A Simple Top-Down Query Answering Procedure for Many-Valued Logic Programming

Umberto Straccia
ISTI - CNR
Via G. Moruzzi, 1
56124 Pisa ITALY
Technical Report: 2006-TR-xx

April 7, 2006

Abstract

We present a simple, yet general top-down query answering procedure for many-valued logic programming, which allow to deal with imprecision and some forms of uncertainty. The main features of the logic are: (*i*) the truth values are taken from a complete lattice; (*ii*) computable functions may appear in the rule bodies to manipulate truth values. To answer queries, we provide a novel and simple tabling-like top-down procedure.

Keywords: deductive databases, logic programming, many-valued logic, imprecision, uncertainty

Category: F.4.1: Mathematical Logic and Formal Languages: Mathematical Logic: [Logic and constraint programming]

Category: I.2.3: Artificial Intelligence: Deduction and Theorem Proving: [Logic programming]

Terms: Theory

1 Introduction

The management of uncertainty and/or imprecision within deduction systems is an important issue whenever the real world information to be represented is of imperfect nature. In logic programming (and deductive databases, in particular), the problem has attracted the attention of many researchers and numerous frameworks have been proposed. Essentially, they differ in the underlying notion of uncertainty theory and imprecision theory and how uncertainty/imprecision values, associated to rules and facts, are managed. Below a list of references and the underlying imprecision and uncertainty theory:

Probability theory: [8, 4, 5, 17, 29, 31, 27, 28, 30, 48, 56, 57, 68, 66, 81, 82, 83, 84, 85, 86, 93, 100, 104, 105, 106, 107, 110, 125, 131];

Fuzzy set theory: [92, 6, 7, 12, 14, 42, 54, 55, 62, 51, 50, 92, 102, 101, 109, 111, 116, 117, 118, 87, 124, 127, 126, 128, 129, 132];

Multi-valued logic: [13, 18, 19, 20, 25, 21, 22, 23, 24, 33, 32, 36, 34, 35, 43, 44, 45, 46, 47, 52, 58, 59, 60, 61, 64, 65, 67, 69, 72, 73, 74, 75, 76, 77, 79, 80, 88, 90, 89, 91, 94, 95, 98, 98, 96, 97, 99, 112, 113, 114, 115, 120, 119, 121, 122];

Possibilistic logic: [2, 3, 1, 15, 40, 108].

We recall that under *uncertainty theory* fall all those approaches in which statements rather than being either true or false, are true or false to some *probability* or *possibility/necessity*, while under *imprecision theory* fall all those approaches in which statements are true to some degree which is taken from a truth space (see [41] for a clarification between the notions of uncertainty and imprecision).

In this work we deal with *imprecision* and, thus, statements have a degree of truth. However, as we will see later on, in some cases we can simulate also some forms of uncertainty (e.g. possibilistic logic programs and probabilistic logic programs under the event independence assumption).

Current frameworks for managing imprecision in logic programming can roughly be classified into *annotation based* (AB) and *implication based* (IB).

In the AB approach (e.g. [60, 61, 103, 104]), a rule is of the form

$$A : f(\beta_1, \ldots, \beta_n) \leftarrow B_1 : \beta_1, \ldots, B_n : \beta_n$$

which asserts "the value of atom A is at least (or is in) $f(\beta_1, \ldots, \beta_n)$, whenever the value of atom B_i is at least (or is in) β_i, $1 \leq i \leq n$". Here f is an n-ary computable function and β_i is either a constant or a variable ranging over an appropriate truth domain.

In the IB approach, (e.g. [18, 24, 68, 69, 97, 124, 126] a rule is of the form

$$A \stackrel{\alpha}{\leftarrow} B_1, ..., B_n$$

which says that the value associated with the implication $B_1 \wedge ... \wedge B_n \rightarrow A$ is α. Computationally, given an assignment I of values to the B_i, the value of A is computed by taking the "conjunction" of the values $I(B_i)$ and then somehow "propagating" it to the rule head. The values the atoms may have are taken from a lattice. More recently, [18, 63, 69, 126] show that most of the frameworks dealing with imprecision and logic programming can be embedded into the IB framework.

To accommodate most frameworks, we have presented a general framework for logic programs with many-valued semantics (see, [120, 119, 121]) where the truth space is a complete lattice (a bilattice [49] is used in case we have to deal with default negation). Sorts, as used in [18, 24], can be simulated by using the join of lattices.

Rules and facts have the very general form

$$A \leftarrow f(B_1, ..., B_n) \ ,$$

where f is an n-ary computable function over lattices and B_i are atoms. Each rule may have a different f. Computationally, given an assignment (interpretation) I of values to the B_i, the value of A is computed by stating that A is at least as true as $f(I(B_1), ..., I(B_n))$ (and, thus, follows the IB approach). The form of the rules is sufficiently expressive to encompass most approaches to many-valued normal logic programming.

Contribution. In this paper we present an simple top-down procedure to answer queries within our formalism. What makes our proposal different from all other ones ([19, 26, 61, 69, 119, 120, 121, 126], – see related work) is that we present a tabulation procedure, which allows us not just to compute *one* answer to a query, but rather *all* answers to a query, where an answer is a query instance together with its degree of truth. Being able to compute all answers to a query is especially very important as ultimately we are interested to rank all answers with respect to their degrees. As illustrative example suppose a user may have the following information need:

"Find *cheap* hotels *near* to the conference location."

Here, *cheap* and *near* are fuzzy predicates. Unlike the classical case, tuples satisfy now these predicates to a score (usually in $[0, 1]$). In the former case the score may depend, e.g., on the price, while in the latter case it may depend e.g. on the distance between the hotel location and the conference location. Therefore, a major problem we have to face with in such cases is that now we want rather the set of all tuples *ranked* according to their *score*, instead of just one answer.

Related work. As seen above, there are many works dealing with imprecision with logic programming with or without negation, either using the AB approach or the IB approach. We recall that it is not difficult to see that the use of arbitrary computable truth combination functions in the body is sufficiently expressive to subsume all those works. Also, it is worth mentioning that very few works address *non-monotonic reasoning*, as [22, 43, 44, 72, 73, 74, 78, 94, 119, 121].

Additionally, in most frameworks, in order to answer to a query, we have to compute the whole intended model (e.g., by a bottom-up fixed-point computation) and then answer with the evaluation of the query in this model. This always requires the computation of a whole model, even if not all the atom's truth is required to determine the answer. Some works that also present top-down procedures are [19, 26, 61, 69, 120, 126], but in none of them non-monotonic negation is considered. The only exception is [119, 121], which deals with normal logic programs over bilattices.

A common characteristics of all these top-down procedures is that they are some variant to the many-valued case of the usual classical SLD resolution [71]. Therefore, the are tailored to compute one answer only. The computation of all answers may led to a well-known non-termination issue in case recursive rules are allowed such as the typical "path" example

$$\texttt{path}(x, y) \leftarrow \texttt{path}(x, z) \wedge \texttt{edge}(z, y) \ .$$

Essentially, any goal containing $\texttt{path}(x, y)$ will unify with the above rule over and over. We avoid this problem by applying a variant of so-called *memoing* techniques (*tabling/magic sets*) developed for classical logic programming (see, e.g. [130] for an overview). Essentially, the basic idea of our procedure is

to collect, during the computation, all correct answers incrementally together in a similar way as it is done for classical Datalog [123] [1].

We proceed as follows. In the next section we present some basic definitions about the logic programming formalism.

2 Preliminaries

Truth spaces. The truth spaces we consider are complete lattices. A *truth space* is a structure $\mathcal{L} = \langle L, \preceq \rangle$ where L is a non-empty set and $\preceq$ is a partial ordering giving $\mathcal{L}$ the structure of a *complete lattice. Meet (or greatest lower bound)* and *join (or least upper bound)* under $\preceq$ are denoted $\wedge$ and $\vee$. *Top and bottom under* $\preceq$ are denoted $\top$ and $\bot$.

The main idea is that an statement $P(a)$, rather than being interpreted as either true or false, will be mapped into a truth value c in L. Examples of typical truth spaces are [2]:

- Classical 0-1: $\mathcal{L}_{\{0,1\}}$ corresponds to the classical truth-space, where 0 stands for 'false', while 1 stands for 'true'.
- Fuzzy: $\mathcal{L}_{[0,1]_{\mathbb{Q}}}$, which relies on the rationals in the unit real interval, is quite frequently used as truth space.
- Four-valued: another frequent truth space is Belnap's $\mathcal{FOUR}$ [9], where L is $\{f, t, u, i\}$ with $f \preceq u \preceq t$ and $f \preceq i \preceq t$. Here, u stands for 'unknown', whereas i stands for inconsistency. We denote the lattice as $\mathcal{L}_B$.
- Many-valued: $L = \langle \{0, \frac{1}{n-1}, \ldots \frac{n-2}{n-1}, 1\}, \leq \rangle$, n positive integer.
- Belief-Doubt: a further popular lattice allows us to reason about *belief and doubt*. Indeed, the idea is to take any lattice $\mathcal{L}$, and to consider the cartesian product $\mathcal{L} \times \mathcal{L}$. For any pair $(b, d) \in \mathcal{L} \times \mathcal{L}$, b indicates the degree of *belief* a reasoning agent has about a sentence s, while d indicates the degree of *doubt* the agent has about s. The order on $\mathcal{L} \times \mathcal{L}$ is determined by $(b, d) \preceq (b', d')$ iff $b \preceq b'$ and $d' \preceq d$, i.e. belief goes up, while doubt goes down. The minimal element is $(\bot, \top)$ (no

[1] See also, `http://tinman.cs.gsu.edu/~raj/8710/f03/datalog-eval.html`

[2] See [66] for more examples.

belief, maximal doubt), while the maximal element is $(\top, \bot)$ (maximal belief, no doubt). We indicate this lattice with $\bar{\mathcal{L}}$.

- Interval: another lattice allows us to reason about *intervals* [43]. Indeed, the idea is to take any lattice $\mathcal{L}$, to consider the cartesian product $\mathcal{L} \times \mathcal{L}$ and for any pair $(b, d) \in \mathcal{L} \times \mathcal{L}$, b indicates the lower bound truth degree of a sentence s, while d indicates the upper bound degree of s. That is, (b, d) denotes the *interval* of truth values $\{c | b \preceq_t c \preceq_t d\}$. The order on $\mathcal{L} \times \mathcal{L}$ is determined by $(b, d) \preceq (b', d')$ iff $b \preceq b'$ and $d \preceq d'$. The minimal element is $(\bot, \top)$ (no constraint), while the maximal element is $(\top, \bot)$ (maximal inconsistency).

We also provide a family $\mathcal{F}$ of $\preceq$-monotone n-ary functions over $\mathcal{L}$ to manipulate truth values.

Generalized logic programs. Fix a truth space $\mathcal{L} = \langle L, \preceq \rangle$. We extend logic programs [71] to the case where *computable functions* $f \in \mathcal{F}$ are allowed to manipulate truth values (see [119, 121]). [3] That is, we allow any $f \in \mathcal{F}$ to appear in the body of a rule to be used to combine the truth of the atoms appearing in the body. The language is sufficiently expressive to accommodate almost all frameworks on many-valued logic programming [119, 121].

A *term*, t, is either a variable or a constant symbol. An *atom*, A, is an expression of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol and all t_i are terms. A *formula*, φ, is an expression built up from the atoms, the truth values $b \in L$ of the truth space and the functions $f \in \mathcal{F}$. The members of the truth space may appear in a formula, as well as functions $f \in \mathcal{F}$. For instance, e.g. in $\mathcal{L}_{[0,1]_\mathbb{Q}}$, the expression

$$\min(p, q) \cdot \max(r, 0.7) + v$$

is a formula φ, where p, q, r and v are atoms. The intuition here is that the truth value of the formula is obtained by determining the truth value of p, q, r and v and then to apply the arithmetic functions to determine the final value of φ.

A *rule* is of the form

$$A \leftarrow \varphi$$

where A is an atom and φ is a formula.

[3] With computable we mean that for any input, the value of f can be determined in finite time.

Example 1 *For instance,*

$$p \leftarrow max(0, q + r - 1)$$

is a rule dictating that p is at least as true as the conjunction of q and r with respect to the Lukasiewicz t-norm $x \wedge y = \max(0, x + y - 1)$ *[53]. On the other hand,*

$$P(x) \leftarrow A(x) + B(x) - A(x) \cdot B(x)$$

dictates that the truth of $P(x)$ *is determined by the algebraic sum of* $A(x)$ *and* $B(X)$ *and may be used to simulate reasoning in logic programming under the probabilistic event independence assumption.*

We note that from a practical point of view, we may have introduced *typed or sorted* terms and predicates in which each argument has a type or sort (as in e.g. [126]). This is useful in practice, but from a theoretical point of view their management is straightforward, so we leave it out for the ease of presentation.

A *generalized normal logic program*, or simply *logic program*, $\mathcal{P}$, is a finite set of rules.

The notions of *Herbrand universe* $H_{\mathcal{P}}$ of $\mathcal{P}$ and *Herbrand base* (as the set of all ground atoms) $B_{\mathcal{P}}$ of $\mathcal{P}$ are as usual. Additionally, given $\mathcal{P}$, the generalized normal logic program $\mathcal{P}^*$ derived from grounding $\mathcal{P}$ is constructed as follows:

1. set $\mathcal{P}^*$ to the set of all ground instantiations of rules in $\mathcal{P}$ [4];
2. replace several rules in $\mathcal{P}^*$ having same head, $A \leftarrow \varphi_1$, $A \leftarrow \varphi_2$, ... with $A \leftarrow \varphi_1 \vee \varphi_2 \vee \ldots$ (recall that $\vee$ is the join operator of the truth space); and
3. if an atom A is not head of any rule in $\mathcal{P}^*$, then add the rule $A \leftarrow \mathtt{f}$ to $\mathcal{P}^*$ (it is a standard practice in logic programming to consider such atoms as *false*).

Note that in $\mathcal{P}^*$, each atom appears in the head of *exactly one* rule and that $\mathcal{P}^*$ is *finite*.

We next recall the usual semantics of logic programs over truth spaces (cf. [119, 120]).

[4] Note that typed terms may reduce the size of $\mathcal{P}^*$.

Interpretations. An *interpretation* I on the truth space $\mathcal{L} = \langle L, \preceq \rangle$ is a mapping from atoms to members of L. I is extended from atoms to formulae in the usual way: (i) for $b \in L$, $I(b) = b$; (ii) for formulae φ and φ', $I(\varphi \wedge \varphi') = I(\varphi) \wedge I(\varphi')$, and similarly for $\vee$; and (iii) for formulae $f(A)$, $I(f(A)) = f(I(A))$, and similarly for n-ary functions. $\preceq$ is extended from $\mathcal{L}$ to the set $\mathcal{I}(\mathcal{L})$ of all interpretations point-wise: (i) $I_1 \preceq I_2$ iff $I_1(A) \preceq I_2(A)$, for every ground atom A.

With $\mathtt{I}_\bot$ we denote the bottom interpretation (it maps any atom into $\bot$). Of course, $\langle \mathcal{I}(\mathcal{L}), \preceq \rangle$ is a complete lattice as well.

Models. I is a *model* of $\mathcal{P}$, denoted $I \models \mathcal{P}$, iff for all $A \leftarrow \varphi \in \mathcal{P}^*$, $I(A) = I(\varphi)$ holds. Note that usually a model has to satisfy $I(\varphi) \preceq I(A)$ only, i.e. $A \leftarrow \varphi \in \mathcal{P}^*$ specifies the necessary condition on A, "A is at least as true as φ". But, as $A \leftarrow \varphi \in \mathcal{P}^*$ is the unique rule with head A, the constraint becomes also sufficient (see also e.g. [44, 77, 78, 121]).

Among all the models, one model plays a special role: namely the $\preceq$-least model $M_\mathcal{P}$ of $\mathcal{P}$. Furthermore, for the sake of this paper, we note that the existence and uniqueness of $M_\mathcal{P}$ is guaranteed by the fixed-point characterization based on the $\preceq$-monotone function $\Phi_\mathcal{P}$: for an interpretation I, for any ground atom A with (unique) $A \leftarrow \varphi \in \mathcal{P}^*$,

$$\Phi_\mathcal{P}(I)(A) = I(\varphi) \ .$$

Then all models of $\mathcal{P}$ are fixed-points of $\Phi_\mathcal{P}$ and vice-versa, and $M_\mathcal{P}$ can be computed in the usual way by iterating $\Phi_\mathcal{P}$ over $\mathtt{I}_\bot$.

In the following, we recall some examples, which both might help informally the reader to get confidence with the formalism and show how our formalism may capture different approaches to the management of imprecision (and some forms of uncertainty) in logic programming (some examples are taken from [69]).

Consider the following logic program with the four rules r_i,

$$\begin{array}{lcl} r_1 & : & A \leftarrow f_1(\alpha_1, B) \\ r_2 & : & A \leftarrow f_2(\alpha_2, C) \\ r_3 & : & B \leftarrow \alpha_3 \\ r_4 & : & C \leftarrow \alpha_4 \end{array}$$

where A, B, C are ground atoms and $\alpha_i \in [0,1]_\mathbb{Q}$.

Example 2 (Classical case, [69]) *Consider* $\mathcal{L}_{\{0,1\}}$ *and* $\alpha_i = 1$, *for* $1 \leq i \leq 4$. *Suppose* f_i *is* min. *Then,* $\mathcal{P}$ *is a program in the standard logic programming framework.*

Example 3 ([40, 69]) *Consider* $\mathcal{L}_{[0,1]}$. *Suppose* $\alpha_1 = 0.8, \alpha_2 = \alpha_3 = 0.7$, *and* $\alpha_4 = 0.8$ *are possibility/necessity degrees associated with the implications. Suppose* f_i *is* min. *Then* $\mathcal{P}$ *is a program in the framework proposed by Dubois et al. [40], which is founded on Zadeh's possibility theory [134]. In a fixed-point evaluation of* $\mathcal{P}$, *the possibility/necessity degrees derived for* A, B, C *are* 0.7, 0.7, 0.8, *respectively.*

Example 4 ([124, 69]) *Consider* $\mathcal{L}_{[0,1]}$ *and suppose* α_i *as defined in Example 3. But, suppose* f_i *is multiplication* $(\cdot)$. *Then* $\mathcal{P}$ *is a program in van Emden's framework [124], which is mathematically founded on the theory of fuzzy sets proposed by Zadeh [133]. In a fixed-point evaluation of* $\mathcal{P}$, *the values derived for* A, B, C *are* 0.56, 0.7, 0.8, *respectively.*

Example 5 (MYCIN [11, 69]) *Consider* $\mathcal{L}_{[0,1]}$, *and suppose* α_i*'s are probabilities defined as in the previous example. Suppose* f_i *is* $(\cdot)$. *However, in order to simulate a probabilistic setting, in particular related to the atom* A, *with independent events, we write the program above as:*

$$\begin{array}{lcl} r_0 & : & A \leftarrow f_s(A', A'') \\ r'_1 & : & A' \leftarrow f_1(0.8, B) \\ r'_2 & : & A'' \leftarrow f_2(0.7, C) \\ r_3 & : & B \leftarrow 0.7 \\ r_4 & : & C \leftarrow 0.8 \end{array}$$

where we use two new atoms A' *and* A'' *to indicate that* A *is head of two rules and use the algebraic sum* $f_s(\alpha, \beta) = \alpha + \beta - \alpha \cdot \beta$ *to sum up the probabilities of deriving* A. *Viewing an atom as an event,* f_s *returns the probability of the occurrence, of any one of two independent events, in the probabilistic sense. Note that* f_s *is the disjunction function used in MYCIN [11]. Let us consider a fixed-point evaluation of* $\mathcal{P}$. *In the first step, we derive* B *and* C *with probabilities* 0.7 *and* 0.8, *respectively. In step 2, applying* r'_1 *and* r'_2, *we obtain two derivations of* A *(namely for* A' *and* A''*), the probability of each of which is* 0.56. *The probability of* A *is then defined as* $f_s(0.56, 0.56) = 0.8064$, *which is indeed the probability that* A *occurs.*

From Example 5 above, it is easy to see that more generally, in order to accommodate independent probabilities, a logic program $\mathcal{P}$ has to be transformed: rules with same head $A \leftarrow \varphi_1$, $A \leftarrow \varphi_2$, ... rather being transformed into $A \leftarrow \varphi_1 \vee \varphi_2 \vee \ldots$, are transformed into $A \leftarrow f_s(\ldots f_s(\varphi_1, \varphi_2) \ldots)$.

In a similar way, we can manage [69].

Example 6 (PDDU [69, 74]) *In [69], a* Parametric Approach to Deductive Databases with Uncertainty *(PDDU) is proposed, where rules have the form*

$$r : A \overset{\alpha_r}{\leftarrow} B_1, ..., B_n; \langle f_d, f_p, f_c \rangle$$

f_d is the disjunction function associated with A and, f_c and f_p are respectively the conjunction and propagation functions associated with the rule r. α_r is the weight of the rule. Roughly, this functions are mappings from $L \times L$ to L and are continuous w.r.t. each one of its arguments and satisfying some constraints such that they behave as conjunction and disjunction functions (see, [69]). The intuition behind a rule is as follows. Ground the program and evaluate each atom B_i. Combine their truth using the conjunction function f_c, i.e. let $c_1 = f_c(B_1, \ldots, B_n)$ (for instance, $c_1 = \min(B_1, \ldots, B_n)$). Then propagate the truth value c_1 to the head using the weight of the rule α_r and the propagation function f_p, i.e. let $c'_1 = f_p(\alpha_r, c_1)$ (for instance, $c'_1 = \alpha_r \cdot c_2$). Repeat this operation for rules heaving A in the head. If $c'_1, \ldots, c'_k$ are all this values, combine them using the disjunction function f_d, $c_A = f_d(c'_1, \ldots, c'_k)$ (for instance, $c_A = \max(c'_1, \ldots, c'_k)$). Informally, a logic program $\mathcal{P}$ in the sense of [69] can be represented in our framework by grounding $\mathcal{P}$ and then transforming rule of the form $r : A \overset{\alpha_r}{\leftarrow} B_1, ..., B_n; \langle f_d, f_p, f_c \rangle$ into

$$A \leftarrow f_p(\alpha_r, f_c(B_1, ..., B_n)) \ .$$

Afterwards, all rules with same head $A \leftarrow \varphi_1$, $A \leftarrow \varphi_2$, ... are transformed into $A \leftarrow f_d(\ldots f_d(\varphi_1, \varphi_2) \ldots)$.

[74] is as [69], but additionally non-monotonic negation is considered as well. We can encode [74] into our framework in the same way as for [69].

Example 7 (Fuzzy Logic Programming [126]) *In [126],* Fuzzy Logic Programming *is proposed, where rules have the form*

$$A \leftarrow f(B_1, ..., B_n)$$

for some specific f and the truth space is $\mathcal{L}_{[0,1]_{\mathbb{Q}}}$. [126] is just a special case of our framework. As an illustrative example consider the following scenario. Assume that we have the following facts, represented in the tables below. There are hotels and conferences, their locations and the distance among locations.

HasLocationH	
HotelID	HasLocationH
h1	hl1
h2	hl2
⋮	⋮

HasLocationC	
ConferenceID	HasLocationC
c1	cl1
c2	cl2
⋮	⋮

Distance		
HasLocationH	HasLocationC	Distance
hl1	cl1	300
hl1	cl2	500
hl2	cl1	750
hl2	cl2	750
⋮	⋮	

Now, suppose that our query is to find hotels close to the conference venue, labeled c1*. We may formulate our query as the rule:*

$$\begin{aligned} \texttt{Query}(h) \leftarrow\ & \min(\\ & \texttt{HasLocationH}(h, hl), \\ & \texttt{HasLocationC}(\texttt{c1}, cl), \\ & \texttt{Distance}(hl, cl, d), \texttt{Close}(d)) \end{aligned}$$

where $\texttt{Close}(x)$ *is defined as*

$$\texttt{Close}(x) = \max(0, 1 - \frac{x}{1000})$$

As a result to that query we get a ranked list of hotels as shown in the table below.

Result List	
HotelID	Closeness degree
h1	0.7
h2	0.25
⋮	⋮

Example 8 ([74]) *Consider* $\mathcal{L}_{[0,1]_\mathbb{Q}}$*, where* $\wedge = \min$ *and* $\vee = \max$*. Consider an insurance company, which has information about its customers used to determine the risk coefficient of each customer. Suppose a value of the risk coefficient is already known, but has to be re-evaluated (the client is a new client and his risk coefficient is given by his precedent insurance company). The company may have:* (i) *data grouped into a set of facts*

$$\begin{aligned} \texttt{Experience(john)} &\leftarrow 0.7 \\ \texttt{Risk(john)} &\leftarrow 0.5 \\ \texttt{Sport\_car(john)} &\leftarrow 0.8\ ; \end{aligned}$$

and (ii) *a set of rules, which after grounding are:*

$$\begin{aligned} \texttt{Good\_driver(john)} &\leftarrow \texttt{Experience(john)} \wedge (0.5 \cdot \texttt{Risk(john)}) \\ \texttt{Risk(john)} &\leftarrow 0.8 \cdot \texttt{Young(john)} \\ \texttt{Risk(john)} &\leftarrow 0.8 \cdot \texttt{Sport\_car(john)} \\ \texttt{Risk(john)} &\leftarrow \texttt{Experience(john)} \wedge (0.5 \cdot \texttt{Good\_driver(john)}) \end{aligned}$$

It turns out that by a bottom-up computation (iterating $\Phi_\mathcal{P}$ *over* $\mathtt{I}_\bot$*) the minimal model is* $M_\mathcal{P}$*, where (for ease, we use first letters only)*

$$\begin{aligned} M_\mathcal{P}(\texttt{R(j)}) &= 0.64 \\ M_\mathcal{P}(\texttt{S(j)}) &= 0.8 \\ M_\mathcal{P}(\texttt{Y(j)}) &= 0 \\ M_\mathcal{P}(\texttt{G(j)}) &= 0.32 \\ M_\mathcal{P}(\texttt{E(j)}) &= 0.7\ . \end{aligned}$$

The following examples shows that ω steps may not be sufficient to compute the minimal model.

Example 9 *Consider* $\mathcal{L}_{[0,1]_\mathbb{Q}}$*, the function* $f(x) = \frac{x+a}{2}$ *(*$0 < a \leq 1, a \in \mathbb{Q}$*), and* $\mathcal{P}_1 = \{A \leftarrow f(A)\}$*. Then the minimal model is attained after* ω *steps of* $\Phi_\mathcal{P}$ *iterations starting from* $\mathtt{I}_\bot(A) = 0$ *and is* $M_\mathcal{P}(A) = a$*.*

Now, consider the function $g(x) = 0$ *if* $x < a$*, and* $g(x) = 1$ *otherwise, and the logic program* $\mathcal{P}_2$ *with rules*

$$\begin{aligned} A &\leftarrow f(A) \\ B &\leftarrow g(A)\ . \end{aligned}$$

Then the minimal model is attained after $\omega+1$ *steps of* $\Phi_\mathcal{P}$ *iterations starting from* $\mathtt{I}_\bot$ *and is* $M_\mathcal{P}(A) = a, M_\mathcal{P}(B) = 1$*.*

However, it is easy not difficult to show that if all functions appearing in $\mathcal{P}$ are continuous, then at most ω steps are necessary to compute the minimal model.

Top-down query answering. A *query* is an expression of the form $?A$ (*query atom*), intended as a question about the truth of the atom A in the minimal model of $\mathcal{P}$. We also allow a query to be a *set* $\{?A_1, \ldots, ?A_n\}$ of query atoms. In that latter case we ask about the truth of all the atoms A_i in the minimal model.

We next recall a basic procedure *Solve* for top-down query answering. Though the procedure and variants of it are reported elsewhere [119, 120, 121], we report it here not only for the sake of completeness of the paper, but also because

- our novel procedure for computing all answers, described in the next section, is inspired on *Solve* and makes its explanation much easier;
- the computational complexity analysis and termination results we are describing here apply to the novel procedure as well.

We anticipate that the main reason why the procedure *Solve* is not suitable to be used for computing all answers to a query $?A$, given $\mathcal{P}$, is that (i) it basically relies on its grounded version $\mathcal{P}^*$, which may be rather huge (exponential with respect to $|\mathcal{P}|$, in general) in applications with many facts; and (ii) it is rather tailored towards ground queries, only.

The top-down procedure [119, 120, 121] is based on a transformation of a program into a system of equations of monotonic functions over complete lattices for which we can compute the least fixed-point in a top-down style. The idea is the following. Consider a complete lattice $\mathcal{L} = \langle L, \preceq \rangle$, a logic program $\mathcal{P}$, its Herbrand base $B_\mathcal{P} = \{A_1, \ldots, A_n\}$ and $\mathcal{P}^*$. Let us associate to each atom $A_i \in B_\mathcal{P}$ a variable x_i, which will take a value in the domain L (sometimes, we will refer to that variable with x_A as well). An interpretation I may be seen as an assignment of truth values to the variables $x_1, ..., x_n$. For the immediate consequence operator $\Phi_\mathcal{P}$, a fixed-point is such that $I = \Phi_\mathcal{P}(I)$, i.e. for all atoms $A_i \in B_P$, $I(A_i) = \Phi_\mathcal{P}(I)(A_i)$. Therefore, we may identify the fixed-points of $\Phi_\mathcal{P}$ as the solutions over L of the system

of equations of the following form:

$$\begin{array}{rcl} x_1 & = & f_1(x_{1_1}, \ldots, x_{1_{a_1}}) \ , \\ & \vdots & \\ x_n & = & f_n(x_{n_1}, \ldots, x_{n_{a_n}}) \ , \end{array} \tag{1}$$

where for $1 \leq i \leq n$, $1 \leq k \leq a_i$, we have $1 \leq i_k \leq n$. Each variable x_{i_k} will take a value in the domain L, each (monotone) function f_i determines the value of x_i (i.e. A_i) given an assignment $I(A_{i_k})$ to each of the a_i variables x_{i_k}. The function f_i implements $\Phi_{\mathcal{P}}(I)(A_i)$. The models of $\mathcal{P}$ are bijectively related to the solutions of the system (1) and the $\preceq$-least solution corresponds to the $\preceq$-least model of $\mathcal{P}$, i.e. $M_{\mathcal{P}}$.

In the general case, we assume that each function $f_i : L^{a_i} \mapsto L$ in Equation (1) is $\preceq$-monotone. We also use f_x in place of f_i, for $x = x_i$. We refer to the monotone system as in Equation (1) as the tuple $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $\mathcal{L}$ is a lattice, $V = \{x_1, ..., x_n\}$ are the variables and $\mathbf{f} = \langle f_1, ..., f_n \rangle$ is the tuple of functions.

As it is well known, a monotonic equation system as (1) has a $\preceq$-least solution, $\mathrm{lfp}(\mathbf{f})$, the $\preceq$-least fixed-point of $\mathbf{f}$ is given as the least upper bound of the $\preceq$-monotone sequence, $\mathbf{y}_0, \ldots, \mathbf{y}_i, \ldots$, where (of course, $\mathbf{f}(\mathbf{y}) = \langle f_1(\mathbf{y}), ..., f_n(\mathbf{y}) \rangle$)

$$\begin{array}{rcl} \mathbf{y}_0 & = & \bot \\ \mathbf{y}_{i+1} & = & \mathbf{f}(\mathbf{y}_i) \end{array}$$

(we point out that in the frequent case of $\mathcal{L}_{[0,1]_{\mathbb{Q}}}$ in which each f_i is a linear function, we may apply directly linear algebra to solve the equational system (1)).

Example 10 *Consider Example 8. Given $\mathcal{P}$, we consider $\mathcal{P}^*$ and represent it as an equational system over $[0,1]_{\mathbb{Q}}$ as follows:*

$$\begin{array}{rcl} x_{\mathtt{E(j)}} & = & 0.7 \\ x_{\mathtt{S(j)}} & = & 0.8 \\ x_{\mathtt{Y(j)}} & = & 0 \\ x_{\mathtt{G(j)}} & = & \min(x_{\mathtt{E(j)}}, 0.5 \cdot x_{\mathtt{R(j)}}) \\ x_{\mathtt{R(j)}} & = & \max(0.5, 0.8 \cdot x_{\mathtt{Y(j)}}, 0.8 \cdot x_{\mathtt{S(j)}}, \min(x_{\mathtt{E(j)}}, 0.5 \cdot x_{\mathtt{G(j)}})) \end{array}$$

It is easily verified that the fixed-points of the $\Phi_{\mathcal{P}}$ operator, i.e. the models of $\mathcal{P}$, are the solutions of the system of equations above and, thus, the least fixed-point, corresponds to the minimal model $M_{\mathcal{P}}$ of $\mathcal{P}$. The bottom-up least fixed-point computation is (the tuples represent $\langle x_{\mathtt{E(j)}}, x_{\mathtt{S(j)}}, x_{\mathtt{Y(j)}}, x_{\mathtt{G(j)}}, x_{\mathtt{R(j)}}\rangle$)

$$\begin{aligned}
\mathbf{y}_0 &= \langle 0,0,0,0,0\rangle \\
\mathbf{y}_1 &= \langle 0.7,0.8,0,0,0.5\rangle \\
\mathbf{y}_2 &= \langle 0.7,0.8,0,0.25,0.64\rangle \\
\mathbf{y}_3 &= \langle 0.7,0.8,0,0.32,0.64\rangle \\
\mathbf{y}_4 &= \mathbf{y}_3 \ ,
\end{aligned}$$

which corresponds to the minimal model of the program, as expected.

Informally, the algorithm works as follows (see Table 1). Assume, we are interested in the value of x_0 in the fixed-point lfp($\mathbf{f}$) of the system. We call the procedure with $Solve(\mathcal{S}, \{x_0\})$. We associate to each variable x_i a marking $\mathtt{v}(x_i)$ denoting the current value of x_i (the mapping $\mathtt{v}$ contains the current value associated to the variables). Initially, $\mathtt{v}(x_i)$ is $\mathbf{0}$. We start with putting x_0 in the *active* list of variables $\mathtt{A}$, for which we evaluate whether the current value of the variable is identical to whatever its right-hand side evaluates to. When evaluating a right-hand side it might of course turn out that we do indeed need a better value of some sons, which will assumed to have the value $\mathbf{0}$ and put them on the list of active nodes to be examined. In doing so we keep track of the dependencies between variables, and whenever it turns out that a variable changes its value all variables that might depend on this variable are put in the active set to be examined. At some point (even if cyclic definitions are present) the active list will become empty and we have actually found part of the fixed-point, sufficient to determine the value of the query x_0. [5].

$Solve(\mathcal{S}, Q)$ uses some auxiliary functions and data structures: given the equational system (1),

- $\mathtt{s}(x)$ denotes the set of *sons* of x, i.e. $\mathtt{s}(x_i) = \{x_{i_1}, \ldots, x_{i_{a_i}}\}$ (the set of variables appearing in the right hand side of the definition of x_i);
- $\mathtt{p}(x)$ denotes the set of *parents* of x, i.e. the set $\mathtt{p}(x) = \{x_i : x \in \mathtt{s}(x_i)\}$ (the set of variables depending on the value of x).

[5] The attentive reader will notice that the *Solve* has commonalities with the so-called *tabulation* procedures, like [16, 19].

Procedure *Solve*$(\mathcal{S}, Q)$
Input: $\preceq$-monotonic system $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $Q \subseteq V$ is the set of query variables
Output: A set $B \subseteq V$, with $Q \subseteq B$ such that the mapping $\mathtt{v}$ restricted to B, equals to lfp($\mathbf{f}$).

1. $\mathtt{A} := Q$, $\mathtt{dg} := Q$, $\mathtt{in} := \emptyset$, **for all** $x \in V$ **do** $\mathtt{v}(x) = \mathbf{0}$, $\mathtt{exp}(x) = \mathtt{false}$
2. **while** $\mathtt{A} \neq \emptyset$ **do**
3. **select** $x_i \in \mathtt{A}$, $\mathtt{A} := \mathtt{A} \setminus \{x_i\}$, $\mathtt{dg} := \mathtt{dg} \cup \mathtt{s}(x_i)$
4. $r := f_i(\mathtt{v}(x_{i_1}), ..., \mathtt{v}(x_{i_{a_i}}))$
5. **if** $r \succ \mathtt{v}(x_i)$ **then** $\mathtt{v}(x_i) := r$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{p}(x_i) \cap \mathtt{dg})$ **fi**
6. **if not** $\mathtt{exp}(x_i)$ **then** $\mathtt{exp}(x_i) = \mathtt{true}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{s}(x_i) \setminus \mathtt{in})$, $\mathtt{in} := \mathtt{in} \cup \mathtt{s}(x_i)$ **fi**

od

Table 1: General top-down algorithm. Grounded version.

- the variable `dg` collects the variables that may influence the value of the query variables;
- the array variable `exp` traces the equations that has been "expanded" (the body variables are put into the active list);
- the variable `in` keeps track of the variables that have been put into the active list so far due to an expansion (to avoid, to put the same variable multiple times in the active list due to function body expansion).

Example 11 *Consider Example 8 and query variable $x_{R(j)}$ (we ask for the risk coefficient of John). In Table 2 we report a sequence of $Solve(\mathcal{S}, \{x_{R(j)}\})$ computation. Each line is a sequence of steps in the 'while loop'. What is left unchanged is not reported.*

The fact that only a part of the model is computed becomes evident, as the computation does not change if we add any program $\mathcal{P}'$ to $\mathcal{P}$ not containing atoms of $\mathcal{P}$, while a bottom-up computation will consider $\mathcal{P}'$ as well.

Given a system $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $\mathcal{L} = \langle L, \preceq \rangle$, let $h(\mathcal{L})$ be the *height* of the truth-value set L, i.e. the length of the longest strictly $\preceq$-increasing chain in L minus 1, where the length of a chain $v_1, ..., v_\alpha, ...$ is the cardinal $|\{v_1, ..., v_\alpha, ...\}|$. The *cardinal* of a set X is the least ordinal α such that α and X are *equipollent*, i.e. there is a bijection from α to X. For instance, $h(\mathcal{FOUR}) = 2$ w.r.t. $\preceq_k$ as well as w.r.t. $\preceq_t$, while $h([0,1]_{\mathbb{Q}}) = \omega$. It can be shown that the above algorithm behaves correctly.

1.	$\mathtt{A} := \{x_{\mathtt{R(j)}}\}, x_i := x_{\mathtt{R(j)}}, \mathtt{A} := \emptyset, \mathtt{dg} := \{x_{\mathtt{R(j)}}, x_{\mathtt{Y(j)}}, x_{\mathtt{S(j)}}, x_{\mathtt{E(j)}}, x_{\mathtt{G(j)}}\}, r := 0.5, \mathtt{v}(x_{\mathtt{R(j)}}) := 0.5,$ $\mathtt{A} := \{x_{\mathtt{G(j)}}\}, \mathtt{exp}(x_{\mathtt{R(j)}}) := 1, \mathtt{A} := \{x_{\mathtt{Y(j)}}, x_{\mathtt{S(j)}}, x_{\mathtt{E(j)}}, x_{\mathtt{G(j)}}\}, \mathtt{in} := \{x_{\mathtt{Y(j)}}, x_{\mathtt{S(j)}}, x_{\mathtt{E(j)}}, x_{\mathtt{G(j)}}\}$
2.	$x_i := x_{\mathtt{Y(j)}}, \mathtt{A} := \{x_{\mathtt{S(j)}}, x_{\mathtt{E(j)}}, x_{\mathtt{G(j)}}\}, r := 0, \mathtt{exp}(x_{\mathtt{Y(j)}}) := 1$
3.	$x_i := x_{\mathtt{S(j)}}, \mathtt{A} := \{x_{\mathtt{E(j)}}, x_{\mathtt{G(j)}}\}, r := 0.8, \mathtt{v}(x_{\mathtt{S(j)}}) := 0.8, \mathtt{A} := \{x_{\mathtt{E(j)}}, x_{\mathtt{G(j)}}, x_{\mathtt{R(j)}}\}, \mathtt{exp}(x_{\mathtt{S(j)}}) := 1$
4.	$x_i := x_{\mathtt{E(j)}}, \mathtt{A} := \{x_{\mathtt{G(j)}}, x_{\mathtt{R(j)}}\}, r := 0.7, \mathtt{v}(x_{\mathtt{E(j)}}) := 0.7, \mathtt{exp}(x_{\mathtt{E(j)}}) := 1$
5.	$x_i := x_{\mathtt{G(j)}}, \mathtt{A} := \{x_{\mathtt{R(j)}}\}, r := 0.25, \mathtt{v}(x_{\mathtt{G(j)}}) := 0.25, \mathtt{exp}(x_{\mathtt{G(j)}}) := 1,$ $\mathtt{in} := \{x_{\mathtt{Y(j)}}, x_{\mathtt{S(j)}}, x_{\mathtt{E(j)}}, x_{\mathtt{G(j)}}, x_{\mathtt{R(j)}}\}$
6.	$x_i := x_{\mathtt{R(j)}}, \mathtt{A} := \emptyset, r := 0.64, \mathtt{v}(x_{\mathtt{R(j)}}) := 0.64, \mathtt{A} := \{x_{\mathtt{G(j)}}\}$
7.	$x_i := x_{\mathtt{G(j)}}, \mathtt{A} := \emptyset, r := 0.32, \mathtt{v}(x_{\mathtt{G(j)}}) := 0.32, \mathtt{A} := \{x_{\mathtt{R(j)}}\}$
8.	$x_i := x_{\mathtt{G(j)}}, \mathtt{A} := \emptyset, r := 0.64$
10.	stop. return v (in particular, $\mathtt{v}(x_{\mathtt{R(j)}}) = 0.64$)

Table 2: Top-down computation related to Example 11.

Proposition 1 ([119, 120]) *Given a monotone system of equations* $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$*, then there is a limit ordinal* λ *such that after* $|\lambda|$ *steps* $Solve(\mathcal{S}, Q)$ *determines a set* $B \subseteq V$*, with* $Q \subseteq B$ *such that the mapping* v *equals* $lfp(\mathbf{f})$ *on* B*, i.e.* $\mathtt{v}_{|B} = lfp(\mathbf{f})_{|B}$*.*

Computational complexity. From a computational point of view, by means of appropriate data structures, the operations on A, v, dg, in, exp, p and s can be performed in constant time. Therefore, step 1 is $O(|V|)$, all other steps, except step 2 and step 4 are $O(1)$. Let $c(f_x)$ be the maximal cost of evaluating function f_x on its arguments, so step 4 is $O(c(f_x))$. It remains to determine the number of loops of step 2. In case the height $h(\mathcal{L})$ of the lattice $\mathcal{L}$ is finite, observe that any variable is increasing in the $\preceq$ order as it enters in the A list (step 5), except it enters due to step 6, which may happen one time only. Therefore, each variable x_i will appear in A at most $a_i \cdot h(\mathcal{L}) + 1$ times, where a_i is the arity of f_i, as a variable is only re-entered into A if one of its son gets an increased value (which for each son only can happen $h(\mathcal{L})$ times), plus the additional entry due to step 6. As a consequence, the worst-case complexity is

$$O(\sum_{x_i \in V} (c(f_i) \cdot (a_i \cdot h(\mathcal{L}) + 1)) \ .$$

Therefore:

Proposition 2 ([119, 120]) *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$. If the computing cost of each function in $\mathbf{f}$ is bounded by c, the arity bounded by a, and the height is bounded by h, then the worst-case complexity of the algorithm Solve is $O(|V|cah)$.*

In case the height of a lattice is not finite, the computation may not terminate after a finite number of steps (see Example 9). Fortunately, under reasonable assumptions on the functions, we may guarantee the termination of *Solve*. We exploit two of such conditions. Consider a monotonic equational system $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$. Consider a function $f \colon L \to L$, where $\langle L, \preceq \rangle$ is a lattice. Let $[\bot]_f$ be the *f-closure of* $\{\bot\}$, i.e. the smallest set that contains $\{\bot\}$ and is closed under f. We say that f *has a finite generation* (see also [10] for more on this issue) iff $[\bot]_f$ is finite. For instance, it can be verified that the functions $\wedge, \vee$ have a finite generation on *any* finite set $X \subseteq L$. More concretely, over $\mathcal{L}_{[0,1]_{\mathbb{Q}}}$, $\min, \max$ and $\max(x+y-1, 0), \min(x+y, 1)$ have a finite generation, while e.g. the product $x \cdot y$ and the algebraic sum $x+y-x \cdot y$ have not. Note also that if f, g have a finite generation over X then so has $f \circ g$. Therefore, given an equational system $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$. If $\mathbf{f}$ has a finite generation, then $[\bot]_{\mathbf{f}}$ is finite. That is, $\{\bot, \mathbf{f}(\bot), \mathbf{f}^2(\bot), ...\}$ is finite. In particular, on induction on the computation of the $\preceq$-least fixed-point of $\mathcal{S}$ it can be shown that at each step of the bottom-up computation of the $\preceq$-least fixed-point, the values of the variables are in $[\bot]_{\mathbf{f}}$. Therefore, the *height* of $[\bot]_{\mathbf{f}}$, $h([\bot]_{\mathbf{f}})$, is finite. On the other hand, it can easily be seen that *Solve* terminates if the sequence, $\bot, \mathbf{f}(\bot), \mathbf{f}^2(\bot), ...$ converges after a finite number of steps. Therefore:

Proposition 3 ([119, 120]) *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$. Then Solve terminates iff $\mathbf{f}$ has a finite generation. If the cost of computing each of the functions in $\mathbf{f}$ is bounded by c and the arity bounded by a then the worst-case complexity of the algorithm Solve is $O(|V|cah)$, where h is the height of $[\bot]_{\mathbf{f}}$.*

The second condition, which guarantees the termination of *Solve*, is inspired directly by [18] and is a special case of above. On bilattices, we

say that a function $f\colon \mathcal{L}^n \to \mathcal{L}$ is *bounded* iff $f(x_1, \ldots, x_n) \preceq \bigwedge_i x_i$. Now, consider a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$. We say that $\mathbf{f}$ is *bounded* iff each f_i is a composition of functions, each of which is either bounded, or a constant in $\mathcal{L}$ or one of $\vee, \wedge$. For instance, the function in Example 9 is not bounded, while $f_i(x, y) = \max(0, x + y - 1) \wedge 0.3$ over $\mathcal{L}_{[0,1]_\mathbb{Q}}$ is. The idea is to prevent the existence of an infinite ascending chain of the form $\bot \prec \mathbf{f}(\bot) \prec \ldots \prec \mathbf{f}^m(\bot) \prec \ldots$. In fact, roughly, consider a $\preceq$-monotone function $\mathbf{f} = \mathbf{g} \circ \mathbf{h}$, where $\mathbf{g}$ is a bounded function, while $\mathbf{h}$ is the composition of constants in $\mathcal{L}$ or functions among $\vee, \wedge$. Then $\bot \preceq \mathbf{f}(\bot) = \mathbf{g} \circ \mathbf{h}(\bot) = \mathbf{g}(\mathbf{h}(\bot)) \preceq \mathbf{h}(\bot)$. But $\mathbf{h}$ has a finite generation and, thus, so has $\mathbf{f}$. The argument for $\mathbf{f} = \mathbf{h} \circ \mathbf{g}$ is similar. Therefore:

Proposition 4 ([119, 120]) *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $\mathbf{f}$ is bounded. Then Solve terminates.*

Note that for bounded functions $\mathbf{f} = \mathbf{g} \circ \mathbf{h}$, the height of $[\bot]_\mathbf{f}$ is given by the height of $[\bot]_\mathbf{h}$. This latter height is bounded by the number $n = |V|$ as $\mathbf{h}^n(\bot) = \mathbf{h}^{n+1}(\bot)$ (this is compatible with [18]). This implies that the worst-case complexity of the algorithm *Solve* is $O(|V|^2 ca)$ in that case.

We are ready now to finalize the query answering procedure. Let $\mathcal{P}$ be a logic program and consider $\mathcal{P}^*$. As already pointed out, each atom appears exactly once in the head of a rule in $\mathcal{P}^*$. The system of equations that we build from $\mathcal{P}^*$ is straightforward. Assign to each atom A a variable x_A and substitute in $\mathcal{P}^*$ each occurrence of A with x_A. Finally, substitute each occurrence of $\leftarrow$ with $=$ and let $\mathcal{S}(\mathcal{P}) = \langle \mathcal{L}, V, \mathbf{f}_\mathcal{P} \rangle$ be the resulting equational system. Of course, $|V| = |B_\mathcal{P}|$, $|\mathcal{S}(\mathcal{P})|$ can be computed in time $O(|\mathcal{P}|)$ and all functions in $\mathcal{S}(\mathcal{P})$ are $\preceq$-monotone. As $\mathbf{f}_\mathcal{P}$ is one to one related to $\Phi_\mathcal{P}$, it follows that the $\preceq$-least fixed-point of $\mathcal{S}(\mathcal{P})$ corresponds to the minimal model of $\mathcal{P}$. The algorithm $Answer(\mathcal{L}, \mathcal{P}, ?A)$, first computes $\mathcal{S}(\mathcal{P})$ and then calls $Solve(\mathcal{S}(\mathcal{P}), \{x_A\})$ and returns the output $\mathtt{v}$ on the query variable, where $\mathtt{v}$ is the output of the call to *Solve*. *Answer* behaves correctly (see Example 11).

Proposition 5 ([119, 120]) *Let $\mathcal{L}$, $\mathcal{P}$ and $?A$ be a be a truth space, a logic program and a query, respectively. Then $M_\mathcal{P}(A) = Answer(\mathcal{L}, \mathcal{P}, \{?A\})$* [6].

[6] The extension to a set of query atoms is straightforward.

From a computational point of view, we can avoid the cost of translating $\mathcal{P}$ into $\mathcal{S}(\mathcal{P})$ as we can directly operate on $\mathcal{P}$. So the cost $O(|\mathcal{P}|)$ can be avoided. In case the height of the lattice is finite, from Proposition 2 it follows immediately that the worst-case complexity for top-down query answering under the minimal model semantics of a logic program $\mathcal{P}$ is $O(|B_\mathcal{P}|cah)$.

Furthermore, often the cost of computing each of the functions of $\mathbf{f}_\mathcal{P}$ is in $O(1)$. By observing that $|B_\mathcal{P}|a$ is in $O(|\mathcal{P}|)$ we immediately have that in this case the complexity is $O(|\mathcal{P}|h)$.

It follows that over the lattice $\mathcal{FOUR}$ ($h = 2$) the top-down algorithm works in linear time as in case the height is a fixed parameter, i.e. a constant. We can conclude that the additional expressive power of logic programs over lattices (with functions with constant cost, and lattices with fixed height) does not increase the computational complexity of classical propositional logic programs, which is linear. The computational complexity of the case where the height of the lattice is not finite is determined by Proposition 3 and Proposition 4. In general, the continuity of the functions in $\mathcal{S}(\mathcal{P})$ guarantees the termination after at most ω steps.

3 Computing all answers

Often a query atom $?Q$ over a logic program $\mathcal{P}$ has associated a rule in $\mathcal{P}$ of the form

$$Q(\mathbf{x}) \quad \leftarrow \quad \varphi(\mathbf{x}, \mathbf{y}) \tag{2}$$

in which we ask for all substitutions θ of the variables $\mathbf{x}$ and truth degrees v, i.e. *answers* $\langle \theta, v \rangle$, such that $M_\mathcal{P}(Q(\mathbf{x})\theta) = v$ holds. That is, by substituting the variables in $\mathbf{x}$ using θ, the evaluation of the query in the minimal model is v. We say that the answer $\langle \theta, v \rangle$ is *correct* iff $M_\mathcal{P}(Q(\mathbf{x})\theta) = v$.

By relying on the method of the previous section, one immediate way to compute all answers is as follows. We consider $\mathcal{P}^*$, which contains all instantiations of the query rule. That is, $\mathcal{P}^*$ contains a rule

$$Q(\mathbf{c}) \quad \leftarrow \quad \bigvee_{\mathbf{c}'} \varphi(\mathbf{c}, \mathbf{c}')$$

for each tuple $\mathbf{c}$ which can be formed from the constants of the Herbrand universe $H_\mathcal{P}$ of $\mathcal{P}$ (similarly, $\mathbf{c}'$ is a tuple of constants in $H_\mathcal{P}$). As a a consequence, in order to find all answers $\langle \theta, v \rangle$ to $?Q$, we have to call the procedure

Answer with the list of query atoms S_Q defined as

$$S_Q := \bigcup_{\mathbf{c}} \{Q(\mathbf{c})\} \ .$$

$Answer(\mathcal{S}(\mathcal{P}), S_Q)$ returns then the degree of truth $v_\mathbf{c}$ of all instantiations $Q(\mathbf{c})$ of $Q(\mathbf{x})$. All the answer tuples $\langle \theta_\mathbf{c}, v_\mathbf{c} \rangle$, where $\theta_\mathbf{c}$ is the substitution $\theta_\mathbf{c} = \{\mathbf{x}/\mathbf{c}\}$, are the answers we are looking for.

A drawback of query answering based on grounded logic programs is that that both the size of S_Q, $|S_Q|$, and that of $\mathcal{P}^*$ may be large. In particular,$\mathcal{P}^*$ may have exponentially more rules than $\mathcal{P}$. In the following we modify the *Answer* algorithm in such a way, which allows us to find all answers $\langle \theta, v \rangle$ to query $?Q$, *without grounding neither the program nor the query.*

For the rest of the paper we assume that $\mathcal{P}$ is made out of an *extensional database* (EDB), $\mathcal{P}_E$, and an *intentional database* (IDB), $\mathcal{P}_I$. The extensional database is a set of facts of the form

$$R(c_1, \ldots, c_n) \leftarrow b \ ,$$

where $R(c_1, \ldots, c_n)$ is a ground atom and b is a truth value. The intentional database is a set of rules for the form

$$P(\mathbf{x}) \leftarrow \varphi(\mathbf{x}, \mathbf{y}) \tag{3}$$

in which the predicates occurring in the extensional database (called *extensional predicates*) do not occur in the head of rules of the intentional database. Essentially, we do not allow that the fact predicates occurring in $\mathcal{P}_E$ can be redefined by $\mathcal{P}_I$. We also assume that the *intentional predicate* symbol P occurs in the head of at most one rule in the intentional database. Due to the expressiveness of rule bodies, it is not difficult to see that by using an equality predicate, logic programs can be put into this form.

For convenience, for each n-ary extensional predicate R, we represent the facts $R(c_1, \ldots, c_n) \leftarrow b$ in $\mathcal{P}$ by means of a relational $n+1$-ary table tab_R, containing the records $\langle c_1, \ldots, c_n, b \rangle$. Thus, the table contains all the instances of R together with their degree.

An *answer set* is a set of answers. Given two answers $\delta_1 = \langle \theta_1, v_1 \rangle$ and $\delta_2 = \langle \theta_2, v_2 \rangle$, we define $\delta_1 \preceq \delta_2$ iff $\theta_1 = \theta_2$ and $v_1 \preceq v_2$. We write $\delta_1 \prec \delta_2$ iff $\theta_1 = \theta_2$ and $v_1 \prec v_2$. The intuition is that δ_2 represents a "better" answer than δ_1. If Δ_1 and Δ_2 are two sets of answers, we write $\Delta_1 \preceq \Delta_2$ iff for all

$\delta_1 \in \Delta_1$ there is $\delta_2 \in \Delta_2$ such that $\delta_1 \preceq \delta_2$. We write $\Delta_1 \prec \Delta_2$ iff $\Delta_1 \preceq \Delta_2$ and there is $\delta_2 \in \Delta$ such that for no $\delta_1 \in \Delta$, $\delta_2 \preceq \delta_1$ holds. Analogously, the intuition is that Δ_2 represents a "better" set of answers than Δ_1. For a given n-ary predicate P and a set of answers Δ_P of P, i.e. a set of answers $\langle \theta, v \rangle$ in which $\theta = \{x_1/c_1, \ldots, x_n/c_n\}$ is a substitution of the n variables with constants ($P\theta$ is ground), for convenience we represent Δ_P as an $n+1$-ary table tab_{Δ_P}, containing the records $\langle c_1, \ldots, c_n, v \rangle$.

For the sake of illustrative purposes, we consider the following well-known example in which we define a path a weighted graph.

Example 12 *Consider the lattice $\mathcal{L}_{[0,1]_\mathbb{Q}}$ and the following rules:*

$$\begin{aligned} \texttt{path}(x,y) &\leftarrow \texttt{edge}(x,y) \\ \texttt{path}(x,y) &\leftarrow \texttt{path}(x,z) \wedge \texttt{edge}(z,y) \end{aligned}$$

These rules have to be transformed into a form in which the path predicate appears in one rule only:

$$\texttt{path}(x,y) \leftarrow \texttt{edge}(x,y) \vee (\texttt{path}(x,z) \wedge \texttt{edge}(z,y)) \tag{4}$$

The logic program $\mathcal{P}$ contains the intentional database containing rule (4) *and the extensional database of edges shown in the relational table $tab_{\texttt{edge}}$ below (the omitted edges have degree* 0*):*

$tab_{\texttt{edge}}$		
a	b	0.3
b	a	0.4
a	c	0.5
c	b	0.6

Of course, $tab_{\texttt{edge}}$ is also the set $tab_{\Delta_{\texttt{edge}}}$ of correct answers of predicate edge*, while it can be verified that the set of correct answers of predicate path is*

given by the relational table:

$tab_{\Delta_{\texttt{path}}}$		
a	a	0.4
a	b	0.5
a	c	0.5
b	a	0.4
b	b	0.4
b	c	0.4
c	a	0.4
c	b	0.6
c	c	0.4

One possibility to compute a correct answer is to follow a SLD-refutation like procedure [71] adapted to the many-valued case, as described for instance in [25, 126]. To compute *all* answers corresponds to compute all SLD-refutations. The downside of this approach is that in case of recursive definitions such as in Example 12 above, we may lead to an infinite loop (due to the recursive definition of `path`).

One way to overcome to this difficulty is to use so-called *memoing* techniques (*tabling/magic sets*) developed for classical logic programming (see, e.g. [130] for an overview).

In this work, we present a tabling like procedure applied to our top-down query answering procedure presented for grounded logic programs. The basic idea of our procedure is similar to the *Solve* algorithm, but now, we try to collect, during the computation, all correct answers incrementally.

At first, consider a general rule of the form (3), i.e. $p(\mathbf{x}) \leftarrow \varphi(\mathbf{x}, \mathbf{y})$. We note that $\varphi(\mathbf{x}, \mathbf{y})$ depends on a computable function f and the predicates $p_1, \ldots, p_k$, which occur in the rule body $\varphi(\mathbf{x}, \mathbf{y})$. Assume that $\Delta_{p_1}, \ldots, \Delta_{p_k}$ are the answers collected so far for the predicates $p_1, \ldots, p_k$. Let us consider a procedure $eval(p, \Delta_{p_1}, \ldots, \Delta_{p_k})$, which computes the set of answers $\langle \{\mathbf{x}/\mathbf{c}\}, v\rangle$ of p, by evaluating the body $\varphi(\mathbf{x}, \mathbf{y})$ over the data provided by $\Delta_{p_1}, \ldots, \Delta_{p_k}$. Formally, let I be an interpretation restricted to the predicates $p_1, \ldots, p_k$ and tuples in such that for all n_i-ary predicates p_i, $I(p_i(\mathbf{c_{i_j}})) = v_i$ if $\langle \mathbf{c_{i_j}}, v_{i_j}\rangle \in \Delta_{p_i}$ and $I(p_i(\mathbf{c_{i_j}})) = 0$ otherwise. Then

$$eval(p, \Delta_{p_1}, \ldots, \Delta_{p_k}) = \{\langle \{\mathbf{x}/\mathbf{c}\}, v\rangle \mid v = max_{\mathbf{c}'} I(\varphi(\mathbf{c}, \mathbf{c}'))\} \ ,$$

where $\mathbf{c}'$ is a tuple of constants occurring in $\bigcup_i \Delta_{p_i}$. For instance, consider Example 12. Assume that both $\Delta_{\texttt{edge}}$ and $\Delta_{\texttt{path}}$ are given by $tab_{\texttt{edge}}$. Then $eval(\texttt{path}, \Delta_{\texttt{edge}}, \Delta_{\texttt{path}})$ returns the set of answers $\Delta'_{\texttt{path}}$ shown below:

$tab_{\Delta'_{\texttt{path}}}$		
a	a	0.3
a	b	0.5
a	c	0.5
b	a	0.4
b	b	0.3
b	c	0.4
c	a	0.4
c	b	0.6

Essentially, $eval(\texttt{path}, \Delta_{\texttt{edge}}, \Delta_{\texttt{path}})$ requires to compute for each tuple satisfying the body of rule (4) its truth degree with respect to the tables $tab_{\Delta_{\texttt{edge}}}$ and $tab_{\Delta_{\texttt{path}}}$. The tuples satisfying the body can be obtained using relational algebra:

$$\pi_{1,2}(tab_{\Delta_{\texttt{edge}}}) \cup \pi_{1,5}(tab_{\Delta_{\texttt{edge}}} \bowtie_{2=4} tab_{\Delta_{\texttt{path}}}) \ .$$

In substance *eval* revises the set of answers for `path`. Note that with respect to Example 12, $tab_{\Delta'_{\texttt{path}}}$ does not have the record $\langle c, c, 0.4\rangle$, and the truth of $\langle a, a\rangle$ and $\langle b, b\rangle$ are smaller. Furthermore, we obtain the correct answers after reiterating the evaluation step once more. That is, it is easily verified that $eval(\texttt{path}, \Delta_{\texttt{edge}}, \Delta'_{\texttt{path}})$ returns all correct answers of *path*.

In the following, we are not going to detail the $eval(p, \Delta_{p_1}, \ldots, \Delta_{p_k})$ procedure, though it has to be carefully be written to minimize the table lookups and joins. It can be obtained by of means of a combination of SQL statements over the tables (similarly to how it works for Datalog) and the application of the functions occurring in the rule body of p. We point out that $eval(p, \Delta_{p_1}, \ldots, \Delta_{p_k})$ can also be seen as a query to a database made out by the relations $tab_{\Delta_{p_1}}, \ldots, tab_{\Delta_{p_k}}$ and that any successive evaluation step corresponds to the execution of the same query over an updated database. We refer the reader to e.g. [37, 38, 39, 70] concerning the problem of repeatedly evaluating the same query to a database that is being updated between successive query requests. In this situation, it may be possible to use the difference between successive database states and the answer to the query in one state to reduce the cost of evaluating the query in the next state.

Procedure *Answer*$(\mathcal{L}, \mathcal{P}, Q)$
Input: Complete lattice $\mathcal{L}$, logic program $\mathcal{P}$, set of query predicate symbols Q
Output: A mapping $\mathtt{v}$ such that it contains all correct answers of predicates in Q.

1. $\mathtt{A} := Q$, $\mathtt{dg} := Q$, $\mathtt{in} := \emptyset$, **for all** predicate symbols p in $\mathcal{P}$ **do** $\mathtt{v}(p) = \emptyset$, $\mathtt{exp}(p) = \mathtt{false}$
2. **while** $\mathtt{A} \neq \emptyset$ **do**
3. **select** $p_i \in \mathtt{A}$, $\mathtt{A} := \mathtt{A} \setminus \{p_i\}$, $\mathtt{dg} := \mathtt{dg} \cup \mathtt{s}(p_i)$
4. **if** (p_i extensional predicate) $\wedge$ ($\mathtt{v}(p_i) = \emptyset$) **then** $\mathtt{v}(p_i) := \Delta_{p_i}$
5. **if** (p_i intentional predicate) **then** $\Delta_{p_i} := eval(p_i, \mathtt{v}(p_{i_1}), ..., \mathtt{v}(p_{i_{k_i}}))$
6. **if** $\Delta_{p_i} \succ \mathtt{v}(p_i)$ **then** $\mathtt{v}(p_i) := \Delta_{p_i}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{p}(p_i) \cap \mathtt{dg})$ **fi**
7. **if not** $\mathtt{exp}(p_i)$ **then** $\mathtt{exp}(p_i) = \mathtt{true}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{s}(p_i) \setminus \mathtt{in})$, $\mathtt{in} := \mathtt{in} \cup \mathtt{s}(p_i)$ **fi**

od

Table 3: General top-down algorithm. Non-grounded version.

We describe now our query answering procedure. Informally, our procedure works as *Solve* except that now the variables are the predicate symbols and the variable $\mathtt{v}$ holds all the answers collected so far for the predicates. At each iteration step we select a new predicate p from the queue $\mathtt{A}$ and evaluate it using the *eval* function with respect to the answers gathered so far. If the evaluation leads to a better answer set for p, we update the current answer set $\mathtt{v}(p)$ and add all predicates p', whose rule body contains p (the parents of p), to the queue $\mathtt{A}$.

The procedure is describe in Table 3. Some explanations of the procedure are in order:

- for predicate symbol p_i, $\mathtt{s}(p_i)$ is the set of predicate symbols occurring in the rule body of p_i, i.e. the *successors* of p_i;
- for predicate symbol p_i, $\mathtt{p}(p_i) = \{p_j : p_i \in \mathtt{s}(p_j)\}$, i.e. the *predecessors* of p_i;
- in step 5, $p_{i_1}, \ldots, p_{i_{k_i}}$ are all predicate symbols occurring in the rule body of p_i, i.e. $\mathtt{s}(p_i) = \{p_{i_1}, \ldots, p_{i_{k_i}}\}$;
- the variables $\mathtt{dg}$, $\mathtt{exp}$ and $\mathtt{in}$ work as for the *Solve* procedure.

Example 13 *Consider Example 12. Let us compute all correct answers of predicate path. So, let* $Q = \{\mathtt{path}\}$. *The execution of* $Answer(\mathcal{L}_{[0,1]_{\mathbb{Q}}}, \mathcal{P}, Q)$ *is shown in Table 4. Table 4 also reports* Δ_{p_i} *and* $\mathtt{v}(p_i)$ *at each iteration* i.

1. $\mathtt{A} := \{\mathtt{path}\}, p_i := \mathtt{path}, \mathtt{A} := \emptyset, \mathtt{dg} := \{\mathtt{path}, \mathtt{edge}\}, \Delta_{\mathtt{path}} := \emptyset$
   $\mathtt{exp}(\mathtt{path}) := \mathtt{1}, \mathtt{A} := \{\mathtt{path}, \mathtt{edge}\}, \mathtt{in} := \{\mathtt{path}, \mathtt{edge}\}$

2. $p_i := \mathtt{path}, \mathtt{A} := \{\mathtt{edge}\}, \Delta_{\mathtt{path}} := \emptyset$

3. $p_i := \mathtt{edge}, \mathtt{A} := \emptyset, \Delta_{\mathtt{edge}} \succ \mathtt{v}(\mathtt{edge}), \mathtt{v}(\mathtt{edge}) := \Delta_{\mathtt{edge}}, \mathtt{A} := \{\mathtt{path}\}, \mathtt{exp}(\mathtt{edge}) := \mathtt{1}$

4. $p_i := \mathtt{path}, \mathtt{A} := \emptyset, \Delta_{\mathtt{path}} \succ \mathtt{v}(\mathtt{path}), \mathtt{v}(\mathtt{path}) := \Delta_{\mathtt{path}}, \mathtt{A} := \{\mathtt{path}\}$

5. $p_i := \mathtt{path}, \mathtt{A} := \emptyset, \Delta_{\mathtt{path}} \succ \mathtt{v}(\mathtt{path}), \mathtt{v}(\mathtt{path}) := \Delta_{\mathtt{path}}, \mathtt{A} := \{\mathtt{path}\}$

6. $p_i := \mathtt{path}, \mathtt{A} := \emptyset, \Delta_{\mathtt{path}} \succ \mathtt{v}(\mathtt{path}), \mathtt{v}(\mathtt{path}) := \Delta_{\mathtt{path}}, \mathtt{A} := \{\mathtt{path}\}$

7. $p_i := \mathtt{path}, \mathtt{A} := \emptyset, \Delta_{\mathtt{path}} = \mathtt{v}(\mathtt{path})$

8. $\mathtt{stop}.\ \mathtt{return}\ \mathtt{v}(\mathtt{path})$

Iter i	Δ_{p_i}	$\mathtt{v}(p_i)$
0.	-	$\mathtt{v}(\mathtt{edge}) = \mathtt{v}(\mathtt{path}) = \emptyset$
1.	$\Delta_{\mathtt{path}} = \emptyset$	-
2.	$\Delta_{\mathtt{path}} = \emptyset$	-
3.	$\Delta_{\mathtt{edge}} = \{\langle a, b, 0.3\rangle, \langle b, a, 0.4\rangle, \langle a, c, 0.5\rangle, \langle c, b, 0.6\rangle\}$	$\mathtt{v}(\mathtt{edge}) = \Delta_{\mathtt{edge}}$
4.	$\Delta_{\mathtt{path}} = \{\langle a, b, 0.3\rangle, \langle b, a, 0.4\rangle, \langle a, c, 0.5\rangle, \langle c, b, 0.6\rangle\}$	$\mathtt{v}(\mathtt{path}) = \Delta_{\mathtt{path}}$
5.	$\Delta_{\mathtt{path}} = \{ \langle a, a, 0.3\rangle, \langle a, b, 0.5\rangle, \langle a, c, 0.5\rangle, \langle b, a, 0.4\rangle, \langle b, b, 0.3\rangle, \langle b, c, 0.4\rangle, \langle c, a, 0.4\rangle, \langle c, b, 0.6\rangle \}$	$\mathtt{v}(\mathtt{path}) = \Delta_{\mathtt{path}}$
6.	$\Delta_{\mathtt{path}} = \{ \langle a, a, 0.4\rangle, \langle a, b, 0.5\rangle, \langle a, c, 0.5\rangle, \langle b, a, 0.4\rangle, \langle b, b, 0.4\rangle, \langle b, c, 0.4\rangle, \langle c, a, 0.4\rangle, \langle c, b, 0.6\rangle, \langle c, c, 0.4\rangle \}$	$\mathtt{v}(\mathtt{path}) = \Delta_{\mathtt{path}}$
7.	$\Delta_{\mathtt{path}} = \{ \langle a, a, 0.4\rangle, \langle a, b, 0.5\rangle, \langle a, c, 0.5\rangle, \langle b, a, 0.4\rangle, \langle b, b, 0.4\rangle, \langle b, c, 0.4\rangle, \langle c, a, 0.4\rangle, \langle c, b, 0.6\rangle, \langle c, c, 0.4\rangle \}$	-

Table 4: Top-down computation related to Example 13.

For completeness, we report the computation for Example 8.

Example 14 *Let us consider a similar example to Example 8, in which the intentional database is written as*

$$
\begin{aligned}
\texttt{Good\_driver(x)} &\leftarrow \texttt{Experience(x)} \wedge (0.5 \cdot \texttt{Risk(x)}) \\
\texttt{Risk(x)} &\leftarrow (0.8 \cdot \texttt{Young(x)}) \vee \\
&\quad\ (0.8 \cdot \texttt{Sport\_car(x)}) \vee \\
&\quad\ (\texttt{Experience(x)} \wedge (0.5 \cdot \texttt{Good\_driver(x)})) \vee \\
&\quad\ \texttt{PriorRisk(x)})
\end{aligned}
$$

and the extensional database is

$tab_{\Delta_{\texttt{Experience}}}$	
john	0.7
tim	0.6
elisa	0.3

$tab_{\Delta_{\texttt{PriorRisk}}}$	
john	0.5
tim	0.4
elisa	0.1

$tab_{\Delta_{\texttt{Sport\_car}}}$	
john	0.8
tim	0.1
elisa	0.6

Suppose we are interested in knowing all risk factors. So, let $Q = \{\texttt{Risk}\}$. *The execution of* $Answer(\mathcal{L}_{[0,1]_{\mathbb{Q}}}, \mathcal{P}, Q)$ *is shown in Table 5. Table 5 also reports* Δ_{p_i} *and* $\mathtt{v}(p_i)$ *at each iteration* i *(for ease, we use first letters of the predicates only).*

Similarly to the grounded case, we have that:

Proposition 6 *Let* $\mathcal{L}$, $\mathcal{P}$ *and* $?Q$ *be a truth space, a logic program and a query, respectively. Then then there is a limit ordinal* λ *such that after* $|\lambda|$ *steps* $Answer(\mathcal{L}, \mathcal{P}, Q)$ *returns the set all of correct answers of* $\mathcal{P}$ *with respect to the predicates in* Q.

From a computational point of view, the analysis is the same as for the *Solve* algorithm. It is worth noting that, as the propositional Example 9 shows, the computation may not terminate after a finite number of steps.

1. $\mathtt{A} := \{\mathtt{R}\}, p_i := \mathtt{R}, \mathtt{A} := \emptyset, \mathtt{dg} := \{\mathtt{R}, \mathtt{Y}, \mathtt{S}, \mathtt{E}, \mathtt{P}, \mathtt{G}\}, \Delta_{\mathtt{R}} := \emptyset$
   $\mathtt{exp}(\mathtt{R}) := 1, \mathtt{A} := \{\mathtt{Y}, \mathtt{S}, \mathtt{E}, \mathtt{P}, \mathtt{G}\}, \mathtt{in} := \{\mathtt{Y}, \mathtt{S}, \mathtt{E}, \mathtt{P}, \mathtt{G}\}$

2. $p_i := \mathtt{Y}, \mathtt{A} := \{\mathtt{S}, \mathtt{E}, \mathtt{P}, \mathtt{G}\}, \Delta_{\mathtt{Y}} := \emptyset, \mathtt{exp}(\mathtt{Y}) := 1$

3. $p_i := \mathtt{S}, \mathtt{A} := \{\mathtt{E}, \mathtt{P}, \mathtt{G}\}, \Delta_{\mathtt{S}} \succ \mathtt{v}(\mathtt{S}), \mathtt{v}(\mathtt{S}) := \Delta_{\mathtt{S}}, \mathtt{A} := \{\mathtt{E}, \mathtt{P}, \mathtt{G}, \mathtt{R}\}, \mathtt{exp}(\mathtt{S}) := 1$

4. $p_i := \mathtt{E}, \mathtt{A} := \{\mathtt{P}, \mathtt{G}, \mathtt{R}\}, \Delta_{\mathtt{E}} \succ \mathtt{v}(\mathtt{E}), \mathtt{v}(\mathtt{E}) := \Delta_{\mathtt{E}}, \mathtt{exp}(\mathtt{E}) := 1$

5. $p_i := \mathtt{P}, \mathtt{A} := \{\mathtt{G}, \mathtt{R}\}, \Delta_{\mathtt{P}} \succ \mathtt{v}(\mathtt{P}), \mathtt{v}(\mathtt{P}) := \Delta_{\mathtt{P}}, \mathtt{exp}(\mathtt{P}) := 1$

6. $p_i := \mathtt{G}, \mathtt{A} := \{\mathtt{R}\}, \Delta_{\mathtt{G}} = \emptyset, \mathtt{exp}(\mathtt{G}) := 1, \mathtt{in} := \{\mathtt{Y}, \mathtt{S}, \mathtt{E}, \mathtt{P}, \mathtt{G}, \mathtt{R}\}$

7. $p_i := \mathtt{R}, \mathtt{A} := \emptyset, \Delta_{\mathtt{R}} \succ \mathtt{v}(\mathtt{R}), \mathtt{v}(\mathtt{R}) := \Delta_{\mathtt{R}}, \mathtt{A} := \{\mathtt{G}\}$

8. $p_i := \mathtt{G}, \mathtt{A} := \emptyset, \Delta_{\mathtt{G}} \succ \mathtt{v}(\mathtt{G}), \mathtt{v}(\mathtt{G}) := \Delta_{\mathtt{G}}, \mathtt{A} := \{\mathtt{R}\}$

9. $p_i := \mathtt{R}, \mathtt{A} := \emptyset, \Delta_{\mathtt{R}} = \mathtt{v}(\mathtt{R}),$

10. stop. return $\mathtt{v}(\mathtt{R})$

Iter i	Δ_{p_i}	$\mathtt{v}(p_i)$
0.	-	$\emptyset$
1.	$\Delta_{\mathtt{R}} = \emptyset$	-
2.	$\Delta_{\mathtt{Y}} = \emptyset$	-
3.	$\Delta_{\mathtt{S}} = \{\langle \mathtt{j}, 0.8\rangle, \langle \mathtt{t}, 0.1\rangle, \langle \mathtt{e}, 0.6\rangle\}$	$\mathtt{v}(\mathtt{S}) = \Delta_{\mathtt{S}}$
4.	$\Delta_{\mathtt{E}} = \langle \mathtt{j}, 0.7\rangle, \langle \mathtt{t}, 0.6\rangle, \langle \mathtt{e}, 0.3\rangle\}$	$\mathtt{v}(\mathtt{E}) = \Delta_{\mathtt{E}}$
5.	$\Delta_{\mathtt{P}} = \langle \mathtt{j}, 0.5\rangle, \langle \mathtt{t}, 0.4\rangle, \langle \mathtt{e}, 0.1\rangle\}$	$\mathtt{v}(\mathtt{P}) = \Delta_{\mathtt{P}}$
6.	$\Delta_{\mathtt{G}} = \emptyset$	-
7.	$\Delta_{\mathtt{R}} = \langle \mathtt{j}, 0.64\rangle, \langle \mathtt{t}, 0.4\rangle, \langle \mathtt{e}, 0.48\rangle\}$	$\mathtt{v}(\mathtt{R}) = \Delta_{\mathtt{R}}$
8.	$\Delta_{\mathtt{G}} = \langle \mathtt{j}, 0.32\rangle, \langle \mathtt{t}, 0.2\rangle, \langle \mathtt{e}, 0.24\rangle\}$	$\mathtt{v}(\mathtt{G}) = \Delta_{\mathtt{G}}$
9.	$\Delta_{\mathtt{R}} = \langle \mathtt{j}, 0.64\rangle, \langle \mathtt{t}, 0.4\rangle, \langle \mathtt{e}, 0.48\rangle\}$	-

Table 5: Top-down computation related to Example 14.

4 Conclusions

We have presented a simple, general, yet effective top-down algorithm to retrieve *all* correct answers to queries for logic programs over lattices with arbitrary continuous functions in the body to manipulate truth values. We believe that its interest relies on its easiness for an effective implementation and on avoiding the non-termination problem of typical SLD-based resolution methods.

As immediate future activity we try to extend the algorithm to deal with non-monotinicity. We plan to rely on [120] as starting point, which is based on an extension of the *Solve* algorithm.

Disclaimer

The list of references above is by no means intended to be all-inclusive. The authors of this overview apologizes both with the authors and with the readers for all the relevant works which are not cited here.

References

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

[2] Teresa Alsinet, Lluís Godo, and Sandra Sandri. On the semantics and automated deduction fo PLFC, a logic of possibilistic uncertainty and fuzzyness. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, 1999.

[3] Teresa Alsinet and Lluis Godo Lluis Godo. A complete calcultis for possibilistic logic programming with fuzzy propositional variables with fuzzy propositional variables. In *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI-00)*, pages 1–10. Morgan Kaufmann, 2000.

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

[5] J. F. Baldwin. A theory of mass assignments for artificial intelligence. *Lecture Notes in Computer Science*, 833:22–34, 1994.

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

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

[8] Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with answer sets. In *Proceedings of the 7th International Conference in Logic Programming and Nonmonotonic Reasoning (LPNMR-04)*, number 2923 in Lecture Notes in Artificial Intelligence, pages 21–33, Fort Lauderdale, FL, USA, 2004. Springer Verlag.

[9] Nuel D. Belnap. A useful four-valued logic. In Gunnar Epstein and J. Michael Dunn, editors, *Modern uses of multiple-valued logic*, pages 5–37. Reidel, Dordrecht, NL, 1977.

[10] Elmar Böhler, Christian Glasser, Bernhard Schwarz, and Klaus Wagner. Generation problems. In *29th International Symposium on Mathematical Foundations of Computer Science (MFCS-04)*, volume 3153 of *Lecture Notes in Computer Science*, pages 392–403. Springer Verlag, 2004.

[11] B.G. Buchanan and E.H. Shortli. A model of inexact reasoning in medicine. *Mathematical Bioscience*, 23:351–379, 1975.

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

[13] J. Calmet, J. Lu, M. Rodriguez, and J. Schü. Signed formula logic programming: operational semantics and applications. In Zbigniew W.

Rás and Maciek Michalewicz, editors, *Proceedings of the Ninth International Symposium on Foundations of Intelligent Systems*, volume 1079 of *LNAI*, pages 202–211, Berlin, 9–13 1996. Springer.

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

[15] Chesnevar Carlos, Simari Guillermo, Alsinet Teresa, and Godo Lluis. A logic programming framework for possibilistic argumentation with vague knowledge. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 76–84, Arlington, Virginia, 2004. AUAI Press.

[16] Weidong Chen and David S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, 1996.

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

[18] Carlos Viegas Damásio, J. Medina, and M. Ojeda Aciego. Sorted multi-adjoint logic programs: Termination results and applications. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA-04)*, number 3229 in Lecture Notes in Computer Science, pages 252–265. Springer Verlag, 2004.

[19] Carlos Viegas Damásio, J. Medina, and M. Ojeda Aciego. A tabulation proof procedure for residuated logic programming. In *Proceedings of the 6th European Conference on Artificial Intelligence (ECAI-04)*, 2004.

[20] Carlos Viegas Damásio, J. Medina, and M. Ojeda Aciego. Termination results for sorted multi-adjoint logic programs. In *Proceedings of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-04)*, pages 1879–1886, 2004.

[21] Carlos Viegas Damásio and Luís Moniz Pereira. A survey of paraconsistent semantics for logic programs. In D. Gabbay and P. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, pages 241–320. Kluwer, 1998.

[22] Carlos Viegas Damásio and Luís Moniz Pereira. Antitonic logic programs. In *Proceedings of the 6th European Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, number 2173 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[23] Carlos Viegas Damásio and Luís Moniz Pereira. Monotonic and residuated logic programs. In Salem Benferhat and Philippe Besnard, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 6th European Conference, ECSQARU 2001, Toulouse, France, September 19-21, 2001, Proceedings*, volume 2143 of *Lecture Notes in Computer Science*, pages 748–759. Springer, 2001.

[24] Carlos Viegas Damásio and Luís Moniz Pereira. Sorted monotonic logic programs and their embeddings. In *Proceedings of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-04)*, pages 807–814, 2004.

[25] C.V. Damásio, J. Medina, and M. Ojeda-Aciego. Termination of logic programs with imperfect information: applications and query procedure. *Journal of Applied Logic*, 2006. To appear.

[26] C.V. Damásio and M. Medina, J. Ojeda-Aciego. A tabulation procedure for first-order residuated logic programs. In *Proceedings of the 11th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-06)*, 2006.

[27] Alex Dekhtyar and Michael I. Dekhtyar. Possible worlds semantics for probabilistic logic programs. In *20th International Conference on Logic Programming*, volume 3132 of *Lecture Notes in Computer Science*, pages 137–148. Springer Verlag, 2004.

[28] Alex Dekhtyar and Michael I. Dekhtyar. Revisiting the semantics of interval probabilistic logic programs. In *8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-05)*, number 3662 in Lecture Notes in Computer Science, pages 330–342. Springer Verlag, 2005.

[29] Alex Dekhtyar, Michael I. Dekhtyar, and V. S. Subrahmanian. Temporal probabilistic logic programs. In Danny De Schreye, editor, *Logic*

Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, pages 109–123, 1999.

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

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

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

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

[34] Marc Denecker, Victor W. Marek, and Mirosław Truszczyński. Uniform semantic treatment of default and autoepistemic logics. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning*, pages 74–84. Morgan Kaufman, 2000.

[35] Marc Denecker, Victor W. Marek, and Mirosław Truszczyński. Ultimate approximations. Technical Report CW 320, Katholieke Iniversiteit Leuven, September 2001.

[36] Marc Denecker, Victor W. Marek, and Mirosław Truszczyński. Ultimate approximations in nonmonotonic knowledge representation systems. In D. Fensel, F. Giunchiglia, D. McGuinness, and M. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 8th International Conference*, pages 177–188. Morgan Kaufmann, 2002.

[37] Guozhu Dong, Leonid Libkin, and Limsoon Wong. Incremental recomputation in local languages. *Inf. Comput.*, 181(2):88–98, 2003.

[38] Guozhu Dong and Jianwen Su. Space-bounded foies (extended abstract). In *PODS '95: Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 139–150, New York, NY, USA, 1995. ACM Press.

[39] Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Annals of Mathematics and Artificial Intelligence*, 14(2-4):187–223, 1995.

[40] Didier Dubois, Jérome Lang, and Henri Prade. Towards possibilistic logic programming. In *Proc. of the 8th Int. Conf. on Logic Programming (ICLP-91)*, pages 581–595. The MIT Press, 1991.

[41] Didier Dubois and Henri Prade. Possibility theory, probability theory and multiple-valued logics: A clarification. *Annals of Mathematics and Artificial Intelligence*, 32(1-4):35–66, 2001.

[42] Rafee Ebrahim. Fuzzy logic programming. *Fuzzy Sets and Systems*, 117(2):215–230, 2001.

[43] M. C. Fitting. The family of stable models. *Journal of Logic Programming*, 17:197–225, 1993.

[44] M. C. Fitting. Fixpoint semantics for logic programming - a survey. *Theoretical Computer Science*, 21(3):25–51, 2002.

[45] Melvin Fitting. A Kripke-Kleene-semantics for general logic programs. *Journal of Logic Programming*, 2:295–312, 1985.

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

[47] Melvin Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.

[48] Norbert Fuhr. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

[49] Matthew L. Ginsberg. Multi-valued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

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

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

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

[53] Petr Hájek. *Metamathematics of Fuzzy Logic*. Kluwer, 1998.

[54] C.J. Hinde. Fuzzy prolog. *International Journal Man.-Machine Stud.*, (24):569–595, 1986.

[55] Mitsuru Ishizuka and Naoki Kanai. Prolog-ELF: incorporating fuzzy logic. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 701–703, Los Angeles, CA, 1985.

[56] Kristian Kersting and Luc De Raedt. Bayesian logic programs. In James Cussens and Alan M. Frisch, editors, *ILP Work-in-progress reports, 10th International Conference on Inductive Logic Programming*, CEUR Workshop Proceedings. CEUR-WS.org, 2000.

[57] Kristian Kersting and Luc De Raedt. Bayesian logic programming: Theory and tools. In L. Getoor and B. Taskar, editors, *An Introduction to Statistical Relational Learning*. MIT Press, 2005.

[58] M.A. Khamsi and D. Misane. Disjunctive signed logic programs. *Fundamenta Informaticae*, 32:349–357, 1996.

[59] M.A. Khamsi and D. Misane. Fixed point theorems in logic programming. *Annals of Mathematics and Artificial Intelligence*, 21:231–243, 1997.

[60] M. Kifer and Ai Li. On the semantics of rule-based expert systems with uncertainty. In *Proc. of the Int. Conf. on Database Theory (ICDT-88)*, number 326 in Lecture Notes in Computer Science, pages 102–117. Springer-Verlag, 1988.

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

[62] Frank Klawonn and Rudolf Kruse. A Łukasiewicz logic based Prolog. *Mathware & Soft Computing*, 1(1):5–29, 1994.

[63] Stanislav Krajči, Rastislav Lencses, and Peter Vojtáš. A comparison of fuzzy and annotated logic programming. *Fuzzy Sets and Systems*, 144:173–192, 2004.

[64] Peter Kulmann and Sandra Sandri. An annotaded logic theorem prover for an extended possibilistic logic. *Fuzzy Sets and Systems*, 144:67–91, 2004.

[65] Laks Lakshmanan. An epistemic foundation for logic programming with uncertainty. In *Foundations of Software Technology and Theoretical Computer Science*, number 880 in Lecture Notes in Computer Science, pages 89–100. Springer-Verlag, 1994.

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

[67] Laks V.S. Lakshmanan and Fereidoon Sadri. Uncertain deductive databases: a hybrid approach. *Information Systems*, 22(8):483–508, 1997.

[68] Laks V.S. Lakshmanan and Nematollaah Shiri. Probabilistic deductive databases. In *Int'l Logic Programming Symposium*, pages 254–268, 1994.

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

[70] Leonid Libkin and Limsoon Wong. On the power of incremental evaluation in SQL-like languages. *Lecture Notes in Computer Science*, 1949:17–??, 2000.

[71] John W. Lloyd. *Foundations of Logic Programming*. Springer, Heidelberg, RG, 1987.

[72] Yann Loyer and Umberto Straccia. Uncertainty and partial non-uniform assumptions in parametric deductive databases. In *Proc. of the 8th European Conference on Logics in Artificial Intelligence (JELIA-02)*, number 2424 in Lecture Notes in Computer Science, pages 271–282, Cosenza, Italy, 2002. Springer-Verlag.

[73] Yann Loyer and Umberto Straccia. The well-founded semantics in normal logic programs with uncertainty. In *Proc. of the 6th International Symposium on Functional and Logic Programming (FLOPS-2002)*, number 2441 in Lecture Notes in Computer Science, pages 152–166, Aizu, Japan, 2002. Springer-Verlag.

[74] Yann Loyer and Umberto Straccia. The approximate well-founded semantics for logic programs with uncertainty. In *28th International Symposium on Mathematical Foundations of Computer Science (MFCS-2003)*, number 2747 in Lecture Notes in Computer Science, pages 541–550, Bratislava, Slovak Republic, 2003. Springer-Verlag.

[75] Yann Loyer and Umberto Straccia. Default knowledge in logic programs with uncertainty. In *Proc. of the 19th Int. Conf. on Logic Programming (ICLP-03)*, number 2916 in Lecture Notes in Computer Science, pages 466–480, Mumbai, India, 2003. Springer Verlag.

[76] Yann Loyer and Umberto Straccia. Epistemic foundation of the well-founded semantics over bilattices. In *29th International Symposium on Mathematical Foundations of Computer Science (MFCS-2004)*, number 3153 in Lecture Notes in Computer Science, pages 513–524, Bratislava, Slovak Republic, 2004. Springer Verlag.

[77] Yann Loyer and Umberto Straccia. Any-world assumptions in logic programming. *Theoretical Computer Science*, 342(2-3):351–381, 2005.

[78] Yann Loyer and Umberto Straccia. Epistemic foundation of stable model semantics. *Journal of Theory and Practice of Logic Programming*, 2006. To appear.

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

[80] James J. Lu, Jacques Calmet, and Joachim Schü. Computing multiple-valued logic programs. *Mathware % Soft Computing*, 2(4):129–153, 1997.

[81] Thomas Lukasiewicz. Many-valued first-order logics with probabilistic semantics. In *Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL'98)*, number 1584 in Lecture Notes in Computer Science, pages 415–429. Springer Verlag, 1998.

[82] Thomas Lukasiewicz. Probabilistic logic programming. In *Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98)*, pages 388–392, Brighton (England), August 1998.

[83] Thomas Lukasiewicz. Many-valued disjunctive logic programs with probabilistic semantics. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, number 1730 in Lecture Notes in Computer Science, pages 277–289. Springer Verlag, 1999.

[84] Thomas Lukasiewicz. Probabilistic and truth-functional many-valued logic programming. In *The IEEE International Symposium on Multiple-Valued Logic*, pages 236–241, 1999.

[85] Thomas Lukasiewicz. Fixpoint characterizations for many-valued disjunctive logic programs with probabilistic semantics. In *In Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, number 2173 in Lecture Notes in Artificial Intelligence, pages 336–350. Springer-Verlag, 2001.

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

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

[88] Zoran Majkic. Coalgebraic semantics for logic programs. In *18th Workshop on (Constraint) Logic Programming ((W(C)LP-05)*, Ulm, Germany, 2004.

[89] Zoran Majkic. Many-valued intuitionistic implication and inference closure in a bilattice-based logic. In *35th International Symposium on Multiple-Valued Logic (ISMVL-05)*, number 214-220, 2005.

[90] Zoran Majkic. Truth and knowledge fixpoint semantics for many-valued logic programming. In *19th Workshop on (Constraint) Logic Programming ((W(C)LP-05)*, number 76-87, Ulm, Germany, 2005.

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

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

[93] Trevor P. Martin. Soft computing, logic programming and the semantic web. In *Proceedings of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-04)*, pages 815–822, 2004.

[94] Cristinel Mateis. Extending disjunctive logic programming by t-norms. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-99)*, number 1730 in Lecture Notes in Computer Science, pages 290–304. Springer-Verlag, 1999.

[95] Cristinel Mateis. Quantitative disjunctive logic programming: Semantics and computation. *AI Communications*, 13:225–248, 2000.

[96] Jesús Medina and Manuel Ojeda-Aciego. Multi-adjoint logic programming. In *Proceedings of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-04)*, pages 823–830, 2004.

[97] Jesús Medina, Manuel Ojeda-Aciego, and Peter Vojtáš. Multi-adjoint logic programming with continuous semantics. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, volume 2173 of *Lecture Notes in Artificial Intelligence*, pages 351–364. Springer Verlag, 2001.

[98] Jesús Medina, Manuel Ojeda-Aciego, and Peter Vojtáš. A procedural semantics for multi-adjoint logic programming. In *Proceedings of the10th Portuguese Conference on Artificial Intelligence on Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving*, pages 290–297. Springer-Verlag, 2001.

[99] Jesús Medina, Manuel Ojeda-Aciego, and Peter Vojtáš. Similarity-based unification: a multi-adjoint approach. *Fuzzy sets and systems*, 1(146):43–62, 2004.

[100] Stephen Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, page 29. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[101] M. Mukaidono. Foundations of fuzzy logic programming. In *Advances in Fuzzy Systems – Application and Theory,*, volume 1. World Scientific, Singapore, 1996.

[102] M. Mukaidono, Z. Shen, and L. Ding. Fundamentals of fuzzy prolog. *Int. J. Approx. Reasoning*, 3(2):179–193, 1989.

[103] Raymond Ng and V.S. Subrahmanian. Stable model semantics for probabilistic deductive databases. In Zbigniew W. Ras and Maria Zemenkova, editors, *Proc. of the 6th Int. Sym. on Methodologies for Intelligent Systems (ISMIS-91)*, number 542 in Lecture Notes in Artificial Intelligence, pages 163–171. Springer-Verlag, 1991.

[104] Raymond Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1993.

[105] Raymond Ng and V.S. Subrahmanian. Stable model semantics for probabilistic deductive databases. *Information and Computation*, 110(1):42–83, 1994.

[106] Liem Ngo. Probabilistic disjunctive logic programming. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfeth Conference (UAI-1996)*, pages 397–404, San Francisco, CA, 1996. Morgan Kaufmann Publishers.

[107] Liem Ngo and Peter Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171(1-2):147–177, 1997.

[108] Pascal Nicolas, Laurent Garcia, and Igor Stéphan. Possibilistic stable models. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 248–253. Morgan Kaufmann Publishers, 2005.

[109] Leonard Paulik. Best possible answer is computable for fuzzy sld-resolution. In Petr Hajék, editor, *Gödel 96: Logical Foundations of Mathematics, Computer Science, and Physics*, number 6 in Lecture Notes in Logic, pages 257–266. Springer Verlag, 1996.

[110] David Poole. Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[111] Paul C. Rhodes and Sabah Merad Menani. Towards a fuzzy logic programming system: a clausal form fuzzy logic. *Knowledge-Based Systems*, 8(4):174–182, 1995.

[112] William C. Rounds and Guo-Qiang Zhang. Clausal logic and logic programming in algebraic domains. *Information and Computation*, 171:183–200, 2001.

[113] Michael Schroeder and Ralf Schweimeier. Fuzzy argumentation and extended logic programming. In *Proceedings of ECSQARU Workshop Adventures in Argumentation*, 2001.

[114] Michael Schroeder and Ralf Schweimeier. Arguments and misunderstandings: Fuzzy unification for negotiating agents. In *Proceedings of the ICLP workshop CLIMA02*. Elsevier, 2002.

[115] Michael Schroeder and Ralf Schweimeier. Fuzzy unification and argumentation for well-founded semantics. In *Proceedings of the Conference on Current Trends in Theory and Practice of Informatics (SOFSEM-04)*, number 2932 in Lecture Notes in Computer Science, pages 102–121. Springer Verlag, 2004.

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

[117] Ehud Y. Shapiro. Logic programs with uncertainties: A tool for implementing rule-based systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 529–532, 1983.

[118] Zuliang Shen, Liya Ding, and Masao Mukaidono. *Fuzzy Computing*, chapter A Theoretical Framework of Fuzzy Prolog Machine, pages 89–100. Elsevier Science Publishers B.V., 1988.

[119] Umberto Straccia. Query answering in normal logic programs under uncertainty. In *8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05)*, number 3571 in Lecture Notes in Computer Science, pages 687–700, Barcelona, Spain, 2005. Springer Verlag.

[120] Umberto Straccia. Uncertainty management in logic programming: Simple and effective top-down query answering. In Rajiv Khosla, Robert J. Howlett, and Lakhmi C. Jain, editors, *9th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems (KES-05), Part II*, number 3682 in Lecture Notes in Computer Science, pages 753–760, Melbourne, Australia, 2005. Springer Verlag.

[121] Umberto Straccia. Query answering under the any-world assumption for normal logic programs. In *Proceedings of the 10th International Conference on Knowledge Representation (KR-06)*, 2006.

[122] Hudson Turner. Signed logic programs. In Maurice Bruynooghe, editor, *Logic Programming: Proc. of the 1994 International Symposium*, pages 61–75. The MIT Press, 1994.

[123] J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1,2. Computer Science Press, Potomac, Maryland, 1989.

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

[125] Joost Vennekens, Sofie Verbaeten, and Maurice Bruynooghe. Logic programs with annotated disjunctions. In *20th International Conference on Logic Programming (ICLP-04)*, volume 3132 of *Lecture Notes in Computer Science*, pages 431–445. Springer Verlag, 2004.

[126] Peter Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124:361–370, 2004.

[127] Peter Vojtáš and Leonard Paulík. Soundness and completeness of non-classical extended SLD-resolution. In *5th International Workshop on Extensions of Logic Programming (ELP'96)*, number 1050 in Lecture Notes in Artificial Intelligence, pages 289–301, Leipzig, Germany, 1996.

[128] Peter Vojtáš and Marta Vomelelová. Transformation of deductive and inductive tasks between models of logic programming with imperfect information. In *Proceedings of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-04)*, pages 839–846, 2004.

[129] Gerd Wagner. Negation in fuzzy and possibilistic logic programs. In T. Martin and F. Arcelli, editors, *Logic programming and Soft Computing*. Research Studies Press, 1998.

[130] David S. Warren. Memoing for logic programs. *Commun. ACM*, 35(3):93–111, 1992.

[131] Beat Wüttrich. Probabilistic knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):691–698, 1995.

[132] H. Yasui, Y. Hamada, and M. Mukaidono. Fuzzy prolog based on lukasiewicz implication and bounded product. *IEEE Trans. Fuzzy Systems, 1995*, 2:949–954.

[133] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[134] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3–28, 1965.