Ann Math Artif Intell (2009) 55:389–417
DOI 10.1007/s10472-008-9099-0

Approximate well-founded semantics, query answering and generalized normal logic programs over lattices

Yann Loyer · Umberto Straccia

Published online: 25 October 2008

Abstract The management of imprecise information in logic programs becomes important whenever the real world information to be represented is of an imperfect nature and the classical crisp *true, false* approximation is not adequate. In this work, we consider normal logic programs over complete lattices, where computable truth combination functions may appear in the rule bodies to manipulate truth values and we will provide a top-down query answering procedure.

Keywords Logic programming · Nonmonotonic logic · Many-valued logic

Mathematics Subject Classifications (2000) 03B50 · 03B52 · 03B70 · 68T27 · 68T30

1 Introduction

The management of uncertainty and/or vagueness within deduction systems is an important issue whenever the real world information to be represented is of an imperfect nature. In logic programming, 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 vagueness theory (*Probability theory*, *Fuzzy set theory*, *Multi-valued logic*, *Possibilistic logic*) and how uncertainty/imprecision values, associated to rules and facts, are managed (see Section 5).

Y. Loyer
PRiSM (CNRS UMR 8144), Université de Versailles Saint Quentin,
45 Avenue des Etats-Unis, 78035 Versailles, France
e-mail: Yann.Loyer@prism.uvsq.fr

U. Straccia (✉)
I.S.T.I. C.N.R., Via G. Moruzzi, 1, 56124 Pisa, Italy
e-mail: straccia@iei.pi.cnr.it

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 "vagueness theory" fall all those approaches in which statements are true to some degree which is taken from a truth space (see [40] for a clarification between the notions of uncertainty and imprecision). In this work we deal with *vagueness* and, thus, statements have a degree of truth.

However, very few proposals provide, in a many-valued setting, both non-monotonic reasoning and a top-down query answering procedure (e.g. [126, 129]).

In this paper, we consider a general framework for normal logic programs with many-valued well-founded semantics, as described in [126, 129]. The truth-space is a complete lattice and 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 I of values to the B_i, the value of A is computed by stating that A is at least as true as $f(I(B_1), ..., I(B_n))$. The form of the rules is sufficiently expressive to encompass most approaches to many-valued normal logic programming.

We point out that [126, 129] provide a top-down query answering procedure in this logic. However they require the grounding of the logic program. Furthermore, queries are ground atoms only. This approach is clearly not satisfactory as the size of the grounded instance of a logic program as well as the number of query instances of a query may be large and generally exponential with respect to the size of the non-ground expressions. We instead provide here a query answering procedure, which avoids the grounding of the program. We present a simple, yet general tabulation-like top-down query answering procedure, which focuses on computing *all* answers of a query. A distinguishing feature of our query answering procedure is that we do not determine all answers by discovering *all proofs*, but rather apply a variant of so-called *memoing* techniques developed for classical logic programming—e.g. [144] 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 [8, 136, 144]. Hence, for instance, we do not rely on any notion of *atom unification*, but rather iteratively access relational tables using relational algebra.

In the remaining, we proceed as follows. In the following section, we give basic definitions about our formalism. In Section 3, we define the intended semantics of normal logic programs. In Section 4 we present a top-down query answering procedure, while Section 6 concludes and addresses future directions of work.

2 Preliminaries

A *truth lattice* is a complete lattice $\mathcal{L} = \langle L, \preceq \rangle$, with L a countable set of truth values, bottom $\bot$, top element $\top$, meet $\wedge$ and join $\vee$. We also assume that $\mathcal{L}$ has a *negation* $\neg$ that reverses the $\preceq$ ordering and verifies $\neg\neg x = x$. In $\mathcal{L} = \langle L, \preceq \rangle$, a function $g\colon L \to L$ is *monotone* if $\forall x, y \in L, x \preceq y$ implies $g(x) \preceq g(y)$. A *fixed-point* of g is an element $x \in L$ such that $g(x) = x$. The basic tool for studying fixed-points of functions on lattices is the well-known Knaster-Tarski theorem [134]. Let g be a

monotone function on a complete lattice $\langle L, \preceq \rangle$. Then g has a fixed-point, the set of fixed-points of f is a complete lattice and, thus, g has a *least* fixed-point. The *least* fixed-point of g can be obtained by iterating g over $\bot$, i.e. is the limit of the non-decreasing sequence $y_0, \ldots, y_i, y_{i+1}, \ldots, y_\lambda, \ldots$, where for a successor ordinal $i \geq 0$, $y_0 = \bot$, $y_{i+1} = g(y_i)$, while for a limit ordinal λ, $y_\lambda = lub\{y_i : i < \lambda\}$. We denote the least fixed-point by $\mathrm{lfp}(g)$. For ease of exposition, we will specify the initial condition y_0 and the next iteration step y_{i+1} only, while the condition on the limit is implicit.

Let $\mathcal{F}$ be a family of continuous n-ary functions $f : \mathcal{L}^n \to \mathcal{L}$. That is (for $n = 1$), for any monotone chain $x_0, x_1, \ldots$ of values in L, $f(\vee_i x_i) = \vee_i f(x_i)$. The n-ary case $n > 1$ is similar. We assume that the standard functions $\wedge$ (meet) and $\vee$ (join) belong to $\mathcal{F}$. Notably, $\wedge$ and $\vee$ are both continuous. We call $f \in \mathcal{F}$ a *truth combination function*, or simply *combination function*.

A *term*, denoted t, is either a variable or a constant symbol. An *atom*, denoted A, is an expression of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol and all t_is are terms. A literal, L, is of the form A or $\neg A$, where A is an atom. A *formula*, φ, is an expression built up from the atoms, the truth values $c \in L$ of the lattice and the functions $f \in \mathcal{F}$. The members of the lattice may appear in a formula, as well as functions $f \in \mathcal{F}$: e.g. in $\mathcal{L}_{[0,1]\cap\mathbb{Q}}$, the expression $\min(p, q) \cdot \max(\neg r, 0.7) + v$ is a formula φ, where p, q, r and v are atoms. The intuition here is that the truth value of the formula $\min(p, q) \cdot \max(\neg r, 0.7) + v$ is obtained by determining the truth value of p, q, r and v and then by applying the arithmetic functions, min, max, $1-$ and product $\cdot$ to determine the value of φ. Note that for ease of exposition, we will use the symbol min both at the syntactic level, writing $\min(p, q)$, as well as in its interpretation (e.g., $I(\min(p, q)) = \min(I(p), I(q))$, where I is an interpretation) with obvious meaning. A *rule* is of the form $A \leftarrow \varphi$, where A is an atom and φ is a formula. The atom A is called the *head*, and the formula φ is called the *body*. A *normal logic program*, denoted $\mathcal{P}$, is a finite set of rules. The *Herbrand universe* $H_\mathcal{P}$ of $\mathcal{P}$ is the set of constants appearing in $\mathcal{P}$. If there is no constant symbol in $\mathcal{P}$ then consider $H_\mathcal{P} = \{a\}$, where a is an arbitrary chosen constant. The *Herbrand base* $B_\mathcal{P}$ of $\mathcal{P}$ is the set of ground instantiations of atoms appearing in $\mathcal{P}$ (ground instantiations are obtained by replacing all variable symbols with constants of the Herbrand universe).

Given $\mathcal{P}$, the normal logic program $\mathcal{P}^*$ is constructed as follows:

1. set $\mathcal{P}^*$ to the set of all ground instantiations of rules in $\mathcal{P}$;
2. if an atom A is not the 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*);
3. 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 lattice in infix notation).

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

An *interpretation* I of a program $\mathcal{P}$ is a function that assigns to all atoms of the Herbrand base of $\mathcal{P}$ a value in $\mathcal{L}$. In logic programming, the intended model is usually the least model of $\mathcal{P}$ w.r.t. $\preceq$.[1] Unfortunately, the introduction of negation may have the consequence that some logic programs do not have a unique minimal model.

[1] $\preceq$ is extended to the set of interpretations as follows: $I \preceq J$ iff for all atoms A, $I(A) \preceq J(A)$.

Example 1 Consider the truth lattice $\mathcal{L}_{[0,1]}$ and the program $\mathcal{P}$

$$\begin{aligned} p &\leftarrow \max(\neg q, r) \\ q &\leftarrow \max(\neg p, s) \\ r &\leftarrow \max(0.3, \min(s, 0.6)) \\ s &\leftarrow s \end{aligned}$$

Informally, an interpretation I is a model of the program if it satisfies every rule, while I satisfies a rule $X \leftarrow Y$ if $I(X) \geq I(Y)$.[2] Thus, concerning the value of s in the above program, we only know that it has to be greater than itself. It follows that the value of s is 0 in any minimal model of $\mathcal{P}$. Concerning the value of r, it follows that the value of r is 0.3 in any minimal model of $\mathcal{P}$. Then, any model I of this program is such that $I(r) \preceq I(p)$, $I(s) \preceq I(q)$, $I(q) \geq 1 - I(p)$. Consequently, there are an infinite number of minimal models such that $I(q) = 1 - I(p)$ and $0.3 \preceq I(p)$.

Concerning the previous example we may note that the truth of p in the minimal models is in the interval [0.3, 1], while for q the interval is [0, 0.7].

The semantics we consider is to provide these intervals as an *approximation* to the truth of the atoms A and B. A well-known approach is to rely on $L \times L$ (see [42–44, 46]). Any element of $L \times L$ is denoted by $[a; b]$ and interpreted as an interval on L, i.e. $[a; b]$ *is interpreted as the set of elements* $x \in L$ *such that* $a \preceq x \preceq b$. For instance, turning back to Example 1 above, in the intended model of $\mathcal{P}$, the truth of p is "approximated" with [0.3; 1], i.e. the truth of p lies in between 0.3 and 1 (similarly for q).

Formally, given a complete lattice $\mathcal{L} = \langle L, \preceq \rangle$, we construct a so-called *bilattice* over $L \times L$, according to a well-known construction method (see [42, 48]). We recall that a bilattice is a triple $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$, where $\mathcal{B}$ is a nonempty set and $\preceq_t$, $\preceq_k$ are both partial orderings giving to $\mathcal{B}$ the structure of a lattice with a top and a bottom [48]. We consider $\mathcal{B} = \mathcal{L} \times \mathcal{L}$ with the following orderings:

1. the *truth ordering* $\preceq_t$, where $[a_1; b_1] \preceq_t [a_2; b_2]$ iff $a_1 \preceq a_2$ and $b_1 \preceq b_2$; and
2. the *knowledge ordering* $\preceq_k$, where $[a_1; b_1] \preceq_k [a_2; b_2]$ iff $a_1 \preceq a_2$ and $b_2 \preceq b_1$, i.e. $[a_1, b_1] \supseteq [a_2, b_2]$.

The intuition of those orders is that truth increases if the interval contains greater values (e.g. $[0.1; 0.4] \preceq_t [0.2; 0.5]$), whereas the knowledge increases when the interval (i.e. in our case the approximation of a truth value) becomes more precise (e.g. $[0.1; 0.4] \preceq_k [0.2; 0.3]$, i.e. we have more knowledge).

The least and greatest elements of $L \times L$ are respectively:

- $\mathtt{f} = [\bot; \bot]$ (false) and $\mathtt{t} = [\top; \top]$ (true), w.r.t. $\preceq_t$;
- $\bot = [\bot; \top]$ (unknown – the less precise interval, i.e. the atom's truth value is unknown) and $\top = [\top; \bot]$ (inconsistent – the empty interval) w.r.t. $\preceq_k$.

The meet $(\wedge, \otimes)$, join $(\vee, \oplus)$ and negation $(\neg)$ on $L \times L$ w.r.t. both orderings are defined by extending the meet, join and negation from L to $L \times L$ in the natural way: let $[a_1; b_1], [a_2; b_2] \in L \times L$, then

[2] Roughly, $X \leftarrow Y$ dictates that "X should be at least as true as Y.

Meet and join on $\preceq_t$: $[a_1; b_1] \wedge [a_2; b_2] = [a_1 \wedge a_2; b_1 \wedge b_2]$ and $[a_1; b_1] \vee [a_2; b_2] = [a_1 \vee a_2; b_1 \vee b_2]$;
Meet and join on $\preceq_k$: $[a_1; b_1] \otimes [a_2; b_2] = [a_1 \wedge a_2; b_1 \vee b_2]$ and $[a_1; b_1] \oplus [a_2; b_2] = [a_1 \vee a_2; b_1 \wedge b_2]$;
Negation: $\neg[a; b] = [\neg b; \neg a]$.

Example 2 For instance, taking $\mathcal{L}_{[0,1]}$,

- $[0.1; 0.4] \vee [0.2; 0.5] = [0.2; 0.5]$,
- $[0.1; 0.4] \wedge [0.2; 0.5] = [0.1; 0.4]$,
- $[0.1; 0.4] \oplus [0.2; 0.5] = [0.2; 0.4]$,
- $[0.1; 0.4] \otimes [0.2; 0.5] = [0.1; 0.5]$ and
- $\neg[0.1; 0.4] = [0.6; 0.9]$.

Finally, we extend the functions $f \in \mathcal{F}$ over L pointwise to $L \times L$: for $f \in \mathcal{F}$ and $[a_1; b_1], [a_2; b_2] \in L \times L$:

$$f([a_1; b_1], [a_2; b_2]) = [f(a_1, a_2); f(b_1, b_2)] .$$

It is easy to verify that these extended functions preserve the original properties of functions $f \in \mathcal{F}$. The following theorem can easily be shown.

Theorem 1 *Consider $L \times L$ with the orderings $\preceq_t$ and $\preceq_k$. Then, the combination functions $\wedge, \vee, \otimes, \oplus$ are continuous (and, thus, monotonic) w.r.t. $\preceq_t$ and $\preceq_k$; any negation function is monotonic w.r.t. $\preceq_k$; and if the negation function satisfies the De Morgan laws, i.e. $\forall a, b \in L. \neg(a \vee b) = \neg a \wedge \neg b$ then the negation function is continuous w.r.t. $\preceq_k$.*

We now define the notion of approximate interpretations.

Definition 1 (Approximate interpretation, $\mathcal{C}_P$) Let $\mathcal{P}$ be a program. An *approximate interpretation* of $\mathcal{P}$ is a total function I from the Herbrand base B_P to the set $L \times L$. The set of all the approximate interpretations of $\mathcal{P}$, is denoted $\mathcal{C}_P$.

Intuitively, assigning the logical value $[a; b]$ to an atom A means that the exact truth value of A lies in between a and b with respect to $\preceq$. Our goal will be to determine, for each atom of the Herbrand base of $\mathcal{P}$ the most precise interval that can be inferred.

We use $\mathtt{I}_{\mathtt{f}}$ and $\mathtt{I}_{\perp}$ to denote the bottom interpretations under $\preceq_t$ and $\preceq_k$ respectively (they map any atom into $\mathtt{f}$ and $\perp$, respectively).

First, we extend the two orderings on $L \times L$ to the set of approximate interpretations $\mathcal{C}_P$ in the usual way: let I_1 and I_2 be in $\mathcal{C}_P$, then

1. $I_1 \preceq_t I_2$ iff $I_1(A) \preceq_t I_2(A)$, for all ground atoms A; and
2. $I_1 \preceq_k I_2$ iff $I_1(A) \preceq_k I_2(A)$, for all ground atoms A.

Under these two orderings $\mathcal{C}_P$ becomes a complete bilattice. The meet and join operations over $L \times L$ for both orderings are extended to $\mathcal{C}_P$ in the usual way (e.g. for any atom A, $(I \oplus J)(A) = I(A) \oplus J(A)$). Negation is extended similarly, for any atom A, $\neg I(A) = I(\neg A)$, and approximate interpretations are extended to elements of L, for any $\alpha \in L$, $I(\alpha) = [\alpha; \alpha]$.

We now identify the models of a program.

Definition 2 (Models of a logic program) Let $\mathcal{P}$ be a program and let I be an approximate interpretation of $\mathcal{P}$. An interpretation I is a *model* of a logic program $\mathcal{P}$, denoted $I \models \mathcal{P}$, iff for the *unique* rule involving A, $A \leftarrow \varphi \in \mathcal{P}^*$, $I(A) = I(\varphi)$ holds.

Note that usually a model has to satisfy $I(\varphi) \preceq_t I(A)$ 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 e.g. [43]).

Third, models of a program are usually also characterized in term of fixed-points of an immediate consequence operator that is used to infer knowledge from the program.

Definition 3 ($T_{\mathcal{P}}$) Let $\mathcal{P}$ be any program. The *immediate consequence operator* $T_{\mathcal{P}}$ is a mapping from $\mathcal{C}_{\mathcal{P}}$ to $\mathcal{C}_{\mathcal{P}}$, defined as follows: for every interpretation I, for every ground atom A, for $A \leftarrow \varphi \in \mathcal{P}^*$

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

Theorem 2 *An interpretation I is a model of $\mathcal{P}$ iff I is a fixed-point of $T_{\mathcal{P}}$.*

Note that by definition of $\mathcal{P}^*$ it follows that if an atom A does not appear as the head of a rule, then $T_{\mathcal{P}}(I)(A) = \mathtt{f}$.

We have the following Theorem.

Theorem 3 *For any program $\mathcal{P}$, $T_{\mathcal{P}}$ is monotonic and, if the De Morgan laws hold, continuous w.r.t. $\preceq_k$.*

Note If we restrict our attention to Datalog with negation, then we have to deal with four values $[f; f]$, $[t; t]$, $[f; t]$ and $[t; f]$ that correspond to the truth values *false, true, unknown* and *inconsistent*, respectively. Then, our interval bilattice coincides with Belnap's logic [10], the notions of satisfaction and model coincide with the classical ones, and our operator $T_{\mathcal{P}}$ reduces to the usual immediate consequence operator Φ defined by Fitting [44].

3 Intended semantics of normal logic programs

We next identify the approximate Kripke-Kleene model and the well-founded model of LPs, by adapting [80] to our case.

Approximate Kripke-Kleene Model [44] The weakest semantics of a normal logic program is the least model of the program w.r.t. the knowledge ordering: the *approximate Kripke-Kleene model* of a logic program $\mathcal{P}$, denoted $KK_{\mathcal{P}}$, is the $\preceq_k$-least model of $\mathcal{P}$. By Theorem 3, that model always exists and coincides with the least fixed-point of $T_{\mathcal{P}}$ with respect to $\preceq_k$. For ease of presentation, we may represent an interpretation also as a set of expressions of the form $A : [x; y]$, where A is a ground atom, indicating that $I(A) = [x; y]$.

Example 3 The following sequence of interpretations I_0, I_1, I_2 shows how the approximate Kripke-Kleene model of Example 1 is computed as the iterated fixed-point of $T_{\mathcal{P}}$, starting from $I_0 = I_\perp$, the $\preceq_k$ minimal interpretation that maps any $A \in B_P$ to $[\perp; \top]$, and $I_{n+1} = T_{\mathcal{P}}(I_n)$ (note that $I_i \preceq_k I_{i+1}$):

$$I_0 = \{p\colon [0; 1], q\colon [0; 1], r\colon [0; 1], s\colon [0; 1]\},$$

$$I_1 = \{p\colon [0; 1], q\colon [0; 1], r\colon [0.3; 0.6], s\colon [0; 1]\},$$

$$I_2 = \{p\colon [0.3; 1], q\colon [0; 1], r\colon [0.3; 0.6], s\colon [0; 1]\},$$

$$I_3 = I_2$$

$$= KK_{\mathcal{P}}\ .$$

Note that $KK_{\mathcal{P}}$ is minimal w.r.t. $\preceq_k$ and contains only the knowledge provided by $\mathcal{P}$, the truth values of q and w lie between 0 and 1, i.e. are unknown, the truth value of p is greater than 0.3 and the truth value of r lies between 0.3 and 0.6.

As well known, the approximate Kripke-Kleene model is usually considered as too weak. In the following, we propose to consider the *Closed World Assumption* (CWA) [105] to complete our knowledge (the CWA assumes that all atoms whose value cannot be inferred from the program are false by default). As we will see in the next section, the CWA also allows us to make the truth interval of an atom more precise.

The CWA as a Source of falsehood [80] We recall here the notion of *support*, introduced in [80], which is equivalent to [42], of a program w.r.t. an interpretation. Given a program $\mathcal{P}$ and an interpretation I that represents our current knowledge, the support of $\mathcal{P}$ w.r.t. I, denoted $s_{\mathcal{P}}(I)$, determines in a principled way how much *false* knowledge, i.e. how much knowledge provided by the CWA, can "safely" be joined to I w.r.t. the program $\mathcal{P}$. Roughly speaking, a part of the CWA is an interpretation J such that $J \preceq_k \mathtt{I_f}$, where $\mathtt{I_f}$ maps any $A \in B_P$ to $[\perp; \perp]$, and we consider that such an interpretation can be safely added to I if $J \preceq_k T_{\mathcal{P}}(I \oplus J)$, i.e. if J does not contradict the knowledge represented by $\mathcal{P}$ and I. Intuitively, a part of the CWA represents an assumption on the falsehood of the atoms. That assumption should be used to increase our knowledge. To this end, it should be added (using $\oplus$) to our current knowledge I to provide more precise approximations of the truth values assigned to each atom. Of course, some care should be taken in order to avoid the introduction of inconsistent knowledge. Thus we propose to test if adding such an assumption to our knowledge is safe, i.e. if the activation of the rules through $T_{\mathcal{P}}$ on the interpretation obtained by adding J to I does not contradict the knowledge that we have assumed ($J \preceq_k T_{\mathcal{P}}(I \oplus J)$). This is formalized as follows.

Definition 4 (safe part) An interpretation J is a *safe part* of the CWA w.r.t. a program $\mathcal{P}$ and an interpretation I iff

1. J is a part of the CWA, i.e. $J \preceq_k \mathtt{I_f}$, and
2. J is safe w.r.t. $\mathcal{P}$ and I, i.e. $J \preceq_k T_{\mathcal{P}}(I \oplus J)$.

Of course, the CWA should be used to complete as much as possible our current knowledge. Thus, we are especially interested in the maximal, safe part of the CWA.

Definition 5 (support) The *support of a program $\mathcal{P}$ w.r.t. an interpretation I*, denoted $s_{\mathcal{P}}(I)$, is the maximal safe part of the CWA w.r.t. a program $\mathcal{P}$ and an interpretation I w.r.t. $\preceq_k$, i.e. it is the maximal interpretation J w.r.t. $\preceq_k$ such that $J \preceq_k \mathtt{I_f}$ and $J \preceq_k T_{\mathcal{P}}(I \oplus J)$.

It is easy to verify (see [80]) that

$$s_{\mathcal{P}}(I) = \bigoplus \{J \mid J \preceq_k \mathtt{I_f} \text{ and } J \preceq_k T_{\mathcal{P}}(I \oplus J)\} .$$

The following theorem provides an algorithm for computing the support.

Theorem 4 ([80]) *$s_{\mathcal{P}}(I)$ coincides with the iterated fixed-point of the function $F_{P,I}$ beginning the computation with $I_{\mathtt{f}}$, where*

$$F_{P,I}(J) = I_{\mathtt{f}} \otimes T_{\mathcal{P}}(I \oplus J) .$$

From Theorems 1 and 3, it can be shown that $F_{P,I}$ is monotone and, if the De Morgan laws hold, continuous w.r.t. $\preceq_k$. It follows that the iteration of the function $F_{P,I}$ starting from $I_{\mathtt{f}}$ decreases w.r.t. $\preceq_k$.

We will refer to $s_{\mathcal{P}}$ as the *closed world operator*.

Corollary 1 *Let $\mathcal{P}$ be a program. The closed world operator $s_{\mathcal{P}}$ is monotone and, if the De Morgan laws hold, continuous w.r.t. the knowledge order $\preceq_k$.*

Example 4 The following sequence of interpretations J_0, J_1, J_2 shows the computation of $s_{\mathcal{P}}(KK_{\mathcal{P}})$, i.e. the additional knowledge that can be considered using the CWA on the Kripke-Kleene semantics $KK_{\mathcal{P}}$ of Example 1 ($I = KK_{\mathcal{P}}$, $J_0 = I_{\mathtt{f}}$ and $J_{n+1} = F_{P,I}(J_n)$):

$$J_0 = \{p\colon [0;0], q\colon [0;0], r\colon [0;0], s\colon [0;0]\},$$

$$J_1 = \{p\colon [0;1], q\colon [0;0.7], r\colon [0;0.3], s\colon [0;0]\},$$

$$J_2 = J_1$$

$$= s_{\mathcal{P}}(KK_{\mathcal{P}})$$

$s_{\mathcal{P}}(KK_{\mathcal{P}})$ asserts that, according to the CWA and w.r.t. $\mathcal{P}$ and $KK_{\mathcal{P}}$, the truth of q and r should be respectively at most 0.7 and 0.3, while the truth of s should be exactly 0. Please, note how the support provides some more precise information about the atoms q, r and s with respect to the Kripke-Kleene semantics provided at the beginning of this section, but leaves p invariant.

Classical setting A well-known way for extracting falsehood using the CWA was defined in the classical setting through the notion of *unfounded set* [138]. We recall

that a set U of atoms is *unfounded* w.r.t. a Datalog program $\mathcal{P}$ and an interpretation I iff for all A in U,

- for $A \leftarrow \varphi \in \mathcal{P}^*$ (note that $\varphi = \varphi_1 \vee \ldots \vee \varphi_n$ and $\varphi_i = L_{i_1} \wedge \ldots \wedge L_{i_n}$), φ_i is false either w.r.t. I or w.r.t. $\neg.U$, for all $1 \leq i \leq n$.[3]

It is easy to prove that (see [80]), in the classical setting:

Theorem 5 ([80]) *Let $\mathcal{P}$ and I be a classical logic program and a classical interpretation, respectively. Let U be a subset of $B_\mathcal{P}$.*

1. *A set U is unfounded w.r.t. $\mathcal{P}$ and I iff $\neg.U$ is a safe part of the CWA w.r.t. $\mathcal{P}$ and I;*[4]
2. *A set U is the greatest unfounded w.r.t. $\mathcal{P}$ and I iff $\neg.U$ is the support of the CWA w.r.t. $\mathcal{P}$ and I, i.e. $s_\mathcal{P}(I) = \neg.U_\mathcal{P}(I)$.*

Approximate Well-Founded Model We have now two ways to infer information from a program $\mathcal{P}$ and an approximate interpretation I: using $T_\mathcal{P}$ and using $s_\mathcal{P}$. To maximize the knowledge derived from $\mathcal{P}$ and the CWA, we consider the family of models that already contain their own support. In that family of models, we are particularly interested in the least one w.r.t. $\preceq_k$.

Definition 6 (Model supported by the CWA) An interpretation I is a *model of a program $\mathcal{P}$ supported by the CWA* iff $I \models P$ and $s_\mathcal{P}(I) \preceq_k I$. The *approximate well-founded model* of a program $\mathcal{P}$, denoted $W_\mathcal{P}$, is the least model of $\mathcal{P}$ supported by the CWA w.r.t. $\preceq_k$, i.e. the $\preceq_k$-least model of $\mathcal{P}$ such that $I \models P$ and $s_\mathcal{P}(I) \preceq_k I$.

If we consider the definition of support in the classical setting, then supported models are classical models of classical logic programs such that $\neg.U_\mathcal{P}(I) \subseteq I$, i.e. the false atoms provided by the greatest unfounded set are already false in the interpretation I. That is, CWA does not further contribute improving I's knowledge about the program $\mathcal{P}$. It is interesting to note how the above definition is nothing else than a generalization from the classical setting to lattices of the notion of well-founded model. Indeed, in [69] it is shown that the well-founded model is the least model satisfying $\neg.U_\mathcal{P}(I) \subseteq I$.

Example 5 Consider the logic program $\mathcal{P}$ with the following rules.

$$\begin{aligned} q(x) &\leftarrow q(x) \vee \neg r(x) \\ p(x) &\leftarrow p(x) \\ r(a) &\leftarrow \mathtt{t} \\ r(b) &\leftarrow \mathtt{f} \end{aligned}$$

In Table 1 we report the approximate Kripke-Kleene and well-founded model of $\mathcal{P}$, marked by bullets.

[3] The interpretation $\neg.U$ is defined by: for all A, if $A \in U$ then $\neg.U(A) = \mathtt{f}$ else $\neg.U(A) = \mathtt{u}$.

[4] Note that this condition can be rewritten as $\neg.U \subseteq T_\mathcal{P}(I \cup \neg.U)$.

Table 1 Approximate Kripke-Kleene and well-founded model of $\mathcal{P}$

	I_i						$KK(\mathcal{P})$	$WF(\mathcal{P})$	$s_\mathcal{P}(I_i)$						$U_\mathcal{P}(I_i)$
	$q(a)$	$q(b)$	$r(a)$	$r(b)$	$p(a)$	$p(b)$			$q(a)$	$q(b)$	$r(a)$	$r(b)$	$p(a)$	$p(b)$	
I_1	$\bot$	t	t	f	$\bot$	$\bot$	•		f	$\bot$	$\bot$	f	f	f	$\{q(a), r(b), p(a), p(b)\}$
I_2	f	t	t	f	f	f		•	f	$\bot$	$\bot$	f	f	f	$\{q(a), r(b), p(a), p(b)\}$

Example 6 Consider $\mathcal{L}_{[0,1]}$ and the bilattice of intervals build from it. Consider the following logic program:

$$\begin{aligned}
\mathcal{P} = p(x) &\leftarrow p(x) \vee q(x)\\
q(x) &\leftarrow (\neg r(x) \wedge p(x)) \vee s(x)\\
r(x) &\leftarrow \neg q(x) \vee t(x)\\
s(a) &\leftarrow [0.3; 0.5]\\
t(a) &\leftarrow [0.2; 0.4]\\
s(b) &\leftarrow \mathtt{t}\\
e(b) &\leftarrow \mathtt{f}
\end{aligned}$$

Note that the approximated Kripke-Kleene model of $\mathcal{P}$ is such that

$$KK_{\mathcal{P}} \supseteq \{p(a)\colon [0.3; 1], q(a)\colon [0.3; 0.8], r(a)\colon [0.2; 0.7],\ p(b)\colon \mathtt{t}, q(b)\colon \mathtt{t}, r(b)\colon \mathtt{f}\},$$

while the approximated well-founded model is such that

$$W_{\mathcal{P}} \supseteq \{p(a)\colon [0.3; 0.5], q(a)\colon [0.3; 0.5], r(a)\colon [0.5; 0.7],\ p(b)\colon \mathtt{t}, b(b)\colon \mathtt{t}, r(b)\colon \mathtt{f}\}\ .$$

Notice that $KK_{\mathcal{P}} \preceq_k WF_{\mathcal{P}}$, as expected.

Now we provide a fixed-point characterization and, thus, a way of computation of the approximate well-founded semantics. It is based on an operator, called approximate well-founded operator, that combines the two operators that have been defined above.

Definition 7 ($AW_{\mathcal{P}}$) Let $\mathcal{P}$ be a program. The *approximate well-founded operator*, denoted $AW_{\mathcal{P}}$, takes in input an approximate interpretation $I \in \mathcal{C}_P$ and returns $AW_{\mathcal{P}}(I) \in \mathcal{C}_P$ defined by

$$AW_{\mathcal{P}}(I) = T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I))\ .$$

Note that for $A \leftarrow \varphi \in \mathcal{P}^*$,

$$(I \oplus s_{\mathcal{P}}(I))(\varphi) = I(\varphi) \oplus s_{\mathcal{P}}(I)(\varphi)$$

holds and, thus, we can rewrite the $AW_{\mathcal{P}}$ operator as

$$AW_{\mathcal{P}}(I) = T_{\mathcal{P}}(I) \oplus s_{\mathcal{P}}(I)\ . \tag{1}$$

Theorem 6 ([80]) *Let $\mathcal{P}$ be a program. An interpretation I is a fixed-point $AW_{\mathcal{P}}$ iff I is a model of $\mathcal{P}$ supported by the CWA.*

Using the properties of monotonicity and continuity of $T_{\mathcal{P}}$ and $s_{\mathcal{P}}$ w.r.t. the knowledge order $\preceq_k$ over $\mathcal{C}_P$, from the fact that $\mathcal{C}_P$ is a complete lattice w.r.t. $\preceq_k$, by the well-known Knaster-Tarski theorem [134], it follows that:

Theorem 7 *Let $\mathcal{P}$ be a program. The approximate well-founded operator $AW_{\mathcal{P}}$ is monotone and, if the De Morgan laws hold, continuous w.r.t. the knowledge order $\preceq_k$. Therefore, $AW_{\mathcal{P}}$ has a least fixed-point w.r.t. the knowledge order $\preceq_k$. Moreover that least fixed-point coincides with the approximate well-founded semantics $W_{\mathcal{P}}$ of $\mathcal{P}$.*

It is illustrative to recall, as in [80], the way our definition of approximate well-founded semantics generalizes the classical setting (using Eq. 1) to logic programs over lattices, where arbitrary, continuous truth combination functions are allowed to occur in the rule body.

I is the well-founded semantics of $\mathcal{P}$		
	Classical logic $\{\mathtt{f}, \bot, \mathtt{t}\}$	**Interval bilattices**
$\preceq_k$-least I s.t.	$I = W_{\mathcal{P}}(I) = T_{\mathcal{P}}(I) \cup \neg.U_{\mathcal{P}}(I)$	$\mathbf{I} = \mathbf{AW}_{\mathcal{P}}(\mathbf{I}) = \mathbf{T}_{\mathcal{P}}(\mathbf{I}) \oplus \mathbf{s}_{\mathcal{P}}(\mathbf{I})$
$\preceq_k$-least model I s.t.	$\neg.U_{\mathcal{P}}(I) \subseteq I$	$\mathbf{s}_{\mathcal{P}}(\mathbf{I}) \preceq_{\mathbf{k}} \mathbf{I}$

Hence, the support may be seen as the added-value to the approximate Kripke-Kleene semantics and evidences the role of CWA in the approximate well-founded semantics.

Example 7 The following sequence of interpretations shows the computation of $W_{\mathcal{P}}$ of Example 1 ($I_0 = I_\bot$ and $I_{n+1} = AW_{\mathcal{P}}(I_n)$).

$$
\begin{aligned}
I_0 &= \{p\colon [0;1], q\colon [0;1], r\colon [0;1], s\colon [0;1]\}\\
s_{\mathcal{P}}(I_0) &= \{p\colon [0;1], q\colon [0;1], r\colon [0;0.3], s\colon [0;0]\}\\
\\
I_1 &= \{p\colon [0;1], q\colon [0;1], r\colon [0.3;0.3], s\colon [0;0]\}\\
s_{\mathcal{P}}(I_1) &= \{p\colon [0;1], q\colon [0;1], r\colon [0;0.3], s\colon [0;0]\}\\
\\
I_2 &= \{p\colon [0.3;1], q\colon [0;1], r\colon [0.3;0.3], s\colon [0;0]\}\\
s_{\mathcal{P}}(I_2) &= \{p\colon [0;1], q\colon [0;0.7], r\colon [0;0.3], s\colon [0;0]\}\\
\\
\mathbf{I_3} &= \{\mathbf{p}\colon [\mathbf{0.3};\mathbf{1}], \mathbf{q}\colon [\mathbf{0};\mathbf{0.7}], \mathbf{r}\colon [\mathbf{0.3};\mathbf{0.3}], \mathbf{s}\colon [\mathbf{0};\mathbf{0}]\}\\
s_{\mathcal{P}}(I_3) &= \{p\colon [0;1], q\colon [0;0.7], r\colon [0;0.3], s\colon [0;0]\}\\
\\
I_4 &= I_3\\
&= W_{\mathcal{P}}
\end{aligned}
$$

The truth of r and s are respectively 0.3 and 0, while the truth of p and q can only be approximated respectively with $[0.3; 1]$ and $[0; 0.7]$. Note that, at each step i, the support $s_{\mathcal{P}}(I_i)$ provided by the CWA to $\mathcal{P}$ and I_i represents some knowledge

that can be used to complete I_i. Also note that $KK_{\mathcal{P}} \preceq_k W_{\mathcal{P}}$, i.e. the approximate well-founded model contains more knowledge than the approximate Kripke-Kleene model (see Example 3)

$$KK_{\mathcal{P}} = \{p\colon [0.3; 1], q\colon [0; 1], r\colon [0.3; 0.6], w\colon [0; 1]\} .$$

Note also that the only difference between these semantics comes from the use of the support as a supplementary way to infer knowledge in the computation of $W_{\mathcal{P}}$.

The approximate Kripke-Kleene model is completed with some default knowledge from the CWA, namely $s_{\mathcal{P}}(I_3) = s_{\mathcal{P}}(KK_{\mathcal{P}})$ (see below), to obtain the approximate well-founded model. Indeed, to stress that role of the support, and thus of the CWA, note that, in our example (see Example 4 for the computation of the support $s_{\mathcal{P}}(KK_{\mathcal{P}})$),

$$W_{\mathcal{P}} = KK_{\mathcal{P}} \oplus s_{\mathcal{P}}(KK_{\mathcal{P}}) ,$$

i.e. that the approximate well-founded model of $\mathcal{P}$ coincides with the Kripke-Kleene model of $\mathcal{P}$ completed with its support.

It is easily be verified that in case of logic programs without negation, no approximation arises related to the atom's truth.

Theorem 8 *If we restrict our attention to logic programs without negation, then for any program $\mathcal{P}$ the approximate well-founded semantics $W_{\mathcal{P}}$ assigns exact values (i.e. of the form $[c; c]$) to all atoms.*

4 Top-down query answering

A *query* is an atom $?Q$ (*query atom*) of the form $q(\mathbf{x})$, intended as a question about the truth degree of all the instances of Q in the intended model of $\mathcal{P}$. We also allow a query to be a *set* $\{?Q_1, \ldots, ?Q_n\}$ of query atoms. In that latter case we ask about the truth degree of all instances of the atoms Q_i in the intended model.

The procedure we devise in this paper is a generalization of the procedures presented in [126, 129]. We anticipate that the main reason why the procedures in [126, 129] are not suitable to be used for computing all answers to a query $?Q$, given $\mathcal{P}$, is that

- Straccia [126, 129] rely on $\mathcal{P}$'s grounded version $\mathcal{P}^*$, which may be rather huge (exponential with respect to $|\mathcal{P}|$, in general) in applications with many facts;
- Straccia [126, 129] answer ground queries only. Strictly speaking, [126, 129] can compute all answers of a query atom $q(\mathbf{x})$ by submitting as query the set of all ground instances $q(\mathbf{c})$. This is clearly not feasible if the Herbrand universe is large.

In the following, we make the following assumptions. We assume the lattice we will deal with is *finite*. From a practical point of view this is a limitation we can live with, especially taking into account that computers have finite resources, and thus, only a finite set of truth degrees can be represented. In particular, this includes also the usual case were we use the rational numbers in $[0, 1] \cap \mathbb{Q}$ under a given fixed precision p of numbers a computer can work with. This will guarantee the termination of

our procedures (otherwise the termination after a finite number of steps cannot be guaranteed always).

Furthermore, we assume that a logic program $\mathcal{P}$ is made out of an *extensional database* (EDB), $\mathcal{P}_E$, and an *intensional 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 interval in $L \times L$. 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 degrees. Without loss of generality, we assume that there cannot be two records $\langle c_1, \ldots, c_n, b_1 \rangle$ and $\langle c_1, \ldots, c_n, b_2 \rangle$ in tab_r with $b_1 \neq b_2$.

The intensional database is a set of rules for the form

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

in which the predicates occurring in the extensional database (called *extensional predicates*) do not occur in the head of rules of the intensional 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 *intensional predicate* symbol p occurs in the head of at most one rule in the intensional database. Due to the expressiveness of rule bodies, it is not difficult to see that, possibly defining an equality predicate $Eq(x, y)$, logic programs can be put into this form.

For an atom A of the form $p(\mathbf{x})$, an *answer* for p is a pair $\langle \theta, b \rangle$, where $\theta = \{\mathbf{x}/\mathbf{c}\}$ is a substitution of the variables $\mathbf{x}$ in $p(\mathbf{x})$ with the constants in $\mathbf{c}$ and $b \in L \times L$ is a truth interval. We say that the answer $\langle \theta, b \rangle$ is *correct* for p with respect to the intended model I of $\mathcal{P}$ iff $I(p(\mathbf{c})) = b$. That is, by substituting the variables in $\mathbf{x}$ using θ, the evaluation of the query in the intended model is b. An *answer set* for p is a set of answers for p. Of course, our goal is to determine the set of all correct answers for the query $?Q$. For a given n-ary predicate p and a set of answers Δ_p of p, for convenience we represent Δ_p as an $n+1$-ary table tab_{Δ_p}, containing the records $\langle c_1, \ldots, c_n, b \rangle$.

Given two answers $\delta_1 = \langle \theta, b_1 \rangle$ and $\delta_2 = \langle \theta, b_2 \rangle$ for the same atom P, we define $\delta_1 \preceq_k \delta_2$ ($\delta_1 \succeq_k \delta_2$) iff $b_1 \preceq_k b_2$ ($b_1 \succeq_k b_2$). We write $\delta_1 \prec_k \delta_2$ ($\delta_1 \succ_k \delta_2$) iff $b_1 \prec_k b_2$ ($b_1 \succ_k b_2$). If Δ_p^1 and Δ_p^2 are two sets of answers for p, we write $\Delta_p^1 \preceq_k \Delta_p^2$ ($\Delta_p^1 \succeq_k \Delta_p^2$) iff for all $\delta_1 \in \Delta_p^1$ there is $\delta_2 \in \Delta_p^2$ such that $\delta_1 \preceq_k \delta_2$ ($\delta_1 \succeq_k \delta_2$). We write $\Delta_p^1 \prec_k \Delta_p^2$ ($\Delta_p^1 \succ_k \Delta_p^2$) iff $\Delta_p^1 \preceq_k \Delta_p^2$ ($\Delta_p^1 \succeq_k \Delta_p^2$) and there is $\delta_2 \in \Delta_p^2$ such that for no $\delta_1 \in \Delta_p^1$, $\delta_2 \preceq_k \delta_1$ ($\delta_2 \succeq_k \delta_1$) holds.

We present now our top-down tabling like procedure tailored to compute all correct answer of a query $?Q$ in the intended model. The basic idea of our procedure is to try to collect, during the computation, all correct answers incrementally together. The procedure can be related to the so-called *memoing* techniques (*tabling/magic sets*) developed for classical logic programming –see e.g. [144] for an overview.

At first, consider a general rule of the form $p(\mathbf{x}) \leftarrow \varphi(\mathbf{x}, \mathbf{y})$. 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 $\langle \{\mathbf{x}/\mathbf{c}\}, b \rangle$ 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 such that for all n_i-ary predicates p_i,

$$I(p_i(\mathbf{c})) = \begin{cases} b\,, \text{ if } \langle \mathbf{c}, b \rangle \in tab_{\Delta_{p_i}} \\ \mathtt{f} \quad \text{if } p_i \text{ is an extensional predicate and } \langle \mathbf{c}, b \rangle \notin tab_{\Delta_{p_i}} \\ \bot \quad \text{otherwise}\,. \end{cases}$$

The intuition in the definition above is that to an atom $p_i(\mathbf{c})$ we assign the current truth value if this truth value is known. Otherwise, we assign to it the default truth, which is $\mathtt{f}$ (if p_i is an extensional predicate). Then

$$eval(p, \Delta_{p_1}, \ldots, \Delta_{p_k}) = \{\langle \{\mathbf{x}/\mathbf{c}\}, b \rangle \mid b = \bigvee_{\mathbf{c}'} I(\varphi(\mathbf{c}, \mathbf{c}')), b \neq \bot\}\,, \tag{3}$$

where $\mathbf{c}'$ is a tuple of constants occurring in $\bigcup_i \Delta_{p_i}$. We omit to report the tuple whose degree is $\bot$. The disjunction $\bigvee_{\mathbf{c}'}$ is required as the free variables $\mathbf{y}$ in $\varphi(\mathbf{x}, \mathbf{y})$ may be seen as existentially quantified.

Example 8 Consider $\mathcal{P} = \{p(x) \leftarrow q(x, y), q(a, b) \leftarrow \mathtt{f}, q(a, c) \leftarrow \mathtt{t}\}$. Assume $\Delta_q = \{\langle (a, b), \mathtt{f} \rangle, \langle (a, c), \mathtt{t} \rangle\}$. Then $eval(p, \Delta_q) = \{\langle a, \mathtt{t} \rangle\}$, which amounts to evaluate $q(a, b) \vee q(a, c)$.

We are not going to further investigate the implementation details of the $eval(p, \Delta_{p_1}, \ldots, \Delta_{p_k})$ procedure, though it has to be carefully written to minimize the number of table look-ups and relational algebraic operations such as joins. It can be obtained by means of a combination of SQL statements over the tables and the application of the truth combination 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, 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.

4.1 Query answering: approximate Kripke-Kleene semantics

We start showing how to compute all answers with respect to the Kripke-Kleene semantics, i.e. the $\preceq_k$-least fixed-point of $T_{\mathcal{P}}$. The procedure is detailed in Table 2 and is based on similar basic principles as [126, 129]. Assume, we are interested in determining all correct answers of $q(\mathbf{x})$ w.r.t. the Kripke-Kleene semantics. We call the procedure with $Answer(\mathcal{P}, Q)$. We start with putting the predicate symbols $q \in Q$ in the *active* list of predicate symbols $\mathtt{A}$. At each iteration step (step 2) 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 (steps 4 or 5). If the evaluation leads to a better answer set for p (step 6), 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 $\mathtt{A}$, i.e. all predicate symbols that might depend on p are put in the active set to be examined. At some point (even if cyclic definitions are present) the active list will become empty

Table 2 General top-down algorithm

Procedure *Answer*($\mathcal{P}$, Q)
Input: Logic program $\mathcal{P}$, set Q of query predicate symbols;
Output: Mapping $\mathtt{v}$ containing all correct answers of predicates in Q w.r.t. lfp($T_{\mathcal{P}}$)

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** $\mathtt{v}(p_i) := tab_{p_i}$
5. **if** (p_i intensional predicate) **then** $\Delta_{p_i} := eval(p_i, \mathtt{v}(p_{i_1}), ..., \mathtt{v}(p_{i_{k_i}}))$
6. **if** $\mathtt{v}(p_i) \prec_k \Delta_{p_i}$ **then** $\mathtt{v}(p_i) := \Delta_{p_i}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{p}(p_i) \cap \mathtt{dg})$
7. **if not** $\mathtt{exp}(p_i)$ **then** $\mathtt{exp}(p_i) = \mathtt{true}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{s}(p_i) \setminus \mathtt{in})$, $\mathtt{in} := \mathtt{in} \cup \mathtt{s}(p_i)$

endwhile

and we have actually found all correct answers of $q(\mathbf{x})$. The procedure in Table 2 uses some auxiliary functions and data structures:

- 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 *sons* of p_i;
- for predicate symbol p_i, $\mathtt{p}(p_i) = \{p_j : p_i \in \mathtt{s}(p_j)\}$, i.e. the *parents* 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. the sons $\mathtt{s}(p_i) = \{p_{i_1}, \ldots, p_{i_{k_i}}\}$ of p_i;
- the variable $\mathtt{dg}$ collects the predicate symbols that may influence the result of the query predicates;
- the array variable $\mathtt{exp}$ traces the rule bodies that have been "expanded" (the predicate symbols occurring in the rule body are put into the active list);
- the variable $\mathtt{in}$ keeps track of the predicate symbols that have been put into the active list so far due to an expansion (to avoid, to put the same predicate symbol multiple times in the active list due to rule body expansion).

Example 9 Consider Example 6. The computation of *Answer*($\mathcal{P}$, $\{p\}$) is shown in Table 3, which also reports Δ_{p_i} and $\mathtt{v}(p_i)$ at each iteration i. Each line is a sequence of steps in the 'while loop'. What is left unchanged is not reported. *Answer*($\mathcal{P}$, $\{p\}$) returns $\mathtt{v}(p) = \{\langle a, [0.3; 1]\rangle, \langle b, [1; 1]\rangle\}$, as expected.

From a computational point of view, the analysis is as in [126, 127, 129]. Given $\mathcal{L} = \langle L \times L, \preceq_k\rangle$, let $h(\mathcal{L})$ be the *height* of the truth-value set L, i.e. the length of the longest strictly $\preceq_k$-increasing chain in $L \times L$ 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. As made clear before, the lattice is always finite and, thus, the height is finite as well. Now, observe that the truth of any ground instance of predicate symbol p_i is increasing in the $\preceq_k$ order as p_i enters in the $\mathtt{A}$ list (step 6), except it enters due to step 7, which may happen one time only. Hence, as for [126, 127, 129], we have that

Proposition 1 *Let $\mathcal{L}$, $\mathcal{P}$ and $?Q$ be a finite truth space, a logic program and a query, respectively. If the computing cost of each truth combination function is bounded by c and the height is bounded by h, then the worst-case complexity of the algorithm Answer($\mathcal{P}$, Q) is $O(|\mathcal{P}^*|ch)$.*

Table 3 Execution related to Example 9

1. $\mathtt{A} := \{p\}, p_i := p, \mathtt{A} := \emptyset, \mathtt{dg} := \{p, q\}, \Delta_p := \emptyset$
   $\mathtt{exp}(p) := 1, \mathtt{A} := \{p, q\}, \mathtt{in} := \{p, q\}$
2. $p_i := p, \mathtt{A} := \{q\}, \Delta_p := \emptyset$
3. $p_i := q, \mathtt{A} := \emptyset, \mathtt{dg} := \{p, q, r, s\}, \Delta_q := \emptyset$
   $\mathtt{exp}(q) := 1, \mathtt{A} := \{r, s\}, \mathtt{in} := \{p, q, r, s\}$
4. $p_i := r, \mathtt{A} := \{s\}, \mathtt{dg} := \{p, q, r, s, t\}, \Delta_r := \emptyset$
   $\mathtt{exp}(r) := 1, \mathtt{A} := \{s, t\}, \mathtt{in} := \{p, q, r, s, t\}$
5. $p_i := s, \mathtt{A} := \{t\}, \mathtt{v}(s) \prec_k \Delta_s, \mathtt{v}(s) := \Delta_s, \mathtt{A} := \{t, q\}, \mathtt{exp}(s) := 1$
6. $p_i := t, \mathtt{A} := \{q\}, \mathtt{v}(t) \prec_k \Delta_t, \mathtt{v}(t) := \Delta_t, \mathtt{A} := \{q, r\}, \mathtt{exp}(t) := 1$
7. $p_i := q, \mathtt{A} := \{r\}, \mathtt{v}(q) \prec_k \Delta_q, \mathtt{v}(q) := \Delta_q, \mathtt{A} := \{r, p\}$
8. $p_i := r, \mathtt{A} := \{p\}, \mathtt{v}(r) \prec_k \Delta_r, \mathtt{v}(r) := \Delta_r, \mathtt{A} := \{p, q\}$
9. $p_i := p, \mathtt{A} := \{q\}, \mathtt{v}(p) \prec_k \Delta_p, \mathtt{v}(p) := \Delta_p, \mathtt{A} := \{q\}$
10. $p_i := q, \mathtt{A} := \emptyset, \mathtt{v}(q) \prec_k \Delta_q, \mathtt{v}(q) := \Delta_q, \mathtt{A} := \{p, r\}$
11. $p_i := p, \mathtt{A} := \{r\}, \mathtt{v}(p) \not\prec_k \Delta_p$
12. $p_i := r, \mathtt{A} := \emptyset, \mathtt{v}(r) \not\prec_k \Delta_r$
13. $\mathtt{stop}.\ \mathtt{return}\ \mathtt{v}(p)$

Iter i	Δ_{p_i}	$\mathtt{v}(p_i)$
0.	–	$\mathtt{v}(p_i) = \emptyset$
1.	$\Delta_p = \emptyset$	–
2.	$\Delta_p = \emptyset$	–
3.	$\Delta_q = \emptyset$	–
4.	$\Delta_r = \emptyset$	–
5.	$\Delta_s = \{\langle a, [0.3; 0.5]\rangle, \langle b, \mathtt{t}\rangle\}$	$\mathtt{v}(s) = \Delta_s$
6.	$\Delta_t = \{\langle a, [0.2; 0.4]\rangle, \langle b, \mathtt{f}\rangle\}$	$\mathtt{v}(t) = \Delta_t$
7.	$\Delta_q = \{\langle a, [0.3; 1]\rangle, \langle b, [1; 1]\rangle\}$	$\mathtt{v}(q) = \Delta_q$
8.	$\Delta_r = \{\langle a, [0.2; 0.7]\rangle, \langle b, [0; 0]\rangle\}$	$\mathtt{v}(r) = \Delta_r$
9.	$\Delta_p = \{\langle a, [0.3; 1]\rangle, \langle b, [1; 1]\rangle\}$	$\mathtt{v}(p) = \Delta_p$
10.	$\Delta_q = \{\langle a, [0.3; 0.8]\rangle, \langle b, [1; 1]\rangle\}$	$\mathtt{v}(q) = \Delta_q$
11.	$\Delta_p = \{\langle a, [0.3; 1]\rangle, \langle b, [1; 1]\rangle\}$	–
12.	$\Delta_r = \{\langle a, [0.2; 0.7]\rangle, \langle b, [0; 0]\rangle\}$	–

We conclude by showing that the procedure *Answer* behaves correctly.

Proposition 2 *Let $\mathcal{L}$, $\mathcal{P}$ and $?Q$ be a finite truth space, a logic program and a query, respectively. Then Answer$(\mathcal{P}, Q)$ returns the set of all correct answers of $\mathcal{P}$ with respect to the predicates in Q and the Kripke-Kleene semantics.*

Proof For each n-ary predicate p consider the unique rule $p(\mathbf{x}) \leftarrow \varphi(\mathbf{x}, \mathbf{y})$ in $\mathcal{P}$. Therefore, for each tuple of constants $\mathbf{c}$, there is an unique rule

$$p(\mathbf{c}) \leftarrow \bigvee_{\mathbf{c}'} \varphi(\mathbf{c}, \mathbf{c}') \tag{4}$$

in $\mathcal{P}^*$. Now, let I be the interpretation such that for all predicates p,

$$I(p(\mathbf{c})) = \begin{cases} v & \text{if } \langle\{\mathbf{x}, \mathbf{c}\}, v\rangle \in \mathtt{v}(p) \\ KK_{\mathcal{P}}(p(\mathbf{c})) & \text{otherwise.} \end{cases}$$

Consider $p \in \mathtt{dg} \supseteq Q$ and $\delta = \langle\{\mathbf{x}/\mathbf{c}\}, v\rangle \in \mathtt{v}(p)$. As the truth of $p(\mathbf{c})$ is increasing in the $\preceq_k$ order during the computation, by definition of the *eval* function (3), after stopping the truth v of the head $p(\mathbf{c})$ evaluates exactly to the truth of the body of rule (4), i.e. $I(p(\mathbf{c})) = \bigvee_{\mathbf{c}'} I(\varphi(\mathbf{c}, \mathbf{c}'))$ and, hence, I is a model of $\mathcal{P}$. Also, it is easy to show by induction on the number of iterations of step 2, that at each iteration i of step 2, for $\langle\{\mathbf{x}, \mathbf{c}\}, v_i\rangle \in \mathtt{v}(p)$, where v_i is the truth degree of $p(\mathbf{c})$ computed at the ith iteration so far, and $p \in \mathtt{dg}$, we always have that $v_i \preceq_k KK_{\mathcal{P}}(p(\mathbf{c}))$. As a consequence, as $KK_{\mathcal{P}}$ is $\preceq_k$ minimal, for each $p \in \mathtt{dg} \supseteq Q$, for each $\delta = \langle\{\mathbf{x}/\mathbf{c}\}, v\rangle \in \mathtt{v}(p)$, $I(p(\mathbf{c})) = KK_{\mathcal{P}}(p(\mathbf{c}))$ has to hold. Therefore, all computed answers of $p \in \mathtt{dg} \supseteq Q$ are correct answers.

Vice-versa, suppose there is $p \in Q$ such that $\langle \{\mathbf{x}/\mathbf{c}\}, v\rangle \notin \mathtt{v}(p)$, while $KK_{\mathcal{P}}(p(\mathbf{c})) \neq \bot$. Then by construction of the *eval* function and definition of I, we can show that $\bot \neq KK_{\mathcal{P}}(p(\mathbf{c})) = I(p(\mathbf{c})) = \bigvee_{\mathbf{c}'} I(\varphi(\mathbf{c}, \mathbf{c}')) = \bot$, a contradiction. Hence, all correct answers with non-$\bot$ value are retrieved.

4.2 Query answering: approximate well-founded semantics

As we have seen in Section 3, the approximate well-founded semantics of a logic program $\mathcal{P}$ is the $\preceq_k$-least fixed-point of the operator

$$AW_{\mathcal{P}}(I) = T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I)) .$$

By Theorem 4, $s_{\mathcal{P}}(I)$ coincides with the iterated fixed-point of the function $F_{P,I}$ beginning the computation with $I_{\mathtt{f}}$, where

$$F_{P,I}(J) = I_{\mathtt{f}} \otimes T_{\mathcal{P}}(I \oplus J) .$$

That is, $s_{\mathcal{P}}(I)$ coincides with the limit of the $\preceq_k$ decreasing sequence

$$\begin{aligned} J_0 &= \mathtt{I}_{\mathtt{f}} , \\ J_{i+1} &= F_{P,I}(J_i) = I_{\mathtt{f}} \otimes T_{\mathcal{P}}(I \oplus J_i) . \end{aligned}$$

As we already have a top-down query answering procedure related to $T_{\mathcal{P}}$, it suffices to determine an analogue related to the support. In the following, we show how we can slightly change the *Answer* procedure to compute the support. That is, we want a top-down procedure that, for a set of atoms $p(\mathbf{x})$, computes all answers $\langle \{\mathbf{x}/\mathbf{c}\}, b\rangle$ such that $s_{\mathcal{P}}(p(\mathbf{c})) = b$.

So, let *Support*$(\mathcal{P}, Q, I)$ be the procedure, which is as the *Answer* procedure except that:

- Step 1 is replaced with
  $\mathcal{P} := \mathcal{P}_I, \mathtt{A} := Q, \mathtt{dg} := Q, \mathtt{in} := \emptyset,$
  for all predicate symbols p in $\mathcal{P}$ **do** $\mathtt{v}(p) = \emptyset, \mathtt{exp}(p) = \mathtt{false}$
  The logic program $\mathcal{P}_I$ is obtained from $\mathcal{P}$ in the following way:
  - for each intensional predicate p in $\mathcal{P}$, replace the rule $p(\mathbf{x}) \leftarrow \varphi(\mathbf{x}, \mathbf{y})$ in $\mathcal{P}$ with the rule

    $$p(\mathbf{x}) \leftarrow \mathtt{f} \otimes (I(p_\varphi)(\mathbf{x}) \oplus \varphi(\mathbf{x}, \mathbf{y})) , \tag{5}$$

    where $I(p_\varphi)(\mathbf{x})$ is a built-in predicate that given a substitution $\mathbf{c}$ for $\mathbf{x}$, returns $\bigvee_{\mathbf{c}'} I(\varphi(\mathbf{c}, \mathbf{c}'))$.
  - for each extensional predicate r in $\mathcal{P}$, replace the rule $r(\mathbf{c}) \leftarrow b$ in $\mathcal{P}$ with the rule

    $$r(\mathbf{c}) \leftarrow \mathtt{f} \otimes b . \tag{6}$$

We point out that the rules above are the result of applying $F_{\mathcal{P},I}$ to the support $s_{\mathcal{P}}(I)$ and to all rules:

$$\begin{aligned} s_{\mathcal{P}}(I)(p(\mathbf{c})) &= [\mathtt{I_f} \otimes T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I))](p(\mathbf{c})) \\ &= \mathtt{f} \otimes [I \oplus s_{\mathcal{P}}(I)](\bigvee_{\mathbf{c}'} \varphi(\mathbf{c}, \mathbf{c}')) \\ &= \mathtt{f} \otimes (I(\bigvee_{\mathbf{c}'} \varphi(\mathbf{c}, \mathbf{c}')) \oplus s_{\mathcal{P}}(I)(\bigvee_{\mathbf{c}'} \varphi(\mathbf{c}, \mathbf{c}'))) \\ &= \mathtt{f} \otimes (\bigvee_{\mathbf{c}'} I(\varphi(\mathbf{c}, \mathbf{c}')) \oplus \bigvee_{\mathbf{c}'} s_{\mathcal{P}}(I)(\varphi(\mathbf{c}, \mathbf{c}'))) \ . \end{aligned}$$

Since the above equation holds for all predicates p and all $\mathbf{c}$, we get rule (5) and (6). Build-in predicates do not count as sons and, thus, do not appear in the A, s, p, v, in, dg variables.

- Step 6 is replaced with

  if $\mathtt{v}(p_i) \succ_k \Delta_{p_i}$ **then** $\mathtt{v}(p_i) := \Delta_{p_i}, \mathtt{A} := \mathtt{A} \cup (\mathtt{p}(p_i) \cap \mathtt{dg})$

  Essentially, in Step 6 we replace $\prec_k$ with $\succ_k$. This modification is motivated by the fact that during the computation of the support, Δ_{p_i} is now $\preceq_k$ decreasing in accordance with Theorem 4.

Example 10 Consider Example 5 and interpretation I_2. We have seen that I_2 is the approximate well-founded model of $\mathcal{P}$. We next want to show the computation of *Support*$(\mathcal{P}, \{q, r\}, I_2)$. We first determine $\mathcal{P}_{I_2}$. As predicate p does not play any role in the computation, we report the modified rule for predicate q and r only. $\mathcal{P}_{I_2}$ related to q and r is

$$\begin{aligned} &q(x) \leftarrow \mathtt{f} \otimes (I_2(q_\varphi)(x) \oplus (q(x) \vee \neg r(x))) \\ &r(a) \leftarrow \bot \\ &r(b) \leftarrow \mathtt{f} \ . \end{aligned}$$

We recall that $I_2(q_\varphi)(a) = I_2(q(a) \vee \neg r(a)) = \mathtt{f}$, while $I_2(q_\varphi)(b) = \mathtt{t}$. Then, it can be verified that (by a straightforward fixed-point computation iterating $F_{\mathcal{P},I}$ starting with $\mathtt{I_f}$) that the set of correct answers of predicate q, r of $\mathcal{P}$ w.r.t. $s_{\mathcal{P}}(I_2)$ are: $\Delta_q = \{\langle a, \mathtt{f}\rangle\}$, $\Delta_r = \{\langle b, \mathtt{f}\rangle\}$. Below is a sequence of *Support*$(\mathcal{P}, \{q, r\}, I_2)$, returning the expected values.

1. $\mathtt{A} := \{q, r\}, p_i := q, \mathtt{A} := \{r\}, \mathtt{dg} := \{q, r\}, \Delta_q \succ_k \mathtt{v}(q)$, $\mathtt{exp}(q) := \mathtt{1}, \mathtt{A} := \{r, q\}, \mathtt{in} := \{q, r\}$
2. $p_i := r, \mathtt{A} := \{q\}, \mathtt{v}(r) \succ_k \Delta_r, \mathtt{v}(r) := \Delta_r, \mathtt{exp}(r) := \mathtt{1}$
3. $p_i := q, \mathtt{A} := \emptyset, \Delta_q = \mathtt{v}(q)$
4. stop. return $\mathtt{v}(q)$

Iter i	Δ_{p_i}	$\mathtt{v}(p_i)$
0.	–	$\mathtt{v}(p_i) = \emptyset$
1.	$\Delta_q = \{\langle a, \mathtt{f}\rangle\}$	$\mathtt{v}(q) = \Delta_q$
2.	$\Delta_r = \{\langle b, \mathtt{f}\rangle\}$	$\mathtt{v}(r) = \Delta_r$
3.	$\Delta_q = \{\langle a, \mathtt{f}\rangle\}$	–

From a computational point of view, as for [129], *Support*$(\mathcal{P}, Q, I)$ has the same complexity as *Answer*$(\mathcal{P}, Q)$, i.e. $O(|\mathcal{P}^*|hc)$. Similarly to Proposition 2, it can be shown that

Proposition 3 *Support*$(\mathcal{P}, Q, I)$ *returns the set of all correct answers of* $\mathcal{P}$ *with respect to the predicates in* Q *and the support* $s_{\mathcal{P}}(I)$ *after a finite number of steps.*

We are now ready to define the top-down procedure $Answer_{WF}(\mathcal{P}, Q)$, which computes all correct answers to a query $?Q$ under the approximate well-founded semantics. We define $Answer_{WF}(\mathcal{P}, Q)$ as $Answer(\mathcal{P}, Q)$, except that Step 5 is replaced with the statements

5. **if** (p_i intensional predicate) **then**
5.1. $\mathtt{Q}' := \mathtt{s}(p_i)$;
5.2. $\mathtt{I} := \mathtt{v}$;
5.3. $\mathtt{supp} := Support(\mathcal{P}, \mathtt{Q}', \mathtt{I})$;
5.4. $\mathtt{v}' := \mathtt{I} \oplus \mathtt{supp}$;
5.5. $\Delta_{p_i} := eval(p_i, \mathtt{v}'(p_{i_1}), ..., \mathtt{v}'(p_{i_{k_i}}))$

These steps correspond to the application of the $AW_{\mathcal{P}}(I) = T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I))$ operator to p_i. Indeed, at first we ask about all the correct answers of the predicates occurring in the body of p_i w.r.t. the support and the current interpretation $\mathtt{I} := \mathtt{v}$ (Steps 5.1 – 5.3). The variable supp holds these answers. Then we join them with I, i.e. we compute $I \oplus s_{\mathcal{P}}(I)$ (Step 5.4), where this latter is defined pointwise: (*i*) $\mathtt{v}' = \mathtt{v}_1 \oplus \mathtt{v}_2$ iff for all p, $\mathtt{v}'(p) = \mathtt{v}_1(p) \oplus \mathtt{v}_2(p) = \{\langle \theta, b\rangle \mid \langle \theta, b_1\rangle \in \mathtt{v}_1(p), \langle \theta, b_2\rangle \in \mathtt{v}_2(p), b = b_1 \oplus b_2\}$ (if $\langle \theta, b_i\rangle \notin \mathtt{v}_i(p)$ then $b_i = \bot$ is assumed). Finally, we evaluate the body of p_i with respect to $I \oplus s_{\mathcal{P}}(I)$ (Step 5.5), i.e. we apply $T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I))$.

Example 11 Consider Example 5. Let us compute all correct answers to the query $q(x)$ w.r.t. the approximate well-founded semantics. As the interpretation I_2 in Example 5 is the well-founded model, we expect to retrieve $\Delta_q = \{\langle a, \mathtt{f}\rangle, \langle b, \mathtt{t}\rangle\}$. Below is the computation of $Answer_{WF}(\mathcal{P}, \{q\})$.

1. $\mathtt{A} := \{q\}, p_i := q, \mathtt{A} := \emptyset, \mathtt{dg} := \{q, r\}, \mathtt{supp} := \{\langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle r(b), \mathtt{f}\rangle\}$, $\mathtt{v}(q) \prec_k \Delta_q, \mathtt{exp}(q) := 1, \mathtt{A} := \{q, r\}, \mathtt{in} := \{q, r\}$
2. $p_i := q, \mathtt{A} := \{r\}, \mathtt{supp} := \{\langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle q(b), \mathtt{t}\rangle, \langle r(b), \mathtt{f}\rangle\}, \Delta_q = \mathtt{v}(q)$
3. $p_i := r, \mathtt{A} := \emptyset, \mathtt{supp} := \{\langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle q(b), \mathtt{t}\rangle, \langle r(b), \mathtt{f}\rangle\}$, $\mathtt{v}(r) \prec_k \Delta_r, \mathtt{v}(r) := \Delta_r, \mathtt{A} := \{q\}, \mathtt{exp}(r) := 1$
4. $p_i := q, \mathtt{A} := \emptyset, \mathtt{supp} := \{\langle q(a), \mathtt{f}\rangle, \langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle q(a), \mathtt{f}\rangle, \langle q(b), \mathtt{t}\rangle, \langle r(a), \mathtt{t}\rangle, \langle r(b), \mathtt{f}\rangle\}$, $\mathtt{v}(q) \prec_k \Delta_q, \mathtt{v}(q) := \Delta_q, \mathtt{A} := \{q\}$
5. $p_i := q, \mathtt{A} := \emptyset, \mathtt{supp} := \{\langle q(a), \mathtt{f}\rangle, \langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle q(a), \mathtt{f}\rangle, \langle q(b), \mathtt{t}\rangle, \langle r(a), \mathtt{t}\rangle, \langle r(b), \mathtt{f}\rangle\}$, $\Delta_q = \mathtt{v}(q)$
6. stop. return $\mathtt{v}(q)$

Iter i	Δ_{p_i}	$\mathtt{v}(p_i)$
0.	–	$\mathtt{v}(p_i) = \emptyset$
1.	$\Delta_q = \{\langle b, \mathtt{t}\rangle\}$	$\mathtt{v}(q) = \Delta_q$
2.	$\Delta_q = \{\langle b, \mathtt{t}\rangle\}$	–
3.	$\Delta_r = \{\langle a, \mathtt{t}\rangle, \langle b, \mathtt{f}\rangle\}$	$\mathtt{v}(r) = \Delta_r$
4.	$\Delta_q = \{\langle a, \mathtt{f}\rangle, \langle b, \mathtt{t}\rangle\}$	$\mathtt{v}(q) = \Delta_q$
5.	$\Delta_q = \{\langle a, \mathtt{f}\rangle, \langle b, \mathtt{t}\rangle\}$	–

Therefore, $Answer_{WF}(\mathcal{P}, \{q\})$ returns $\mathtt{v}(q) = \{\langle a, \mathtt{f}\rangle, \langle b, \mathtt{t}\rangle\}$, as expected.

From a computational point of view, as for [129], as now for each iteration we have a call to the support, $Answer_{WF}(\mathcal{P}, Q)$ runs in time $O(|\mathcal{P}^*|^2 h^2 c)$. Furthermore, by Proposition 3, and similarly to Proposition 2, it is not difficult to show that:

Proposition 4 *After a finite number of steps, $Answer_{WF}(\mathcal{P}, Q)$ returns the set of all correct answers of $\mathcal{P}$ with respect to the predicates in Q and the approximate well-founded semantics.*

5 Related work

In logic programming, the management of imperfect information has attracted the attention of many researchers and numerous frameworks have been proposed. Addressing all of them is almost impossible, due to both the large number of works published in this field (early works date back to early 80-ties [124]) and the different approaches proposed. Essentially they differ in the underlying notion of uncertainty theory and vagueness theory (probability theory, possibilistic logic, fuzzy logic and multi-valued logic) and how uncertainty/vagueness values, associated to rules and facts, are managed.

Below a list of references and the underlying imprecision and uncertainty theory in logic programming frameworks. The list of references is by no means intended to be all-inclusive.[5]

Probability theory: [4, 5, 9, 16, 27–31, 47, 54–56, 65, 67, 83–85]
[86–89, 98, 106, 110–113, 116, 117, 139, 145]
Possibilistic logic: [1–3, 14, 39, 114]
Fuzzy set theory: [6, 7, 11, 13, 15, 41, 49, 50, 52, 53, 61, 97, 107, 108, 115]
[92, 118, 123–125, 133, 137, 140–143, 146]
Multi-valued logic: [12, 17–26, 32–36]
[42–46, 51, 57–60, 63, 64, 66, 68]
[74–82, 93–96, 99–103, 103, 104]
[119–122, 126–132, 135]

Here we are dealing with vagueness, so we will not address the former two categories. Concerning the latter two categories, while there is a large literature related to the management of vagueness in logic programs, there are rule forms that are general enough to cover a large amount of them.

Indeed, current frameworks for managing vagueness in logic programming can roughly be classified into *annotation based* (AB) and *implication based* (IB).

- In the AB approach (e.g. [59, 60, 109, 110]), 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. [17, 23, 67, 68, 102, 137, 140] a rule is of the form

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

  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

[5] The author apologizes both to the authors and with the readers for all the relevant works, which are not cited here.

from a lattice. More recently, [17, 62, 68, 140] show that most of the frameworks dealing with imprecision and logic programming can be embedded into the IB framework. Our work falls into the IB approach.

In some cases, e.g. [68] there is also a function g, which dictates how to aggregate the truth values in case an atom is head of several rules. So, for instance, given the rules $A \leftarrow \phi_1$ and $A \leftarrow \phi_2$, they are roughly equivalent to $A \leftarrow g(\phi_1, \phi_2)$ rather than to $A \leftarrow \phi_1 \vee \phi_2$, as in our case.

There are also some extensions to many-valued positive disjunctive logic programs [99, 100, 128], while [90, 128] based on an extension of stable models semantics. However, no top-down query answering procedure is provided.

Very few works address *non-monotonic reasoning*, as [21, 42, 43, 71–80, 90, 99, 126, 128, 129], where the underlying truth-space are lattices, and its formulations goes over *bilattices* [48], like in [21] and this work. While [74–76, 80, 126, 127] uses logic programs or normal logic programs over bilattices directly under the IB framework.

Sorts, as used in [17, 23], can be simulated by using the join of lattices.

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 work presenting top-down procedures are [18, 60, 68, 127, 140]), but in very few of them non-monotonic negation is considered [126, 129], as already pointed out.

Another rising problem is the problem to compute the top-k ranked answers to a query, without computing the score of all answers. This allows to answer queries such as "find the top-k closest hotels to the conference location". Solutions to this problem for negation free logic programs can be found in [91, 130, 132]. No solution is yet known for normal logic programs.

6 Conclusions and future work

We have considered a general framework to deal with normal logic programs evaluated over complete lattices and with non-monotone negation. Atoms are assigned with truth interval approximations. Our main contribution is a very general tabling-like top-down method for answering queries.

The next step for future work is address the problem of computing the top-k ranked answers to a query, without computing the score of all answers as we did here. Another point is to extend our formalism to disjunctive logic programs with default negation were the head of a rule is a disjunction. It would be interesting to see whether our top-down query method can be extended to this general form (or at least to disjunctive logic programs) as well.

Acknowledgements We would like to thank to the anonymous reviewers of our earlier version, who pointed out to related work we were not aware of.

References

1. Alsinet, T., Godo, L.: Towards an automated deduction system for first-order possibilistic logic programming with fuzzy constants. Int. J. Intell. Syst. **17**(9), 887–924 (2002)

2. Alsinet, T., Godo, L., Sandri, S.: 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). Morgan Kaufmann, San Francisco (1999)
3. Alsinet, T., Godo, L.: 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), pp. 1–10. Morgan Kaufmann, San Francisco (2000)
4. Baldwin, J.F.: Evidential support of logic programming. Fuzzy Sets Syst. **24**(1), 1–26 (1987)
5. Baldwin, J.F.: A theory of mass assignments for artificial intelligence. Lect. Notes Comput. Sci. **833**, 22–34 (1994)
6. Baldwin, J.F., Martin, T.P., Pilsworth, B.W.: Fril—Fuzzy and Evidential Reasoning in Artificial Intelligence. Research Studies, Taunton (1995)
7. Baldwin, J.F., Martin, T.P., Pilsworth, B.W.: Applications of fuzzy computation: knowledge based systems: knowledge representation. In: Ruspini, E.H., Bonnissone, P., Pedrycz, W. (eds.) Handbook of Fuzzy Computing. IOP, Bristol (1998)
8. Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.D.: Magic sets and other strange ways to implement logic programs (extended abstract). In: Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS-86), New York, NY, USA, pp. 1–15. ACM, New York (1986)
9. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. In: Proceedings of the 7th International Conference in Logic Programming and Nonmonotonic Reasoning (LPNMR-04). Lecture Notes in Artificial Intelligence, Fort Lauderdale, FL, USA, No. 2923 pp. 21–33. Springer, Berlin Heidelberg New York (2004)
10. Belnap, N.D.: How a computer should think. In: Ryle, G. (ed.) Contemporary Aspects of Philosophy, pp. 30–56. Oriel, Stocksfield (1977)
11. Bueno, F., Cabeza, D., Carro, M., Hermenegildo, M., López-García, P., Puebla, G.: The Ciao prolog system. Reference manual. Technical Report CLIPS3/97.1, School of Computer Science, Technical University of Madrid (UPM) (1997). http://www.cliplab.org/Software/Ciao/
12. Calmet, J., Lu, J., Rodriguez, M., Schü, J.: Signed formula logic programming: operational semantics and applications. In: Rás, Z.W., Michalewicz, M. (eds.) Proceedings of the Ninth International Symposium on Foundations of Intelligent Systems. LNAI, Berlin, vol. 1079, pp. 202–211, 9–13. Springer, Berlin Heidelberg New York (1996)
13. Cao, T.H.: Annotated fuzzy logic programs. Fuzzy Sets Syst. **113**(2), 277–298 (2000)
14. Chesnevar, C., Simari, G., Alsinet, T., Godo, L.: A logic programming framework for possibilistic argumentation with vague knowledge. In: Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI-04), Arlington, Virginia, pp. 76–84. AUAI, Arlington (2004)
15. Chortaras, A., Stamou, G.B., Stafylopatis, A.: Integrated query answering with weighted fuzzy rules. In: 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-07). Lecture Notes in Computer Science, No. 4724 pp. 767–778. Springer, Berlin Heidelberg New York (2007)
16. Damásio, C.V., Pereira, L.M.: Hybrid probabilistic logic programs as residuated logic programs. Stud. Log. **72**(1), 113–138 (2002)
17. Damásio, C.V., Medina, J., Ojeda Aciego, M.: Sorted multi-adjoint logic programs: Termination results and applications. In: Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA-04). Lecture Notes in Computer Science, No. 3229 pp. 252–265. Springer, Berlin Heidelberg New York (2004)
18. Damásio, C.V., Medina, J., Ojeda Aciego, M.: A tabulation proof procedure for residuated logic programming. In: Proceedings of the 6th European Conference on Artificial Intelligence (ECAI-04), Valencia, 22–27 August 2004
19. Damásio, C.V., Medina, J., Ojeda Aciego, M.: Termination results for sorted multi-adjoint logic programs. In: Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-04), pp. 1879–1886, Perugia, 4–9 July 2004
20. Damásio, C.V., Pereira, L.M.: A survey of paraconsistent semantics for logic programs. In: Gabbay, D., Smets, P. (eds.) Handbook of Defeasible Reasoning and Uncertainty Management Systems, pp. 241–320. Kluwer, Deventer (1998)
21. Damásio, C.V., Pereira, L.M.: Antitonic logic programs. In: Proceedings of the 6th European Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01). Lecture Notes in Computer Science, No. 2173. Springer, Berlin Heidelberg New York (2001)

22. Damásio, C.V., Pereira, L.M.: Monotonic and residuated logic programs. In: Benferhat, S., Besnard, P. (eds.) Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 6th European Conference, ECSQARU 2001, Toulouse, France, 19–21 September 2001. Proceedings, Lecture Notes in Computer Science, vol. 2143, pp. 748–759. Springer, Berlin Heidelberg New York (2001)
23. Damásio, C.V., Pereira, L.M.: Sorted monotonic logic programs and their embeddings. In: Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-04), pp. 807–814, Perugia, 4–9 July 2004
24. Damásio, C.V., Medina, J., Ojeda-Aciego, M.: Termination of logic programs with imperfect information: applications and query procedure. J. Appl. Logic **5**(3), 435–458 (2007) September
25. Damásio, C.V., Medina, M., Ojeda-Aciego, J.: A tabulation procedure for first-order residuated logic programs. In: Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-06), Paris, 2–7 July 2006
26. Damásio, C.V., Medina, M., Ojeda-Aciego, J.: Termination of logic programs with imperfect information: applications and query procedure. J. Appl. Logic. **7**(5), 435–458 (2007)
27. Dekhtyar, A., Dekhtyar, M.I.: Possible worlds semantics for probabilistic logic programs. In: 20th International Conference on Logic Programming. Lecture Notes in Computer Science, vol. 3132, pp. 137–148. Springer, Berlin Heidelberg New York (2004)
28. Dekhtyar, A., Dekhtyar, M.I.: Revisiting the semantics of interval probabilistic logic programs. In: 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-05). Lecture Notes in Computer Science, No. 3662, pp. 330–342. Springer, Berlin Heidelberg New York (2005)
29. Dekhtyar, A., Dekhtyar, M.I., Subrahmanian, V.S.: Temporal probabilistic logic programs. In: De Schreye, D. (ed.) Logic Programming: The 1999 International Conference, pp. 109–123, Las Cruces, 29 November–4 December 1999
30. Dekhtyar, A., Subrahmanian, V.S.: Hybrid probabilistic programs. J. Log. Program. **43**(3), 187–250 (2000)
31. Dekhtyar, M.I., Dekhtyar, A., Subrahmanian, V.S.: Hybrid probabilistic programs: algorithms and complexity. In: Laskey, K.B., Prade, H. (eds.) Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99), S.F., Cal., 30– 1 1999, pp. 160–169. Morgan Kaufmann, San Francisco
32. Denecker, M., Marek, V., Truszczyński, M.: Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In: Minker, J. (ed.) Logic-Based Artifical Intelligence, pp. 127–144. Kluwer, Deventer (2000)
33. Denecker, M., Pelov, N., Bruynooghe, M.: Ultimate well-founded and stable semantics for logic programs with aggregates. In: Codognet, P. (ed.) Logic Programming, 17th International Conference, ICLP 2001, Paphos, Cyprus, 2001 November 26–December 1. Proceedings, Lecture Notes in Computer Science, vol. 2237. Springer, Berlin Heidelberg New York (2001)
34. Denecker, M., Marek, V.W., Truszczyński, M.: Uniform semantic treatment of default and autoepistemic logics. In: Cohn, A.G., Giunchiglia, F., Selman, B. (eds.) Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning, pp. 74–84. Morgan Kaufman, San Francisco (2000)
35. Denecker, M., Marek, V.W., Truszczyński, M.: Ultimate approximations. Technical Report CW 320, Katholieke Iniversiteit Leuven (2001)
36. Denecker, M., Marek, V.W., Truszczyński, M.: Ultimate approximations in nonmonotonic knowledge representation systems. In: Fensel, D., Giunchiglia, F., McGuinness, D., Williams, M. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the 8th International Conference, pp. 177–188. Morgan Kaufmann, San Francisco (2002)
37. Dong, G., Libkin, L., Wong, L.: Incremental recomputation in local languages. Inf. Comput. **181**(2), 88–98 (2003)
38. Dong, G., Su, J., Topor, R.W.: Nonrecursive incremental evaluation of datalog queries. Ann. Math. Artif. Intell. **14**(2–4), 187–223 (1995)
39. Dubois, D., Lang, J., Prade, H.: Towards possibilistic logic programming. In: Proc. of the 8th Int. Conf. on Logic Programming (ICLP-91), pp. 581–595. MIT, Cambridge (1991)
40. Dubois, D., Prade, H.: Possibility theory, probability theory and multiple-valued logics: A clarification. Ann. Math. Artif. Intell. **32**(1–4), 35–66 (2001)
41. Ebrahim, R.: Fuzzy logic programming. Fuzzy Sets Syst. **117**(2), 215–230 (2001)
42. Fitting, M.C.: The family of stable models. J. Logic Program. **17**, 197–225 (1993)
43. Fitting, M.C.: Fixpoint semantics for logic programming—a survey. Theor. Comp. Sci. **21**(3), 25–51 (2002)

44. Fitting, M.: A Kripke-Kleene-semantics for general logic programs. J. Logic Program. **2**, 295–312 (1985)
45. Fitting, M.: Pseudo-Boolean valued Prolog. Stud. Log. **XLVII**(2), 85–91 (1987)
46. Fitting, M.: Bilattices and the semantics of logic programming. J. Logic Program. **11**, 91–116 (1991)
47. Fuhr, N.: Probabilistic Datalog: Implementing logical information retrieval for advanced applications. J. Am. Soc. Inf. Sci. **51**(2), 95–110 (2000)
48. Ginsberg, M.L.: Multi-valued logics: a uniform approach to reasoning in artificial intelligence. Comput. Intell. **4**, 265–316 (1988)
49. Guller, D.: Procedural semantics for fuzzy disjunctive programs. In: Baaz, M., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning 9th International Conference, LPAR 2002, Tbilisi, Georgia, 14–18 October 2002. Proceedings, Lecture Notes in Computer Science, vol. 2514, pp. 247–261. Springer, Berlin Heidelberg New York (2002)
50. Guller, D.: Semantics for fuzzy disjunctive programs with weak similarity. In: Abraham, A., Köppen, M. (eds.) Hybrid Information Systems, First International Workshop on Hybrid Intelligent Systems, Adelaide, Australia, 2001 December 11–12, Proceedings, Advances in Soft Computing, pp. 285–299. Physica, Würzburg (2002)
51. Hähnle, R.: Uniform notation of tableaux rules for multiple-valued logics. In: Proc. International Symposium on Multiple-Valued Logic, Victoria, pp. 238–245. IEEE, Los Alamitos (1991)
52. Hinde, C.J.: Fuzzy prolog. Int. J. Man-Mach. Stud. **24**, 569–595 (1986)
53. Ishizuka, M., Kanai, N.: Prolog-ELF: incorporating fuzzy logic. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85), pp. 701–703, Los Angeles, CA, 18–23 August 1985
54. Kern-Isberner, G., Lukasiewicz, T.: Combining probabilistic logic programming with the power of maximum entropy. Artif. Intell. **157**(1–2), 139–202 (2004)
55. Kersting, K., De Raedt, L.: Bayesian logic programs. In: Cussens, J., Frisch, A.M. (eds.) ILP Work-in-progress reports, 10th International Conference on Inductive Logic Programming, CEUR Workshop Proceedings. CEUR-WS.org (2000)
56. Kersting, K., De Raedt, L.: Bayesian logic programming: Theory and tools. In: Getoor, L., Taskar, B. (eds.) An Introduction to Statistical Relational Learning. MIT, Cambridge (2005)
57. Khamsi, M.A., Misane, D.: Disjunctive signed logic programs. Fundam. Inform. **32**, 349–357 (1996)
58. Khamsi, M.A., Misane, D.: Fixed point theorems in logic programming. Ann. Math. Artif. Intell. **21**, 231–243 (1997)
59. Kifer, M., Li, A.: 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, pp. 102–117. Springer, Berlin Heidelberg New York (1988)
60. Kifer, M., Subrahmanian, V.S.: Theory of generalized annotated logic programming and its applications. J. Logic Program. **12**, 335–367 (1992)
61. Klawonn, F., Kruse, R.: A Łukasiewicz logic based Prolog. Mathw. Soft Comput. **1**(1), 5–29 (1994)
62. Krajči, S., Lencses, R., Vojtáš, P.: A comparison of fuzzy and annotated logic programming. Fuzzy Sets Syst. **144**, 173–192 (2004)
63. Kulmann, P., Sandri, S.: An annotaded logic theorem prover for an extended possibilistic logic. Fuzzy Sets Syst. **144**, 67–91 (2004)
64. Lakshmanan, L.: An epistemic foundation for logic programming with uncertainty. In: Foundations of Software Technology and Theoretical Computer Science. Lecture Notes in Computer Science, no. 880, pp. 89–100. Springer, Berlin Heidelberg New York (1994)
65. Lakshmanan, L.V.S., Sadri, F.: On a theory of probabilistic deductive databases. Theory Pract. Log. Program. **1**(1), 5–42 (2001)
66. Lakshmanan, L.V.S., Sadri, F.: Uncertain deductive databases: a hybrid approach. Inf. Syst. **22**(8), 483–508 (1997)
67. Lakshmanan, L.V.S., Shiri, N.: Probabilistic deductive databases. In: Int'l. Logic Programming Symposium, pp. 254–268. MIT, Cambridge (1994)
68. Lakshmanan, L.V.S., Shiri, N.: A parametric approach to deductive databases with uncertainty. IEEE Trans. Knowl. Data Eng. **13**(4), 554–570 (2001)
69. Leone, N., Rullo, P., Scarcello, F.: Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. Inf. Comput. **135**(2), 69–112 (1997)
70. Libkin, L., Wong, L.: On the power of incremental evaluation in SQL-like languages. Lect. Notes Comput. Sci. **1949**, 17–30 (2000)

71. Loyer, Y., Spyratos, N.: Hypothesis-Founded Semantics for Datalog Programs with Negation. In: 27th International Symposium on Mathematical Foundations of Computer Science (MFCS-2002). Lecture Notes in Computer Science, Warsaw, Poland, no. 2420, pp. 483–494. Springer, Berlin Heidelberg New York (2002)
72. Loyer, Y., Spyratos, N., Stamate, D.: Parametrized semantics of logic programs–a unifying framework. Theor. Comput. Sci. **308**(1–3), 429–447 (2003)
73. Loyer, Y., Spyratos, N., Stamate, D.: Hypothesis-based semantics of logic programs in multivalued logics. ACM Trans. Comput. Log. **5**(3), 508–527 (2004)
74. Loyer, Y., Straccia, U.: Uncertainty and partial non-uniform assumptions in parametric deductive databases. In: Proc. of the 8th European Conference on Logics in Artificial Intelligence (JELIA-02). Lecture Notes in Computer Science, Cosenza, Italy, no. 2424, pp. 271–282. Springer, Berlin Heidelberg New York (2002)
75. Loyer, Y., Straccia, U.: The well-founded semantics in normal logic programs with uncertainty. In: Proc. of the 6th International Symposium on Functional and Logic Programming (FLOPS-2002). Lecture Notes in Computer Science, Aizu, Japan, no. 2441, pp. 152–166. Springer, Berlin Heidelberg New York (2002)
76. Loyer, Y., Straccia, U.: The approximate well-founded semantics for logic programs with uncertainty. In: 28th International Symposium on Mathematical Foundations of Computer Science (MFCS-2003). Lecture Notes in Computer Science, Bratislava, Slovak Republic, no. 2747, pp. 541–550. Springer, Berlin Heidelberg New York (2003)
77. Loyer, Y., Straccia, U.: Default knowledge in logic programs with uncertainty. In: Proc. of the 19th Int. Conf. on Logic Programming (ICLP-03). Lecture Notes in Computer Science, Mumbai, India, no. 2916, pp. 466–480. Springer, Berlin Heidelberg New York (2003)
78. Loyer, Y., Straccia, U.: Epistemic foundation of the well-founded semantics over bilattices. In: 29th International Symposium on Mathematical Foundations of Computer Science (MFCS-2004). Lecture Notes in Computer Science, Bratislava, Slovak Republic, no. 3153, pp. 513–524. Springer, Berlin Heidelberg New York (2004)
79. Loyer, Y., Straccia, U.: Any-world assumptions in logic programming. Theor. Comput. Sci. **342**(2–3), 351–381 (2005)
80. Loyer, Y., Straccia, U.: Epistemic foundation of stable model semantics. J. Theor. Pract. Log. Program. **6**, 355–393 (2006)
81. Lu, J.J.: Logic programming with signs and annotations. J. Log. Comput. **6**(6), 755–778 (1996)
82. Lu, J.J., Calmet, J., Schü, J.: Computing multiple-valued logic programs. Mathw. Soft Comput. **2**(4), 129–153 (1997)
83. Lukasiewicz, T.: Many-valued first-order logics with probabilistic semantics. In: Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL'98). Lecture Notes in Computer Science, no. 1584, pp. 415–429. Springer, Berlin Heidelberg New York (1998)
84. Lukasiewicz, T.: Probabilistic logic programming. In: Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98), Brighton (England), pp. 388–392 (1998, August)
85. Lukasiewicz, T.: Many-valued disjunctive logic programs with probabilistic semantics. In: Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99). Lecture Notes in Computer Science, no. 1730, pp. 277–289. Springer, Berlin Heidelberg New York (1999)
86. Lukasiewicz, T.: Probabilistic and truth-functional many-valued logic programming. In: The IEEE International Symposium on Multiple-Valued Logic, pp. 236–241. IEEE, Piscataway (1999)
87. Lukasiewicz, T.: Fixpoint characterizations for many-valued disjunctive logic programs with probabilistic semantics. In: Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01). Lecture Notes in Artificial Intelligence, no. 2173, pp. 336–350. Springer, Berlin Heidelberg New York (2001)
88. Lukasiewicz, T.: Probabilistic logic programming under inheritance with overriding. In: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI-01), San Francisco, CA, USA, pp. 329–336. Morgan Kaufmann, San Francisco (2001)
89. Lukasiewicz, T.: Probabilistic logic programming with conditional constraints. ACM Trans. Comput. **2**(3), 289–339 (2001)
90. Lukasiewicz, T., Straccia, U.: Tightly integrated fuzzy description logic programs under the answer semantics for the semantic web. In: Proceedings of the First International Conference

on Web Reasoning and Rule Systems (RR-07). Lecture Notes in Computer Science, no. 4524, pp. 289—298. Springer, Berlin Heidelberg New York (2007)
91. Lukasiewicz, T., Straccia, U.: Top-k retrieval in description logic programs under vagueness for the semantic web. In: Proceedings of the 1st International Conference on Scalable Uncertainty Management (SUM-07). Lecture Notes in Computer Science, no. 4772, pp. 16–30. Springer, Berlin Heidelberg New York (2007)
92. Magrez, P., Smets, P.: Fuzzy modus ponens: a new model suitable for applications in knowledge-based systems. Int. J. Intell. Syst. **4**, 181–200 (1989)
93. Majkic, Z.: Coalgebraic semantics for logic programs. In: 18th Workshop on (Constraint) Logic Programming ((W(C)LP-05), Ulm, May 2004
94. Majkic, Z.: Many-valued intuitionistic implication and inference closure in a bilattice-based logic. In: 35th International Symposium on Multiple-Valued Logic (ISMVL-05), no. 214–220, Calgary, 19–21 May 2005
95. Majkic, Z.: Truth and knowledge fixpoint semantics for many-valued logic programming. In: 19th Workshop on (Constraint) Logic Programming ((W(C)LP-05), no. 76–87, Ulm, 21–25 February 2005
96. Marek, V.W., Truszczyński, M.: Logic programming with costs. Technical report, University of Kentucky. ftp://al.cs.engr.uky.edu/cs/manuscripts/lp-costs.ps (2000)
97. Martin, T.P., Baldwin, J.F., Pilsworth, B.W.: The implementation of FProlog –a fuzzy prolog interpreter. Fuzzy Sets Syst. **23**(1), 119–129 (1987)
98. Martin, T.P.: Soft computing, logic programming and the semantic web. In: Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-04), pp. 815–822, Perugia, 4–9 July 2004
99. Mateis, C.: Extending disjunctive logic programming by t-norms. In: Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-99). Lecture Notes in Computer Science, no. 1730, pages 290–304. Springer, Berlin Heidelberg New York (1999)
100. Mateis, C.: Quantitative disjunctive logic programming: semantics and computation. AI Commun. **13**, 225–248 (2000)
101. Medina, J., Ojeda-Aciego, M.: Multi-adjoint logic programming. In: Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-04), pp. 823–830, Perugia, 4–9 July 2004
102. Medina, J., Ojeda-Aciego, M., Vojtáš, P.: Multi-adjoint logic programming with continuous semantics. In: Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01). Lecture Notes in Artificial Intelligence, vol. 2173, pp. 351–364. Springer, Berlin Heidelberg New York (2001)
103. Medina, J., Ojeda-Aciego, M., Vojtáš, P.: 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, pp. 290–297. Springer, Berlin Heidelberg New York (2001)
104. Medina, J., Ojeda-Aciego, M., Vojtáš, P.: Similarity-based unification: a multi-adjoint approach. Fuzzy Sets Syst. **1**(146), 43–62 (2004)
105. Minker, J.: On indefinite data bases and the closed world assumption. In: Proc. of the 6th Conf. on Automated Deduction (CADE-82). Lecture Notes in Computer Science, no. 138. Springer, Berlin Heidelberg New York (1982)
106. Muggleton, S.: Stochastic logic programs. In: De Raedt, L. (ed.) Proceedings of the 5th International Workshop on Inductive Logic Programming, p. 29. Department of Computer Science, Katholieke Universiteit Leuven (1995)
107. Mukaidono, M.: Foundations of fuzzy logic programming. In: Advances in Fuzzy Systems—Application and Theory, vol. 1. World Scientific, Singapore (1996)
108. Mukaidono, M., Shen, Z., Ding, L.: Fundamentals of fuzzy prolog. Int. J. Approx. Reason. **3**(2), 179–193 (1989)
109. Ng, R., Subrahmanian, V.S.: Stable model semantics for probabilistic deductive databases. In: Ras, Z.W., Zemenkova, M. (eds.) Proc. of the 6th Int. Sym. on Methodologies for Intelligent Systems (ISMIS-91). Lecture Notes in Artificial Intelligence, no. 542, pp. 163–171. Springer, Berlin Heidelberg New York (1991)
110. Ng, R., Subrahmanian, V.S.: Probabilistic logic programming. Inf. Comput. **101**(2), 150–201 (1993)

111. Ng, R., Subrahmanian, V.S.: Stable model semantics for probabilistic deductive databases. Inf. Comput. **110**(1), 42–83 (1994)
112. Ngo, L.: Probabilistic disjunctive logic programming. In: Uncertainty in Artificial Intelligence: Proceedings of the Twelfeth Conference (UAI-1996), San Francisco, CA, pp. 397–404. Morgan Kaufmann, San Francisco (1996)
113. Ngo, L., Haddawy, P.: Answering queries from context-sensitive probabilistic knowledge bases. Theor. Comput. Sci. **171**(1–2), 147–177 (1997)
114. Nicolas, P., Garcia, L., Stéphan, I.: Possibilistic stable models. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05), pp. 248–253. Morgan Kaufmann, San Francisco (2005)
115. Paulik, L.: Best possible answer is computable for fuzzy sld-resolution. In: Hajék, P. (ed.) Gödel 96: Logical Foundations of Mathematics, Computer Science, and Physics. Lecture Notes in Logic, no. 6, pp. 257–266. Springer, Berlin Heidelberg New York (1996)
116. Poole, D.: Probabilistic horn abduction and bayesian networks. Artif. Intell. **64**(1), 81–129 (1993)
117. Poole, D.: The independent choice logic for modelling multiple agents under uncertainty. Artif. Intell. **94**(1–2), 7–56 (1997)
118. Rhodes, P.C., Merad Menani, S.: Towards a fuzzy logic programming system: a clausal form fuzzy logic. Knowl.-Based Syst. **8**(4), 174–182 (1995)
119. Rounds, W.C., Zhang, G.-Q.: Clausal logic and logic programming in algebraic domains. Inf. Comput. **171**, 183–200 (2001)
120. Schroeder, M., Schweimeier, R.: Fuzzy argumentation and extended logic programming. In: Proceedings of ECSQARU Workshop Adventures in Argumentation, Toulouse, September 2001
121. Schroeder, M., Schweimeier, R.: Arguments and misunderstandings: Fuzzy unification for negotiating agents. In: Proceedings of the ICLP workshop CLIMA02. Elsevier, Amsterdam (2002)
122. Schroeder, M., Schweimeier, R.: Fuzzy unification and argumentation for well-founded semantics. In: Proceedings of the Conference on Current Trends in Theory and Practice of Informatics (SOFSEM-04). Lecture Notes in Computer Science, no. 2932, pp. 102–121. Springer, Berlin Heidelberg New York (2004)
123. Sessa, M.I.: Approximate reasoning by similarity-based sld resolution. Theor. Comput. Sci. **275**, 389–426 (2002)
124. Shapiro, E.Y.: Logic programs with uncertainties: a tool for implementing rule-based systems. In: Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83), pp. 529–532, Karlsruhe, 8–12 August 1983
125. Shen, Z., Ding, L., Mukaidono, L.: A theoretical framework of fuzzy prolog machine. In: Fuzzy Computing, pp. 89–100. Elsevier, Amsterdam (1988)
126. Straccia, U.: Query answering in normal logic programs under uncertainty. In: 8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05). Lecture Notes in Computer Science, Barcelona, Spain, no. 3571, pp. 687–700. Springer, Berlin Heidelberg New York (2005)
127. Straccia, U.: Uncertainty management in logic programming: Simple and effective top-down query answering. In: Khosla, R., Howlett, R.J., Jain, L.C. (eds.) 9th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems (KES-05), Part II. Lecture Notes in Computer Science, Melbourne, Australia, no. 3682, pp. 753–760. Springer, Berlig Heidelberg New York (2005)
128. Straccia, U.: Annotated answer set programming. In: Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-06), pp. 1212–1219. E.D.K., Paris (2006)
129. Straccia, U.: Query answering under the any-world assumption for normal logic programs. In: Proceedings of the 10th International Conference on Principles of Knowledge Representation (KR-06), pp. 329–339. AAAI, Menlo Park (2006)
130. Straccia, U.: Towards top-k query answering in deductive databases. In: Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics (SMC-06), pp. 4873–4879. IEEE, Piscataway (2006)
131. Straccia, U.: A top-down query answering procedure for normal logic programs under the any-world assumption. In: Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-07). Lecture Notes in Computer Science, no. 4724, pp. 115–127. Springer, Berlin Heidelberg New York (2007)

132. Straccia, U.: Towards vague query answering in logic programming for logic-based information retrieval. In: World Congress of the International Fuzzy Systems Association (IFSA-07). Lecture Notes in Computer Science, Cancun, Mexico, no. 4529, pp. 125–134. Springer, Berlin Heidelberg New York (2007)
133. Subramanian, V.S.: On the semantics of quantitative logic programs. In: Proc. 4th IEEE Symp. on Logic Programming, pp. 173–182. Computer Society Press, Los Angeles (1987)
134. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pac. J. Math. **5**, 285–309 (1955)
135. Turner, H.: Signed logic programs. In: Bruynooghe, M. (ed.) Logic Programming: Proc. of the 1994 International Symposium, pp. 61–75. MIT, Piscataway (1994)
136. Ullman, J.D.: Principles of Database and Knowledge Base Systems, vols. 1,2. Computer Science Press, Potomac (1989)
137. van Emden, M.H.: Quantitative deduction and its fixpoint theory. J. Log. Program. **4**(1), 37–53 (1986)
138. van Gelder, A., Ross, K.A., Schlimpf, J.S.: The well-founded semantics for general logic programs. J. ACM. **38**(3), 620–650 (1991)
139. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: 20th International Conference on Logic Programming (ICLP-04). Lecture Notes in Computer Science, vol. 3132, pp. 431–445. Springer, Berlin Heidelberg New York (2004)
140. Vojtáš, P.: Fuzzy logic programming. Fuzzy Sets Syst. **124**, 361–370 (2001)
141. Vojtáš, P., Paulík, L.: Soundness and completeness of non-classical extended SLD-resolution. In: 5th International Workshop on Extensions of Logic Programming (ELP'96). Lecture Notes in Artificial Intelligence, no. 1050, pp. 289–301, Leipzig, 28–30 March 1996
142. Vojtáš, P., Vomelelová, M.: 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 Management of Uncertainty in Knowledge-Based Systems, (IPMU-04), pp. 839–846, Perugia, 4–9 July 2004
143. Wagner, G.: Negation in fuzzy and possibilistic logic programs. In: Martin, T., Arcelli, F. (eds.) Logic Programming and Soft Computing. Research Studies, Taunton (1998)
144. Warren, D.S.: Memoing for logic programs. Commun. ACM. **35**(3), 93–111 (1992)
145. Wüttrich, B.: Probabilistic knowledge bases. IEEE Trans. Knowl. Data Eng. **7**(5), 691–698 (1995)
146. Yasui, H., Hamada, Y., Mukaidono, M.: Fuzzy prolog based on lukasiewicz implication and bounded product. IEEE Trans. Fuzzy Syst. **2**, 949–954 (1995)