Top-Down Query Answering for Logic Programs over Bilattices

Umberto Straccia

I.S.T.I. - C.N.R.,
Via G. Moruzzi,1 I-56124 Pisa (PI) ITALY
straccia@isti.cnr.it

Techincal Report

November 9, 2004

Abstract

Bilattices are generalizations of classical logics allowing reasoning with partial, incomplete, uncertain and/or inconsistent information and have interesting mathematical properties for both practical as well as theoretical investigations. In this paper we present a very simple, yet general, top-down query answering procedure under the Kripke-Kleene semantics as well as under the well-founded semantics for logic programs over bilattices.

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

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

Terms: Theory

Keywords: Logic programs, uncertainty, bilattices, top-down query answering

1 Introduction

The management of uncertainty 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 (e.g. probability theory [25, 34, 44, 45, 46, 47, 48, 49, 55, 56, 64], fuzzy set theory [8, 57, 60, 62, 63], multi-valued logic [12, 13, 14, 15, 16, 17, 29, 30, 32, 33, 35, 37, 38, 39, 40, 41, 42, 43, 50, 51, 52, 53, 54], possibilistic logic [19]) and how uncertainty values, associated to rules and facts, are managed.

Apart from the different notion of uncertainty they rely on, these frameworks differ in the way in which uncertainty is associated with the facts and rules of a program. With respect to this latter point, these frameworks can be classified into *annotation based* (AB) and *implication based* (IB), which we briefly summarize below. In the AB approach, a rule is of the form $A : f(\beta_1, \ldots, \beta_n) \leftarrow B_1 : \beta_1, \ldots, B_n : \beta_n$, which asserts "the certainty of atom A is at least (or is in) $f(\beta_1, \ldots, \beta_n)$, whenever the certainty 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 certainty domain. Examples of AB frameworks include [29, 30, 55, 56]. In the IB approach, a rule is of the form $A \stackrel{\alpha}{\leftarrow} B_1, ..., B_n$, which says that the certainty associated with the implication $B_1 \wedge ... \wedge B_n \rightarrow A$ is α. Computationally, given an assignment v of certainties to the B_is, the certainty of A is computed by taking the "conjunction" of the certainties $v(B_i)$ and then somehow "propagating" it to the rule head. The truth-values are taken from a certainty lattice. Examples of the IB frameworks include [34, 35, 54, 60] (see [12, 35] for a more detailed comparison between the two approaches). More recently, [12, 17, 35, 62] show that most of the frameworks can be embedded into the IB framework (some exceptions deal with probability theory). 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 negation operation is defined). The need of non-monotonic formalisms for real-world applications is commonly accepted: our knowledge about the world is almost always *incomplete* and, thus, we are forced to reason in the *absence of complete information*. Exception to this limitation are [48, 55] in which the stable semantics has been considered, but limited to the case where the underlying uncertainty formalism is probability theory. That semantics has been considered also in [63], where a semi-possibilistic logic has been proposed, a particular negation operator has been introduced and a fixed min/max-evaluation of conjunction and disjunction is adopted. In [16] the underlying truth-space are lattices, while [37, 38, 39] are based on *bilattices* [27], a slightly more general structure than lattices.

In this paper we will present a goal-oriented/top-down query procedure for logic programs over bilattices [20, 21, 22, 23]. Bilattices are obvious generalizations of three-valued truth to many-valued truth allowing reasoning with partial, incomplete, uncertain and/or inconsistent (notably, paraconsistent logic programming [1, 3, 6, 15]) information and having interesting mathematical properties for both practical as well as theoretical investigations [27] (see also [39], as an example of extension of logic programs over bilattices to the IB framework). The procedure we will present is quite general and inspired to [2], which presents a top-down procedure for computing the minimal fixed-points of a system of equations of monotonic functions over lattices. We adapt it and extend it to the query answering under the *well-founded semantics* [61] over bilattices [20, 21, 23]. We will also address the computational complexity issue. To the best of our knowledge the only work addressing the above issue are [13, 30, 35, 62], but no non-monotonicity is considered.

The structure of the paper is as follows. In order to make the paper self-contained, in the next section, we will briefly recall definitions and properties of bilattices and logic programs over bilattices. Section 3 is the main part of this work, where we present our top-down query procedure and the computational complexity analysis, while Section 4 concludes.

2 Preliminaries

We start with some well-known basic definitions and properties of lattices, bilattices and logic programs.

2.1 Lattices

A *lattice* is a partially ordered set $\mathcal{L} = \langle L, \preceq \rangle$ such that every two element set $\{x, y\} \subseteq L$ has a *least upper bound*, $lub_{\preceq}(x, y)$ (called the *join* of x and y), and a *greatest lower bound*, $glb_{\preceq}(x, y)$ (called the *meet* of x and y). For ease, we will write $x \prec y$ if $x \preceq y$ and $x \neq y$. A lattice $\langle L, \preceq \rangle$ is *complete* if every subset of L has both least upper and greatest lower bounds. Consequently, a complete lattice has a least element, $\bot$, and a greatest element $\top$. For ease, throughout the paper, given a complete lattice $\langle L, \preceq \rangle$ and a subset of elements $S \subseteq L$, with $\preceq$-*least* and $\preceq$-*greatest* we will always mean $glb_{\preceq}(S)$ and $lub_{\preceq}(S)$, respectively. With $\min_{\preceq}(S)$ we denote the set of minimal elements in S w.r.t. $\preceq$, i.e. $\min_{\preceq}(S) = \{x \in S \colon \nexists y \in S \; s.t. \; y \prec x\}$. Note that while $glb_{\preceq}(S)$ is unique, $|\min_{\preceq}(S)| > 1$ may hold. If $\min_{\preceq}(S)$ is a singleton $\{x\}$, for convenience we may also write $x = \min_{\preceq}(S)$ in place of $\{x\} = \min_{\preceq}(S)$. An *operator* on a lattice $\langle L, \preceq \rangle$ is a function from L to L, $f \colon L \to L$. An operator f on L is *monotone*, if for every pair of elements $x, y \in L$, $x \preceq y$ implies $f(x) \preceq f(y)$, while f is *antitone* if $x \preceq y$ implies $f(y) \preceq f(x)$. A *fixed-point* of f is an element $x \in L$ such that $f(x) = x$.

The basic tool for studying fixed-points of operators on lattices is the well-known Knaster-Tarski theorem [59].

Proposition 1 ([59]) *Let f be a monotone operator 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* $\preceq$-least *and a* $\preceq$-greatest *fixed-point. The* $\preceq$-least *(respectively,* $\preceq$-greatest*) fixed-point can be obtained by iterating f over $\bot$ (respectively, $\top$), i.e. is the limit of the non-decreasing (respectively, non-increasing) sequence $x_0, \ldots, x_i, x_{i+1}, \ldots, x_\lambda, \ldots$, where for a successor ordinal $i \geq 0$,*

$$\begin{aligned} x_0 &= \bot, \\ x_{i+1} &= f(x_i) \end{aligned}$$

(respectively, $x_0 = \top$), while for a limit ordinal λ,

$$x_\lambda = lub_{\preceq}\{x_i \colon i < \lambda\} \; (respectively, x_\lambda = glb_{\preceq}\{x_i \colon i < \lambda\}) \; . \tag{1}$$

We denote the $\preceq$-least and the $\preceq$-greatest fixed-point by $\mathrm{lfp}_{\preceq}(f)$ and $\mathrm{gfp}_{\preceq}(f)$, respectively.

Often, throughout the paper, we will define monotone operators, whose sets of fixed-points define certain classes of models of a logic program. As a consequence, please note that this will also mean that a least model *always* exists for such classes. Additionally, for ease, for the monotone operators defined in this study, we will specify the initial condition x_0 and the next iteration step x_{i+1} only, while Equation 1 is always considered as implicit.

2.2 Bilattices

We work in the well-studied context of bilattices [27]. The simplest non-trivial bilattice, called $\mathcal{FOUR}$, is due to Belnap [5], who introduced a logic intended to deal with incomplete and/or inconsistent information – see also [4, 15]. $\mathcal{FOUR}$ already illustrates many of the basic properties concerning bilattices. Essentially, $\mathcal{FOUR}$ extends the classical truth set $\{\mathtt{f}, \mathtt{t}\}$ to $\{\mathtt{f}, \mathtt{t}, \bot, \top\}$, where $\bot$ stands *unknown*, and $\top$ stands for *inconsistent*. $\mathcal{FOUR}$ has two quite intuitive and natural 'orthogonal' orderings, $\preceq_k$ and $\preceq_t$ (see Figure 1), each giving to $\mathcal{FOUR}$ the structure of a complete lattice. The two orders are the so-called *knowledge ordering* $\preceq_k$ and the *truth ordering* $\preceq_t$. If $x \preceq_k y$ then y represents 'more information' than x. On the other hand, if $x \preceq_t y$ then y represents 'more truth' than x. For instance, in $\mathcal{FOUR}$, $\bot \preceq_k \mathtt{f} \preceq_k \top$, $\bot \preceq_k \mathtt{t} \preceq_k \top$, $\mathtt{f} \preceq_t \bot \preceq_t \mathtt{t}$ and $\mathtt{f} \preceq_t \top \preceq_t \mathtt{t}$.

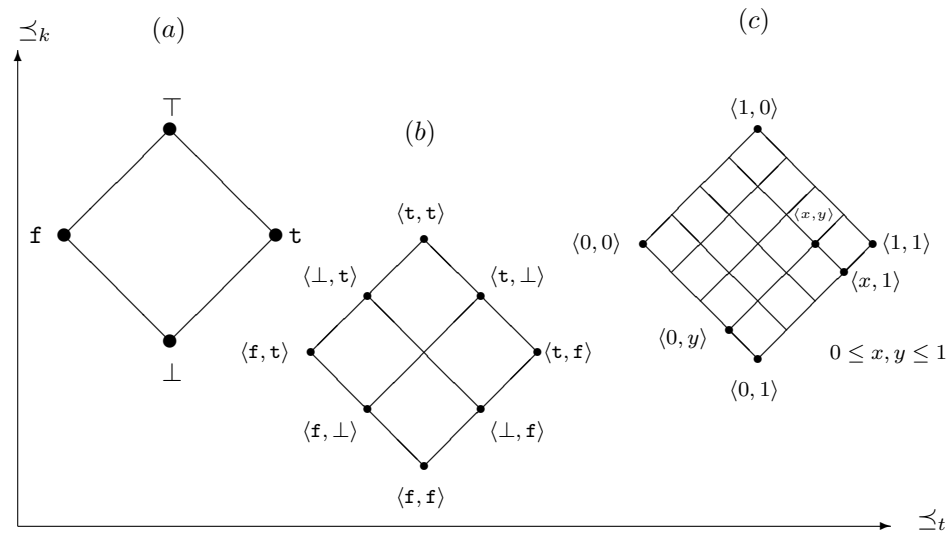

Figure 1: Bilattices. (a) $\mathcal{FOUR}$, (b) $\{\mathtt{f},\bot,\mathtt{t}\} \odot \{\mathtt{f},\bot,\mathtt{t}\}$ and (c) $\mathcal{K}([0,1] \cap \mathbb{Q})$.

Formally [21, 27], a *bilattice* is a structure $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$ where $\mathcal{B}$ is a non-empty, countable set and $\preceq_t$ and $\preceq_k$ are both partial orderings giving $\mathcal{B}$ the structure of a *complete lattice* with a top and bottom element. *Meet (or greatest lower bound)* and *join (or least upper bound)* under $\preceq_t$, denoted $\wedge$ and $\vee$, correspond to extensions of classical conjunction and disjunction. On the other hand, *meet and join under* $\preceq_k$ are denoted $\otimes$ and $\oplus$. $x \otimes y$ corresponds to the maximal information x and y can agree on, while $x \oplus y$ simply combines the information represented by x with that represented by y. *Top and bottom under* $\preceq_t$ are denoted $\mathtt{t}$ and $\mathtt{f}$, and *top and bottom under* $\preceq_k$ are denoted $\top$ and $\bot$, respectively. We will assume that bilattices are *infinitary distributive bilattices* in which all distributive laws connecting $\wedge, \vee, \otimes$ and $\oplus$ hold. We also assume that every bilattice satisfies the *infinitary interlacing conditions*, i.e. each of the lattice operations $\wedge, \vee, \otimes$ and $\oplus$ is monotone w.r.t. both orderings. An example of interlacing condition is: $x \preceq_t y$ and $x' \preceq_t y'$ implies $x \otimes x' \preceq_t y \otimes y'$. Finally, we assume that each bilattice has a *negation*, i.e. an operator $\neg$ that reverses the $\preceq_t$ ordering, leaves unchanged the $\preceq_k$ ordering, and verifies $\neg\neg x = x$.

Additionally, we provide a family $\mathcal{F}$ of $\preceq_k$ and $\preceq_t$-continuous binary and unary functions $f\colon \mathcal{B} \times \mathcal{B} \to \mathcal{B}$ and $f\colon \mathcal{B} \to \mathcal{B}$, that is, for any $\preceq_k$-monotone chain $x_0, x_1, \ldots$ of values in $\mathcal{B}$, $f(\oplus_i x_i) = \oplus_i f(x_i)$ and for any $\preceq_t$-monotone chain $x_0, x_1, \ldots$ of values in $\mathcal{B}$, $f(\vee_i x_i) = \vee_i f(x_i)$. The binary case is similar. Notably, it is not difficult to see that $\wedge, \vee, \otimes$ and $\vee$ are both $\preceq_k$-continuous and $\preceq_t$-continuous, while $\neg$ is $\preceq_k$-continuous but not $\preceq_t$-continuous (it is $\preceq_t$-antitone).

Bilattices come up in natural ways. Indeed, there are two general, but different, construction methods, which allow to build a bilattice from a lattice and are widely used. We just sketch them here in order to give a feeling of their application (see also [20, 27]).

The first bilattice construction method comes from [27]. Suppose we have two complete distributive lattices $\langle L_1, \preceq_1 \rangle$ and $\langle L_2, \preceq_2 \rangle$. Think of L_1 as a lattice of values we use when we measure the degree of belief of a statement, while think of L_2 as the lattice we use when we measure the degree of doubt of it. Now, we define the structure $L_1 \odot L_2$ as follows. The structure is $\langle L_1 \times L_2, \preceq_t, \preceq_k \rangle$, where

- $\langle x_1, x_2 \rangle \preceq_t \langle y_1, y_2 \rangle$ if $x_1 \preceq_1 y_1$ and $y_2 \preceq_2 x_2$;
- $\langle x_1, x_2 \rangle \preceq_k \langle y_1, y_2 \rangle$ if $x_1 \preceq_1 y_1$ and $x_2 \preceq_2 y_2$.

In $L_1 \odot L_2$ the idea is: knowledge goes up if both degree of belief and degree of doubt go

up; truth goes up if the degree of belief goes up, while the degree of doubt goes down. It is easily verified that $L_1 \odot L_2$ is a bilattice. Furthermore, if $L_1 = L_2 = L$, i.e. we are measuring belief and doubt in the same way, then negation can be defined as $\neg\langle x, y\rangle = \langle y, x\rangle$. That is, negation switches the roles of belief and doubt. In Figure 1 we report the bilattice based on $L_1 = L_2 = \{\mathtt{f}, \bot, \mathtt{t}\}$ and order $\preceq_1 = \preceq_2 = \preceq$, where $\mathtt{f} \preceq \bot \preceq \mathtt{t}$. Notably, under this approach fall work on paraconsistent logic programming [1, 15] and anti-tonic logic programming [16].

The second construction method has been sketched in [27] and addressed in more details in [24], and is probably the more used one. Suppose we have a complete distributive lattice of truth values $\langle L, \preceq\rangle$ (like e.g. in Many-valued Logics [28]). Think of these values as the 'real' values we are interested in, but due to lack of knowledge we are able just to 'approximate' the exact values. That is, rather than considering a pair $\langle x, y\rangle \in L \times L$ as indicator for degree of belief and doubt, $\langle x, y\rangle$ is interpreted as the set of elements $z \in L$ such that $x \preceq z \preceq y$. Therefore, a pair $\langle x, y\rangle$ is interpreted as an *interval*. An interval $\langle x, y\rangle$ may be seen as an approximation of an exact value. Formally, given a distributive lattice $\langle L, \preceq\rangle$, the *bilattice of intervals*, denoted $\mathcal{K}(L)$, is $\langle L \times L, \preceq_t, \preceq_k\rangle$, where:

- $\langle x_1, x_2\rangle \preceq_t \langle y_1, y_2\rangle$ if $x_1 \preceq y_1$ and $x_2 \preceq y_2$;
- $\langle x_1, x_2\rangle \preceq_k \langle y_1, y_2\rangle$ if $x_1 \preceq y_1$ and $y_2 \preceq x_2$.

The intuition of those orders is that truth increases if the interval contains greater values, whereas the knowledge increases when the interval becomes more precise. Negation can be defined as $\neg\langle x, y\rangle = \langle \neg y, \neg x\rangle$, where $\neg$ is a negation operator on L. As an example, in Figure 1 we report the bilattice $\mathcal{K}([0, 1] \cap \mathbb{Q})$. This approach has been used in e.g. [38, 39, 40]), where L is the unit interval $[0, 1] \cap \mathbb{Q}$ with standard ordering, $L \times L$ is interpreted as the set of (closed) sub-intervals of $[0, 1] \cap \mathbb{Q}$, and the pair $\langle x, y\rangle$ is interpreted as a lower and an upper bound of the exact value of the certainty value. Notably these works also show how to extend many logic program formalisms for the management of uncertainty over a lattice L with negation. Just take the bilattice $\mathcal{K}(L)$ and extend the functions point-wise, e.g. $f([x, y]) = [f(x), f(y)]$ [1].

2.3 Logic programs and models

We start with the definitions given in [20] and extend it to the case arbitrary functions $f \in \mathcal{F}$ are allowed in logic programs. For ease the presentation, will limit our attention to propositional logic programs. The first order case can be handled by grounding.

Logic programs. Consider an alphabet of propositional letters. An *atoms*, denoted A is a propositional letter. 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 literals, the members of a bilattice $\mathcal{B}$ using $\wedge, \vee, \otimes$ and $\oplus$ and the functions $f \in \mathcal{F}$. Note that members of the bilattice may appear in a formula, as well as functions: e.g. in $\mathcal{FOUR}$, $f(p \wedge q, r \otimes \mathtt{f}) \oplus v$ is a formula.

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 *logic program*, denoted with $\mathcal{P}$, is a finite set of rules. The *Herbrand base* of $\mathcal{P}$ (denoted $B_{\mathcal{P}}$) is the set of atoms occurring in $\mathcal{P}$.

Given $\mathcal{P}$, the set $\mathcal{P}^*$ is constructed as follows; (i) if an atom A is not head of any rule in $\mathcal{P}^*$, then add the rule $A \leftarrow \mathtt{f}$ to $\mathcal{P}^*$; [2] and (ii) replace several rules in $\mathcal{P}^*$ having same head, $A \leftarrow \varphi_1$, $A \leftarrow \varphi_2$, ... with $A \leftarrow \varphi_1 \vee \varphi_2 \vee \ldots$. Note that in $\mathcal{P}^*$, each atom appears in the head of *exactly one* rule.

Interpretations. An *interpretation of a logic program* on the bilattice $\langle \mathcal{B}, \preceq_t, \preceq_k\rangle$ is a mapping from atoms to members of $\mathcal{B}$. An interpretation I is extended from atoms to formulae as follows:

[1] Of course, the same can be done by using the belief-doubt bilattice construction.

[2] It is a standard practice in logic programming to consider such atoms as *false*.

(i) for $b \in \mathcal{B}$, $I(b) = b$; (ii) for formulae φ and φ', $I(\varphi \wedge \varphi') = I(\varphi) \wedge I(\varphi')$, and similarly for $\vee, \otimes, \oplus$ and $\neg$; and (iii) for formulae $f(\varphi)$, $I(f(\varphi)) = f(I(\varphi))$, and similarly for binary functions. The truth and knowledge orderings are extended from $\mathcal{B}$ to the set $\mathcal{I}(\mathcal{B})$ of all interpretations point-wise: (i) $I_1 \preceq_t I_2$ iff $I_1(A) \preceq_t I_2(A)$, for every ground atom A; and (ii) $I_1 \preceq_k I_2$ iff $I_1(A) \preceq_k I_2(A)$, for every ground atom A. We define $(I_1 \wedge I_2)(A) = I_1(A) \wedge I_2(A)$, and similarly for the other operations. With $\mathtt{I_f}$ and $\mathtt{I}_\bot$ we denote the bottom interpretations under $\preceq_t$ and $\preceq_k$ respectively (they map any atom into $\mathtt{f}$ and $\bot$, respectively). It is easy to see that $\langle \mathcal{I}(\mathcal{B}), \preceq_t, \preceq_k \rangle$ is an infinitary interlaced and distributive bilattice as well.

Classical setting. Note that in a *classical logic program* the body is a conjunction of literals. Therefore, if $A \leftarrow \varphi \in \mathcal{P}^*$ (except for the case $A \leftarrow \mathtt{f} \in \mathcal{P}^*$), then $\varphi = \varphi_1 \vee \ldots \vee \varphi_n$ and $\varphi_i = L_{i_1} \wedge \ldots \wedge L_{i_n}$. Furthermore, a *classical total interpretation* is an interpretation over $\mathcal{FOUR}$ such that an atom is mapped into either $\mathtt{f}$ or $\mathtt{t}$. A *partial classical interpretation* is a classical interpretation where the truth of some atom may be left unspecified. This is the same as saying that the interpretation maps all atoms into either $\mathtt{f}, \mathtt{t}$ or $\bot$.

Models. 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 the above definition of model follows the so-called *Clark-completion* procedure [11], where we replace in $\mathcal{P}^*$ each occurrence of $\leftarrow$ with $\leftrightarrow$. Indeed, 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". Under the Clark-completion, the constraint becomes also sufficient as the unique rule involving A in $\mathcal{P}^*$ *completely* defines A (see e.g. [21]).

Query. 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 a logic program $\mathcal{P}$. We also allow a query to be a *set* $\{?A_1, \ldots, ?A_n\}$ of query atoms. In that latter case we ask about the truth of all the atoms $A_i \in q$ in the intended model of a logic program $\mathcal{P}$.

2.4 Semantics of logic programs

The semantics of a logic program $\mathcal{P}$ is usually determined by selecting a particular model, or a set of models, of $\mathcal{P}$. In our context we will consider three possible intended semantics, namely the Kripke-Kleene, the Well-founded semantics and stable models, in $\preceq_k$-increasing order. Notably, the well-founded semantics is the $\preceq_k$-least stable model.

To ease our presentation, we will rely on the following simple running example to illustrate the concepts we introduce in the paper.

Example 1 (running example) *Consider the following logic program $\mathcal{P}$ with the following rules.*

$$p \leftarrow p$$
$$q \leftarrow \neg r$$
$$r \leftarrow \neg q \wedge \neg p$$

In the following table, we report the different interpretations and models presented in this paper: models, Kripke-Kleene (KK), Well-Founded (WF) semantics and stable models [20, 21, 22, 41, 61].

Kripke-Kleene semantics. The Kripke-Kleene semantics [20, 22] has a simple and intuitive characterization, as it corresponds to the $\preceq_k$-least model of a logic program, i.e. the *Kripke-Kleene model* of a logic program $\mathcal{P}$ is $KK(\mathcal{P}) = \min_{\preceq_k}\{I : I \models \mathcal{P}\}$. The *existence and uniqueness* of $KK(\mathcal{P})$ is guaranteed by the fixed-point characterization below, by means of the *immediate consequence operator* $\Phi_\mathcal{P}$. For an interpretation I, for any ground atom A

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

$I_i \models \mathcal{P}$	I_i p	q	r	KK	WF	stable models
I_1	$\bot$	$\bot$	$\bot$	•		
I_2	$\bot$	t	f			
I_3	f	$\bot$	$\bot$		•	•
I_4	f	f	t			•
I_5	f	t	f			•
I_6	f	$\top$	$\top$			•
I_7	t	t	f			
I_8	$\top$	t	f			
I_9	$\top$	$\top$	$\top$			

Table 1: Models, Kripke-Kleene (KK), Well-Founded (WF) semantics and stable models.

where $A \leftarrow \varphi \in \mathcal{P}^*$. [3] It can be shown that (based on [20, 39]) that

Proposition 2 *In the space of interpretations, the operator $\Phi_\mathcal{P}$ is $\preceq_k$-continuous, the set of fixed-points of $\Phi_\mathcal{P}$ is a complete lattice under $\preceq_k$ and, thus, $\Phi_\mathcal{P}$ has a $\preceq_k$-least (and $\preceq_k$-greatest) fixed-point; and I is a model of a program $\mathcal{P}$ iff I is a fixed-point of $\Phi_\mathcal{P}$.*

Therefore, the Kripke-Kleene model of $\mathcal{P}$ coincides with the least fixed-point of $\Phi_\mathcal{P}$ under $\preceq_k$, which can be computed in the usual way by iterating $\Phi_\mathcal{P}$ over $\mathtt{I}_\bot$ and is attained after at most ω iterations.

Example 2 *Consider the bilattice $\mathcal{K}([0,1] \cap \mathbb{Q})$, the function $f \in \mathcal{F}$, $f(\langle x, 1\rangle) = \langle \frac{x+a}{2}, 1\rangle$ ($0 < a \leq 1, a \in \mathbb{Q}$), and the logic program $\mathcal{P} = \{A \leftarrow f(A)\}$. Then the Kripke-Kleene model is attained after ω steps of $\Phi_\mathcal{P}$ iterations over $\mathtt{I}_\bot = \langle 0, 1\rangle$ and is such that $KK(\mathcal{P})(A) = \langle a, 1\rangle$.*

Stable models and the well-founded semantics. The *stable model semantics* approach, has been defined first by Gelfond and Lifschitz [26] with respect to the classical two valued truth space $\{\mathtt{f}, \mathtt{t}\}$ and extended by Fitting to bilattices [20, 21]. Informally, an interpretation I is a *stable model* of a logic program $\mathcal{P}$ if $I = I'$, where I' is computed according to the so-called *Gelfond-Lifschitz transformation*:

1. substitute (fix) in $\mathcal{P}^*$ the negative literals by their evaluation with respect to I. Let $\mathcal{P}^I$ be the resulting *positive* program, called *reduct* of $\mathcal{P}$ w.r.t. I; and
2. compute the truth-minimal model I' of $\mathcal{P}^I$.

For instance, given $\mathcal{P}$ and I_3 in Example 1, $\mathcal{P}^{I_3}$ is $\{(p \leftarrow p), (q \leftarrow \bot), (r \leftarrow \bot \wedge \mathtt{t})\}$, whose $\preceq_t$-least model is I_3. Therefore, I_3 is a stable model. On the other hand, it can be verified that the $\preceq_t$-least model of $\mathcal{P}^{I_1}$ ($= \mathcal{P}^{I_3}$), is I_3, so I_1 is *not* a stable model.

Formally, Fitting [20] relies on a binary immediate consequence operator $\Psi_\mathcal{P}$, which accepts two input interpretations over a bilattice, the first one is used to assign meanings to positive literals, while the second one is used to assign meanings to negative literals. Let I and J be two interpretations in the bilattice $\langle \mathcal{I}(\mathcal{B}), \preceq_t, \preceq_k\rangle$. The notion of *pseudo-interpretation* $I \triangle J$ over

[3] Recall that all atoms are head of exactly one rule in $\mathcal{P}^*$.

the bilattice is defined as follows (I gives meaning to positive literals, while J gives meaning to negative literals): for a pure ground atom A:

$$\begin{array}{lcl}(I \triangle J)(A) & = & I(A) \\ (I \triangle J)(\neg A) & = & \neg J(A)\ .\end{array}$$

Pseudo-interpretations are extended to non-literals in the obvious way. [4] For instance, $(I \triangle J)(f(\neg A \wedge B)) = f((I \triangle J)(\neg A \wedge B)) = f((I \triangle J)(\neg A) \wedge (I \triangle J)(B)) = f(\neg J(A) \wedge I(B))$. We can now define $\Psi_{\mathcal{P}}$ as follows. For $I, J \in \mathcal{I}(\mathcal{B})$, $\Psi_{\mathcal{P}}(I, J)$ is the interpretation, which for any ground atom A is such that

$$\Psi_{\mathcal{P}}(I, J)(A) = (I \triangle J)(\varphi)\ ,$$

where $A \leftarrow \varphi \in \mathcal{P}^*$. Note that $\Phi_{\mathcal{P}}$ is a special case of $\Psi_{\mathcal{P}}$, as by construction $\Phi_{\mathcal{P}}(I) = \Psi_{\mathcal{P}}(I, I)$. Similarly to [20], it can be shown that

Proposition 3 *In the space of interpretations the operator $\Psi_{\mathcal{P}}$ is $\preceq_k$-continuous in both arguments, $\preceq_t$-continuous in its first argument and $\preceq_t$-antitone in its second argument.*

To define the stable model semantics, Fitting [20] introduces the $\Psi'_{\mathcal{P}}$ operator, whose fixed-points will be the stable models of a program. For any interpretation I, $\Psi'_{\mathcal{P}}(I)$ is the $\preceq_t$-least fixed-point of the operator $\lambda x.\Psi_{\mathcal{P}}(x, I)$, i.e.

$$\Psi'_{\mathcal{P}}(I) = \mathrm{lfp}_{\preceq_t}(\lambda x.\Psi_{\mathcal{P}}(x, I))\ .$$

Due to the $\preceq_t$-continuity of $\Psi_{\mathcal{P}}$ on its first argument, $\Psi'_{\mathcal{P}}$ is well defined. $\Psi'_{\mathcal{P}}(I)$ can be computed by iterating $\Psi_{\mathcal{P}}(x, I)$ over $\mathtt{I}_{\mathtt{f}}$ and the limit is attained in at most ω iterations. Additionally, it can be shown that

Proposition 4 *The operator $\Psi'_{\mathcal{P}}$ is $\preceq_k$-continuous, $\preceq_t$-antitone and every fixed-point of $\Psi'_{\mathcal{P}}$ is also a fixed-point of $\Phi_{\mathcal{P}}$, i.e. a model of $\mathcal{P}$.*

A *stable model* for a logic program $\mathcal{P}$ is a fixed-point of $\Psi'_{\mathcal{P}}$. Therefore, the set of fixed-points of $\Psi'_{\mathcal{P}}$, i.e. the set of stable models of $\mathcal{P}$, is a complete lattice under $\preceq_k$ and, thus, $\Psi'_{\mathcal{P}}$ has a $\preceq_k$-least (and $\preceq_k$-greatest) fixed-point, which is denoted $WF(\mathcal{P})$. $WF(\mathcal{P})$ is the *Well-Founded model* of $\mathcal{P}$ and, by definition coincides with the $\preceq_k$-least stable model, i.e. $WF(\mathcal{P}) = \min_{\preceq_k}(\{I\colon I \text{ stable model of } \mathcal{P}\})$.

Finally, the well-founded model and the $\preceq_k$-greatest stable model can be computed by iterating $\Psi'_{\mathcal{P}}$ starting from $\mathtt{I}_{\bot}$ and $\mathtt{I}_{\top}$, respectively, and the limit is attained in at most ω iterations.

Example 3 *Let us consider the bilattice of intervals $\mathcal{K}([0, 1] \cap \mathbb{Q})$ Consider the following logic program $\mathcal{P}$,*

$$\begin{array}{lcl}A & \leftarrow & A \vee B \\ B & \leftarrow & (\neg C \wedge A) \vee \langle 0.3, 0.5\rangle \\ C & \leftarrow & \neg B \vee \langle 0.2, 0.4\rangle\end{array}$$

The table below shows the computation of the Kripke-Kleene semantics of $\mathcal{P}$, $KK(\mathcal{P})$, as $\preceq_k$-least fixed-point of $\Phi_{\mathcal{P}}$.

A	B	C	K_i
$\langle 0, 1\rangle$	$\langle 0, 1\rangle$	$\langle 0, 1\rangle$	K_0
$\langle 0, 1\rangle$	$\langle 0.3, 1\rangle$	$\langle 0.2, 1\rangle$	K_1
$\langle 0.3, 1\rangle$	$\langle 0.3, 0.8\rangle$	$\langle 0.2, 0.7\rangle$	K_2
$\langle 0.3, 1\rangle$	$\langle 0.3, 0.8\rangle$	$\langle 0.2, 0.7\rangle$	$K_3 = K_2 = KK(\mathcal{P})$

[4] Note that negation may appear in front of a literal only.

Note that knowledge increases during the computation as the intervals becomes more precise, i.e. $K_i \preceq_k K_{i+1}$.

The following table shows us the computation of the well-founded semantics of $\mathcal{P}$, $WF(\mathcal{P})$, *as* $\preceq_k$*-least fixed-point of* $\Psi'_{\mathcal{P}}$.

$v_i^{W_j}$	A	B	C	A	B	C	W_j
$v_0^{W_0}$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$	W_0
$v_1^{W_0}$	$\langle 0,0\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0,1\rangle$				
$v_2^{W_0}$	$\langle 0.3,0.5\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0,1\rangle$				
$v_3^{W_0}$	$\langle 0.3,0.5\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0,1\rangle$				
$v_0^{W_1}$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0,1\rangle$	W_1
$v_1^{W_1}$	$\langle 0,0\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.5,0.7\rangle$				
$v_2^{W_1}$	$\langle 0.3,0.5\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.5,0.7\rangle$				
$v_3^{W_1}$	$\langle 0.3,0.5\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.5,0.7\rangle$				
$v_0^{W_2}$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.5,0.7\rangle$	W_2
$v_1^{W_2}$	$\langle 0,0\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.5,0.7\rangle$				
$v_2^{W_2}$	$\langle 0.3,0.5\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.5,0.7\rangle$				
$v_3^{W_2}$	$\langle 0.3,0.5\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.5,0.7\rangle$				
				$\langle 0.3,0.5\rangle$	$\langle 0.3,0.5\rangle$	$\langle 0.5,0.7\rangle$	$W_3 = W_2 = WF(\mathcal{P})$

Notice that $W_i \preceq_k W_{i+1}$ *and* $KK(\mathcal{P}) \preceq_k WF(\mathcal{P})$, *as expected.* $\diamond$

3 Top-down query answering

Given a logic program $\mathcal{P}$ and an intended model (either Kripke-Kleene or Well-founded model), 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 general top-down method, which relies on the computation of just a partial 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 a modification of the method described in [2] and slightly inspired on [31]. The former presents a quite general top-down (local) procedure for computing the answer to a query with respect to the minimal fixed-point of a system of equations of monotonic functions over lattices, while the latter is a (global) bottom-up computation.

Let $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$ be a bilattice and let $\mathcal{P}$ be a logic program. Consider the Herbrand base $B_{\mathcal{P}} = \{A_1, \ldots, A_n\}$ of $\mathcal{P}$. We have seen that we can restrict our attention to $\mathcal{P}^*$ in which any atom A_i appears exactly once in the head of a rule. Let us associate to each atom $A_i \in B_{\mathcal{P}}$ a variable x_i, which will take a value in the domain $\mathcal{B}$ (sometimes, we will refer to that variable with x_A as well). An interpretation I may be seen as an assignment of bilattice values to the variables $x_1, ..., x_n$. For an immediate consequence operator O, e.g. $\Phi_{\mathcal{P}}$, a fixed-point is such that $I = O(I)$, i.e. for all atoms $A_i \in B_{\mathcal{P}}$, $I(A_i) = O(I)(A_i)$. As a consequence, we may identify the fixed-points of an immediate consequence operator O as the solutions over $\mathcal{B}$ of the system of equations of the following form:

$$\begin{array}{rcl} x_1 & = & f_1(x_{1_1}, \ldots, x_{1_{a_1}}) \,, \\ & \vdots & \\ x_n & = & f_n(x_{n_1}, \ldots, x_{n_{a_n}}) \,, \end{array} \tag{2}$$

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 $\mathcal{B}$, 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. For instance, by considering the logic program in Example 1, the fixed-points of the $\Phi_{\mathcal{P}}$ operator are the solutions over a bilattice of the system of equations ($p \mapsto x_1, q \mapsto x_2, r \mapsto x_3$)

$$\begin{array}{lcl} x_1 & = & x_1 \ , \\ x_2 & = & \neg x_3 \ , \\ x_3 & = & \neg x_2 \wedge \neg x_1 \ . \end{array} \tag{3}$$

It is easily verified that all nine interpretations I_i in Eaxmple 1 are bijectively related to the solutions of the system (3) over $\mathcal{FOUR}$ and $(x_1, x_2, x_3) = (\bot, \bot, \bot)$ is the $\preceq_k$-least solution and corresponds to the Kripke-Kleene model of $\mathcal{P}$.

In the following, we will adapt the general, easy to implement procedure [2] for the top-down computation of the value of variable x in the $\preceq$-least solution to the system (2), given a lattice $\mathcal{L} = \langle L, \preceq \rangle$. Then, we will customize it to the particular case of the Kripke-Kleene semantics and the Well-founded semantics.

We use some auxiliary functions. $\mathtt{s}(x)$ denotes the set of *sons* of x, i.e.

$$\mathtt{s}(x_i) = \{x_{i_1}, \ldots, x_{i_{a_i}}\} \ .$$

$\mathtt{p}(x)$ denotes the set of *parents* of x, i.e. the set

$$\mathtt{p}(x) = \{x_i \colon x \in \mathtt{s}(x_i)\} \ .$$

In the general case, we assume that each function $f_i \colon L^{a_i} \mapsto L$ in Equation (2) 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 (2) as the tuple $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$, where $\mathcal{L}$ is a lattice, $V = \{x_1, ..., x_n\}$ are the variables and $\vec{f} = \langle f_1, ..., f_n \rangle$ is the tuple of functions.

As it is well known, a monotonic equation system as (2) has a $\preceq$-least solution, $\mathrm{lfp}_{\preceq}(\vec{f})$, the $\preceq$-least fixed-point of $\vec{f}$ is given as the least upper bound of the $\preceq$-monotone sequence, $\vec{x}_0, \ldots, \vec{x}_1, \ldots$, where

$$\begin{array}{lcl} \vec{x}_0 & = & \vec{\bot} \\ \vec{x}_{i+1} & = & \vec{f}(\vec{x}_i) \ . \end{array} \tag{4}$$

We are ready to describe informally the algorithm. Assume that we are interested in the value of variable x_0 in the least fixed-point of the system. We associate to each variable x_i a marking $\mathtt{v}(x_i)$, which denotes the current value of x_i. Initially, the value of each variable 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 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 particular variable are put in the active set to be examined. At some point the active list will become empty and, and we have actually found part of the fixed-point, sufficient to determine the value of the query x_0.

The general algorithm is given in Table 2. Note that 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). Note also that our algorithm is indeed a simplification of the one in [2] as this latter allows also to deal with partial functions. That is, it tries to evaluate as view sons as possible, by allowing some variable to have an undefined truth-value. Informally, this is useful whenever one would like to take advantage of the fact that e.g. $\mathtt{f} \wedge x$ is evaluated to $\mathtt{f}$, whatever the value for x is and, thus x has not to be evaluated. Our approach can be extended to this case as well but some technicalities have to be introduced to cope with partial functions, which do not fit into the space constraints of this paper.

	Procedure $Solve(\mathcal{S}, Q)$
	Input: $\preceq$-monotonic system $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$, where $Q \subseteq V$ is the set of query variables;
	Output: A set $B \subseteq V$, with $Q \subseteq B$ such that the mapping $\mathtt{v}$ 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 2: General Top-down algorithm.

The attentive reader will notice that the $Solve$ procedure has much in common with the so-called *tabulation* procedures, like [9, 10, 13, 58]. Indeed, it is a generalization of it to arbitrary monotone equational systems over lattices.

Given a system $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$, 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 countable set X is the least ordinal α such that α and X are *equipollent*, i.e. there is a bijection from α to X. For instance, $h(\mathcal{FOUR}) = 2$ w.r.t. $\preceq_k$ as well as w.r.t. $\preceq_t$, while $h(\mathcal{K}([0,1] \cap \mathbb{Q})) = \omega$. Likewise to [2], it can be shown that the above algorithm behaves correctly.

Proposition 5 *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$, then there is a limit ordinal λ such that after $|\lambda|$ steps $Solve(\mathcal{S}, Q)$ determines a set $B \subseteq V$, with $Q \subseteq B$ such that the mapping $\mathtt{v}$ equals $lfp_{\preceq}(\vec{f})$ on B, i.e. $\mathtt{v}_{|B} = lfp_{\preceq}(\vec{f})_{|B}$.*

Example 4 *Let us consider Example 3 and the associated equational system $\mathcal{S}$ with 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. $x_i := x_B, \mathtt{A} := \{x_A\}, \mathtt{dg} := \{x_A, x_B, x_C\}, r := \langle 0.3, 1\rangle, \mathtt{v}(x_B) := \langle 0.3, 1\rangle, \mathtt{A} := \{x_A, x_C\}, \mathtt{exp}(x_B) := \mathtt{true}, \mathtt{in} := \{x_A, x_B, x_C\}$
3. $x_i := x_C, \mathtt{A} := \{x_A\}, r := \langle 0.2, 0.7\rangle, \mathtt{v}(x_C) := \langle 0.2, 0.7\rangle, \mathtt{A} := \{x_A, x_B\}, \mathtt{exp}(x_C) := \mathtt{true}$
4. $x_i := x_B, \mathtt{A} := \{x_A\}, r := \langle 0.3, 0.8\rangle, \mathtt{v}(x_B) := \langle 0.3, 0.8\rangle, \mathtt{A} := \{x_A, x_C\}$
5. $x_i := x_C, \mathtt{A} := \{x_A\}, r := \langle 0.2, 0.7\rangle$
6. $x_i := x_A, \mathtt{A} := \emptyset, r := \langle 0.3, 1\rangle, \mathtt{v}(x_A) := \langle 0.3, 1\rangle, \mathtt{A} := \{x_A, x_B\}$
7. $x_i := x_B, \mathtt{A} := \{x_A\}, r := \langle 0.3, 0.8\rangle,$
8. $x_i := x_A, \mathtt{A} := \emptyset, r := \langle 0.3, 1\rangle$
10. $\mathtt{stop}.\ \mathtt{return}\ \mathtt{v}(x_A, x_B, x_C) = \langle \langle 0.3, 1\rangle, \langle 0.3, 0.8\rangle, \langle 0.2, 0.7\rangle \rangle$

Note that $Solve(\mathcal{S}, \{x_A\})$ answers w.r.t. the Kripke-Kleene semantics as we considered the $\preceq_k$ ordering.

From a Computational point of view, it is easily verified that by means of appropriate data structures, the operations on `A`, `v`, `dg`, `in`, `exp`, `p` and `son` can be performed in constant time. Therefore, Step 1. is $O(|V|)$, all other steps, except Step 2. and Step 4. are $O(1)$. Let $c(f_x)$ be the maximal cost of evaluating function f_x on its arguments, so Step 4. is $O(c(f_x))$. It remains to determine the number of loops of Step 2. In case the height $h(\mathcal{L})$ of the bilattice $\mathcal{L}$ is finite, observe that any variable is increasing in the $\preceq$ order as it enters in the `A` list (Step 5.), except the case 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

$$O(\sum_{x_i \in V} (c(f_i) \cdot (a_i \cdot h(\mathcal{L}) + 1)) \,. \tag{5}$$

As a consequence,

Proposition 6 *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$. If the cost of computing each of the functions in $\vec{f}$ is bounded by c, the arity bounded by a, and the height is bounded by h, then the worst-case complexity of the algorithm $Solve$ is $O(|V|cah)$.*

In case the height of a bilattice is not finite, the computation may not terminate after a finite number of steps (see Example 2). Fortunately, under reasonable assumptions on the functions, we may guarantee the termination of $Solve$. We exploit two of such conditions. Consider a monotonic equational system $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$. Consider a function $f\colon L \to L$, where $\langle L, \preceq \rangle$ is a lattice. Let $[\bot]_f$ be the *f-closure of* $\{\bot\}$, i.e. the smallest set that contains $\{\bot\}$ and is closed under f. We say that *f has a finite generation* (see also [7] for more on this issue) iff $[\bot]_f$ is finite. For instance, it can be verified that the functions $\wedge, \vee, \otimes, \oplus, \neg$ have a finite generation on *any* finite set $X \subseteq \mathcal{B}$. Note also that if f, g have a finite generation over X then so has $f \circ g$.

Therefore, given an equational system $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$. If $\vec{f}$ has a finite generation, then $[\bot]_{\vec{f}}$ is finite. That is, $\{\bot, \vec{f}(\vec{\bot}), \vec{f}^2(\vec{\bot}), ...\}$ is finite. In particular, on induction on the computation of the $\preceq$-least fixed-point of $\mathcal{S}$ it can be shown that at each step of the bottom-up computation of the $\preceq$-least fixed-point, the values of the variables are in $[\bot]_{\vec{f}}$. Therefore, the *height* of $[\bot]_{\vec{f}}$, $h([\bot]_{\vec{f}})$, is finite. On the other hand, it can easily be seen that $Solve$ terminates if the sequence defined by Equation (4), $\vec{\bot}, \vec{f}(\vec{\bot}), \vec{f}^2(\vec{\bot}), ...$ converges after a finite number of steps. Therefore,

Proposition 7 *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$. Then $Solve$ terminates iff $\vec{f}$ has a finite generation. If the cost of computing each of the functions in $\vec{f}$ is bounded by c and the arity bounded by a then the worst-case complexity of the algorithm $Solve$ is $O(|V|cah)$, where h is the height of $[\bot]_{\vec{f}}$.*

Example 5 *Consider $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$, where $\mathcal{L}$ has a finite height h. Consider a chain $x_1 \preceq x_2 \preceq ... \preceq x_h$ in $\mathcal{L}$. Assume $\vec{f}$ has exactly one equation $x = f(x)$, where $f(x_i) = x_{i+1}$, $f(x_h) = x_h$ and $f(x) = \bot$, for $x \neq x_i$ ($1 \leq i \leq n$). Then f has a finite generation, the variable x enters h times into the `A` list, and $Solve$ terminates after h iterations.*

The second condition on the functions, which guarantees the termination of $Solve$, is inspired directly by [12, 14] and is a special case of above. On bilattices, we say that a function $f\colon \mathcal{B}^n \to \mathcal{B}$ is *bounded* iff $f(x_1, \ldots, x_n) \preceq_k \otimes_i x_i$. Now, consider a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$. We say that $\vec{f}$ is *bounded* iff each f_i is a composition of functions, each of which is either bounded, or a constant in $\mathcal{B}$ or one of $\vee, \wedge, \oplus, \otimes$ and $\neg$. For instance, the function in Example 2 is not bounded, while $f_i(\langle x, y \rangle) = \langle \max(0, x + y - 1), 1 \rangle \wedge \langle 0.3, 0.4 \rangle$ over $\mathcal{K}([0, 1] \cap \mathbb{Q})$ is. The idea is to prevent the existence of an infinite ascending chain of the form $\vec{\bot} \prec_k \vec{f}(\vec{\bot}) \prec_k \ldots \prec_k \vec{f}^m(\vec{\bot}) \prec_k \ldots$. This is indeed the case. Roughly, consider a $\preceq_k$-monotone function $\vec{f} = \vec{g} \circ \vec{h}$, where $\vec{g}$ is a bounded function, while $\vec{h}$ is the composition of constants in $\mathcal{B}$ or functions among $\vee, \wedge, \oplus, \otimes$ and $\neg$. Then $\vec{\bot} \preceq_k \vec{f}(\vec{\bot}) = \vec{g} \circ \vec{h}(\vec{\bot}) = \vec{g}(\vec{h}(\vec{\bot})) \preceq_k \vec{h}(\vec{\bot})$. But $\vec{h}$ has a finite generation and, thus, so has $\vec{f}$. The argument for $\vec{f} = \vec{h} \circ \vec{g}$ is similar. Therefore,

Proposition 8 *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \vec{f} \rangle$, where $\vec{f}$ is bounded. Then $Solve$ terminates. The cost is as for Proposition 7.*

Note that for bounded functions $\vec{f} = \vec{g} \circ \vec{h}$, the height of $[\bot]_{\vec{f}}$ is given by the height of $[\bot]_{\vec{h}}$. We belief that this latter height is bounded by the number $n = |V|$ as we conjecture that $\vec{h}^n(\vec{\bot}) = \vec{h}^{n+1}(\vec{\bot})$ (this is compatible with [12, 14]). This would imply that the worst-case complexity of the algorithm $Solve$ is $O(|V|^2 ca)$ in that case.

3.1 Top-down query answering under the 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 $\mathcal{P}^*$. As already pointed out, each atom appear 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 $\mathcal{S}_{KK}(\mathcal{P}) = \langle \mathcal{L}, V, \vec{f}_{\mathcal{P}} \rangle$ be the resulting equational system (see Equation 3). 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 $\preceq_k$-continuous. As $\vec{f}_{\mathcal{P}}$ is one to one related to $\Phi_{\mathcal{P}}$, it follows that the $\preceq_k$-least fixed-point of $\mathcal{S}_{KK}(\mathcal{P})$ corresponds to the Kripke-Kleene semantics of $\mathcal{P}$. The algorithm $Solve_{KK}(\mathcal{P}, ?A)$, first computes $\mathcal{S}_{KK}(\mathcal{P})$ and then calls $Solve(\mathcal{S}_{KK}(\mathcal{P}), \{x_A\})$ and returns the output $\mathtt{v}$ on the query variable, where $\mathtt{v}$ is the output of the call to $Solve$. It can be shown that $Solve_{KK}$ behaves correctly (see Example 4).

Proposition 9 *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 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. In case the height of the bilattice is finite, from Proposition 6 it follows immediately that

Proposition 10 *The worst-case complexity for top-down query answering under the Kripke-Kleene semantics of a logic program $\mathcal{P}$ is $O(|B_{\mathcal{P}}| cah)$.*

Furthermore, note that is reasonable to assume that the cost of computing each of the functions of $\vec{f}_{\mathcal{P}}$ is in $O(1)$. By observing that $|B_{\mathcal{P}}|a$ is in $O(|\mathcal{P}|)$ we immediately have that

Proposition 11 *If the height is bounded by h, then the worst-case complexity for top-down query answering under the Kripke-Kleene semantics of a logic program $\mathcal{P}$ is $O(|\mathcal{P}|h)$.*

It follows that over the bilattice $\mathcal{FOUR}$ ($h = 2$) the top-down algorithm works in linear time. Moreover, if the height is a fixed parameter, i.e. a constant, we can conclude that the additional expressive power of Kripke-Kleene semantics of logic programs over bilattices (with functions with constant cost) does not increase the computational complexity of classical logic programs [18].

The computational complexity of the case where the height of the bilattice is not finite is determined by Proposition 7 and Proposition 8. In general, the continuity of the functions in $\mathcal{S}_{KK}(\mathcal{P})$ guarantees the termination after at most ω steps.

3.2 Top-down query answering under the Well-founded semantics

We address now the issue of a top-down computation of the value of a query under the well-founded semantics. As we have seen, according to Fitting's formulation, the well-founded semantics of a logic program $\mathcal{P}$ is the $\preceq_k$-least fixed-point of the operator $\Psi'_{\mathcal{P}}$ defined as

$$\Psi'_{\mathcal{P}}(I) = \mathrm{lfp}_{\preceq_t}(\lambda x.\Psi_{\mathcal{P}}(x, I)) \ , \tag{6}$$

where $\Psi_{\mathcal{P}}(I, J)(A) = (I \triangle J)(\varphi)$. Before we are going to present our top-down procedure for the well-founded semantics, we roughly explain the approach. To this purpose, let us consider Example 1. Assume that our query is $?r$ and consider the related equational system (3). So, our query variable is x_3. Following the $Solve$ algorithm, x_3 becomes the active variable. We have to introduce a major change in Step 4. Indeed, it is not hard to see that, due to Equation (6) above, in order to compute $r := \neg x_2 \wedge \neg x_1$, we have to compute the values of x_1 and x_2 w.r.t. the $\preceq_t$-least fixed-point of another equational system, where the current partial evaluation $\mathtt{v}$ acts as the interpretation I. This means that we have to make a call to another instance of the $Solve$ algorithm, which computes the values of x_1 and x_2 w.r.t. to the current evaluation $\mathtt{v}(x_1, x_2, x_3)$. In our case, we consider the equational system (3) in which negated variables have been replaced with their value w.r.t. to the current evaluation and, thus, we replace $\neg x_1, \neg x_2$ and $\neg x_3$ with $\mathtt{v}(x_1)$ and $\mathtt{v}(x_2)$, and $\mathtt{v}(x_3)$ respectively. Once the sub-routine call gives us back the values of the arguments x_1, x_2 of the function f_3 we compute $r{:=}\ \neg x_2 \wedge \neg x_1$ and continue with Step 5.

Let us formalize the above illustrated concept. Given a logic program $\mathcal{P}$, given a truth value assignment I, let us denote $\mathcal{S}(\mathcal{P}^I)$ the equational system obtained from $\mathcal{S}_{KK}(\mathcal{P})$ in which all occurrences of $\neg x$ have been replaced with $\neg I(x)$, except that $\mathcal{S}(\mathcal{P}^I)$ is based on the $\preceq_t$ order.

It can be shown that

Proposition 12 *$Solve(\mathcal{S}(\mathcal{P}^I), Q)$ outputs a set $B \subseteq V$, with $Q \subseteq B$, s.t. the mapping $\mathtt{v}$ equals to the $\preceq_t$-least fixed-point on B of the functions in $\mathcal{S}(\mathcal{P}^I)$ and, thus, $\mathtt{v}_{|B} = \Psi'_{\mathcal{P}}(I)_{|B}$.*

Moreover, from a computational complexity point of view, the same properties of $Solve$ hold for $Solve(\mathcal{S}(\mathcal{P}^I), Q)$ as well. This completes the first part.

Finally, $Solve_{WF}(\mathcal{P}, ?A)$ is as $Solve_{KK}(\mathcal{P}, ?A)$, except that Step $4.$ is replaced with

4.1. $\mathtt{Q} := \mathtt{s}(x_i)$; $\mathtt{I} := \mathtt{v}$;

4.2. $\mathtt{v}' := Solve(\mathcal{S}(\mathcal{P}^{\mathtt{I}}), \mathtt{Q})$;

4.3. $r{:=}\ f_i(\mathtt{v}'(x_{i_1}), ..., \mathtt{v}'(x_{i_{a_i}}))$.

It can be shown that

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

Example 6 *Let us consider Example 3 and the associated equational system $\mathcal{S}$ with query variable x_A. Below is a sequence of $Solve_{WW}(\mathcal{P}, ?A)$ computation. It resembles the one we have seen in Example 6. Each line is a sequence of steps in the 'while loop'. What is left unchanged is not reported.*

1. $\mathtt{A}{:=}\{x_A\}, x_i{:=}x_A, \mathtt{A}{:=}\emptyset, \mathtt{dg}{:=}\{x_A, x_B\}, \mathtt{Q}{:=}\{x_A, x_B\}, \mathtt{v'}{:=}\langle\langle 0.3, 0.5\rangle, \langle 0.3, 0.5\rangle, \langle 0, 1\rangle\rangle$, $r{:=}\langle 0.3, 0.5\rangle, \mathtt{v}(x_A){:=}\langle 0.3, 0.5\rangle, \mathtt{A}{:=}\{x_A, x_B\}, \mathtt{exp}(x_A){:=}\mathtt{true}, \mathtt{in}{:=}\{x_A, x_B\}$
2. $x_i{:=}x_B, \mathtt{A}{:=}\{x_A\}, \mathtt{dg}{:=}\{x_A, x_B, x_C\}, \mathtt{Q}{:=}\{x_A, x_C\}, \mathtt{v'}{:=}\langle\langle 0.3, 0.5\rangle, \langle 0.3, 0.5\rangle, \langle 0.5, 0.7\rangle\rangle$, $r{:=}\langle 0.3, 0.5\rangle, \mathtt{v}(x_B){:=}\langle 0.3, 0.5\rangle, \mathtt{A}{:=}\{x_A, x_C\}, \mathtt{exp}(x_B){:=}\mathtt{true}, \mathtt{A}{:=}\{x_A, x_C\}$, $\mathtt{in}{:=}\{x_A, x_B, x_C\}$
3. $x_i{:=}x_C, \mathtt{A}{:=}\{x_A\}, \mathtt{Q}{:=}\{x_B\}, \mathtt{v'}{:=}\langle\langle 0.3, 0.5\rangle, \langle 0.3, 0.5\rangle, \langle 0.5, 0.7\rangle\rangle$, $r{:=}\langle 0.5, 0.7\rangle, \mathtt{v}(x_C){:=}\langle 0.5, 0.7\rangle, \mathtt{A}{:=}\{x_A, x_B\}, \mathtt{exp}(x_C){:=}\mathtt{true}$
4. $x_i{:=}x_B, \mathtt{A}{:=}\{x_A\}, \mathtt{Q}{:=}\{x_A, x_C\}, \mathtt{v'}{:=}\langle\langle 0.3, 0.5\rangle, \langle 0.3, 0.5\rangle, \langle 0.5, 0.7\rangle\rangle, r{:=}\langle 0.3, 0.5\rangle$
5. $x_i{:=}x_A, \mathtt{A}{:=}\emptyset, \mathtt{Q}{:=}\{x_A, x_B\}, \mathtt{v'}{:=}\langle\langle 0.3, 0.5\rangle, \langle 0.3, 0.5\rangle, \langle 0.5, 0.7\rangle\rangle, r{:=}\langle 0.3, 0.5\rangle$
6. $\mathtt{stop}.\ \mathtt{return}\ \mathtt{v}(x_A, x_B, x_C)_{|x_A} = \langle\langle 0.3, 0.5\rangle, \langle 0.3, 0.5\rangle, \langle 0.5, 0.7\rangle\rangle_{|x_A} = \langle 0.3, 0.5\rangle$

We conclude with addressing the computational complexity of $Solve_{WF}$. The analysis parallels the one we have made for $Solve_{KK}$. We first address the case the height of a bilattice is finite. Like $Solve_{KK}$, each variable x_j will appear in $\mathtt{A}$ 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 analogously to Proposition 10 is $O(|B_{\mathcal{P}}|cah)$. Therefore, $Solve_{WW}$ runs in time $O(|B_{\mathcal{P}}|^2 a^2 h^2 c)$. Therefore,

Proposition 14 *The worst-case complexity for top-down query answering under the well-founded semantics of a logic program $\mathcal{P}$ over bilattices is $O(|\mathcal{P}|^2 h^2 c)$, where h is the height of the bilattice.*

If the bilattice is fixed, then the height parameter is a constant. Furthermore, it is reasonable to assume that c is $O(1)$ and, thus, the worst-case complexity reduces to $O(|\mathcal{P}|^2)$. However, note that for the classical case we can do better, as worked out in [36]. It remains open whether those results extends to our case as well.

In the case the height of a bilattice is not finite, the continuity of the functions $f \in \mathcal{F}$ guarantees that each recursive call to $Solve$ requires at most ω steps. Thus, we have at most ω_ω steps for $Solve_{WF}$. In case the functions have a finite generation or are bounded, Proposition 7 and Proposition 8 can be applied.

4 Conclusions and outlook

We have presented a general, top-down algorithm to answer queries to monotone equational systems over lattices and bilattices and, thus, for logic programs over thereof. We believe that its interest relies on the fact that most approaches to paraconsistency and uncertainty of logic programs with negation can be reduced to the bilattice framework and, thus, the presented algorithm gives us an easy to implement query-solving procedure for them. However, we are aware that the "quadratic bound" for the well-founded semantics case may not be completely satisfactory, especially in the light of the results [36]. It is, thus, interesting to investigate whether modifications to our algorithm $Solve_{WF}$ inspired by [36] or by [58] may give advantages from a computational point of view.

References

[1] João Alcantâra, Carlos Viegas Damásio, and Luís Moniz Pereira. Paraconsistent logic programs. In *Proc. of the 8th European Conference on Logics in Artificial Intelligence*

(JELIA-02), number 2424 in Lecture Notes in Computer Science, pages 345–356, Cosenza, Italy, 2002. Springer-Verlag.

[2] Henrik R. Andersen. Local computation of simultaneous fixed-points. Technical Report PB-420, DAIMI, October 1992.

[3] Ofer Arieli. Paraconsistent declarative semantics for extended logic programs. *Annals of Mathematics and Artificial Intelligence*, 36(4):381–417, 2002.

[4] Ofer Arieli and Arnon Avron. The value of the four values. *Artificial Intelligence Journal*, 102(1):97–141, 1998.

[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] H. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.

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

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

[9] Weidong Chen, Terrance Swift, and David Scott Warren. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24(3):161–199, 1995.

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

[11] K.L. Clark. Negation as failure. In Hervé Gallaire and Jack Minker, editors, *Logic and data bases*, pages 293–322. Plenum Press, New York, NY, 1978.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulas. *Journal of Logic Programming*, 3(1):267–284, 1984.

[19] 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.

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

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

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

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

[24] Melvin Fitting. Kleene's logic, generalized. *Journal of Logic and Computation*, 1(6):797–810, 1992.

[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] Reiner Hänle and Gonzalo Escalada-Imaz. Deduction in many-valued logics: a survey. *Mathware and Soft Computing*, IV(2):69–97, 1997.

[29] 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.

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

[31] Gary A. Kildall. A unified approach to global program optimization. In *Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 194–206. ACM Press, 1973.

[32] 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.

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

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

[35] 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.

[36] Zbigniew Lonc and Mirosław Truszczyński. On the problem of computing the well-founded semantics. *Theory and Practice of Logic Programming*, 5(1):591–609, 2001.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

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

[44] 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.

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

[46] 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.

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

[48] 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.

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

[50] 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.

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

[52] Jesús Medina, Manuel Ojeda-Aciego, and Peter Vojtás. 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.

[53] 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.

[54] 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.

[55] 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.

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

[57] 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.

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

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

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

[61] 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, January 1991.

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

[63] 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.

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