Query Answering in Normal Logic Programs Under Uncertainty

Umberto Straccia

ISTI - CNR, Via G. Moruzzi, 1 I-56124 Pisa (PI), Italy

Abstract. We present a simple, yet general top-down query answering procedure for normal logic programs over lattices and bilattices, where functions may appear in the rule bodies. Its interest relies on the fact that many approaches to paraconsistency and uncertainty in logic programs with or without non-monotonic negation are based on bilattices or lattices, respectively.

1 Introduction

The management of uncertainty in logic programming has attracted the attention of many researchers and numerous frameworks have been proposed. Essentially, they differ in the underlying notion of uncertainty [1] (e.g. probability theory [15, 18], fuzzy set theory [22], multi-valued logic [4, 12, 13, 14], possibilistic logic [8]) and how uncertainty values, associated to rules and facts, are managed. Roughly, these frameworks can be classified into *annotation based* (AB) and *implication based* (IB). In the AB approach (e.g. [12, 18]), 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 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. In the IB approach (see [4, 13] for a more detailed comparison between the two approaches), 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_i, 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. More recently, [4, 13, 22] 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, as they do not address any mode of *non-monotonic negation*. Exception to this limitation are e.g. [16, 17] in which the stable semantics has been considered, but limited to the case where the underlying uncertainty formalism is probability theory; in [7] the underlying truth-space are lattices, but its formulations goes over *bilattices* [11] (a slightly more general structure than lattices); while [14] uses normal logic programs over bilattices under the IB framework.

[1] See e.g. [19] for an extensive list of references.

L. Godo (Ed.): ECSQARU 2005, LNAI 3571, pp. 687–700, 2005.

In many frameworks, in order to answer to a query, we have to compute the whole intended model by a bottom-up fixed-point computation and then answer with the evaluation of the query in this model. This always requires to compute a whole model, even if not all the atom's truth is required to determine the answer. To the best of our knowledge the only work presenting top-down procedures are [5, 12, 13, 22], but in none of them non-monotonic negation is considered.

In this paper we present a general top-down query answering procedure for normal logic programs over lattices and bilattices [9, 11] in the IB framework. Its interest relies on the fact that many approaches to paraconsistency and uncertainty in logic programs with or without non-monotonic negation are based on bilattices or lattices, respectively.

We proceed as follows. In the next section, we will briefly recall definitions and properties of lattices, bilattices and normal 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

Lattice. In a complete lattice $\mathcal{L} = \langle L, \preceq \rangle$, with L a countable set, bottom $\bot$ and top element $\top$, a function $f\colon L \to L$ is *monotone*, if $\forall x, y \in L$, $x \preceq y$ 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 functions on lattices is the well-known Knaster-Tarski theorem [20]. 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 $\preceq$-*least* fixed-point. The $\preceq$-*least* fixed-point 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_λ, $\ldots$, where for a successor ordinal $i \geq 0$, $y_0 = \bot$, $y_{i+1} = f(y_i)$, while for a limit ordinal λ, $y_\lambda = lub_{\preceq}\{y_i \colon i < \lambda\}$. We denote the $\preceq$-least fixed-point by $\mathrm{lfp}_{\preceq}(f)$. For ease, we will specify the initial condition y_0 and the next iteration step y_{i+1} only, while the condition for a limit ordinal is considered as implicit.

Bilattice. A *bilattice* [11] is a structure $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$ where $\mathcal{B}$ is a non-empty, countable set and $\preceq_t$ (the *truth order*) and $\preceq_k$ (the *knowledge order*) 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 `t` and `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 $\neg\neg x = x$. For instance, the simplest non-

trivial bilattice, called $\mathcal{FOUR}$, is due to Belnap [1], who introduced a logic intended to deal with incomplete and/or inconsistent information – see also [6]. $\mathcal{FOUR}$ 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*. 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 $\mathtt{f} \preceq_t \top \preceq_t \mathtt{t}$. Furthermore, we have that $\neg\mathtt{f} = \mathtt{t}$, $\neg\bot = \bot$, $\neg\top = \top$. In addition to the usual bilattice approach, we provide a family $\mathcal{F}$ of $\preceq_k$ and $\preceq_t$-continuous binary and unary functions $f{:}\,\mathcal{B} \times \mathcal{B} \to \mathcal{B}$ and $f{:}\,\mathcal{B} \to \mathcal{B}$ to manipulate truth values. 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, $\wedge, \vee, \otimes$ 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). Interestingly, bilattices come up in two natural ways and are widely used. We just sketch them here in order to give a feeling of their application. The first bilattice construction method comes from [11]. 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 (i) $\langle x_1, x_2\rangle \preceq_t \langle y_1, y_2\rangle$ if $x_1 \preceq_1 y_1$ and $y_2 \preceq_2 x_2$, while (ii) $\langle x_1, x_2\rangle \preceq_k \langle y_1, y_2\rangle$ 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. Notably, under this approach fall work on paraconsistent logic programming (see, e.g. [6]) and anti-tonic logic programming (see, e.g. [7]). In the second construction method, suppose we have a complete distributive lattice of truth values $\langle L, \preceq\rangle$ (like e.g. in Many-valued Logics). Think of these values as the 'actual' 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: (i) $\langle x_1, x_2\rangle \preceq_t \langle y_1, y_2\rangle$ if $x_1 \preceq y_1$ and $x_2 \preceq y_2$, while (ii) $\langle x_1, x_2\rangle \preceq_k \langle y_1, y_2\rangle$ 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. This approach has been used, e.g. in [14], where L is the unit interval $[0, 1]_{\mathbb{Q}} = [0, 1] \cap \mathbb{Q}$ with standard ordering.

Logic Programs. Fix a bilattice. We start with the definitions given in [9] and extend it to the case *arbitrary computable functions* $f \in \mathcal{F}$ are allowed in rule bodies to manipulate the truth values. For the ease of presentation, we limit our attention to propositional logic programs. The first order case can be handled by grounding. So, consider an alphabet of propositional letters. An *atom*, 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. The intuition here is that the truth value of the formula $f(p \wedge q, r \otimes \mathtt{f}) \oplus v$ is obtained by determining the truth value of $p \wedge q$ and $r \otimes \mathtt{f}$, then apply the function f to them and join the result to the truth value of v. A *rule* is of the form $A \leftarrow \varphi$, where A is an atom and φ is a formula. The atom A is called the *head*, and the formula φ is called the *body*. A *normal logic program* (in the following, simply *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.

Example 1 ([14]). *Consider $\mathcal{K}([0,1]_{\mathbb{Q}})$, with $\wedge = \min$ and $\vee = \max$. Consider an insurance company, which has information about its customers used to determine the risk coefficient of each customer. Suppose a value of the risk coefficient is already known, but has to be re-evaluated (the client is a new client and his risk coefficient is given by his precedent insurance company). The company may have: (i) data grouped into a set of facts $\{(\mathtt{Experience(john)} \leftarrow [0.7, 0.7]), (\mathtt{Risk(john)} \leftarrow [0.5, 0.5]), (\mathtt{Sport\_car(john)} \leftarrow [0.8, 0.8])\}$; and (ii) a set of rules, which after grounding are:*

$$
\begin{array}{ll}
\mathtt{Good\_driver(john)} & \leftarrow \mathtt{Experience(john)} \wedge \neg \mathtt{Risk(john)} \\
\mathtt{Risk(john)} & \leftarrow 0.8 \cdot \mathtt{Young(john)} \\
\mathtt{Risk(john)} & \leftarrow 0.8 \cdot \mathtt{Sport\_car(john)} \\
\mathtt{Risk(john)} & \leftarrow \mathtt{Experience(john)} \wedge \neg \mathtt{Good\_driver(john)}
\end{array}
$$

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: (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 $\mathtt{I}_{\mathtt{f}}$ 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.

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 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. [9]).

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

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

Semantics of Logic Programs. The semantics of a logic program $\mathcal{P}$ is determined by selecting a particular model, or a set of models, of $\mathcal{P}$. In our context we will consider two possible intended semantics over bilattices, namely the *Kripke-Kleene* (KK) and the *Well-Founded semantics* (WF) [9, 21], which are well-established semantics for non-monotonic negation over bilattices. It is well-know that the WF semantics is more informative (provides more knowledge) than the KK semantics.

Example 2. *Consider $\mathcal{FOUR}$ and $\mathcal{P} = \{(p \leftarrow p), (q \leftarrow \neg r), (r \leftarrow \neg q \wedge \neg p)\}$. Let us identify an interpretation I as a triple for the truth-values of $\langle p, q, r\rangle$. Then the models of $\mathcal{P}$ are $I_1 = \langle \bot, \bot, \bot\rangle$, $I_2 = \langle \bot, \mathtt{t}, \mathtt{f}\rangle$, $I_3 = \langle \mathtt{f}, \bot, \bot\rangle$, $I_4 = \langle \mathtt{f}, \mathtt{f}, \mathtt{t}\rangle$, $I_5 = \langle \mathtt{f}, \mathtt{t}, \mathtt{f}\rangle$, $I_6 = \langle \mathtt{f}, \top, \top\rangle$, $I_7 = \langle \mathtt{t}, \mathtt{t}, \mathtt{f}\rangle$, $I_8 = \langle \top, \mathtt{t}, \mathtt{f}\rangle$ and $I_9 = \langle \top, \top, \top\rangle$. The KK semantics will be I_1, while the WF semantics will be I_3. Note that $I_1 \preceq_k I_3$.*

Formally, the *Kripke-Kleene semantics* 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, by means of the *immediate consequence operator* $\Phi_\mathcal{P}$. For an interpretation I, for any ground atom A with (unique) $A \leftarrow \varphi \in \mathcal{P}^*$, $\Phi_\mathcal{P}(I)(A) = I(\varphi)$. We can show that (based on [9, 14]) the function $\Phi_\mathcal{P}$ is $\preceq_k$-continuous over $\mathcal{I}(\mathcal{B})$, 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 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 3. *Consider $\mathcal{K}([0,1]_\mathbb{Q})$, the function $f(\langle x, 1\rangle) = \langle \frac{x+a}{2}, 1\rangle$ ($0 < a \leq 1, a \in \mathbb{Q}$), and $\mathcal{P} = \{A \leftarrow f(A)\}$. Then the KK model is attained after ω steps of $\Phi_\mathcal{P}$ iterations over $\mathtt{I}_\bot = \langle 0, 1\rangle$ and is $KK(\mathcal{P})(A) = \langle a, 1\rangle$.*

The *Well-Founded semantics* over bilattices is derived directly from Fitting's formulation [9]. Informally, an interpretation I is the *well-founded model* of a logic program $\mathcal{P}$ if I it the $\preceq_k$-least interpretation satisfying $I = I'$, where I' is computed according to the so-called *Gelfond-Lifschitz transformation*: (i) 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 (ii) compute the truth-minimal model I' of $\mathcal{P}^I$. For instance, given $\mathcal{P}$ and I_3 in Example 2, $\mathcal{P}^{I_3}$ is $\{(p \leftarrow p), (q \leftarrow \bot), (r \leftarrow \bot \wedge \mathtt{t})\}$, whose $\preceq_t$-least model is I_3. Also I_3 is the $\preceq_k$-least model satisfying the above condition. Therefore, I_3 is well-founded model. Note that the $\preceq_t$-least model of $\mathcal{P}^{I_1}$ ($= \mathcal{P}^{I_3}$), is I_3, so I_1 does not satisfy the Gelfond-Lifschitz transformation. Formally, Fitting [9] relies on a binary immediate consequence operator $\Psi_\mathcal{P}$, which accepts two input interpretations I and J 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 the bilattice is defined as follows: for an atom A: $(I \triangle J)(A) = I(A)$ and $(I \triangle J)(\neg A) = \neg J(A)$. Pseudo-interpretations are extended to non-literals in the obvious way. 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 atom A with $A \leftarrow \varphi \in \mathcal{P}^*$, satisfies $\Psi_{\mathcal{P}}(I, J)(A) = (I \triangle J)(\varphi)$. 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 [9], we can show that 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 well-founded semantics, Fitting [9] further introduces the $\Psi'_{\mathcal{P}}$ operator, whose $\preceq_k$-least fixed-point will be the WF model 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.

$$\Psi'_{\mathcal{P}}(I) = \text{lfp}_{\preceq_t}(\lambda x.\Psi_{\mathcal{P}}(x, I)) \; . \tag{1}$$

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 $\mathtt{I_f}$ and the limit is attained in at most ω iterations. In particular, we can show that 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 $\mathcal{P}$. Therefore, the set of fixed-points of $\Psi'_{\mathcal{P}}$ is a complete lattice under $\preceq_k$ and, thus, $\Psi'_{\mathcal{P}}$ has a $\preceq_k$-least fixed-point, which is denoted $WF(\mathcal{P})$. $WF(\mathcal{P})$ is the *Well-Founded model* of $\mathcal{P}$. Of course, the well-founded model can be computed by iterating $\Psi'_{\mathcal{P}}$ starting from $\mathtt{I}_\bot$ and the limit is attained in at most ω iterations.

Example 4. *Consider* $\mathcal{K}([0,1]_{\mathbb{Q}})$ *and* $\mathcal{P} = \{ \; (A \leftarrow A \vee B), \; (B \leftarrow (\neg C \wedge A) \vee \langle 0.3, 0.5\rangle), \; (C \leftarrow \neg B \vee \langle 0.2, 0.4\rangle) \; \}$. *Then the computation of* $KK(\mathcal{P})$*, as* $\preceq_k$*-least fixed-point of* $\Phi_{\mathcal{P}}$*, converges to* $KK(\mathcal{P})(A, B, C) = \langle\langle 0.3, 1\rangle, \langle 0.3, 0.8\rangle, \langle 0.2, 0.7\rangle\rangle$. *The computation of* $WF(\mathcal{P})$*, as* $\preceq_k$*-least fixed-point of* $\Psi'_{\mathcal{P}}$*, converges to* $WF(\mathcal{P})(A, B, C) = \langle\langle 0.3, 0.5\rangle, \langle 0.3, 0.5\rangle, \langle 0.5, 0.7\rangle\rangle$. *Notice that* $KK(\mathcal{P}) \preceq_k WF(\mathcal{P})$*, as expected.*

Example 5. *Consider Example 1. It turns out that the KK semantics is* I_1*, while the WF semantics is* I_2*, where (for ease, we use first letter only)* $I_1(\mathtt{R(j)}) = [0.64, 0.8]$, $I_1(\mathtt{S(j)}) = [0.8, 0.8]$, $I_1(\mathtt{Y(j)}) = [0, 1]$, $I_1(\mathtt{G(j)}) = [0.2, 0.36]$, $I_1(\mathtt{E(j)}) = [0.7, 0.7]$, *while* $I_2(\mathtt{R(j)}) = [0.64, 0.7]$, $I_2(\mathtt{S(j)}) = [0.8, 0.8]$, $I_2(\mathtt{Y(j)}) = [0, 0]$, $I_2(\mathtt{G(j)}) = [0.3, 0.36]$, $I_2(\mathtt{E(j)}) = [0.7, 0.7]$. *Note that* $I_1 \preceq_k I_2$. *In fact,* I_2 *establish that* `john`*'s degree of* `Risk` *is in between* $[0.64, 0.7]$*, while* I_1 *is less* precise. *Also note that* $I_2(\mathtt{Y(j)}) = [0, 0]$ *(*$=$ `false`*), while* $I_1(\mathtt{Y(j)}) = [0, 1]$ *(*$=$ `unknown`*).*

3 Top-Down Query Answering

Given a logic program $\mathcal{P}$ and either the KK or the WF 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, 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 and bilattices for which we compute the least fixed-point in a top-down style. The idea is the following. 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 $\mathcal{P}$ 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 $\mathcal{B}$ (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 $A_i \in B_\mathcal{P}$, $I(A_i) = O(I)(A_i)$. Therefore, we may identify the fixed-points of O as the solutions over $\mathcal{B}$ of the system of equations of the following form:

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

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 2, the fixed-points of the $\Phi_\mathcal{P}$ operator are the solutions over a bilattice of the system of equations ($p \mapsto x_1, q \mapsto x_2, r \mapsto x_3$)

$$x_1 = x_1 \,, \;\; x_2 = \neg x_3 \,, \;\; x_3 = \neg x_2 \wedge \neg x_1 \,. \tag{3}$$

It is easily verified that all nine interpretations I_i in Example 2 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 $\mathcal{P}$.

Now, at first present the general procedure for the top-down computation of the value of variable in the $\preceq$-least solution of 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. $\mathtt{s}(x_i) = \{x_{i_1}, \ldots, x_{i_{a_i}}\}$ (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 (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 $\mathcal{S} = \langle \mathcal{L}, V, \boldsymbol{f} \rangle$, where $\mathcal{L}$ is a lattice, $V = \{x_1, ..., x_n\}$ are the variables and $\boldsymbol{f} = \langle f_1, ..., f_n \rangle$ is the tuple of functions. As it is well known, a monotonic equation system as (2) has a $\preceq$-least solution, $\mathrm{lfp}_\preceq(\boldsymbol{f})$, the $\preceq$-least fixed-point of $\boldsymbol{f}$ is given as the least upper bound of the $\preceq$-monotone sequence, $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i, \ldots$, where $\boldsymbol{y}_0 = \bot$ and $\boldsymbol{y}_{i+1} = \boldsymbol{f}(\boldsymbol{y}_i)$.

Example 6. *Consider Example 4. The equational system is* $\{x_A = x_A \vee x_B, \;\; x_B = (\neg x_C \wedge x_A) \vee \langle 0.3, 0.5 \rangle, \;\; x_C = \neg x_B \vee \langle 0.2, 0.4 \rangle\}$. *The* $\preceq_k$*-least fixed-point computation is* $\boldsymbol{y}_0 = \bot = \langle [0,1]_\mathbb{Q}, [0,1]_\mathbb{Q}, [0,1]_\mathbb{Q} \rangle$ *(the triples represent* (x_A, x_B, x_C)*),*

$\boldsymbol{y}_1 = \langle [0,1]_{\mathbb{Q}}, [0.3,1]_{\mathbb{Q}}, [0.2,1]_{\mathbb{Q}} \rangle$, $\boldsymbol{y}_2 = \langle [0.3,1]_{\mathbb{Q}}, [0.3,0.8]_{\mathbb{Q}}, [0.2,0.7]_{\mathbb{Q}} \rangle$ *and* $\boldsymbol{y}_3 = \boldsymbol{y}_2$*, which corresponds to the KK model of the program, as expected.*

Informally our algorithm works as follows. 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 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 algorithm is given below.

Procedure $Solve(\mathcal{S}, Q)$
Input: $\preceq$-monotonic system $\mathcal{S} = \langle \mathcal{L}, V, \boldsymbol{f} \rangle$, 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

The variable $\mathtt{dg}$ collects the variables that may influence the value of the query variables, the array variable $\mathtt{exp}$ traces the equations that has been "expanded" (the body variables are put into the active list), while the variable $\mathtt{in}$ 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 [3, 5]. Indeed, it is a generalization of it to arbitrary monotone equational systems over lattices.

Example 7. *Consider Example 6 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. $x_i := x_B, \mathtt{A} := \{x_A\}, \mathtt{dg} := \{x_A, x_B, x_C\}, r := \langle 0.3, 1\rangle, \mathtt{v}(x_B) := \langle 0.3, 1\rangle, \mathtt{A} := \{x_A, x_C\}, \mathtt{exp}(x_B) := \mathtt{true}, \mathtt{in} := \{x_A, x_B, x_C\}$
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. $x_i := x_A, \mathtt{A} := \emptyset, r := \langle 0.3, 1\rangle$
9. `stop`. `return` $\mathtt{v}(x_A, x_B, x_C) = \langle\langle 0.3, 1\rangle, \langle 0.3, 0.8\rangle, \langle 0.2, 0.7\rangle\rangle$

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 $\mathcal{S} = \langle \mathcal{L}, V, \boldsymbol{f}\rangle$, 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 $h(\mathcal{K}([0,1]_\mathbb{Q})) = \omega$. It can be shown that the above algorithm behaves correctly.

Proposition 8. *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \boldsymbol{f}\rangle$, then there is a limit ordinal λ such that after $|\lambda|$ steps $Solve(\mathcal{S}, Q)$ determines a set $B \subseteq V$, with $Q \subseteq B$ such that the mapping* v *equals lfp$_\preceq(\boldsymbol{f})$ on B, i.e. $\mathtt{v}_{|B} =$ lfp$_\preceq(\boldsymbol{f})_{|B}$.*

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

Proposition 9. *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \boldsymbol{f}\rangle$. If the computing cost of each function in $\boldsymbol{f}$ 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 3). 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 $\mathcal{S} = \langle \mathcal{L}, V, \boldsymbol{f}\rangle$. 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 [2] 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 $X \subseteq \mathcal{B}$. More concretely, over the interval bilattice on $[0,1]_\mathbb{Q}$, $\min, \max, 1 - x$ and Lukasiewicz t-norm and t-conorm, $\max(x + y - 1, 0), \min(x + y, 1)$ have a finite generation, while e.g. the product t-norm $x \cdot y$ and its t-conorm $x + y - x \cdot y$ have not. Note also that

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

Proposition 10. *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \boldsymbol{f} \rangle$. Then $Solve$ terminates iff $\boldsymbol{f}$ has a finite generation. If the cost of computing each of the functions in $\boldsymbol{f}$ 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 $[\bot]_{\boldsymbol{f}}$.*

The second condition, which guarantees the termination of $Solve$, is inspired directly by [4] 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 $\mathcal{S} = \langle \mathcal{L}, V, \boldsymbol{f} \rangle$. We say that $\boldsymbol{f}$ 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 3 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 $\mathcal{K}([0,1]_{\mathbb{Q}})$ is. The idea is to prevent the existence of an infinite ascending chain of the form $\bot \prec_k \boldsymbol{f}(\bot) \prec_k \ldots \prec_k \boldsymbol{f}^m(\bot) \prec_k \ldots$. In fact, roughly, consider a $\preceq_k$-monotone function $\boldsymbol{f} = \boldsymbol{g} \circ \boldsymbol{h}$, where $\boldsymbol{g}$ is a bounded function, while $\boldsymbol{h}$ is the composition of constants in $\mathcal{B}$ or functions among $\vee, \wedge, \oplus, \otimes$ and $\neg$. Then $\bot \preceq_k \boldsymbol{f}(\bot) = \boldsymbol{g} \circ \boldsymbol{h}(\bot) = \boldsymbol{g}(\boldsymbol{h}(\bot)) \preceq_k \boldsymbol{h}(\bot)$. But $\boldsymbol{h}$ has a finite generation and, thus, so has $\boldsymbol{f}$. The argument for $\boldsymbol{f} = \boldsymbol{h} \circ \boldsymbol{g}$ is similar. Therefore:

Proposition 11. *Given a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \boldsymbol{f} \rangle$, where $\boldsymbol{f}$ is bounded. Then $Solve$ terminates.*

Note that for bounded functions $\boldsymbol{f} = \boldsymbol{g} \circ \boldsymbol{h}$, the height of $[\bot]_{\boldsymbol{f}}$ is given by the height of $[\bot]_{\boldsymbol{h}}$. We believe that this latter height is bounded by the number $n = |V|$ as we conjecture that $\boldsymbol{h}^n(\bot) = \boldsymbol{h}^{n+1}(\bot)$ (this is compatible with [4]). 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 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 $\mathcal{S}_{KK}(\mathcal{P}) = \langle \mathcal{L}, V, \boldsymbol{f}_{\mathcal{P}} \rangle$ 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 $\boldsymbol{f}_{\mathcal{P}}$ 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 $Solve$. $Solve_{KK}$ behaves correctly (see Example 7).

Proposition 12. *Let $\mathcal{P}$ and $?A$ be a logic program and a query, respectively. Then $KK(\mathcal{P})(A) = Solve_{KK}(\mathcal{P}, \{?A\})$* [3].

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 9 it follows immediately that 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, often the cost of computing each of the functions of $\boldsymbol{f}_{\mathcal{P}}$ is in $O(1)$. By observing that $|B_{\mathcal{P}}|a$ is in $O(|\mathcal{P}|)$ we immediately have that in this case the complexity is $O(|\mathcal{P}|h)$. It follows that over the 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 propositional logic programs, which is linear. The computational complexity of the case where the height of the bilattice is not finite is determined by Proposition 10 and Proposition 11. 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 $\Psi'_{\mathcal{P}}(I) = \text{lfp}_{\preceq_t}(\lambda x.\Psi_{\mathcal{P}}(x, I))$. 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 2. 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.: it is not hard to see that, due to Equation (1), 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. That is, 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 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 $\neg I(x)$, but $\mathcal{S}(\mathcal{P}^I)$ is based on the $\preceq_t$ order rather than on $\preceq_k$. Then it can be verified that $Solve(\mathcal{S}(\mathcal{P}^I), Q)$ 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 $\mathtt{v}_{|B} = \Psi'_{\mathcal{P}}(I)_{|B}$. Moreover, from a computational

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

complexity point of view, the same properties of $Solve$ hold for $Solve(\mathcal{S}(\mathcal{P}^I), Q)$ as well. Finally, $Solve_{WF}(\mathcal{P}, ?A)$ is as $Solve_{KK}(\mathcal{P}, ?A)$, except that Step 4. is replaced with the statements $\mathtt{Q}:= \mathtt{s}(x_i)$; $\mathtt{I}:= \mathtt{v}$; $\mathtt{v}':= Solve(\mathcal{S}(\mathcal{P}^{\mathtt{I}}), \mathtt{Q})$; $r:= f_i(\mathtt{v}'(x_{i_1}), ..., \mathtt{v}'(x_{i_{a_i}}))$. It can be shown that the following holds:

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 14. *Consider Example 6 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 7. 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. $\mathtt{stop}.\ \mathtt{return}\ \mathtt{v}(x_A, x_B, x_C)_{|x_A} = \langle\langle 0.3, 0.5\rangle, \langle 0.3, 0.5\rangle, \langle 0.5, 0.7\rangle\rangle_{|x_A} = \langle 0.3, 0.5\rangle$

The computational complexity analysis of $Solve_{WF}$ parallels the one we have made for $Solve_{KK}$. If the height of a bilattice is finite then, 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 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 bilattice 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 $O(|\mathcal{P}|^2)$. 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^2 steps for $Solve_{WF}$. In case the functions have a finite generation or are bounded, Proposition 10 and Proposition 11 can be applied.

4 Conclusions

We have presented a general top-down algorithm to answer queries for normal logic programs over lattices as well as over bilattices (for which no top-down algorithm was known yet). We believe that its interest relies on the fact that many approaches to paraconsistency and uncertainty of logic programming with or without non-monotonic negation are based on bilattices or lattices, respectively. Therefore, the presented algorithms give us general query-solving procedures for many of them.

References

1. N. D. Belnap. A useful four-valued logic. In G. Epstein and J. M. Dunn, editors, *Modern uses of multiple-valued logic*, pages 5–37. Reidel, Dordrecht, NL, 1977.
2. E. Böhler, C. Glaer, B. Schwarz, and K. Wagner. Generation problems. In *29th Int. Symp. on Mathematical Foundations of Computer Science (MFCS-04)*, LNCS 3153, pages 392–403. Springer Verlag, 2004.
3. W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, 1996.
4. C. V. Damásio, J. Medina, and M. O. Aciego. Sorted multi-adjoint logic programs: Termination results and applications. In *Proc. of the 9th Europ. Conf. on Logics in Art. Intelligence (JELIA-04)*, LNCS 3229, pages 252–265. Springer Verlag, 2004.
5. C. V. Damásio, J. Medina, and M. O. Aciego. A tabulation proof procedure for residuated logic programming. In *Proc. of the 6th Europ. Conf. on Art. Intelligence (ECAI-04)*, 2004.
6. C. V. Damásio and L. M. 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.
7. C. Viegas Damásio and L. M. Pereira. Antitonic logic programs. In *Proc. of the 6th Int. Conf. on logic programming and Nonmonotonic Reasoning (LPNMR-01)*, LNCS 2173. Springer-Verlag, 2001.
8. D. Dubois, J. Lang, and H. Prade. Towards possibilistic logic programming. In *Proc. of the 8th Int. Conf. on Logic Programming (ICLP-91)*, pages 581–595. The MIT Press, 1991.
9. M. C. Fitting. Fixpoint semantics for logic programming - a survey. *Theoretical Computer Science*, 21(3):25–51, 2002.
10. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the 5th Int. Conf. on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
11. M. L. Ginsberg. Multi-valued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.
12. M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
13. Laks V.S. Lakshmanan and N. Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.
14. Y. Loyer and U. Straccia. The approximate well-founded semantics for logic programs with uncertainty. In *28th Int. Symp. on Mathematical Foundations of Computer Science (MFCS-2003)*, LNCS 2747, pages 541–550, 2003. Springer-Verlag.
15. T. Lukasiewicz. Probabilistic logic programming. In *Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98)*, pages 388–392, 1998.
16. T. Lukasiewicz. Fixpoint characterizations for many-valued disjunctive logic programs with probabilistic semantics. In *Proc. of the 6th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, LNAI 2173, pages 336–350. Springer-Verlag, 2001.
17. R. Ng and V.S. Subrahmanian. Stable model semantics for probabilistic deductive databases. In *Proc. of the 6th Int. Sym. on Methodologies for Intelligent Systems (ISMIS-91)*, LNAI 542, pages 163–171. Springer-Verlag, 1991.
18. R. Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1993.
19. U. Straccia. Top-down query answering for logic programs over bilattices. Technical Report, ISTI-CNR, Pisa, Italy, 2004.

20. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, (5):285–309, 1955.
21. A. van Gelder, K. A. Ross, and J. S. Schlimpf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, January 1991.
22. P. Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124:361–370, 2004.