_{\mathbb{P}}(\triangle)(A) > \triangle(A) = \bot$ then a new rule has been used.

Inductive case: Assume that the result is true for any propositional symbol and $n = k$; in order to prove the result for $k+1$, assume that $T_{\mathbb{P}}{}^{k+1}(\triangle)(A) > T_{\mathbb{P}}{}^k(\triangle)(A)$.

By the singleton hypothesis, there exists a rule labeled by an index $i \in \{1,\ldots,s\}$ such that

$$T_{\mathbb{P}}{}^{k+1}(\triangle)(A) = \vartheta_i \,\&\, T_{\mathbb{P}}{}^k(\triangle)(@_i[g^i_1(\mathcal{D}^i_1),\ldots,g^i_{r_i}(\mathcal{D}^i_{r_i}), C^i_1,\ldots,C^i_{m_i}])$$

now, for that rule indexed by i, by definition of $T_{\mathbb{P}}$ as a least upper bound, we have

$$T_{\mathbb{P}}{}^k(\triangle)(A) \geq \vartheta_i \,\&\, T_{\mathbb{P}}{}^{k-1}(\triangle)(@_i[g^i_1(\mathcal{D}^i_1),\ldots,g^i_{r_i}(\mathcal{D}^i_{r_i}), C^i_1,\ldots,C^i_{m_i}])$$

then, by the monotonicity of the connectives in the body, either there exists $j \in \{1,\ldots,r_i\}$ such that

$$g^i_j(T_{\mathbb{P}}{}^k(\triangle)(\mathcal{D}^i_j)) > g^i_j(T_{\mathbb{P}}{}^{k-1}(\triangle)(\mathcal{D}^i_j))$$

or there exists $j \in \{1,\ldots,m_i\}$ such that

$$T_{\mathbb{P}}{}^k(\triangle)(C^i_j) > T_{\mathbb{P}}{}^{k-1}(\triangle)(C^i_j)$$

In the former case, it is obvious that $|\Xi^A_{n+1}| > |\Xi^A_n|$; in the latter case, then the induction hypothesis on the propositional variable C^i_j applies. □

Proof of the Theorem: The previous lemma is the key of the proof of the main theorem of this section:

- 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. □

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. A termination result motivated by the hybrid probabilistic logic programs has been presented, the consequences of this theoretical result and some of its possible modifications will be studied in the particular framework of HPLP. The embedding of other proposals in the literature into our framework will be explored in subsequent work.

References

[1] 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

[2] 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.

[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] 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] 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.

[10] 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.

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

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