Termination of logic programs with imperfect information: applications and query procedure

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

[a] *Centro Inteligência Artificial. Universidade Nova de Lisboa. Portugal*

[b] *Dept. Matemática Aplicada. Universidad de Málaga. Spain*

Abstract

A general logic programming framework allowing for the combination of several adjoint lattices of truth-values is presented. The language is sorted, enabling the combination of several reasoning forms in the same knowledge base. The contribution of the paper is two-fold: on the one hand, sufficient conditions guaranteeing termination of all queries for the fix-point semantics for a wide class of sorted multi-adjoint logic programs are presented and related to some well-known probability-based formalisms; in addition, we specify a general non-deterministic tabulation goal-oriented query procedure for sorted multi-adjoint logic programs over complete lattices. We prove its soundness and completeness as well as independence of the selection ordering. We apply the termination results to probabilistic and fuzzy logic programming languages, enabling the use of the tabulation proof procedure for query answering.

Key words: Logic Programming, Probabilistic Reasoning, Fuzzy Reasoning, Termination, Tabulation Proof Procedures

Email addresses: cd@di.fct.unl.pt (C.V. Damásio), jmedina@ctima.uma.es (J. Medina), aciego@ctima.uma.es (M. Ojeda-Aciego).

[1] Partially supported by FSE/FEDER project TARDE (POSI/EEI/12097/2001), by European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net).

[2] Partially supported by Spanish project TIC2003-09001-C02-01.

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 so-called inexact or fuzzy or approximate reasoning, involving either fuzzy [16,23] or annotated [11] or similarity-based [18] or probabilistic logic programming. Several proposals have appeared in the literature for dealing with probabilistic information, namely Hybrid Probabilistic Logic Programs [9], Probabilistic Deductive Databases [12], and Probabilistic Logic Programs with conditional constraints [14].

This paper does not intend to survey the literature of this area (the interested reader might consult for instance [6] for a starting point). We have just picked three or four important cases to illustrate the applicability of our techniques. Also, we center the discussion on the monotonic case, ignoring default negation. Default negation introduces other significant theoretical problems in the general setting in which we are working, problems whose solutions depend on the availability of results we are going to present for the first time here. The major results show that general abstract approaches to the semantics of logic programming are still needed and (still) produce very interesting research problems and challenges.

Residuated and monotonic logic programs [3] and multi-adjoint logic programs [15] were introduced as general frameworks which abstract out 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.

To make the discussion more concrete, consider the common-sense knowledge that good papers reviewed by good referees are accepted for publication. In the ideal classical two-valued approach, one would represent this by the definite logic programming rule [22]:

$$paper\_accepted \leftarrow good\_work, good\_referees$$

We know that in real-life things do not work exactly like this and some way of dealing with uncertainty and imprecision is necessary. For instance, Quantitative Deduction Rules [21] allows us to express the following rule:

$$paper\_accepted \xleftarrow{0.9} good\_work \ \& \ good\_referees$$

The idea here is that propositions have truth-values in the unit interval $[0, 1]$ and weights can be assigned to rules. The above rule states that $good\_work$ conjoined with $good\_referees$ entails that a paper is accepted with confidence

degree 0.9. This is more evident if we make clear the connectives used in the rules, as it is enforced by Fuzzy Logic Programming [24]. The following rule is equivalent to the previous quantitative deduction rule:

$$paper\_accepted \xleftarrow{0.9}_{p} good\_work \,\&_G\, good\_referees$$

The above rule is satisfied whenever the value of *paper_accepted* is greater or equal than 0.9 times (product implication) the minimum (Gödel conjunction) of the truth-values of *good_work* and $good\_referees$. In Fuzzy Logic Programming, we can choose a t-norm for combining the weight of the rule with the result of another t-norm applied to all the propositions in the body. These t-norms can be selected by the user, instead of being fixed *a priori* like in the Quantitative Deduction Rule framework. Unfortunately, we cannot restrict the discussion to linearly ordered lattices of truth-values, since for instance dealing with probabilities requires using intervals of probability. This is particularly clear in the Probabilistic Deductive Databases system [12], where we can express rules like:

$$\Big(paper\_accepted \xleftarrow{\langle[0.7,0.95],[0.03,0.2]\rangle} good\_work, good\_referees; ind, pc\Big)$$

Here we have a complex confidence value containing two probability intervals, one for the case where *paper_accepted* is true and other for the case where *paper_accepted* is false (this representation allows for incomplete information). The label *ind* indicates that *good_work* is assumed to be probabilistically independent from $good\_referees$, while *pc* specifies the way how the results of the several rules for *paper_accepted* are to be (disjunctively) combined.

Despite the differences between these languages, some interesting concepts and mechanisms are common to all of them:

SORTED: Different forms of weights, confidence values, truth values, or degrees and corresponding operators;

MULTI-ADJOINT: Different implication symbols with partially ordered weights associated to rules;

LOGIC PROGRAMS: Rules with a single propositional variable in the head, and bodies constructed from arbitrary combinations of monotonic functions.

This justifies the need to consider such a very abstract framework in order to be able to state and prove general results, which can apply to such apparently disparate approaches. In particular, we aim at obtaining termination properties of the fix-point semantics of a sorted version of multi-adjoint logic programming. The termination results are subtle since there are infinite programs that terminate for every query as well as finite ones that do not. Having these results we can tell the knowledge engineers what kind of programs they can build and which connectives they may use. Although we restrict ourselves

to the ground case, we nevertheless allow infinite programs; thus there is not loss of generality.

The first 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 [7]. In our sorted approach each sort identifies an underlying lattice of truth-values (weights) that must satisfy the adjoint conditions (see below). The termination results rely on the careful combination of both syntactical and semantical conditions, which appear to summarise the techniques found in the literature for specific cases [16,10,12]. In particular, we illustrate the application of the termination theorems to obtain known termination results for some of the previously stated approaches and languages.

Any logic programming language should be accompanied with query answering procedures. However two fundamental problems must be addressed: by allowing infinite truth-values spaces queries may not terminate; by allowing partial orders, the contributions of several rules must be combined together to obtain the answer for some queries. By exploring the previous results, the second important contribution of this work is a tabulation goal-oriented query procedure, which tackles both problems. In particular, this tabulation procedure terminates for a significant class of sorted multi-adjoint logic programs, showing termination of query answering for several fuzzy and probabilistic logic programming languages in the literature.

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 subsequent section presents the tabulation-based query procedure, together with some illustrative examples. 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 [5], is introduced below. We use the quantitative deduction rules in order to illustrate the concepts to be introduced.

The notions of signature and Σ-algebra 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, (*sorts*), and* F *is a collection of function declaration 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. A pair* $\langle f, \tau\rangle$ *belonging to* F *will be usually denoted as* $f\colon \tau$*.*

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 non-empty set called the carrier of sort* s*,*
(2) and I *is a function which assigns a map* $I(f)\colon A^{s_1} \times \cdots \times A^{s_k} \to A^s$ *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*.*

For quantitative deduction rules we have a signature with a single sort, say u, and the function types:

$$\begin{array}{lll} \&_P\colon u \times u \to u & \leftarrow_P\colon u \times u \to u & 0.0\colon u \\ \&_G\colon u \times u \to u & \leftarrow_G\colon u \times u \to u & \vdots \\ & & 1.0\colon u \end{array}$$

With this signature we construct the Σ-algebra *unit* where:

- the carrier A^u of sort u is the unit interval $[0, 1]$;
- the constant and function symbols are interpreted as follows:

$$\begin{array}{rcl} I(\&_P)\colon [0,1]\times[0,1] & \longrightarrow & [0,1] \\ (x, y) & \mapsto & x \cdot y \end{array} \qquad \begin{array}{rcl} I(0.0) & : & [0,1] \\ & \mapsto & \text{real number } 0 \end{array}$$

$$\begin{array}{rcl} I(\leftarrow_P)\colon [0,1]\times[0,1] & \longrightarrow & [0,1] \\ (x, y) & \mapsto & \min(1, x/y) \end{array} \qquad \vdots$$

$$\begin{array}{rcl} I(\&_G)\colon [0,1]\times[0,1] & \longrightarrow & [0,1] \\ (x, y) & \mapsto & \min(x, y) \end{array} \qquad \begin{array}{rcl} I(1.0) & : & [0,1] \\ & \mapsto & \text{real number } 1 \end{array}$$

$$I(\leftarrow_G)\colon [0,1]\times[0,1] \longrightarrow [0,1]$$
$$(x,y) \mapsto \begin{cases} 1 & \text{if } x \geq y \\ x & \text{otherwise} \end{cases}$$

The $\&_P$ and $\leftarrow_P$ symbols denote, respectively, the product t-norm and Goguen's implication. The other operators $\&_G$ and $\leftarrow_G$ are the minimum t-norm and Gödel's implication. Despite of the representation used above with just a decimal digit, we assume that for every real number there is a corresponding constant symbol denoting it in the signature. Of course, there are uncountably many constant symbols, which obviously cannot be represented in a denumerable language. We ignore these important technical details in the following discussion, since they will not affect our results. In practical terms, this simply means that "some" programs cannot be represented in the computer. It is a usual practical assumption, to work only with rational numbers.

2.2 Multi-Adjoint Lattices and Multi-Adjoint Algebras

The Σ-algebra concept does not define any particular relation between the interpretation of function symbols defined in the signature. For our purposes we need to make clear the relationship between the arrow symbols, weights and the values resulting from the evaluation of body formulas. 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 $(y \leftarrow z) \succeq x$ iff $y \succeq (x \,\&\, z)$*

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

The first two conditions specify the usual properties of "conjunction" and "implication". The adjoint condition is more interesting and allows us to use many-valued versions of *modus ponens*. The value x can be understood as the weight associated to the rules, and therefore condition $(a3)$ expresses that in order to satisfy the rule the value of the consequent (head) must be larger than or equal to the value of the rule weight conjoined to the value of the body. Dropping any of the sides of the equivalence in condition $(a3)$ destroys the expected properties of models of our programs (see [6]). This is the basic inference rule used in sorted multi-adjoint logic programs.

Extending the results in [3,5,23] 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 5 *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.*

The adjoint condition $(a3)$ can now be fully understood. By setting x to $\top$ we obtain the equivalence:

$$(y \leftarrow_i z) \succeq \top \text{ iff } y \succeq \top \,\&_i\, z \text{ iff } y \succeq z$$

Thus, the truth-value of the arrow symbol is $\top$ iff the value of the head is greater than or equal to the value of the body. This is the expected generalisation of the classical two-valued material implication connective. The adjoint condition lets us introduce the notions of weight and satisfiable rule.

We have also seen in the examples in the introductory section, that it is desirable to allow extra operators besides those necessary in the multi-adjoint lattice definition. The structure which captures this possibility is that of a multi-adjoint algebra.

Definition 6 *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* $i = 1, \ldots, n^s$ *(and possibly some extra operators) such that the tuple* $\mathcal{L}^s$

$$(L^s, \preceq^s, I(\leftarrow_1^s), I(\&_1^s), \ldots, I(\leftarrow_{n^s}^s), I(\&_{n^s}^s))$$

is a multi-adjoint lattice.

For the *unit* Σ-algebra, recall that we have a single sort u with carrier $[0, 1]$. The corresponding partial order $\preceq^u$ is the usual ordering between real numbers in the unit interval, which is a complete lattice. The structure

$$([0, 1], \leq, I(\leftarrow_P), I(\&_P), I(\leftarrow_G), I(\&_G))$$

is a multi-adjoint lattice. In this setting we have two implication symbols, but more can easily be added by introducing the appropriate adjoint pairs; for instance, a missing one is Łukasiewicz's adjoint pair.

Furthermore, multiple sorts can be found underlying the probabilistic deductive databases framework of [12] 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, as well as allowing for more operators.

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

3 Syntax and Semantics of Sorted Multi-Adjoint Logic Programs

Sorted multi-adjoint logic programs were introduced in [7,8], and an enhanced presentation is given below. Our 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[3] $\mathfrak{F} = Terms(\Sigma, \Pi)$. 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 7 *As we are working with two Σ-algebras 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.*

In the sequel we take the liberty of using infix notation whenever it simplifies presentation.

[3] This corresponds to the algebra freely generated from Π and the set of function symbols in Σ, respecting sort assignments.

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 8 *Given a multi-adjoint Σ-algebra $\mathfrak{L}$, 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 L^s;*
(3) The head *of the rule A is a propositional symbol of Π of sort s;*
(4) The body *$\mathcal{B}$ is an implication-free 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.*

Facts are rules with body $\top^s$, the top element of lattice L^s. A *query* (or *goal*) is a propositional symbol intended as a question $?A$ prompting the system. In order to simplify notation, we alternatively represent a rule $\langle A \leftarrow_i^s \mathcal{B}, \vartheta\rangle$ by $A \overset{\vartheta}{\leftarrow}{}_i^s \mathcal{B}$.

Sometimes, we will represent bodies of formulas as $@[B_1, \ldots, B_n]$, where[4] the B_is are the propositional variables occurring in the body and $@$ is the aggregator obtained as a composition. This sets the syntax for our programs.

Example 9 *The following quantitative deduction program illustrates these concepts:*

$$\langle good\_work \leftarrow_P 1.0, 0.9\rangle$$
$$\langle good\_referees \leftarrow_P 1.0, 1.0\rangle$$
$$\langle paper\_accepted \leftarrow_P good\_work \;\&_G\; good\_referees, 0.9\rangle$$

The above program has two facts and a rule. The propositional variables are $good\_work$, $good\_referees$ and $paper\_accepted$ all of sort u; the underlying multi-adjoint algebra unit has been introduced before. Intuitively, $good\_work$ should be assigned the truth-value at least 0.9*; $good\_referees$ the value* 1.0 *and the truth-value $paper\_accepted$ at least* 0.81*.*

We now proceed to formalise these intuitions.

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

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

Definition 10 *An* interpretation *is a mapping* $I\colon \Pi \to \bigcup_s L^s$ *such that* $I(p) \in L^s$ *for every propositional symbol* p *of sort* 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 (see for instance [17] for a proof), each of these interpretations can be uniquely extended to the whole set of formulas $\mathfrak{F}$. The valuation function obtained in this way from an interpretation I is denoted by $\hat{I}$.

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

Definition 11 *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* $\bot^s \in L^s$*.*

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

Definition 12 *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*.*

Example 13 *Consider the interpretation* I_1 *which maps the propositional symbols as follows:*

$$I_1(good\_work) = 0.95 \quad I_1(good\_referees) = 1.0 \quad I_1(paper\_accepted) = 0.92$$

It is clear that all the rules in the program of Example 9 are satisfied. Let us analyse the last one:

$$\langle paper\_accepted \leftarrow_P good\_work \;\&_G\; good\_referees, 0.9 \rangle$$

We have that:

$$\begin{aligned}
& 0.9 \leq \hat{I}_1\,(paper_accepted \leftarrow_P good_work \;\&_G\; good_referees) \\
\textit{iff}\;\; & 0.9 \cdot \hat{I}_1\,(good_work \;\&_G\; good_referees) \leq \hat{I}_1\,(paper_accepted) \\
\textit{iff}\;\; & 0.9 \cdot \min(I_1(good_paper), I_1(good_referee)) \leq I_1(paper_accepted) \\
\textit{iff}\;\; & 0.855 \leq 0.92
\end{aligned}$$

Thus, the rule is satisfied by the given interpretation,

Definition 14 *An element $\alpha \in L^s$ 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 $\alpha \preceq^s I(A)$.*

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

Definition 15 *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*

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

where $\bigsqcup_s$ is the least upper bound in the lattice L^s.

The fundamental result is that the $T_{\mathbb{P}}$ operator is monotonic, since the conjunctors associated with the arrow symbols are monotonically increasing, as well as the functions denoted by the body formulae.

Theorem 16 (Monotonicity of $T_{\mathbb{P}}$) *Let I_1 and I_2 be two interpretations in $\mathcal{I}_{\mathfrak{L}}$, and $\mathbb{P}$ be a sorted multi-adjoint logic program. Operator $T_{\mathbb{P}}$ is monotonic: if $I_1 \sqsubseteq I_2$ then $T_{\mathbb{P}}(I_1) \sqsubseteq T_{\mathbb{P}}(I_2)$.*

The semantics of a sorted multi-adjoint logic program can be characterised, as usual, by the post-fixpoints of $T_{\mathbb{P}}$:

Theorem 17 *An interpretation I of $\mathcal{I}_{\mathfrak{L}}$ is a model of a sorted multi-adjoint logic program $\mathbb{P}$ iff $T_{\mathbb{P}}(I) \sqsubseteq I$.*

By the Knaster-Tarski fix-point theorem, $T_{\mathbb{P}}$ has a least fix-point which, by the previous theorem is also a least model. Thus:

Definition 18 (Declarative Semantics) *The semantics of a sorted multi-adjoint logic program $\mathbb{P}$ is given by the least model $M_{\mathbb{P}}$ of $\mathbb{P}$, which always exists.*

Example 19 *For the program of Example 9 the minimal model M maps*

$$M(good\_work) = 0.9 \quad M(good\_referees) = 1.0 \quad M(paper\_accepted) = 0.81$$

The $T_{\mathbb{P}}$ operator provides a way of "operationally" obtaining the least fix-point of any program:

Theorem 20 (Fix-point Semantics) *Let $\mathbb{P}$ be a sorted multi-adjoint logic program, and consider the transfinite sequence of interpretations of $\mathcal{I}_{\mathfrak{L}}$:*

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

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

Example 21 *The computation of the minimal model of the program of Example 9 is:*

	$good\_work$	$good\_referees$	$paper\_accepted$
$T_{\mathbb{P}}\uparrow^0 =$	0.0	0.0	0.0
$T_{\mathbb{P}}\uparrow^1 =$	0.9	1.0	0.0
$T_{\mathbb{P}}\uparrow^2 =$	0.9	1.0	0.81
$T_{\mathbb{P}}\uparrow^3 =$	0.9	1.0	0.81

The computation stops after three iterations.

The major difference from standard classical logic programming is that our $T_{\mathbb{P}}$ operator might not be continuous, and therefore more than ω iterations may be necessary to "reach" the least fix-point. This possibility is unavoidable if one wants to retain generality. All the other important results carry over to our sorted multi-adjoint logic programs. The single-sorted $T_{\mathbb{P}}$ operator is proved to be monotonic and continuous under very general hypotheses, see [15], 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. For obvious practical reasons, we will explore conditions that guarantee that programs reach their fix-points in at most ω iterations.

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 fix-point 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 (see Example 23).

In [7,8] several results providing sufficient conditions guaranteeing that every query can be answered after a finite number of iterations were announced. In particular, this means that for finite programs the least fix-point 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 termination property we investigate is stated in the following definition, and corresponds to the notion of fixpoint-reachability of Kifer and Subrahmanian [11]:

Definition 22 *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 the classical definite logic programming case it is guaranteed that $T_{\mathbb{P}}$ terminates for every query. However, in our general setting we may have infinite programs that terminate for every query while finite ones may not:

Example 23 *Consider the following infinite program over the unit interval multi-adjoint algebra and the countable number of propositional symbols A_i:*

$$\langle A_1 \leftarrow_P 1.0, 0.1\rangle$$
$$\langle A_2 \leftarrow_P A_1, 0.1\rangle$$
$$\langle A_3 \leftarrow_P A_2, 0.1\rangle$$
$$\vdots$$
$$\langle A_{i+1} \leftarrow_P A_i, 0.1\rangle$$
$$\vdots$$

The least fix-point is only attained at iteration ω of $T_{\mathbb{P}}$, however a fixed query $?A_n$ gets evaluated with value 0.1^n after finitely many steps (specifically at iteration n).

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 for each rule with head A and the 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 24 *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 program in Example 23 has finite dependencies since propositional symbol A_{i+1} depends solely on the values of $A_0, \ldots, A_i$ through $i+1$ rules (edges).

The fact that a propositional symbol has finite dependencies gives us some guarantees that we can finitely compute 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, because we may have an infinite number of truth-values.

Example 25 *[11] Consider the following program, again with respect to the unit single-sorted multi-adjoint Σ-algebra extended with the addition and division of real numbers by 2:*

$$\langle A \leftarrow_P \frac{1+A}{2}, 1.0 \rangle$$

The iterations of the $T_{\mathbb{P}}$ operator are:

	$T_{\mathbb{P}}\uparrow^0$	$T_{\mathbb{P}}\uparrow^1$	$T_{\mathbb{P}}\uparrow^2$	$\cdots$	$T_{\mathbb{P}}\uparrow^n$	$\cdots$	$T_{\mathbb{P}}\uparrow^\omega$	$T_{\mathbb{P}}\uparrow^{\omega+1}$
A	0.0	$\frac{1}{2}$	$\frac{3}{4}$	$\cdots$	$\frac{2^n-1}{2^n}$	$\cdots$	1.0	1.0

Clearly, we obtain a strictly increasing sequence which converges to 1.0*, but this value is only attained at the step ω. Intuitively, this query cannot be evaluated in a finite number of steps.*

The following definition identifies an important class of sorted multi-adjoint logic programs for which we can show that these infinite ascending chains cannot occur, and thus ensure termination.

Definition 26 *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 $f \colon \overbrace{s \times \cdots \times s}^{n} \to s$, i.e. are closed operations in each sort, satisfying the following boundary conditions for every $v \in L^s$ and $k = 0, \ldots, n-1$:*

$$I(f)(\underbrace{\top^s, \ldots, \top^s}_{k}, v, \underbrace{\top^s, \ldots, \top^s}_{n-k-1}) \preceq^s v$$

  where $\top^s$ is the top element of 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 $v \in L^s$ and finite composition of casting functions with overall sort $s \to s$.

In local 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 has to use explicitly the casting functions in the appropriate places; moreover, recall that the connectives are not assumed to be continuous. Local multi-adjoint algebras are basically imposing that operators cannot give more "information" than any of the arguments. The same applies to the composition of casting functions; if one starts with a value v in some sort and then converts it an arbitrary number of times, obtaining a value of the original sort, then this cast value must not be greater than the starting value v. No information gain (increase in truth-values) is obtained. This is particularly important for applications where one has discrete and continuous carriers of domains, e.g. discretisation of continuous domains.

The underlying idea of our first termination result 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 27 *Let $\mathbb{P}$ be a sorted 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 $\{\vartheta \mathbin{\dot{\&}^s_i} \hat{I}(\mathcal{B}) \mid \langle A \leftarrow^s_i \mathcal{B}, \vartheta \rangle \in \mathbb{P}\}$*
- *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 $\vartheta \mathbin{\dot{\&}^s_i} \hat{I}(\mathcal{B})$ 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.*

With this definition, we are able to state a first termination result about sorted multi-adjoint logic programs.

Theorem 28 *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* $T_{\mathbb{P}}^n(\triangle)$ *is a singleton, then* $T_{\mathbb{P}}$ *terminates for every query.*

PROOF. 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 $T_{\mathbb{P}}^{n+1}(\triangle)(A) \succ^s T_{\mathbb{P}}^n(\triangle)(A)$ for $A \in \Pi$, then the culprit collection for $T_{\mathbb{P}}^{n+1}(\triangle)(A)$ 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. The formalisation of this argument is given below:

Firstly, let us prove by induction that, if $T_{\mathbb{P}}^{n+1}(\triangle)(A) \succ^s T_{\mathbb{P}}^n(\triangle)(A)$ for $A \in \Pi$, then the culprit collection for $T_{\mathbb{P}}^{n+1}(\triangle)(A)$ has cardinality at least $n+1$.

Base case: For $n = 0$, consider $A \in \Pi^s$ and assume $T_{\mathbb{P}}^1(\triangle)(A) \succ^s T_{\mathbb{P}}^0(\triangle)(A) = \triangle(A)$ 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 $T_{\mathbb{P}}^n(\triangle)(B) \succ^t T_{\mathbb{P}}^{n-1}(\triangle)(B)$, then the culprit collection for $T_{\mathbb{P}}^n(\triangle)(B)$ has at least n different rules for all sorts t and $B \in \Pi$.

Let $A \in \Pi^s$ and assume $T_{\mathbb{P}}^{n+1}(\triangle)(A) \succ^s T_{\mathbb{P}}^n(\triangle)(A)$, then there is at least one rule in the program, $\langle A \leftarrow_i^s \mathcal{B}, \vartheta \rangle$, such that $T_{\mathbb{P}}^{n+1}(\triangle)(A) = \vartheta \mathbin{\dot{\&}_i^s} \widehat{T_{\mathbb{P}}^n(\triangle)}(\mathcal{B})$. Summing up, we have:

$$T_{\mathbb{P}}^{n+1}(\triangle)(A) = \vartheta \mathbin{\dot{\&}_i^s} \widehat{T_{\mathbb{P}}^n(\triangle)}(\mathcal{B}) \succ^s T_{\mathbb{P}}^n(\triangle)(A) \succ^s \vartheta \mathbin{\dot{\&}_i^s} \widehat{T_{\mathbb{P}}^{n-1}(\triangle)}(\mathcal{B}).$$

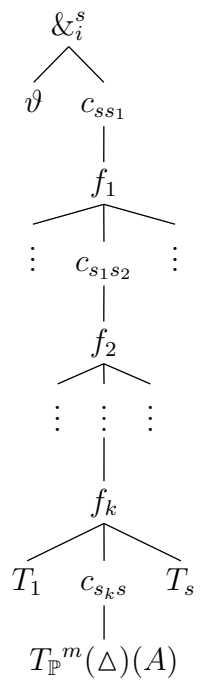
By monotonicity of both $T_{\mathbb{P}}$ and $\dot{\&}_i^s$ then there must be at least one propositional symbol $C \in \Pi^u$ occurring in the body $\mathcal{B}$ which changed value from step $n-1$ to step n, i.e. $T_{\mathbb{P}}^n(\triangle)(C) \succ^u T_{\mathbb{P}}^{n-1}(\triangle)(C)$.

Applying the induction hypothesis, at least n different rules are in the culprit collection of $T_{\mathbb{P}}^n(\triangle)(C)$, and belong to the dependency graph for A since C

occurs in the body of a rule for A. We will prove that $\langle A \leftarrow_i^s \mathcal{B}, \vartheta \rangle$ is not in that culprit collection.

By contradiction, assume the existence of $m < n+1$ such that $\langle A \leftarrow_i^s \mathcal{B}, \vartheta \rangle$ is also a culprit for ${T_{\mathbb{P}}}^m(\triangle)(A)$.

In this case, we can view the computation performed by the $T_{\mathbb{P}}$ operator as the evaluation of the term showed at the right, where each $c_{s_i s_j}$ 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

$$T_{\mathbb{P}}^{n+1}(\triangle)(A) \preceq^s \dot{c}_{ss_1} \left(\cdots (\dot{c}_{s_k s} \left((T_{\mathbb{P}}^m(\triangle)A) \right)) \right)$$

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

$$T_{\mathbb{P}}^{n+1}(\triangle)(A) \preceq^s T_{\mathbb{P}}^m(\triangle)(A) \qquad (1)$$

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

$$T_{\mathbb{P}}^{n+1}(\triangle)(A) \succ^s T_{\mathbb{P}}^n(\triangle)(A) \succeq^s T_{\mathbb{P}}^m(\triangle)(A).$$

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

$$\dot{T}_1 \preceq^{s_k} \top^{s_k} \quad \ldots \quad \dot{T}_s \preceq^{s_k} \top^{s_k}$$

By the boundary conditions we conclude immediately that

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

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. $\square$

This theorem has a set of important corollaries, firstly, we can obtain results about the complexity of reasoning:

Corollary 29 *If the conditions of Theorem 28 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.*

One way of guaranteeing that we have at most one element in $R^I_{\mathbb{P}}(A)$ is to enforce that $\preceq^s$ is a total order. The next corollary results from finiteness of number of rules for A:

Corollary 30 *Let $\mathbb{P}$ be a sorted multi-adjoint logic program with respect to a local multi-adjoint Σ-algebra $\mathfrak{L}$ and to the set of sorted propositional symbols Π, and having finite dependencies. If all the carrier lattices L^s are totally ordered then $T_{\mathbb{P}}$ terminates for every query.*

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

Corollary 31 *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.*

Clearly, programs where only t-norms over the unit interval are used in weighted rules are catered by the previous result, extending the results of [16].

Example 32 *Consider the following program:*

$$\langle a \leftarrow_P b \;\&_G\; c, 0.8\rangle$$
$$\langle a \leftarrow_P 1.0, 0.5\rangle$$
$$\langle b \leftarrow_P a, 0.7\rangle$$
$$\langle c \leftarrow_P 1.0, 1.0\rangle$$

Since we have four rules, and the min operator obeys to the boundary condition of Theorem 28, let us confirm that we need at most 4 iterations to obtain the fixpoint:

	a	b	c
$T_{\mathbb{P}}\uparrow^0 =$	0.0	0.0	0.0
$T_{\mathbb{P}}\uparrow^1 =$	0.5	0.0	1.0
$T_{\mathbb{P}}\uparrow^2 =$	0.5	0.35	1.0
$T_{\mathbb{P}}\uparrow^3 =$	0.5	0.35	1.0

As we shall see, Theorem 28 and its Corollaries can be used to obtain the Probabilistic Deductive Databases termination theorem [12], 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 [10] 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 33 *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 range.*

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. the image of g is a finite set). 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 range, and the same happens with C in $g(f(C))$ and E in $g(E)$, 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 34 *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. The idea is to consider an arbitrary propositional symbol A and the corresponding range dependency sub-graph 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.

The formalisation of the proof proceeds by induction on the depth of a propositional symbol B in the range dependency sub-graph for A.

Depth 0**:** This means that either there is no rule for the propositional symbol, or no propositional symbols occur in bodies for B or all propositional symbols occur "in the scope of" a function symbol having finite range. It is immediate to see that in any of these cases only finitely many values for B can be produced for each rule. Since the number of rules for B is assumed to be finite, the set containing the combination of all these values by the least

upper bound operation has also finite cardinality, and thus the set $T_{\mathbb{P}}(B)$ is also finite.

Depth $n+1$: All the propositional symbols occurring in the range dependency sub-graph for B have depth at most n. By induction hypothesis, all these symbols can take only a finite number of values. Since all propositional symbols occurring in the body of a rule for B are in the dependency graph, or "in the scope of" a function symbol with finite range, then all the bodies of rules for B in $\mathbb{P}$ also have a finite number of possible evaluations. By a similar argument to the base case, we immediately conclude that $T_{\mathbb{P}}(B)$ can take only a finite number of values. □

Corollary 35 *If $\mathbb{P}$ is a sorted multi-adjoint logic program such that all function symbols in the underlying Σ-algebra have finite range, then $T_{\mathbb{P}}$ terminates for every query.*

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

Example 36 *Consider the following variant of the program of Example 25:*

$$\langle A \leftarrow_P f\left(\frac{1+A}{2}\right), 1.0\rangle$$

Suppose function symbol f denotes the function fin defined as

$$fin(x) = \begin{cases} 0 & \text{if } x < 0.5 \\ 0.5 & \text{if } 0.5 \leq x < 1.0 \\ 1.0 & \text{if } x = 1.0 \end{cases}$$

Since f denotes a function with finite range, then the range dependency graph is empty. Therefore, the $T_{\mathbb{P}}$ operator terminates for every query, as the next iterations show:

	A
$T_{\mathbb{P}}\uparrow^0$	$= 0.0$
$T_{\mathbb{P}}\uparrow^1$	$= 0.5$
$T_{\mathbb{P}}\uparrow^2$	$= 0.5$

It is worth to note that the conditions of the theorem do not imply that program $\mathbb{P}$ is acyclic, as in the example. 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 34, boundary conditions have an important role, allowing to obtain a new result combining

Theorems 28 and 34. Specifically, the termination result can be obtained as well 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 37 *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 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 28. 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 [5], 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 38 *The* counting sets for $\mathbb{P}$ and A *for all $n \in \mathbb{N}$, denoted Ξ^A_n, are*

[5] Note we are using again the dependency graph, not the *range* dependency graph.

defined as follows:

$$\Xi_n^A = \{k < n \mid \text{ there is } g_j^i \in G(\mathbb{P}^A) \text{ such that } \dot{g_j^i}\ (\widehat{T_{\mathbb{P}}{}^k(\triangle)}(\mathcal{D}_j^i)) > \dot{g_j^i}\ (\widehat{T_{\mathbb{P}}{}^{k-1}(\triangle)}(\mathcal{D}_j^i))\}$$

With this definition we can state the main lemma needed in the proof of Theorem 37.

Lemma 39 *Under the hypotheses of Theorem 37, if* $T_{\mathbb{P}}{}^{n+1}(\triangle)(A) > T_{\mathbb{P}}{}^n(\triangle)(A)$ *then either* $|\Xi_{n+1}^A| > |\Xi_n^A|$ *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 is a rule indexed by $i \in \{1, \ldots, s\}$ such that

$$T_{\mathbb{P}}{}^{k+1}(\triangle)(A) = \vartheta_i \ \dot{\&}\ \widehat{T_{\mathbb{P}}{}^k(\triangle)}(@_i[g_1^i(\mathcal{D}_1^i), \ldots, g_{r_i}^i(\mathcal{D}_{r_i}^i), C_1^i, \ldots, C_{m_i}^i])$$

now, for the rule indexed by i, by definition of $T_{\mathbb{P}}$ as a l.u.b., we have

$$T_{\mathbb{P}}{}^k(\triangle)(A) \geq \vartheta_i \ \dot{\&}\ \widehat{T_{\mathbb{P}}{}^{k-1}(\triangle)}(@_i[g_1^i(\mathcal{D}_1^i), \ldots, g_{r_i}^i(\mathcal{D}_{r_i}^i), C_1^i, \ldots, C_{m_i}^i])$$

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

$$\dot{g_j^i}\ (\widehat{T_{\mathbb{P}}{}^k(\triangle)}(\mathcal{D}_j^i)) > \dot{g_j^i}\ (\widehat{T_{\mathbb{P}}{}^{k-1}(\triangle)}(\mathcal{D}_j^i))$$

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

$$T_{\mathbb{P}}{}^k(\triangle)(C_j^i) > T_{\mathbb{P}}{}^{k-1}(\triangle)(C_j^i)$$

In the former case, it is obvious that $|\Xi_{k+1}^A| > |\Xi_k^A|$; in the latter case, then the induction hypothesis on the propositional variable C_j^i applies. □

Proof of Theorem 37 The previous lemma provides the key idea:

- 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. □

As a final remark, we can pre-process the body of rules which have occurrences of the least upper bound operators, by introducing rules for each such occurrence. Suppose you have the following rule with respect to unit interval Σ-algebra, extended with the max operator [6]:

$$\langle A \leftarrow_P B \ \&_G \ max(C, D), 0.7 \rangle$$

This can be substituted by the following rules, where $maxCD$ is a new proposition symbol

$$\langle A \leftarrow_P B \ \&_G \ maxCD, 0.7 \rangle$$

$$\langle maxCD \leftarrow_P C, 1.0 \rangle$$

$$\langle maxCD \leftarrow_P D, 1.0 \rangle$$

In this way, we can generalise all the previous results by allowing least upper bound operations in the body. This is an observation due to Umberto Straccia.

In the next section we apply the previous 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 [9], Probabilistic Deductive Databases [12], and Probabilistic Logic Programs with conditional constraints [14]. Both Hybrid Probabilistic Logic Programs, Probabilistic Deductive Databases, and Ordinary Probabilistic Logic Programs can be captured by Residuated Monotonic Logic Programs, as shown in [6]. 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

[6] The least upper-bound operator in the unit interval.

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 [14] 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 [14] 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 both 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

$$(A \mid B_1 \wedge \ldots \wedge B_n)[c, 1] \text{ or } (A \mid \top)[c, 1] \tag{2}$$

Under positively correlated probabilistic interpretations (PCP-interpretations), reasoning becomes tractable and truth-functional. Ordinary conditional constraints (2) of ordinary probabilistic logic programs under PCP-interpretation can be immediately translated to a sorted multi-adjoint logic programming rule

$$\langle A \leftarrow_P B_1 \ \&_G \ \ldots \ \&_G \ B_{n-1} \ \&_G \ B_n, c \rangle$$

over the multi-adjoint *unit* Σ-algebra. The previous rule can also be represented as:

$$\langle A \leftarrow_P c \ \&_P \ (B_1 \ \&_G \ \ldots \ \&_G \ B_{n-1} \ \&_G \ B_n), 1.0 \rangle$$

Clearly, as remarked in [14], the resulting rule is equivalent to a rule of van Emden's Quantitative Deduction [21]. It is pretty clear that in these circumstances all the conditions of Theorem 28 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 [21].

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 [16]. 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 [12] 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[7]. 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 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:

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

[7] 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 will not assume these constraints.

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 [12] assume that 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 [12]. 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

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

is encoded as the following four rules:

$$A^\alpha \xleftarrow[G]{1.0}{}^{m} \ a \ \&_P \ B_1^\alpha \ \&_P \ \ldots \ \&_P \ B_n^\alpha \qquad A^\beta \xleftarrow[G]{1.0}{}^{m} \ b \ \&_P \ B_1^\beta \ \&_P \ \ldots \ \&_P \ B_n^\beta$$

$$A^\gamma \xleftarrow[K]{0.0}{}^{M} \ c \ \vee_P \ B_1^\gamma \ \vee_P \ \ldots \ \vee_P \ B_n^\gamma \qquad A^\delta \xleftarrow[K]{0.0}{}^{M} \ d \ \vee_P \ B_1^\delta \ \vee_P \ \ldots \ \vee_P \ B_n^\delta$$

The functions $\leftarrow_G^m$ and $\&_P$ denote again Gödel's implication (with min conjunctor) and product or Goguen's t-norm. Regarding the sort M, we have implication symbol $\leftarrow_K^M$ denoting Kleene-Dienes implication, i.e.

$$I(\leftarrow_K^M)(x,y) = \max(1-y, x)$$

while $\vee_P$ 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 function 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 28 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 [12]. 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 28.

5.3 Termination of Hybrid Probabilistic Logic Programs

Hybrid Probabilistic Logic Programs [9] 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 pair-wise 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 [8].

The termination results presented in [10] assume finite ground programs. From a difficult analysis of the complex fix-point 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 34 almost directly, given the embedding of Hybrid Probabilistic Logic Programs into Residuated ones presented in [5]. This embedding generates rules of the following four types

$$(1)\ F \overset{[a,b]}{\longleftarrow} s_{\mu_1}\left(\overline{\overline{F_1}}\right) \sqcap \ldots \sqcap s_{\mu_k}\left(\overline{\overline{F_k}}\right) \qquad (3)\ F \overset{[0,b]}{\longleftarrow} s_{\mu_1}\left(\overline{\overline{E_1}}\right) \sqcap \ldots \sqcap s_{\mu_m}\left(\overline{\overline{E_m}}\right)$$

$$(2)\ F \overset{[a,1]}{\longleftarrow} s_{\mu_1}\left(\overline{\overline{E_1}}\right) \sqcap \ldots \sqcap s_{\mu_m}\left(\overline{\overline{E_m}}\right) \qquad (4)\ F \overset{[1,0]}{\longleftarrow} c_\rho\left(\overline{\overline{G}}, \overline{\overline{H}}\right)$$

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 range, and thus when

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

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 [9,5] 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 34 it immediately follows that $T_{\mathbb{P}}$ terminates for every finite ground program, as we intended to show. Just as a side remark, Theorem 37 can also be applied if only conjunctive basic formulas occur in the program, without requiring any reasoning about the shape of the transformed program and its dependencies.

6 A tabling procedure for sorted multi-adjoint logic programming

In the previous sections we have presented several termination results as well as embeddings. The major practical problem is that the $T_{\mathbb{P}}$ operator may take ω iterations to converge, even when all queries terminate. So, an immediate application of the bottom-up fix-point semantics will not be able in some circumstances to determine the computed answer of a particular query after a finite amount of time (because there are an infinite number of propositional symbols). With finite dependencies, one could restrict the computation to subprogram $\mathbb{P}^A$, but this still suffers from a lot of re-computation of the body of rules.

Here we aim at the use of tabulation (tabling, or memoising) methods to increase the efficiency of the previously proposed proof procedures. Tabulation is a technique which is receiving increasing attention in the logic programming and deductive database communities [1,2,19,20]. The underlying idea is, essentially, that atoms of selected tabled predicates as well as their answers are stored in a table. When an identical atom is recursively called, the selected atom is not resolved against program clauses; instead, all corresponding answers computed so far are looked up in the table and the associated answer substitutions are applied to the atom. The process is repeated for all subsequent computed answer substitutions corresponding to the atom. Furthermore, the use of tabulation allows the combination of the contributions of the several rules for a propositional variable, which is essential in the non-boolean case.

In this section, we provide a tabulation goal-oriented query procedure and show that it is terminating for all queries if and only if the immediate consequences operator terminates for every query. On the basis of this property and the previous termination results, we show that the tabulation procedures terminate for a significant class of sorted multi-adjoint logic programs. As a particular case, query answering in several fuzzy and probabilistic logic programming languages are proven to terminate.

6.1 Description of the procedure

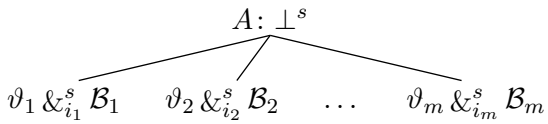
Regarding the definition of an appropriate query procedure for our logic programs, there are two major problems to address: termination and efficiency. On the one hand, the $T_{\mathbb{P}}$ operator is bottom-up but not goal-oriented. Furthermore, in every step the bodies of rules are all re-computed. On the other hand, the usual SLD based implementations of Fuzzy Logic Programming languages (e.g. [24]) are goal-oriented, but inherit the problems of non-termination and re-computation of goals. For tackling these issues, the tabulation implementation technique has been proposed in the deductive databases and logic programming communities [1,2,20]. More recently, an extension of SLD for implementing generalised annotated logic programs has been proposed in [11,19], we will follow these ideas in order to implement our tabling procedure. Other implementation techniques have been proposed for dealing with uncertainty in logic programming, for instance translation into Disjunctive Stable Models [13], but rely on the properties of specific truth-value domains.

In this section we present a general tabulation procedure for our sorted multi-adjoint logic programs. The data structure we will use for the description of the method is that of a *forest*, that is, a finite set of trees. Each one of these trees has a root labelled with a propositional symbol together with a truth-value from the underlying lattice (called the *current value* for the *tabulated* symbol); the rest of the nodes of each of these trees are labelled with an "extended" formula in which some of the propositional symbols have been substituted by its corresponding value. For the description of the adaptation of the tabulation procedure to the framework of multi-adjoint logic programming, we will assume a program $\mathbb{P}$ consisting of a finite number of weighted rules having the form $H \xleftarrow{\vartheta}_{i}^{s} \mathcal{B}$ together with a query $?A$. The purpose of the computational procedure is to give (if possible) the greatest truth-value for A that can be inferred from the information in the program $\mathbb{P}$.

For the sake of clarity in the presentation, we will introduce the following notation: given a propositional symbol A, of a given sort, we will denote by $\mathbb{P}(A)$ the set of rules in $\mathbb{P}$ which have head A. The tabulation procedure uses four basic operations: Create New Tree, New Subgoal, Value Update, and Answer Return. The first operation creates a tree for the first invocation of a given goal. New Subgoal is applied whenever a propositional variable in the body of a rule is found without a corresponding tree in the forest, and resorts to the previous operation. Value update is used to propagate the truth-values of answers to the root of the corresponding tree. Finally, answer return substitutes a propositional variable by the current truth-value in the corresponding tree. We now describe formally the operations:

6.2.1 Rule 1: Create New Tree.

Given a propositional symbol A of sort s, let the set of rules for A be $\mathbb{P}(A) = \{A \xleftarrow{\vartheta_j} {}^{s}_{i_j}\ \mathcal{B}_j \mid j = 1, \ldots, m\}$, construct the tree below, and append it to the current forest. If the forest did not exist, then generate a forest with that tree.



6.2.2 Rule 2: New Subgoal.

Select a non-tabulated propositional symbol C occurring in a leaf of some tree (this means that there is no tree in the forest with the root node labelled with C), then create a new tree as indicated in Rule 1, and append it to the forest.

6.2.3 Rule 3: Value Update.

If there are no propositional symbols in a leaf, then evaluate the corresponding formula (assume that its value is, say, s) and then update the current value (say r) of the propositional symbol at the root of the tree by the value of $\mathrm{lub}(r, s)$, computed in the carrier lattice of that propositional symbol.

6.2.4 Rule 4: Answer Return.

Select in any non-root node a propositional symbol C which is tabulated, and consider that the current value of C is r.

- If the propositional symbol has been selected in a leaf node $\mathcal{N}[\ldots, C, \ldots]$, then extend the branch with the node shown in the figure below.

$$\begin{array}{c} \mathcal{N}[\ldots, C, \ldots] \\ | \\ \mathcal{N}[\ldots, r, \ldots] \end{array}$$

- Otherwise, if the propositional symbol has been selected in a non-leaf node $\mathcal{N}[\ldots, C, \ldots]$ such as that in the left of Fig. 1 then, if $s \preceq r$, then update the whole branch substituting the constant s by r, as in the right of Fig. 1.

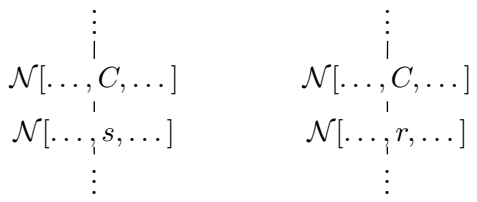


Figure 1. Answer Return operation

It is worth to interpret the execution of each of the previous rules in terms of the better known fix-point semantics.

For instance, the only rule which changes the values of the roots of the trees in the forest is Rule 3. Note that, the only nodes with several immediate successors are the root nodes, these successors correspond to the different rules with head identical to the label of the root node. From there downwards, the extension is done by Rule 4, which either updates the nodes of an existing branch or extends the branch with one new node.

Remark 40 *It is convenient to note that in the leaf of each branch there is a conjunction of the weight of the rule which determined the branch with an instantiation of the body of the rule.*

6.3 A non-deterministic procedure for tabulation

Now, we can state the general non-deterministic procedure for calculating the answer to a given query by using a tabulation technique in terms of the previous rules.

Initial step Create the initial forest with the *create new tree* rule, applied to the query.

Next steps Non-deterministically select a propositional symbol and apply one of the rules 2, 3, or 4.

As we shall show, the order of application of the rules is irrelevant. There are other improvements that can be made to the basic tabulation proof procedure. In particular, all nodes whose value of the body cannot surpass the current value of the root node can be safely removed. A sound rule for determining the maximum value the body can achieve consists in substituting all the propositional variables occurring in the node by $\top^s$. This rule can reduce the search space further more. This pruning rule can be enhanced if there is information available about completed tables in the forest, i.e. the ones which have reached the fix-point.

6.4 Soundness and completeness

As in any non-deterministic procedure, it is necessary to show that the obtained result is independent from the different choices made during the execution of the algorithm. With this aim, we state two propositions, which will provide, as a consequence, the independence of the ordering of applications of steps in the tabulation proof procedure as well as soundness and completeness.

Definition 41 *Given a sorted multi-adjoint logic program $\mathbb{P}$ and a query $?A$. We say that the tabling procedure has constructed a* terminated *forest for $\mathbb{P}$ and $?A$ when no rules of the tabling proof procedure can be applied.*

Proposition 42

(1) The current values of a terminated forest generate a model of $\mathbb{P}^A$. That is, the current values are greater than or equal to those given by the least fix-point of the immediate consequences operator $T_{\mathbb{P}}$.
(2) Given a forest (terminated or not), then for all roots $C_j : r_j$ we have that there exists an iteration k of the $T_{\mathbb{P}}$ operator such that $r_j \leq T_{\mathbb{P}} \uparrow^k (C_j)$.

PROOF. (1) By construction of the forest, each tree has its root labelled with a propositional symbol, say A, and its immediate successors encode the different rules and facts in $\mathbb{P}(A)$ as a chain which ends in an expression without propositional symbols (they have been substituted by values). During the execution of the procedure, Rule 3 is applied to update the current value of the propositional symbol at the root, only if this value is less than the value of the expression in the leaf. Obviously, by definition of the semantics and Remark 40, we obtain that any rule or fact is satisfied. The result follows from the fact that the least fix-point of the operator $T_{\mathbb{P}}$ is the minimal model of the program.

(2) By induction on the number of operations used to generate the forest.

For the base case, assume that only one rule has been used to generate the forest. In this case, we only have one tree in the forest, whose current value is $\bot^s$. Obviously, any iteration of $T_{\mathbb{P}}(\triangle)$ has a value greater than or equal to $\bot^s$ on A of sort s, so we are done.

For the inductive case, consider that the result is true for any forest generated in n steps, and let us prove the result for any forest $\mathcal{F}$ generated in $n+1$ steps.

Our induction hypothesis will be that there exists an integer k such that $r_j \leq {T_{\mathbb{P}}}^k(C_j)$ for all roots $C_j : r_j$ in a forest generated in n steps.

Let us look at the last rule applied for generating $\mathcal{F}$. There is only one case we have to consider, for the only rule which actually can change the current values of the propositional symbols in the roots is Rule 3.

After an application of Rule 3, exactly one propositional variable, say C, has got its current value changed. The new current value is the value of the expression in a leaf of the tree, which has the form $\vartheta_j \,\&^s_{i_j}\, \mathcal{B}_j[r_{j_1}, \ldots, r_{j_m}]$ where the values $r_{j_1}, \ldots, r_{j_m}$ are current values stored in the forest. Therefore, by the induction hypothesis, the monotonicity of the $T_{\mathbb{P}}$ operator, and its very definition we have

$$\vartheta_j \;\dot{\&}^s_{i_j}\, \mathcal{B}_j[r_{j_1}, \ldots, r_{j_m}] \leq \vartheta_j \;\dot{\&}^s_{i_j}\; \mathcal{B}_j[{T_{\mathbb{P}}}^k(C_{j_1}), \ldots, {T_{\mathbb{P}}}^k(C_{j_m})] \leq$$

$$\leq \bigsqcup_s \left\{ \vartheta_j \;\dot{\&}^s_{i_j}\; \mathcal{B}_j[{T_{\mathbb{P}}}^k(C_{j_1}), \ldots, {T_{\mathbb{P}}}^k(C_{j_m})] \mid A \xleftarrow{\vartheta_j}{}^{s}_{i_j}\; \mathcal{B}_j \in \mathbb{P} \right\} = {T_{\mathbb{P}}}^{k+1}(C)$$

As an easy consequence of the previous proposition we obtain the following result, where we recall that $\mathbb{P}^A$ is the set of rules in the dependency graph for propositional symbol A:

Theorem 43 *Consider a sorted multi-adjoint logic program $\mathbb{P}$ and query $?A$*

(1) Every terminated forest for $?A$ calculates exactly the minimal model for program $\mathbb{P}^A$.
(2) The tabulation procedure terminates for a query $?A$ if and only if the minimal model of $\mathbb{P}^A$ is reached by iterating the $T_{\mathbb{P}}$ operator a finite number of times.

The results in Section 4 can guarantee that, under the assumptions of the various theorems, our tabulation proof procedure also terminates. Therefore, our tabulation proof procedure can be used for query-answering with respect to the several formalisms described in Section 5.

We now illustrate the tabulation procedure at work, showing how our tabulation proof procedure handles mutual recursions in a program corresponding to the probabilistic framework of Lakshmanan and Sadri:

Example 44 *Consider the following p-program:*

$$\begin{array}{l}
\Big(a \xleftarrow{\langle[0.8,0.9],[0.0,0.1]\rangle} b, c \;;\; ind, pc\Big) \\
\Big(a \xleftarrow{\langle[0.1,0.3],[0.4,0.6]\rangle} \;;\; ind, pc\Big) \\
\Big(b \xleftarrow{\langle[0.9,1.0],[0.0,0.0]\rangle} \;;\; ind, pc\Big) \\
\Big(c \xleftarrow{\langle[0.7,0.8],[0.0,1.0]\rangle} a \;;\; ind, pc\Big) \\
\Big(c \xleftarrow{\langle[0.3,0.6],[0.2,0.7]\rangle} \;;\; ind, pc\Big)
\end{array}$$

Instead of applying the translation of Section 5.2, we sketch the construction of a new multi-adjoint Σ-algebra, where the underlying complete lattice is the lattice of confidence levels of Probabilistic Deductive Databases under truth-ordering. The translation of the above p-program has the following form:

$$\begin{array}{l}
a \xleftarrow{\top_t} \langle[0.8, 0.9], [0.0, 0.1]\rangle \wedge_{ind} b \wedge_{ind} c \\
a \xleftarrow{\top_t} \langle[0.1, 0.3], [0.4, 0.6]\rangle \\
b \xleftarrow{\top_t} \langle[0.9, 1.0], [0.0, 0.0]\rangle \\
c \xleftarrow{\top_t} \langle[0.7, 0.8], [0.0, 1.0]\rangle \wedge_{ind} a \\
c \xleftarrow{\top_t} \langle[0.3, 0.6], [0.2, 0.7]\rangle
\end{array}$$

Notice that all rules have confidence level $\top_t = \langle[1, 1], [0, 0]\rangle$, meaning that the rule is satisfied iff the value of the body is $\leq_t$ than the head. Furthermore, the conjunctor associated with the implication symbol is the greatest lower bound in truth-ordering, i.e. positive correlation conjunctive mode, and not $\wedge_{ind}$. Since it is not essential to provide an explicit definition of implication, we leave the details to the reader (see also Section 5.2).

Suppose it is intended to determine the truth-degree of proposition a. The computation is started by applying Rule 1 to a and a possible forest generated by the algorithm is presented in Figure 2. All the nodes are annotated by a possible order of creation, and the selected nodes by Rule 2 are underlined. Since $\top_t \otimes_t v = v$, we omit these expressions in the Figure (introduced by Rule 1). Other executions exist, but the computations will terminate in any case and generate the same truth-degrees for all propositional symbols.

The first nodes (i) (ii) and (iii) were created by the Create New Tree *operation*

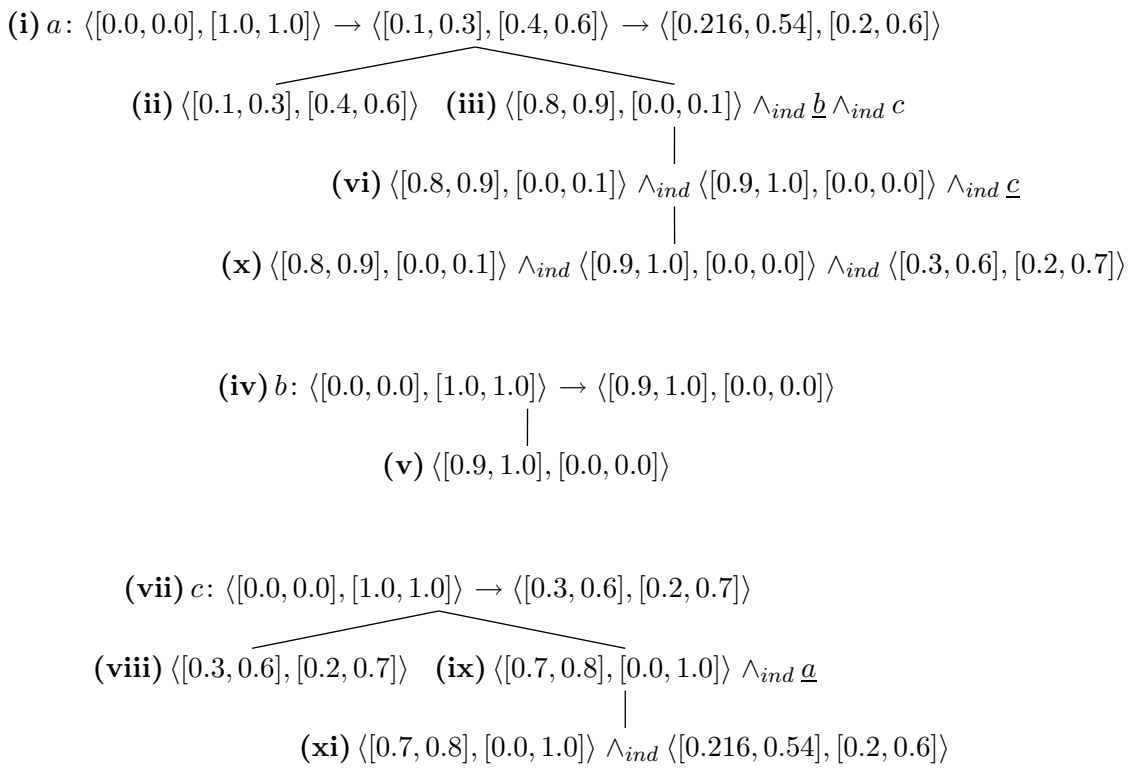


Figure 2. Example forest for query a

(Rule 1). Applying Rule 3 to node (ii) we update the truth-degree for a from $\langle[0.0,0.0],[1.0,1.0]\rangle$ *to* $\langle[0.1,0.3],[0.4,0.6]\rangle$. *The* New SubGoal *selects propositional variable b at node (iii) and creates the new tree with root (iv). The computation proceeds and we get for b the truth-degree* $\langle[0.9,1.0],[0.0,0.0]\rangle$, *by a simple application of* Value Update *to node (v). The* Answer Return *applied to node (iii) generates the new node (vi).*

The procedure now determines the truth-degree of c, and the reader can easily follow the steps. Notice that, New Subgoal *operation at node (ix) does not create a new tree. The truth-value of c is back propagated to node (vi) and originates a new answer update for a. Notice that node (xi) evaluates to* $\langle[0.216,0.54],[0.2,0.73]\rangle$, *and thus the value for a is a mixture of the previous tabled value with the new one. This value is then consumed by node (i) but the truth-value obtained at node (xi) is smaller than the current root node value of c, and the computation terminates since no more operations are applicable. This is expected from the discussion in Section 6.4.*

7 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 such as van Emden's Quantitative Deduction, Possibilistic Logic Programming, Non-classical SLD resolution, Ordinary Probabilistic Logic Programs and Probabilistic Deductive Databases; for all these situations, reasoning is polynomial in the size of the ground program. Last but not least, we have described a general non-deterministic tabulation goal-oriented query procedure for sorted multi-adjoint logic programs over complete lattices. We prove its soundness and completeness as well as independence of the selection ordering.

As future work, on the one hand, a first goal is the attempt to extend this technique to the first order case; on the other hand, we are also interested in gaining a better understanding of Fuzzy Rule Systems to be translated into our framework. An implementation of the tabulation procedure is underway using the GAP package of XSB Prolog [19], as well as a distributed implementation for the use in the Semantic Web. A major distinguishing feature of our tabulation proof-procedure is that it is defined for arbitrary combinations of operators in the body of programs; however, theoretical and/or experimental comparison with existent approaches to the computation of minimal models for fuzzy logic programs are still needed.

Finally, we have ignored in this article the issue of default (or non-monotonic) negation. The introduction of non-monotonic negation raises new problems, but we have already started the research in this direction. In fact, a well-founded and a stable model like semantics allowing non-monotonic constructs in the body of programs were defined in [4]. The termination results are potentially applicable to this non-monotonic (or antitonic) setting, in particular for the well-founded based semantics. Using the results of the present work, we are able to show immediately that each iteration of the well-founded fixpoint operator terminates. However, it has to be shown additionally that this sequence itself terminates, which is by no means a trivial result. We intend to address this problem in the nearby future.

References

[1] R. Bol and L. Degerstedt. The underlying search for magic templates and tabulation. In *Proc. of ICLP93*, pages 793–811, 1993.

[2] W. Chen, T. Swift, and D. S. Warren. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24(3):161–199, 1995.

[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. Antitonic Logic Programs. In *Logic Programming and Nonmonotonic Reasoning, 6th International Conference* (LPNMR'01). *Lect. Notes in Artificial Intelligence* 2173:379-392, Springer 2001.

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

[6] 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, pages 807–814, 2004.

[7] 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, pages 1879–1886, 2004.

[8] C.V. Damásio, J. Medina and M. Ojeda-Aciego. Sorted multi-adjoint logic programs: termination results and applications. *Lect. Notes in Artificial Intelligence* 3229:260-273, 2004.

[9] A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. *J. of Logic Programming*, 43:187–250, 2000.

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

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

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

[13] T. Lukasiewicz. Fixpoint Characterizations for Many-valued Disjunctive Logic Programs with Probabilistic Semantics In *Lect. Notes in Artificial Intelligence* 2173:336–350, 2001.

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

[15] 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.

[16] L. Paulík. Best possible answer is computable for SLD-resolution. *Lecture Notes in Logic*, 6:257–266, 1996.

[17] H. Rasiowa. An Algebraic Approach to NonClassical Logics. *Studies in Logic and the Foundations of Mathematics*, vol. 78. North-Holland, Amsterdam, 1974.

[18] M. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1–2):389–426, 2002.

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

[20] H. Tamaki and T. Sato. OLD resolution with tabulation. In *Proc. of ICLP'86*, pages 84–98, 1986.

[21] M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.

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

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

[24] P. Vojtás and L. Paulík. Soundness and completeness of non-classical extended SLD-resolution. In *Proc. of the Ws. on Extensions of Logic Programming (ELP'96)*. LNCS 1050:289–301, 1996.