Approximate Well-founded Semantics, Query Answering and Generalized Normal Logic Programs over Lattices

Yann Loyer[1] and Umberto Straccia[2]

[1] PRiSM, Université de Versailles, Saint Quentin
45 Avenue des Etats-Unis,
78035 Versailles, FRANCE

[2] I.S.T.I. - C.N.R.,
Via G. Moruzzi, 1
I-56124 Pisa, ITALY

Technical Report: ISTI-2005-TR-xx

October 25, 2005

Abstract

The management of imprecise information in logic programs becomes to be important whenever the real world information to be represented is of imperfect nature and the classical crisp *true, false* approximation is not adequate.

In this work, we consider generalized normal logic programs over complete lattices, where computable truth combination functions may appear in the rule bodies to manipulate truth values. It is an unifying umbrella for existing approaches for many-valued normal logic programs. We will provide declarative and fixed-point semantics of the logic and provide a simple and effective top-down query answering procedure by a transformation to an equational system over lattices.

Categories: F.4.1-Mathematical Logic and Formal Languages-Mathematical Logic-Logic and constraint programming, I.2.3-Artificial Intelligence-Deduction and Theorem Proving-Logic programming

Keywords: logic programming, nonmonotonic logic, many-valued logic, top-down

1 Introduction

The management of uncertainty and/or imprecision within deduction systems is an important issue whenever the real world information to be represented is of imperfect nature. In logic programming, the problem has attracted the attention of many researchers and numerous frameworks have been proposed. Essentially, they differ in the underlying notion of uncertainty theory and imprecision theory (*Probability theory* [3, 15, 16, 17, 25, 30, 37, 50, 51, 52, 53, 54, 55, 63, 67, 68, 69, 70, 80, 83], *Fuzzy set theory* [7, 20, 28, 29, 33, 56, 65, 64, 72, 73, 78, 81, 82, 84], *Multi-valued logic* [9, 10, 11, 12, 13, 14, 21, 22, 23, 24, 31, 32, 35, 36, 38, 42, 43, 44, 45, 46, 47, 48, 49, 57, 58, 61, 59, 60, 75, 74], *Possibilistic logic* [1, 2, 18, 71]) and how uncertainty/imprecision values, associated to rules and facts, are managed.

Under "uncertainty theory" fall all those approaches in which statements rather than being either true or false, are true or false to some probability or possibility/necessity, while under "imprecision theory" fall all those approaches in which statements are true to some degree which is taken from a truth space (see [19] for a clarification between the notions of uncertainty and imprecision). In this work we deal with *imprecision* and, thus, statements have a degree of truth.

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

- In the AB approach (e.g. [31, 32, 66, 67]), a rule is of the form

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

  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. [9, 14, 37, 38, 60, 78, 81] 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 from a lattice. More recently, [9, 34, 38, 81] show that most of the frameworks dealing with imprecision and logic programming can be embedded into the IB framework.

However, most of the approaches stress an important limitation for real-world applications, as they do not address any mode of *non-monotonic reasoning*. In particular, no default negation operation is defined. Exception to this limitation are [13, 41, 42, 43, 44, 74, 75], where the underlying truth-space are lattices, and its formulations goes over *bilattices* [27] (a less general structure than lattices).

Additionally, in most frameworks, in order to answer to a query, we have to compute the whole intended model (e.g., by a bottom-up fixed-point computation) and then answer with the evaluation of the query in this model. This always requires the computation of a whole model, even if not all the atom's truth is required to determine the answer (some work presenting top-down procedures are [10, 32, 38, 75, 81], but in none of them non-monotonic negation is considered (and [74] deals with normal logic programs over bilattices).

The contribution of this work is as follows. We present a general framework for generalized normal logic programs with many-valued semantics. We generalize the well-known well-founded semantics for classical normal logical programs [79] to the many-valued case. 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 all approaches to many-valued normal logic programming. Additionally, we present a very general top-down method for answering queries, by a transformation of a normal logic program into an equational system over lattices and then by developing a top-down query answering procedure for such equational systems.

As seen above, there are many works dealing with imprecision with logic programming with or without negation, either using the AB approach or the IB approach. The use of arbitrary computable truth combination functions in the body is sufficiently expressive to subsume all those work dealing with normal logic programs. To the best of our knowledge there is no other work which has the expressive power of our formalism and also presents a top-down query answering procedure for normal logic programs. Most works deal with logic programming without negation, though may provide some technique to answer queries in a top-down manner, as e.g. [10, 32, 38, 81]. On the other hand, we are not aware of other works dealing with normal logic programs, like [13], providing a top-down query answering procedure, too.

Closest to our approach are [74, 75]. While [74] considers positive programs only, but the top-down procedure is a special case of the one we presented here, we consider here non-monotonic negation as well as in [75]. The main differences of this work to [75] are as follows. (i) In [75] we consider bilattices as truth space, while here we rely on complete lattices only. (ii) Furthermore, in [75] the semantics is given by relying on a generalization of the Gelfond-Lifschitz transform [26] due to Fitting [21, 22]. Here we follow a different approach, but semantically equivalent to [75], based on the so-called notion of *support* [41]. The support is a generalization of the notion of *unfounded sets* [79] to characterize the well-founded semantics of classical logic programs. The effect of this choice is the development of a top-down procedure, which is similar to the tabulation procedure [76], but generalized to *many-valued normal logic programs with arbitrary computable truth combination functions in rule bodies*.

In the remaining, we proceed as follows. In the following section, we give some basic definitions about our formalism and some illustrative examples. Section 3 contains the definitions of interpretation and model of a program. In Section 4, we define the intended semantics of normal logic programs. In Section 5 we present a top-down query answering procedure, while Section 6 concludes and addresses future directions of work.

2 Preliminaries

2.1 Truth lattice

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 $\vee$. The main idea is that an statement $P(a)$, rather than being interpreted as either true or false, will be mapped into a truth value $c \in L$. The intended meaning is that c indicates to which extend $P(a)$ is true. Finally, we also assume that $\mathcal{L}$ has a *negation*, i.e. an operator $\neg$ that reverses the $\preceq$ ordering and verifies $\neg\neg x = x$.

Typical truth lattices are the following.

Classical 0-1: $\mathcal{L}_{\{0,1\}}$ corresponds to the classical truth-space, where 0 stands for 'false', while 1 stands for 'true' and $\neg 0 = 1$.

Fuzzy: $\mathcal{L}_{[0,1]\cap\mathbb{Q}}$, which relies on the unit interval, is quite frequently used as truth lattice with $\neg x = 1 - x$.

Four-valued: another frequent truth lattice is Belnap's $\mathcal{FOUR}$ [5], where L is $\{f, t, u, i\}$ with $f \preceq u \preceq t$ and $f \preceq i \preceq t$. Here, u stands for 'unknown', whereas i stands for inconsistency. Concerning negation, we have $\neg f = t$, $\neg u = u$ and $\neg i = i$. We denote the lattice as $\mathcal{L}_B$.

Many-valued: $L = \langle \{0, \frac{1}{n-1}, \ldots \frac{n-2}{n-1}, 1\}, \leq \rangle$, n positive integer and $\neg x = 1 - x$.

In a complete lattice $\mathcal{L} = \langle L, \preceq \rangle$, a function $f \colon L \to L$ is *monotone*, if $\forall x, y \in L$, $x \preceq y$ implies $f(x) \preceq f(y)$. A *fixed-point* of f is an element $x \in L$ such that $f(x) = x$. The basic tool for studying fixed-points of functions on lattices is the well-known Knaster-Tarski theorem [77]. Let f be a monotone function on a complete lattice $\langle L, \preceq \rangle$. Then f has a fixed-point, the set of fixed-points of f is a complete lattice and, thus, f has a *least* fixed-point. The *least* fixed-point of f can be obtained by iterating f 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 $i \geq 0$, $y_0 = \bot$, $y_{i+1} = f(y_i)$, while for a limit ordinal λ, $y_\lambda = lub\{y_i \colon i < \lambda\}$. We denote the least fixed-point by $\mathrm{lfp}(f)$. 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.

2.2 Generalized normal logic programs

Fix a lattice $\mathcal{L} = \langle L, \preceq \rangle$. We assume that $\mathcal{F}$ is a family of continuous n-ary functions $f \colon \mathcal{L}^n \to \mathcal{L}$. 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*.

We extend standard logic programs [40] to the case where *arbitrary computable functions* $f \in \mathcal{F}$ are allowed to manipulate truth values. That is, we allow any $f \in \mathcal{F}$ to appear in the body of a rule to be used to combine the truth of the atoms appearing in the body and to propagate the result to the atom in the head.

Consider an arbitrary first order language that contains infinitely many variables, constants, and predicate symbols. 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_i 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 to apply the arithmetic functions, $\min, \max, 1-$ and product $\cdot$ to determine the value of φ. Note that for ease of exposition, we will use e.g. the symbol $\min$ both at the syntactic level, writing $\min(p, q)$, as well as in its interpretation (e.g., $I(\min(I(p), I(q))) = \min(I(p), I(q))$, where I is an interpretation –see Section 3) 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 *generalized normal logic program*, or simply *normal logic program*, denoted with $\mathcal{P}$, 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 generalized 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 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, $A \leftarrow \varphi_1$, $A \leftarrow \varphi_2, \ldots$ 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.

In the following, we recall some examples, which both might help informally the reader to get confidence with the formalism and show how our formalism may capture different approaches to the management of imprecision (and some forms of uncertainty) in logic programming (some examples are taken from [38]).

Consider the following logic program with the four rules r_i,

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

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

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

Example 2 ([18]) *Consider $\mathcal{L}_{[0,1]}$. Suppose $\alpha_1 = 0.8, \alpha_2 = \alpha_3 = 0.7$, and $\alpha_4 = 0.8$ are possibility/necessity degrees associated with the implications. Suppose f_i is $\min$. Then $\mathcal{P}$ is a program in the framework proposed by Dubois et al. [18], which is founded on Zadeh's possibility theory [86]. In a fixed-point evaluation of $\mathcal{P}$, the possibility/necessity degrees derived for A, B, C are 0.7, 0.7, 0.8, respectively.*

Example 3 ([78]) *Consider $\mathcal{L}_{[0,1]}$ and suppose α_i as defined in Example 2. But, suppose f_i is multiplication $(\cdot)$. Then $\mathcal{P}$ is a program in van Emden's framework [78], which is mathematically founded on the theory of fuzzy sets proposed by Zadeh [85]. In a fixed-point evaluation of $\mathcal{P}$, the values derived for A, B, C are 0.56, 0.7, 0.8, respectively.*

Example 4 (MYCIN [6]) *Consider $\mathcal{L}_{[0,1]}$, and suppose α_i's are probabilities defined as in the previous example. Suppose f_i is $(\cdot)$. However, in order to simulate a probabilistic setting, in particular related to the atom A, with independent events, we write the program above as:*

$$\begin{array}{lcl} r_0 & : & A \leftarrow f_s(A', A'') \\ r_1' & : & A' \leftarrow f_1(0.8, B) \\ r_2' & : & A'' \leftarrow f_2(0.7, C) \\ r_3 & : & B \leftarrow 0.7 \\ r_4 & : & C \leftarrow 0.8 \end{array}$$

where we use two new atoms A' and A'' to indicate that A is head of two rules and use the algebraic sum $f_s(\alpha, \beta) = \alpha + \beta - \alpha \cdot \beta$ to sum up the probabilities of deriving A. Viewing an atom as an event, f_s returns the probability of the occurrence, of any one of two independent events, in the probabilistic sense. Note that f_s is the disjunction function used in MYCIN [6]. Let us consider a fixed-point evaluation of $\mathcal{P}$. In the first step, we derive B and C with probabilities 0.7 and 0.8, respectively. In step 2, applying r_1' and r_2', we obtain two derivations of A (namely for A' and A''), the probability of each of which is 0.56. The probability of A is then defined as $f_s(0.56, 0.56) = 0.8064$, which is indeed the probability that A occurs.

From Example 4 above, it is easy to see that more generally, in order to accommodate independent probabilities, a logic program $\mathcal{P}$ has to be transformed into it's grounded version $\mathcal{P}^*$, but where ground rules with same head $A \leftarrow \varphi_1$, $A \leftarrow \varphi_2$, $\ldots$. rather being transformed into $A \leftarrow \varphi_1 \vee \varphi_2 \vee \ldots$, are transformed into $A \leftarrow f_s(\ldots f_s(\varphi_1, \varphi_2) \ldots\ldots)$.

In a similar way, we can manage [38].

Example 5 (PDDU [38, 44]) *In [38], a* Parametric Approach to Deductive Databases with Uncertainty *(PDDU) is proposed, where rules have the form*

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

f_d is the disjunction function associated with A and, f_c and f_p are respectively the conjunction and propagation functions associated with the rule r. α_r is the weight of the rule. Roughly, this functions are mappings from $L \times L$ to L and are continuous w.r.t. each one of its arguments and satisfying some constraints such that they behave as conjunction and disjunction functions (see, [38]). The intuition behind a rule is as follows. Ground the program and evaluate each atom B_i. Combine their truth using the conjunction function f_c, i.e. let $c_1 = f_c(B_1, \ldots, B_n)$ (for instance, $c_1 = \min(B_1, \ldots, B_n)$). Then propagate the truth value c_1 to the head using the weight of the rule α_r and the propagation function f_p, i.e. let $c'_1 = f_p(\alpha_r, c_1)$ (for instance, $c'_1 = \alpha_r \cdot c_2$). Repeat this operation for rules heaving A in the head. If $c'_1, \ldots, c'_k$ are all this values, combine them using the disjunction function f_d, $c_A = f_d(c'_1, \ldots, c'_k)$ (for instance, $c_A = \max(c'_1, \ldots, c'_k)$). It is then straightforward to see that a logic program $\mathcal{P}$ in the sense of [38] can be represented in our framework by grounding $\mathcal{P}$ and then transforming rule of the form $r : A \overset{\alpha_r}{\leftarrow} B_1, ..., B_n; \langle f_d, f_p, f_c \rangle$ into

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

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

[44] is as [38], but additionally non-monotonic negation is considered as well. We can encode [44] into our framework in the same way as for [38]. Additionally, we would like to note that [44] does not provide any top-down query answering procedure as we do.

Example 6 (Fuzzy Logic Programming [81]) *In [81],* Fuzzy Logic Programming *is proposed, where rules have the form*

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

for some specific f and the truth space is $\mathcal{L}_{[0,1] \cap \mathbb{Q}}$. [81] is just a special case of our framework. Also, [81] does not support negation.

As an illustrative example consider the following scenario. Assume that we have the following facts, represented in the tables below. There are hotels and *conferences, their locations and the distance among locations.*

HasLocationH	
HotelID	HasLocationH
h1	hl1
h2	hl2
⋮	⋮

HasLocationC	
ConferenceID	HasLocationC
c1	cl1
c2	cl2
⋮	⋮

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

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

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

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

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

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

Result List	
HotelID	Closeness degree
h1	0.7
h2	0.25
⋮	⋮

Finally, consider the following example whose semantics will be studied later on.

Example 7 *Consider an insurance company, which has information about its customers used to determine the risk coefficient of each customer. The company has: (i) data grouped into a set F of facts; and (ii) a set R of rules. Suppose the company has the following database (which is a program $P = F \cup R$), where a value of the risk coefficient may be already known, but has to be re-evaluated (the client may be a new client and his risk coefficient is given by his precedent insurance company). The truth lattice is $\mathcal{L}_{[0,1]\cap\mathbb{Q}}$.*

$$F = \begin{cases} \texttt{Experience(John)} & \leftarrow & 0.7 \\ \texttt{Risk(John)} & \leftarrow & 0.5 \\ \texttt{Sport\_car(John)} & \leftarrow & 0.8 \end{cases}$$

$$R = \begin{cases} \texttt{Good\_driver(X)} & \leftarrow & \min(\texttt{Experience(X)}, \neg\texttt{Risk(X)}) \\ \texttt{Risk(X)} & \leftarrow & 0.8 \cdot \texttt{Young(X)} \\ \texttt{Risk(X)} & \leftarrow & 0.8 \cdot \texttt{Sport\_car(X)} \\ \texttt{Risk(X)} & \leftarrow & \min(\texttt{Experience(X)}, \neg\texttt{Good\_driver(X)}) \end{cases}$$

Then in $\mathcal{P}^*$ the rules become

$$R^* = \begin{cases} \texttt{Good\_driver(John)} & \leftarrow & \min(\texttt{Experience(John)}, \neg\texttt{Risk(John)}) \\ \texttt{Risk(John)} & \leftarrow & \max(\\ & & \quad 0.8 \cdot \texttt{Young(John)}, \\ & & \quad 0.8 \cdot \texttt{Sport\_car(John)}, \\ & & \quad \min(\texttt{Experience(John)}, \\ & & \qquad \neg\texttt{Good\_driver(John)})) \end{cases}$$

Using another disjunction function associated to the rules with head `Risk`*, such as the algebraic sum $f_s(x, y) = x + y - xy$, might have been more appropriate in such an example (i.e. we accumulate the risk factors, rather than take the* $\max$ *only), but we will use* $\max$ *in order to facilitate the reader's comprehension later on when we compute the semantics of $\mathcal{P}$.*

3 Interpretations

The semantics of a program $\mathcal{P}$ is determined by selecting a particular interpretation of $\mathcal{P}$ in the set of models of $\mathcal{P}$, where 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, that chosen 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 8 (Running example) *Consider the truth lattice $\mathcal{L}_{[0,1]}$ and the program $\mathcal{P}$*

$$\begin{array}{rcl} A & \leftarrow & \max(\neg B, C) \\ B & \leftarrow & \max(\neg A, D) \\ C & \leftarrow & \max(0.3, \min(D, 0.6)) \\ D & \leftarrow & D \end{array}$$

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

Concerning the previous example we may note that the truth of A in the minimal models is in the interval $[0.3, 1]$, while for B the interval is $[0, 0.7]$. The semantics we device, is to provide these intervals as an *approximation* to the truth of the atoms A and B.

We propose to rely on $L \times L$. 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 8 above, in the intended model of $\mathcal{P}$, the truth of A is "approximated" with $[0.3; 1]$, i.e. the truth of A lies in between 0.3 and 1 (similarly for B).

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 [21, 27]). 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 [27]. 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 $b_2 \preceq b_1$.

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:

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

- $\mathtt{f} = [\bot; \bot]$ (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

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 9 *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 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 holds.

Theorem 1 *Consider $L \times L$ with the orderings $\preceq_t$ and $\preceq_k$.*

1. *the combination functions $\wedge, \vee, \otimes, \oplus$ are continuous (and, thus, monotonic) w.r.t. $\preceq_t$ and $\preceq_k$;*
2. *any negation function is monotonic w.r.t. $\preceq_k$;*
3. *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. $\preceq_k$.*

PROOF. We proof only the last item, as the others are immediate. Consider a chain of intervals $x_0 \preceq_k x_1 \preceq_k \ldots$, where $x_j = [a_j; b_j]$ with $a_j, b_j \in L$. To show the continuity of the extended negation function w.r.t. $\preceq_k$, we show that $\neg \oplus_{j \geq 0} x_j = \oplus_{j\geq 0} \neg x_j$. Indeed, the following holds:

$$\begin{array}{rcl}
\neg \oplus_{j\geq 0} x_j & = & \neg[\vee_{j\geq 0} a_j; \wedge_{j\geq 0} b_j] \\
 & = & [\neg \wedge_{j\geq 0} b_j; \neg \vee_{j\geq 0} a_j] \\
 & = & [\vee_{j\geq 0} \neg b_j; \wedge_{j\geq 0} \neg a_j] \\
 & = & \oplus_{j\geq 0}[\neg b_j; \neg a_j] \\
 & = & \oplus_{j\geq 0}\neg[a_j; b_j] \\
 & = & \oplus_{j\geq 0}\neg x_j
\end{array}$$

□

We now define the notion of approximate interpretations.

Definition 1 (Approximate interpretation) *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.

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

At first, we extend the two orderings on $L \times L$ to the set of approximate interpretations $\mathcal{C}_P$ in a 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 $\alpha \in L$, $I(\alpha) = [\alpha; \alpha]$.

At second, we 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 by $I \models \mathcal{P}$, 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. [22]).

At 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 *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 $T_{\mathcal{P}}$.*

PROOF. $I \models \mathcal{P}$ iff for all $A \leftarrow \varphi \in \mathcal{P}^*$, $I(A) = I(\varphi) = T_{\mathcal{P}}(I)(A)$ and, thus, $I \models \mathcal{P}$ iff $I = T_{\mathcal{P}}(I)$. □

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. $\preceq_k$.*

PROOF. The proof of monotonicity is straightforward. To prove the continuity w.r.t. $\preceq_k$, consider a chain of interpretations $I_0 \preceq_k I_1 \preceq_k \ldots$. We show that for any $A \in B_P$,

$$T_{\mathcal{P}}(\oplus_{j\geq 0} I_j)(A) = \oplus_{j\geq 0} T_{\mathcal{P}}(I_j)(A) \; . \tag{1}$$

As $\mathcal{C}_P$ is a complete lattice, the sequence $I_0 \preceq_k I_1 \preceq_k \ldots$ has a least upper bound, say $\bar{I} = \oplus_{j\geq 0} I_j$. For any $B \in B_P$, we have $\oplus_{j\geq 0} I_j(B) = \bar{I}(B)$ (as $\oplus$ is continuous) and, from Theorem 1, $\oplus_{j\geq 0} I_j(\neg B) = \oplus_{j\geq 0} \neg I_j(B) = \neg \oplus_{j\geq 0} I_j(B)$ $= \neg \bar{I}(B)$ and, thus, for any atom (and similarly for any for truth value) A,

$$\oplus_{j\geq 0} I_j(A) = \bar{I}(A) \; . \tag{2}$$

Now, consider $A \leftarrow f(B1, \ldots, B_n) \in \mathcal{P}^*$. Let us evaluate the left hand side of Equation 1.

$$\begin{array}{rcl} T_{\mathcal{P}}(\oplus_{j\geq 0} I_j)(A) & = & T_{\mathcal{P}}(\bar{I})(A) \\ & = & f(\bar{I}(B_1), \ldots, \bar{I}(B_n)). \end{array} \tag{3}$$

On the other hand side,

$$\oplus_{j \geq 0} T_{\mathcal{P}}(I_j)(A) = \oplus_{j \geq 0} f(I_j(B_1), \ldots, I_j(B_n)) \ .$$

But, f is continuous w.r.t. $\preceq_k$, and, thus, by Equation 2 and by Equation 3,

$$\begin{array}{rcl} \oplus_{j \geq 0} T_{\mathcal{P}}(I_j)(A) & = & \oplus_{j \geq 0} f(I_j(B_1), \ldots, I_j(B_n)) \\ & = & f(\oplus_{j \geq 0} I_j(B_1), \ldots, \oplus_{j \geq 0} I_j(B_n)) \\ & = & f(\bar{I}(B_1), \ldots, \bar{I}(B_n)) \\ & = & T_{\mathcal{P}}(\oplus_{j \geq 0} I_j)(A) \ . \end{array}$$

Therefore, Equation 1 holds and, thus, $T_{\mathcal{P}}$ is continuous. □

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 [4], 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 [23].

4 Intended semantics of normal logic programs

Approximate Kripke-Kleene Model. 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 $\preceq_k$.

Note that this least model with respect to $\preceq_k$ corresponds to an extension of the classical Kripke-Kleene semantics [23] of Datalog programs with negation to normal logic programs: for any Datalog program with negation $\mathcal{P}$, the least fixed-point of $T_{\mathcal{P}}$ w.r.t. $\preceq_k$ is a model of $\mathcal{P}$ that coincides with the Kripke-Kleene model of $\mathcal{P}$ [23].

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 10 *The following sequence of interpretations I_0, I_1, I_2 shows how the approximate Kripke-Kleene model of the running Example 8 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 $I_i \preceq_k I_{i+1}$):*

$$\begin{aligned}
I_0 &= \{A\colon[0;1], B\colon[0;1], C\colon[0;1], D\colon[0;1]\},\\
I_1 &= \{A\colon[0;1], B\colon[0;1], C\colon[0.3;0.6], D\colon[0;1]\},\\
I_2 &= \{A\colon[0.3;1], B\colon[0;1], C\colon[0.3;0.6], D\colon[0;1]\},\\
I_3 &= I_2\\
&= KK_{\mathcal{P}}\ .
\end{aligned}$$

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 B and D lie between 0 and 1, i.e. are unknown, the truth value of A is greater than 0.3 and the truth value of C 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) [62] 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 Closed World Assumption as a Source of falsehood. The main topic we address here is to define the notion of *support*, introduced in [41], 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 $[\bot;\bot]$, 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 *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. $J \preceq_k \mathtt{I}_{\mathtt{f}}$, 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 *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 $J \preceq_k \mathtt{I}_{\mathtt{f}}$ and $J \preceq_k T_{\mathcal{P}}(I \oplus J)$.*

It is easy to verify (see [41]) that

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

The following theorem, which can be shown as in [41], provides an algorithm for computing the support.

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*

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

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 11 *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 the running Example 8 ($I = KK_{\mathcal{P}}$, $J_0 = I_{\mathtt{f}}$ and $J_{n+1} = F_{P,I}(J_n)$):*

$$\begin{aligned} J_0 &= \{A\colon[0;0], B\colon[0;0], C\colon[0;0], D\colon[0;0]\}, \\ J_1 &= \{A\colon[0;1], B\colon[0;0.7], C\colon[0;0.3], D\colon[0;0]\}, \\ J_2 &= J_1 \\ &= s_{\mathcal{P}}(KK_{\mathcal{P}}) \end{aligned}$$

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

Classical setting. A well-known way for extracting falsehood using the CWA was defined in the classical setting through the notion of *unfounded set* [79]. 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 [41]), in the classical setting:

Theorem 5 ([41]) *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 propose to 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 *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

[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 $\neg.U \subseteq T_{\mathcal{P}}(I \cup \neg.U)$.

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 [39] it is shown that the well-founded model is the least model satisfying $\neg.U_{\mathcal{P}}(I) \subseteq I$.

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 *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

$$AW_{\mathcal{P}}(I) = T_{\mathcal{P}}(I) \oplus s_{\mathcal{P}}(I) \ . \tag{4}$$

The following theorems can be shown, as a in [41].

Theorem 6 *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.*

PROOF. $\Rightarrow$.) Assume $I = AW_{\mathcal{P}}(I)$. From the safeness of $s_{\mathcal{P}}(I)$, it follows that $s_{\mathcal{P}}(I) \preceq_k T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I)) = AW_{\mathcal{P}}(I) = I$. Therefore, $I = T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I)) = T_{\mathcal{P}}(I)$. By Theorem 2, I is a model of $\mathcal{P}$ and, thus by definition I is a model of $\mathcal{P}$ supported by the CWA.

$\Leftarrow$.) Assume $I \models P$ and $s_{\mathcal{P}}(I) \preceq_k I$. Then, using Theorem 2, $I = T_{\mathcal{P}}(I) = T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I)) = AW_{\mathcal{P}}(I)$. $\Box$

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 [77], 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 [41], the way our definition of approximate well-founded semantics generalizes the classical setting (using Equation 4) 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.	$\neg.U_{\mathcal{P}}(I) \subseteq I$	$\mathbf{s}_{\mathcal{P}}(\mathbf{I}) \preceq_{\mathbf{k}} \mathbf{I}$

Our result indicates that 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 12 *The following sequence of interpretations shows the computation of $W_{\mathcal{P}}$ of Example 8 ($I_0 = I_\bot$ and $I_{n+1} = AW_{\mathcal{P}}(I_n)$).*

$$\begin{array}{lcl}
I_0 & = & \{A\colon[0;1], B\colon[0;1], C\colon[0;1], D\colon[0;1]\} \\
s_{\mathcal{P}}(I_0) & = & \{A\colon[0;1], B\colon[0;1], C\colon[0;0.3], D\colon[0;0]\} \\
 & & \\
I_1 & = & \{A\colon[0;1], B\colon[0;1], C\colon[0.3;0.3], D\colon[0;0]\} \\
s_{\mathcal{P}}(I_1) & = & \{A\colon[0;1], B\colon[0;1], C\colon[0;0.3], D\colon[0;0]\} \\
 & & \\
I_2 & = & \{A\colon[0.3;1], B\colon[0;1], C\colon[0.3;0.3], D\colon[0;0]\} \\
s_{\mathcal{P}}(I_2) & = & \{A\colon[0;1], B\colon[0;0.7], C\colon[0;0.3], D\colon[0;0]\} \\
 & & \\
\mathbf{I_3} & = & \{\mathbf{A}\colon[\mathbf{0.3};\mathbf{1}], \mathbf{B}\colon[\mathbf{0};\mathbf{0.7}], \mathbf{C}\colon[\mathbf{0.3};\mathbf{0.3}], \mathbf{D}\colon[\mathbf{0};\mathbf{0}]\} \\
s_{\mathcal{P}}(I_3) & = & \{A\colon[0;1], B\colon[0;0.7], C\colon[0;0.3], D\colon[0;0]\} \\
 & & \\
I_4 & = & I_3 \\
 & = & W_{\mathcal{P}}
\end{array}$$

The truth of C and D are respectively 0.3 and 0, while the truth of A and B 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 10)*

$$KK_{\mathcal{P}} \;=\; \{A\colon[0.3;1], B\colon[0;1], C\colon[0.3;0.6], D\colon[0;1]\}\ .$$

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 11 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. □

Example 13 *Consider the program $P = R \cup F$ given in Example 7. The computation of the approximate well-founded semantics $W_{\mathcal{P}}$ of $\mathcal{P}$ gives the following result*[5]*:*

$$\begin{array}{lll} W_{\mathcal{P}} & = \{ & \texttt{Risk}(\texttt{John})\colon[0.64;0.7], \\ & & \texttt{Sport\_car}(\texttt{John})\colon[0.8;0.8], \\ & & \texttt{Young}(\texttt{John})\colon[0;0], \\ & & \texttt{Good\_driver}(\texttt{John})\colon[0.3;0.36], \\ & & \texttt{Experience}(\texttt{John})\colon[0.7;0.7]\ \}\ , \end{array}$$

which establishes that `John`*'s degree of* `Risk` *is in between* $[0.64, 0.7]$. □

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 to all atoms.*

5 Top-down query answering

The objective of this section is to provide a top-down procedure to answer queries. A *query*, denoted q, is an expression of the form $?A$ (*query atom*), intended as a question about the truth of the atom A in the selected intended model of $\mathcal{P}$. We also

[5]For ease of presentation, we use the first letter of predicates and constants only.

allow a query to be a *set* $\{?A_1, \ldots, ?A_n\}$ of query atoms. In that latter case we ask about the truth of all the atoms A_i in the intended model of a logic program $\mathcal{P}$. The intended model is either the approximate Kripke-Kleene model or the approximate well-founded model.

Given a logic program $\mathcal{P}$, one way to answer to a query $?A$ is to compute the intended model I of $\mathcal{P}$ by a bottom-up fixed-point computation and then answer with $I(A)$. This always requires to compute a whole model, even if in order to determine $I(A)$, not all the atom's truth is required. Our goal is to present a simple, yet general top-down method, which relies on the computation of just a part of an intended model. Essentially, we will try to determine the value of a single atom by investigating only a part of the program $\mathcal{P}$. Our method is based on a transformation of a program into a system of equations of monotonic functions over lattices for which we compute the least fixed-point in a top-down style.

For it we take inspiration on the method [74] and customized it to our case. The main difference to it is that [74] relies on Fitting's [21, 22] formulation based on a generalization of the Gelfond-Lifschitz formulation of stable models [26], while here we have to deal with the equivalent formulation based on the notion of support (in classical terms, the generalization of the notion of unfounded set, see, [41]).

We assume the lattices 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 –see Example17).

The idea is the following. Let $\mathcal{L} = \langle L, \preceq \rangle$ be a complete lattice and let $\langle L \times L, \preceq_t, \preceq_k \rangle$ be the interval bilattice derived from it. Let $\mathcal{P}$ be a logic program. Consider the Herbrand base $B_{\mathcal{P}} = \{A_1, \ldots, A_n\}$ of $\mathcal{P}$ and consider $\mathcal{P}^*$. Let us associate to each atom $A_i \in B_{\mathcal{P}}$ a variable x_i, which will take a value in the domain $L \times L$ (sometimes, we will refer to that variable with x_A as well). An interpretation I may be seen as an assignment of intervals to the variables $x_1, ..., x_n$. For an immediate consequence operator O, e.g. $T_{\mathcal{P}}$, a fixed-point is such that $I = O(I)$, i.e. for all atoms $A_i \in B_P$, $I(A_i) = O(I)(A_i)$. Therefore, we may identify the fixed-points of O as the solutions over $L \times L$ of the system of equations of the following form:

$$\begin{array}{rcl} x_1 & = & f_1(x_{1_1}, \ldots, x_{1_{a_1}}) \; , \\ & \vdots & \\ x_n & = & f_n(x_{n_1}, \ldots, x_{n_{a_n}}) \; , \end{array} \tag{5}$$

where for $1 \leq i \leq n$, $1 \leq k \leq a_i$, we have $1 \leq i_k \leq n$. Each variable x_{i_k}

will take a value in the domain $L \times L$, each (monotone) function f_i determines the value of x_i (i.e. A_i) given an assignment $I(A_{i_k})$ to each of the a_i variables x_{i_k}. The function f_i implements $O(I)(A_i)$. Of course, we are especially interested in the computation of the least fixed-point of the above system.

Example 14 *Consider $\mathcal{L}_{[0,1]}$ and the bilattice of intervals build from it. Consider the following logic program:*

$$\begin{array}{rcl} \mathcal{P} & = & A \leftarrow A \vee B \\ & & B \leftarrow (\neg C \wedge A) \vee 0.3 \vee \neg D \\ & & C \leftarrow \neg B \vee 0.2 \vee \neg E \\ & & D \leftarrow 0.5 \\ & & E \leftarrow 0.6 \end{array}$$

For ease of exposition, we can use directly intervals in the logic program and, thus, write [6]

$$\begin{array}{rcl} \mathcal{P} & = & A \leftarrow A \vee B \\ & & B \leftarrow (\neg C \wedge A) \vee [0.3; 0.5] \\ & & C \leftarrow \neg B \vee [0.2; 0.4] \end{array}$$

This is harmless as the semantics is based on intervals. Then the corresponding equational system is of the form

$$\begin{array}{rcl} x_A & = & x_A \vee x_B \ , \\ x_B & = & (\neg x_C \wedge x_A) \vee [0.3; 0.5] \ , \\ x_C & = & \neg x_B \vee [0.2; 0.4] \ . \end{array}$$

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

$$KK_{\mathcal{P}} = \{A\colon [0.3; 1], B\colon [0.3; 0.8], C\colon [0.2; 0.7]\} \ ,$$

while the approximated well-founded model is

$$W_{\mathcal{P}} = \{A\colon [0.3; 0.5], B\colon [0.3; 0.5], C\colon [0.5; 0.7]\} \ .$$

Notice that $KK_{\mathcal{P}} \preceq_k WF_{\mathcal{P}}$, as expected. Also, both are fixed-points of the above equational system and $KK_{\mathcal{P}}$ is the $\preceq_k$-least fixed point.

[6] Note that $\neg D$ introduces the upper bound 0.5, while $\neg E$ introduces the upper bound 0.4.

In the following, at first we recall the general procedure for the top-down computation of the value of variable in the $\preceq$-least solution of the equational system (5), given a lattice $\mathcal{L} = \langle L, \preceq \rangle$ [74, 75]. Then, we will customize it for computing the approximate Kripke-Kleene semantics, the support and eventually the approximate well-founded semantics.

In the following we use some auxiliary functions: given the equational system (5),

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

In the general case, we assume that each function $f_i \colon L^{a_i} \mapsto L$ in Equation (5) is $\preceq$-monotone. We also use f_x in place of f_i, for $x = x_i$. We refer to the monotone system as in Equation (5) as the tuple $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $\mathcal{L}$ is a lattice, $V = \{x_1, ..., x_n\}$ are the variables and $\mathbf{f} = \langle f_1, ..., f_n \rangle$ is the tuple of functions.

As it is well known, a monotonic equation system as (5) has a $\preceq$-least solution, lfp$(\mathbf{f})$, the $\preceq$-least fixed-point of $\mathbf{f}$ is given as the least upper bound of the $\preceq$-monotone sequence, $\mathbf{y}_0, \ldots, \mathbf{y}_i, \ldots$, where

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

Example 15 *Consider Example 14. The $\preceq_k$-least fixed-point computation is (the triples represent $\langle x_A, x_B, x_C \rangle$),*

$$\begin{array}{lcl} \mathbf{y}_0 & = & \bot = \langle [0;1], [0;1], [0;1] \rangle \\ \mathbf{y}_1 & = & \langle [0;1], [0.3;1], [0.2,1] \rangle \\ \mathbf{y}_2 & = & \langle [0.3;1], [0.3;0.8], [0.2;0.7] \rangle \\ \mathbf{y}_3 & = & \mathbf{y}_2 \,, \end{array}$$

which corresponds to the approximate Kripke-Kleene model of the program, as expected.

Informally our top-down algorithm works as follows (see Table 1). Assume we are interested in the value of x_0 in the least fixed-point of the system. We associate to each variable x_i a marking $\mathtt{v}(x_i)$ denoting the current value of x_i (the mapping $\mathtt{v}$ contains the current value associated to the variables). Initially, $\mathtt{v}(x_i)$ is $\bot$. We start with putting x_0 in the *active* list of variables $\mathtt{A}$, for which we evaluate whether the current value of the variable is identical to whatever its right-hand side evaluates

```
    Procedure $Solve(\mathcal{S}, Q)$
    Input: $\preceq$-monotonic system $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $Q \subseteq V$ is the set of query variables;
    Output: A set $B \subseteq V$, with $Q \subseteq B$ such that the mapping v equals $\mathrm{lfp}_{\preceq}(f)$ on $B$.
1.      $\mathtt{A}{:}= Q$, $\mathtt{dg}{:}= Q$, $\mathtt{in}{:}= \emptyset$, for all $x \in V$ do $\mathtt{v}(x) = \bot$, $\mathtt{exp}(x) = \mathtt{false}$
2.      while $\mathtt{A} \neq \emptyset$ do
3.        select $x_i \in \mathtt{A}$, $\mathtt{A}{:}= \mathtt{A} \setminus \{x_i\}$, $\mathtt{dg}{:}= \mathtt{dg} \cup \mathtt{s}(x_i)$
4.        $r{:}= f_i(\mathtt{v}(x_{i_1}), ..., \mathtt{v}(x_{i_{a_i}}))$
5.        if $r \succ \mathtt{v}(x_i)$ then $\mathtt{v}(x_i){:}= r$, $\mathtt{A}{:}= \mathtt{A} \cup (\mathtt{p}(x_i) \cap \mathtt{dg})$ fi
6.        if not $\mathtt{exp}(x_i)$ then $\mathtt{exp}(x_i) = \mathtt{true}$, $\mathtt{A}{:}= \mathtt{A} \cup (\mathtt{s}(x_i) \setminus \mathtt{in})$, $\mathtt{in}{:}= \mathtt{in} \cup \mathtt{s}(x_i)$ fi
        od
```

Table 1: General top-down algorithm.

to. When evaluating a right-hand side it might of course turn out that we do indeed need a better value of some sons, which will assumed to have the value $\bot$ and put them on the list of active nodes to be examined. In doing so we keep track of the dependencies between variables, and whenever it turns out that a variable changes its value (actually, it can only $\preceq$-increase) all variables that might depend on this variable are put in the active set to be examined. At some point (even if cyclic definitions are present) the active list will become empty and we have actually found part of the fixed-point, sufficient to determine the value of the query x_0.

The additional data structures are used as follows:

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

The attentive reader will notice that the $Solve$ procedure has much in common with the so-called *tabulation* procedures, like [8, 10]. Indeed, it is a generalization of it to arbitrary monotone equational systems over lattices. The algorithm is given in Table 1.

Example 16 *Consider Example 14 and query variable x_A. Below is a sequence of $Solve(\mathcal{S}, \{x_A\})$ computation w.r.t. $\preceq_k$. Each line is a sequence of steps in the 'while loop'. What is left unchanged is not reported.*

1. $\mathtt{A}:=\{x_A\}, x_i:=x_A, \mathtt{A}:=\emptyset, \mathtt{dg}:=\{x_A,x_B\}, r:=\bot, \mathtt{exp}(x_A):=\mathtt{true}$, $\mathtt{A}:=\{x_A,x_B\}, \mathtt{in}:=\{x_A,x_B\}$

2. $x_i:=x_B, \mathtt{A}:=\{x_A\}, \mathtt{dg}:=\{x_A,x_B,x_C\}, r:=[0.3;1], \mathtt{v}(x_B):=[0.3;1]$, $\mathtt{A}:=\{x_A,x_C\}, \mathtt{exp}(x_B):=\mathtt{true}, \mathtt{in}:=\{x_A,x_B,x_C\}$

3. $x_i:=x_C, \mathtt{A}:=\{x_A\}, r:=[0.2;0.7], \mathtt{v}(x_C):=[0.2;0.7], \mathtt{A}:=\{x_A,x_B\}$, $\mathtt{exp}(x_C):=\mathtt{true}$

4. $x_i:=x_B, \mathtt{A}:=\{x_A\}, r:=[0.3;0.8], \mathtt{v}(x_B):=[0.3;0.8], \mathtt{A}:=\{x_A,x_C\}$

5. $x_i:=x_C, \mathtt{A}:=\{x_A\}, r:=[0.2;0.7]$

6. $x_i:=x_A, \mathtt{A}:=\emptyset, r:=[0.3;1], \mathtt{v}(x_A):=[0.3;1], \mathtt{A}:=\{x_A,x_B\}$

7. $x_i:=x_B, \mathtt{A}:=\{x_A\}, r:=[0.3;0.8]$,

8. $x_i:=x_A, \mathtt{A}:=\emptyset, r:=[0.3;1]$

10. $\mathtt{stop.\ return}\ \mathtt{v}(x_A,x_B,x_C)=\langle[0.3;1],[0.3;0.8],[0.2;0.7]\rangle$

The fact that only a part of the model is computed becomes evident, as the computation does not change if we add any program $\mathcal{P}'$ to $\mathcal{P}$ in which A, B and C do not occur.

Given a system $\mathcal{S}=\langle\mathcal{L},V,\mathbf{f}\rangle$, where $\mathcal{L}=\langle L,\preceq\rangle$, let $h(\mathcal{L})$ be the *height* of the truth-value set L, i.e. the length of the longest strictly $\preceq$-increasing chain in L minus 1, where the length of a chain $v_1,...,v_\alpha,...$ is the cardinal $|\{v_1,...,v_\alpha,...\}|$. The *cardinal* of a set X is the least ordinal α such that α and X are *equipollent*, i.e. there is a bijection from α to X. For instance, $|\{0,1\}|=2$, while in general $|[0,1]\cap\mathbb{Q}|=\omega$ and $|([0,1]\cap\mathbb{Q})\times([0,1]\cap\mathbb{Q}|)=\omega$. However, as stated at the beginning of the section, the lattice is always finite and, thus, the height is always finite.

In [74] it is shown that the algorithm $Solve(\mathcal{S},Q)$ behaves correctly. Also we recall that from a computational point of view, by means of appropriate data structures, the operations on `A`, `v`, `dg`, `in`, `exp`, `p` and `s` can be performed in constant time. Therefore, Step 1. is $O(|V|)$, all other steps, except Step 2. and Step 4. are $O(1)$. Let $c(f_x)$ be the maximal cost of evaluating function f_x on its arguments, so Step 4. is $O(c(f_x))$. It remains to determine the number of loops of Step 2. As the height $h(\mathcal{L})$ of $\mathcal{L}$ is finite, observe that any variable is increasing in the $\preceq$ order as it enters in the `A` list (Step 5.), except it enters due to Step 6., which may happen one time only. Therefore, each variable x_i will appear in `A` at most $a_i\cdot h(\mathcal{L})+1$ times, where a_i is the arity of f_i, as a variable is only re-entered into `A` if one of its son gets an increased value (which for each son only can happen $h(\mathcal{L})$ times), plus the additional entry due to Step 6. As a consequence, the worst-case complexity is $O(\sum_{x_i \in V}(c(f_i) \cdot (a_i \cdot h(\mathcal{L}) + 1))$. Therefore:

Theorem 9 ([74]) *Consider a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$.*

1. *There is a finite limit ordinal λ such that after $|\lambda|$ steps $Solve(\mathcal{S}, Q)$ determines a set $B \subseteq V$, with $Q \subseteq B$ such that the mapping $\mathtt{v}$ equals lfp$_{\preceq}(\mathbf{f})$ on B, i.e. $\mathtt{v}_{|B} =$ lfp$_{\preceq}(\mathbf{f})_{|B}$.*
2. *If the computing cost of each function in $\mathbf{f}$ is bounded by c, the arity bounded by a, and the height is bounded by h, then the worst-case complexity of the algorithm $Solve$ is $O(|V|cah)$.*

Note that in case the height of a lattice is not finite, the computation may not terminate after a finite number of steps as the following example shows.

Example 17 ([32]) *Consider $\mathcal{L}_{[0,1]}$, the functions ($0 < a \leq 1, a \in \mathbb{Q}$)*

$$f(x) = \frac{x + a}{2}$$

$$g(x) = \begin{cases} 1 & \textit{if } x \geq a \\ 0 & \textit{if } x < a \end{cases}$$

Consider the two logic programs

$$\mathcal{P}_1 = \{A \leftarrow f(A)\}$$

$$\begin{array}{lll} \mathcal{P}_2 & = \{ & A \leftarrow f(A) \\ & & B \leftarrow g(A) \quad \} \end{array}$$

Then the approximated Kripke-Kleene model of $\mathcal{P}_1$ is attained after ω steps of $T_{\mathcal{P}}$ iterations over $\mathtt{I}_{\perp} = [0;1]$ and is such that $KK_{\mathcal{P}}(A) = [a;1]$, while ($g$ is not continuous) the approximated Kripke-Kleene model of $\mathcal{P}_2$ is attained after $\omega + 1$ steps of $T_{\mathcal{P}}$ iterations ($KK_{\mathcal{P}}(A) = [a;1]$, $KK_{\mathcal{P}}(B) = [1;1]$). However, under the assumption of a fixed finite precision p to represent rationals numbers in $[0,1]$, the computation converges after a finite number of steps for both $\mathcal{P}_1$ and $\mathcal{P}_2$.

5.1 Query answering: approximate Kripke-Kleene semantics

We start with the Kripke-Kleene semantics, for which we have almost anticipated how we will proceed. Let $\mathcal{P}$ be a logic program and consider it's grounding $\mathcal{P}^*$. As already pointed out, each atom appears exactly once in the head of a rule in $\mathcal{P}^*$. The system of equations that we build from $\mathcal{P}^*$ is straightforward. Assign to each atom A a variable x_A and substitute in $\mathcal{P}^*$ each occurrence of A with x_A. Finally, substitute each occurrence of $\leftarrow$ with $=$ and let $\mathcal{S}_{KK}(\mathcal{P}) = \langle \mathcal{L}, V, \mathbf{f}_{\mathcal{P}} \rangle$ be the resulting equational system. Of course, $|V| = |B_{\mathcal{P}}|$, $|\mathcal{S}_{KK}(\mathcal{P})|$ can be computed in time $O(|\mathcal{P}|)$ and all functions in $\mathcal{S}_{KK}(\mathcal{P})$ are $\preceq_k$-monotone. As $\mathbf{f}_{\mathcal{P}}$ is one to one related to $T_{\mathcal{P}}$, it follows that the $\preceq_k$-least fixed-point of $\mathcal{S}_{KK}(\mathcal{P})$ corresponds to the approximated Kripke-Kleene semantics of $\mathcal{P}$. The algorithm $Solve_{KK}(\mathcal{P}, ?A)$, first builds the equational system $\mathcal{S}_{KK}(\mathcal{P})$ and then calls $Solve(\mathcal{S}_{KK}(\mathcal{P}), \{x_A\})$ and returns the output v on the query variable, where v is the output of the call to $Solve$. $Solve_{KK}$ behaves correctly (see Example 16).

Theorem 10 *Let $\mathcal{P}$ and $?A$ be a logic program and a query, respectively. Then $KK_{\mathcal{P}}(A) = Solve_{KK}(\mathcal{P}, \{?A\})$.*

The extension of Theorem 10 to a set of query atoms is straightforward.

From a computational point of view, we can avoid the cost of translating $\mathcal{P}$ into $\mathcal{S}_{KK}(\mathcal{P})$ as we can directly operate on $\mathcal{P}$. So the cost $O(|\mathcal{P}|)$ can be avoided. As from our assumption the lattice is finite, by Theorem 9 it follows immediately that the worst-case complexity for top-down query answering under the approximate Kripke-Kleene semantics of a logic program $\mathcal{P}$ is $O(|B_{\mathcal{P}}|cah)$. Furthermore, often the cost of computing each of the functions of $\mathbf{f}_{\mathcal{P}}$ is in $O(1)$. By observing that $|B_{\mathcal{P}}|a$ is in $O(|\mathcal{P}|)$ we immediately have that in this case the complexity is $O(|\mathcal{P}|h)$. If the height is a fixed parameter, i.e. a constant, we can conclude that the additional expressive power of approximate Kripke-Kleene semantics of logic programs over lattices (with functions with constant cost) does not increase the computational complexity of classical propositional logic programs, which is linear.

5.2 Query answering: approximate well-founded semantics

We address now the issue of a top-down computation of the value of a query under the approximate well-founded semantics.

As we have seen in Section 4, 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 11, $s_{\mathcal{P}}(I)$ 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 sequence

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

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. That is, we want a top-down procedure answering that for a give query atom $?A$ answers with $s_{\mathcal{P}}(A)$, i.e. the truth of A in the support of $\mathcal{P}$ w.r.t. I. To this purpose, it suffices to build an equational system whose least fixed-point is the support and then apply the top-down query answering procedure described in Table 1.

At first, note that the sequence J_i is $\preceq_t$-increasing (see [41]). Therefore, $s_{\mathcal{P}}(I)$ is the $\preceq_t$-least fixed-point of the function $F_{P,I}(\cdot)$. Therefore, $s_{\mathcal{P}}(I) = F_{P,I}(s_{\mathcal{P}}(I))$. Now, consider $A \leftarrow f(B_1, \ldots, B_n) \in \mathcal{P}^*$. Let us introduce variables $x_A, x_{B_1}, ..., x_{B_n}$. The intended meaning of a variable is that of denoting the value of the atom in the support, e.g. x_A will hold the value $s_{\mathcal{P}}(I)(A)$. Given $A \leftarrow f(B_1, \ldots, B_n) \in \mathcal{P}^*$ we consider the equation

$$x_A = \mathtt{f} \otimes [f(I(B_1), ..., I(B_n)) \oplus f(x_{B_1}, ..., x_{B_n}))] \; . \tag{6}$$

The above equation is the result of applying $F_{P,I}(J_i)$ to all rules using the fact that

$$\begin{array}{rcl} J_{i+1}(A) & = & \mathtt{f} \otimes (I \oplus J_i)(f(B_1, \ldots, B_n)) \\ & = & \mathtt{f} \otimes [I(f(B_1, \ldots, B_n)) \oplus J_i(f(B_1, \ldots, B_n))] \\ & = & \mathtt{f} \otimes [f(I(B_1), \ldots, I(B_n)) \oplus f(J_i(B_1), \ldots, J_i(B_n))] \end{array}$$

and then replace $J_i(B_j)$ with the variable x_{B_j} and $J_{i+1}(A)$ with the variable x_A, as at the limit J_i will be the support.

Example 18 *Consider Example 14 and an interpretation I. Then the corresponding equational system for computing the support is*

$$\begin{array}{rcl} x_A & = & \mathtt{f} \otimes [I(A \vee B) \oplus (x_A \vee x_B)] \; , \\ x_B & = & \mathtt{f} \otimes [I((\neg C \wedge A) \vee [0.3; 0.5]) \oplus ((\neg x_C \wedge x_A) \vee [0.3; 0.5])] \; , \\ x_C & = & \mathtt{f} \otimes [I(\neg B \vee [0.2; 0.4]) \oplus (\neg x_B \vee [0.3; 0.5])] \; . \end{array}$$

Let us show that, indeed, a bottom-up computation of the least fixed-point of the above equational systems is the support, which corresponds to the computation of $F_{P,I}(\cdot)$ starting with $\mathtt{I_f}$. To the ease of presentation, we consider $I = \mathtt{I}_\perp$. The $\preceq_t$-least fixed-point computation is (the triples represent x_A, x_B and x_C),

$$\begin{array}{rcl} J_0 & = & \perp = \langle [0;1], [0;1], [0;1] \rangle \\ J_1 & = & \langle [0;0], [0;0.5], [0,1] \rangle \\ J_2 & = & \langle [0;0.5], [0;0.5], [0;1] \rangle \\ J_3 & = & \langle [0;0.5], [0;0.5], [0;1] \rangle \\ J_4 & = & J_3\ . \end{array}$$

J_3 is indeed the support of $\mathcal{P}$ w.r.t. $\mathtt{I_f}$.

It can then easily be shown that:

Theorem 11 *For any program $\mathcal{P}$ and interpretation I, the support $s_\mathcal{P}(I)$ of $\mathcal{P}$ w.r.t. I is the $\preceq_t$-least fixed-point of the equational system obtained by replacing each rule $A \leftarrow f(B_1, \ldots, B_n) \in \mathcal{P}^*$ with the Equation 6.*

It follows then immediately that we have a top-down procedure to compute the truth of an atom in the support $s_\mathcal{P}(I)$. We denote the equational system by using Equation 6 above as $Supp^I_\mathcal{P}$. Then it follows that:

Theorem 12 *For a set of query variables Q, $Solve(Supp^I_\mathcal{P}, Q)$ outputs a set $B \subseteq V$, with $Q \subseteq B$, such that the mapping $\mathtt{v}$ equals to the $\preceq_t$-least fixed-point, i.e. the support $s_\mathcal{P}(I)$ on B: $\mathtt{v}_{|B} = s_\mathcal{P}(I)_{|B}$.*

As a side product we obtain a top-down algorithm for the computation of the well-founded set.

From a computational complexity point of view, the same properties of $Solve$ hold for $Solve(Supp^I_\mathcal{P}, Q)$ as well.

We are now ready to define the top-down procedure, $Solve_{WF}(\mathcal{P}, ?A)$, to compute the answer to an atom A under the approximate well-founded semantics. We define $Solve_{WF}(\mathcal{P}, ?A)$ as $Solve_{KK}(\mathcal{P}, ?A)$, except that Step 4. is replaced with the statements

```
4.1.   $\mathtt{S} := \mathtt{s}(x_i)$;
4.2.   $\mathtt{I} := \mathtt{v}$;
4.3.   $\mathtt{v}' := Solve(Supp^I_\mathcal{P}, \mathtt{S})$;
4.4.   $r := f_i(\mathtt{v}(x_{i_1}) \oplus \mathtt{v}'(x_{i_1}), ..., \mathtt{v}(x_{i_{a_i}}) \oplus \mathtt{v}'(x_{i_{a_i}}))$
```

These steps correspond to one step application of the $AW_{\mathcal{P}}(I) = T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I))$ operator to the variable x_i. Indeed, we have that

$$x_i = f_i(x_{i_1}, ..., x_{i_{a_i}})$$

is the definition of x_i in the equational system. Then, at first we ask about the value of the variables $x_{i_1}, ..., x_{i_{a_i}}$ in the support w.r.t. the current interpretation $\mathtt{I} := \mathtt{v}$ (Steps 4.1. - 4.3). The variable $\mathtt{v}'$ holds these values. Finally, we evaluate $T_{\mathcal{P}}(I \oplus s_{\mathcal{P}}(I))(x_i) = f_i(\mathtt{v}(x_{i_1}) \oplus \mathtt{v}'(x_{i_1}), ..., \mathtt{v}(x_{i_{a_i}}) \oplus \mathtt{v}'(x_{i_{a_i}}))$.

It follows easily then that:

Theorem 13 *Let $\mathcal{P}$ and $?A$ be a logic program and a query, respectively. Then $W_{\mathcal{P}}(A) = Solve_{WF}(\mathcal{P}, ?A)$.*

Example 19 *Consider Example 14 and query variable x_A. Below is a sequence of $Solve_{WW}(\mathcal{P}, ?A)$ computation. It resembles the one we have seen in Example 16. Each line is a sequence of steps in the 'while loop'. What is left unchanged is not reported.*

1. $\mathtt{A} := \{x_A\}, x_i := x_A, \mathtt{A} := \emptyset, \mathtt{dg} := \{x_A, x_B\}, \mathtt{Q} := \{x_A, x_B\}, \mathtt{v}' := \langle [0; 0.5], [0; 0.5], [0; 1]\rangle, r := [0; 0.5], \mathtt{v}(x_A) := [0; 0.5], \mathtt{A} := \{x_A, x_B\}, \mathtt{exp}(x_A) := \mathtt{true}, \mathtt{in} := \{x_A, x_B\}$

2. $x_i := x_B, \mathtt{A} := \{x_A\}, \mathtt{dg} := \{x_A, x_B, x_C\}, \mathtt{Q} := \{x_A, x_C\}, \mathtt{v}' := \langle [0; 0.5], [0; 0.5], [0; 1]\rangle, r := [0.3; 0.5], \mathtt{v}(x_B) := [0.3; 0.5], \mathtt{A} := \{x_A, x_C\}, \mathtt{exp}(x_B) := \mathtt{true}, \mathtt{A} := \{x_A, x_C\}, \mathtt{in} := \{x_A, x_B, x_C\}$

3. $x_i := x_C, \mathtt{A} := \{x_A\}, \mathtt{Q} := \{x_B\}, \mathtt{v}' := \langle [0; 0.5], [0; 0.5], [0; 1]\rangle, r := [0.5; 0.7], \mathtt{v}(x_C) := [0.5; 0.7], \mathtt{A} := \{x_A, x_B\}, \mathtt{exp}(x_C) := \mathtt{true}$

4. $x_i := x_B, \mathtt{A} := \{x_A\}, \mathtt{Q} := \{x_A, x_C\}, \mathtt{v}' := \langle [0; 0.5], [0; 0.5], [0; 0.7]\rangle, r := [0.3; 0.5]$

5. $x_i := x_A, \mathtt{A} := \emptyset, \mathtt{Q} := \{x_A, x_B\}, \mathtt{v}' := \langle [0; 0.5], [0; 0.5], [0; 0.7]\rangle, r := [0.3; 0.5], \mathtt{v}(x_A) := [0.3; 0.5], \mathtt{A} := \{x_A, x_B\}$

6. $x_i := x_A, \mathtt{A} := \{x_B\}, \mathtt{Q} := \{x_A, x_B\}, \mathtt{v}' := \langle [0; 0.5], [0; 0.5], [0; 0.7]\rangle, r := [0.3; 0.5]$

7. $x_i := x_B, \mathtt{A} := \emptyset, \mathtt{Q} := \{x_A, x_C\}, \mathtt{v}' := \langle [0; 0.5], [0; 0.5], [0; 0.7]\rangle, r := [0.3; 0.5]$

8. $\mathtt{stop.\ return}\ \mathtt{v}(x_A, x_B, x_C)_{|x_A} = \langle [0; 0.5], [0; 0.5], [0; 0.7]\rangle_{|x_A} = [0.3; 0.5]$

Note that the answer to $?A$, namely $[0.3; 0.5]$, is now more precise than the one ($[0.3; 1]$) under the approximate Kripke-Kleene model (See Example 16), as expected.

The computational complexity analysis of $Solve_{WF}$ parallels the one we have made for $Solve_{KK}$. As the height of a lattice is finite then, like $Solve_{KK}$, each variable x_j will appear in A at most $a_j \cdot (h(\mathcal{L}) + 1)$ times and, thus, the worst-case complexity is $O(\sum_{x_j \in V}(c(f_j) \cdot (a_j \cdot (h(\mathcal{L}) + 1))$. But now, the cost of $c(f_j)$ is the cost of a recursive call to $Solve$, which is $O(|B_{\mathcal{P}}|cah)$. Therefore, $Solve_{WF}$ runs in time $O(|B_{\mathcal{P}}|^2 a^2 h^2 c)$. That is, $Solve_{WF}$ runs in time $O(|\mathcal{P}|^2 h^2 c)$. If the lattice is fixed, then the height parameter is a constant. Furthermore, often we can assume that c is $O(1)$ and, thus, the worst-case complexity reduces to $O(|\mathcal{P}|^2)$.

6 Conclusions and future work

We have presented a general framework to deal with normal logic programs evaluated over complete lattices. Main features of our extension are: (i) our framework covers all many-valued frameworks we are aware of dealing with imprecision in normal logic programming; (ii) as we deal with non-monotone negation, atoms are assigned with truth interval approximations; (iii) the CWA is used to complete the knowledge to infer the most precise approximations as possible; (iv) the continuity of the immediate consequence operator is preserved in case the truth combination functions are continuous and lattices are distributive; and (v) we have presented a very general top-down method for answering queries, by a transformation of a normal logic program into an equational system over lattices.

The next step for future work is to extend our formalism to disjunctive logic programs with default negation were the head of a rule is a disjunction, or even more generally to rules of the form

$$f_1(A_1, \ldots, A_m) \leftarrow f_2(B_1, \ldots, B_m) \ .$$

It would be interesting to see whether our idea of using equational systems over lattices can be extended to this general form (or at least to disjunctive logic programs) as well.

References

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

[2] Teresa Alsinet and Lluis Godo Lluis Godo. A complete calcultis for possibilistic logic programming with fuzzy propositional variables with fuzzy

propositional variables. In *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI-00)*, pages 1–10. Morgan Kaufmann, 2000.

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

[4] Nuel D. Belnap. How a computer should think. In Gilbert Ryle, editor, *Contemporary aspects of philosophy*, pages 30–56. Oriel Press, Stocksfield, GB, 1977.

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

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

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

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

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

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

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

[12] Carlos Viegas Damásio and Luís Moniz Pereira. A survey of paraconsistent semantics for logic programs. In D. Gabbay and P. Smets, editors, *Handbook*

of Defeasible Reasoning and Uncertainty Management Systems, pages 241–320. Kluwer, 1998.

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

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

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

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

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

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

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

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

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

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

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

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

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

[26] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

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

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

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

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

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

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

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

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

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

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

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

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

[39] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation*, 135(2):69–112, 1997.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[62] Jack Minker. On indefinite data bases and the closed world assumption. In Springer-Verlag, editor, *Proc. of the 6th Conf. on Automated Deduction (CADE-82)*, number 138 in Lecture Notes in Computer Science, 1982.

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

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

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

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

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

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

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

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

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

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

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

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

[75] Umberto Straccia. Uncertainty management in logic programming: Simple and effective top-down query answering. In Rajiv Khosla, Robert J. Howlett,

and Lakhmi C. Jain, editors, *9th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems (KES-05), Part II*, number 3682 in Lecture Notes in Computer Science, pages 753–760, Melbourne, Australia, 2005. Springer Verlag.

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

[77] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, (5):285–309, 1955.

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

[79] Allen van Gelder, Kenneth A. Ross, and John S. Schlimpf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

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

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

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

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

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

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

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