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

Technical 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) $\beta_i$, $1 \leq i \leq n$.” Here $f$ is an $n$-ary computable function and $\beta_i$ 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 $A \alpha \leftarrow B_1, \ldots, B_n$, which says that the certainty associated with the implication $B_1 \land \ldots \land B_n \rightarrow A$ is $\alpha$. Computationally, given an assignment $v$ of certainties to the $B_i$s, 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 \( L = (L, \preceq) \) such that every two element set \( \{x, y\} \subseteq L \) has a least upper bound, \( \text{lub}_\preceq (x, y) \) (called the join of \( x \) and \( y \)), and a greatest lower bound, \( \text{glb}_\preceq (x, y) \) (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 \( (L, \preceq) \) 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 \( (L, \preceq) \) and a subset of elements \( S \subseteq L \), with \( \preceq \)-least and \( \preceq \)-greatest we will always mean \( \text{glb}_\preceq (S) \) and \( \text{lub}_\preceq (S) \), respectively. With \( \text{min}_\preceq (S) \) we denote the set of minimal elements in \( S \) w.r.t. \( \preceq \) i.e., \( \text{min}_\preceq (S) = \{x \in S : \nexists y \in S \; s.t. \; y \prec x\} \). Note that while \( \text{glb}_\preceq (S) \) is unique, \( |\text{min}_\preceq (S)| \geq 1 \) may hold. If \( \text{min}_\preceq (S) \) is a singleton \( \{x\} \), for convenience we may also write \( x = \text{min}_\preceq (S) \) in place of \( \{x\} = \text{min}_\preceq (S) \). An operator on a lattice \( (L, \preceq) \) is a function from \( L \) to \( L \), \( f: L \rightarrow L \). 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 \( (L, \preceq) \). 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 \( i \geq 0 \),

\[
\begin{align*}
x_0 & = \bot, \\
x_{i+1} & = f(x_i)
\end{align*}
\]

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

\[
x_\lambda = \text{lub}_\preceq \{x_i : i < \lambda\} \quad (\text{respectively, } x_\lambda = \text{glb}_\preceq \{x_i : i < \lambda\}) .
\]

We denote the \( \preceq \)-least and the \( \preceq \)-greatest fixed-point by \( \text{lfp}_\preceq (f) \) and \( \text{gfp}_\preceq (f) \), 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 \( \text{FOUR} \), is due to Belnap [5], who introduced a logic intended to deal with incomplete and/or inconsistent information – see also [4, 15]. \( \text{FOUR} \) already illustrates many of the basic properties concerning bilattices. Essentially, \( \text{FOUR} \) extends the classical truth set \( \{\bot, \top\} \) to \( \{\bot, \bot, \top, \top\} \), where \( \bot \) stands unknown, and \( \top \) stands for inconsistent. \( \text{FOUR} \) has two quite intuitive and natural ‘orthogonal’ orderings, \( \preceq_k \) and \( \preceq_t \) (see Figure 1), each giving to \( \text{FOUR} \) 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 \( \text{FOUR} \), \( \bot \preceq_k \bot \preceq_k \top, \bot \preceq_k \top \preceq_k \top, \top \preceq_t \bot \preceq_t \bot \preceq_t \top \), \( \bot \preceq_t \bot \preceq_t \top \) and \( \bot \preceq_t \top \preceq_t \top \).
A bilattice is a structure \( \langle B, \preceq_l, \preceq_r \rangle \) where \( B \) is a non-empty, countable set and \( \preceq_l \) and \( \preceq_r \) are both partial orderings giving \( B \) the structure of a complete lattice with a top and bottom element. Meet (or greatest lower bound) and join (or least upper bound) under \( \preceq_l \), denoted \( \land \) and \( \lor \), correspond to extensions of classical conjunction and disjunction. On the other hand, meet and join under \( \preceq_r \) 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 \( \preceq_r \) are denoted \( t \) and \( f \), and top and bottom under \( \preceq_l \) are denoted \( \top \) and \( \bot \), respectively. We will assume that bilattices are infinitary distributive bilattices in which all distributive laws connecting \( \land, \lor, \otimes \) and \( \oplus \) hold. We also assume that every bilattice satisfies the infinitary interlacing conditions, i.e., each of the lattice operations \( \land, \lor, \otimes \) and \( \oplus \) is monotone w.r.t. both orderings. An example of interlacing condition is: \( x \preceq_l y \) and \( x' \preceq_r y' \) implies \( x \otimes x' \preceq_l y \otimes y' \). Finally, we assume that each bilattice has a negation, i.e., an operator \( \neg \) that reverses the \( \preceq_l \) ordering, leaves unchanged the \( \preceq_r \) ordering, and verifies \( \neg \neg x = x \).

Additionally, we provide a family \( \mathcal{F} \) of \( \preceq_r \)-continuous binary and unary functions \( f: B \times B \to B \) and \( f: B \to B \), that is, for any \( \preceq_k \)-monotone chain \( x_0, x_1, \ldots \) of values in \( B \), \( f(\oplus_i x_i) = \oplus_i f(x_i) \) and for any \( \preceq_r \)-monotone chain \( x_0, x_1, \ldots \) of values in \( B \), \( f(\vee_i x_i) = \vee_i f(x_i) \). The binary case is similar. Notably, it is not difficult to see that \( \land, \lor, \otimes \) and \( \oplus \) are both \( \preceq_k \)-continuous and \( \preceq_l \)-continuous, while \( \neg \) is \( \preceq_k \)-continuous but not \( \preceq_l \)-continuous (it is \( \preceq_l \)-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 \( \langle L_1 \times L_2, \preceq_{1 \odot 2} \rangle \), where

- \( \langle x_1, x_2 \rangle \preceq_{1 \odot 2} \langle y_1, y_2 \rangle \) 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
belief and doubt in the same way, then negation can be defined as \( \neg(x, y) = (y, x) \). That is, negation switches the roles of belief and doubt. In Figure 1 we report the bilattice based on \( L_1 = L_2 = \{ \bot, 1, \top \} \) and order \( \preceq_1 = \preceq_2 = \preceq \), where \( \bot \preceq \top \preceq \top \). 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 \( (x, y) \in L \times L \) as indicator for degree of belief and doubt, \( (x, y) \) is interpreted as the set of elements \( z \in L \) such that \( x \preceq z \preceq y \). Therefore, a pair \( (x, y) \) is interpreted as an interval. An interval \( (x, y) \) may be seen as an approximation of an exact value. Formally, given a distributive lattice \( \langle L, \preceq \rangle \), the bilattice of intervals, denoted \( K(L) \), is \( \langle L \times L, \preceq_1, \preceq_2 \rangle \), where:

- \( \langle x_1, x_2 \rangle \preceq_1 \langle y_1, y_2 \rangle \) if \( x_1 \preceq y_1 \) and \( x_2 \preceq y_2 \);
- \( \langle x_1, x_2 \rangle \preceq_2 \langle y_1, y_2 \rangle \) 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 \( \neg(x, y) = (\neg y, \neg x) \), where \( \neg \) is a negation operator on \( L \). As an example, in Figure 1 we report the bilattice \( K([0, 1] \cap \mathbb{Q}) \). 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 \( (x, y) \) 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 \( K(L) \) 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, \( \ell \), is of the form \( A \) or \( \neg A \), where \( A \) is an atom. A **formula**, \( \varphi \), is an expression built up from the literals, the members of a bilattice \( B \) using \( \land, \lor, \otimes \) 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 FOUR, \( f(p \land q, r \otimes f) \oplus v \) is a formula.

A **rule** is of the form \( A \leftarrow \varphi \), where \( A \) is an atom and \( \varphi \) is a formula. The atom \( A \) is called the head, and the formula \( \varphi \) 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 \( A \leftarrow \top \) to \( \mathcal{P}^* \);² and (ii) replace several rules in \( \mathcal{P}^* \) having same head, \( A \leftarrow \varphi_1, A \leftarrow \varphi_2, \ldots \) with \( A \leftarrow \varphi_1 \lor \varphi_2 \lor \ldots \). 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 \( \langle B, \preceq_1, \preceq_2 \rangle \) is a mapping from atoms to members of \( B \). An interpretation \( I \) is extended from atoms to formulae as follows:

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

²It is a standard practice in logic programming to consider such atoms as false.
functions. The truth and knowledge orderings are extended from B to the set \( \mathcal{I}(B) \) of all interpretations point-wise: \( (i) \ I_1 \preceq_i I_2 \) iff \( I_1(A) \preceq_1 I_2(A) \), 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 \( (I_1 \land I_2)(A) = I_1(A) \land I_2(A) \), and similarly for the other operations. With \( I_1 \) and \( I_\perp \) we denote the bottom interpretations under \( \preceq_1 \) and \( \preceq_k \) respectively (they map any atom into \( \top \) and \( \bot \), respectively). It is easy to see that \( (\mathcal{I}(B), \preceq_1, \preceq_k) \) 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 \( A \Leftarrow \bot \in \mathcal{P}^* \)), then \( \varphi = \varphi_1 \lor \ldots \lor \varphi_n \) and \( \varphi_i = L_{i1} \land \ldots \land L_{in} \). Furthermore, a classical total interpretation is an interpretation over \( \mathcal{FOL} \) such that an atom is mapped into either \( \bot \) or \( \top \). 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 \( \bot \), \( \top \), 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 \( \Leftarrow \). Indeed, usually a model has to satisfy \( I(\varphi) \preceq_k I(A) \) only, i.e. \( A \Leftarrow \varphi \in \mathcal{P}^* \) specifies the necessary condition on \( A \), “\( A \) is at least as true as \( \varphi \).” 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.

\[
p \Leftarrow p
\]
\[
q \Leftarrow \neg r
\]
\[
r \Leftarrow \neg q \land \neg p
\]

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 \( A \)

\[
\Phi_\mathcal{P}(I)(A) = I(\varphi),
\]
where $A \leftarrow \varphi \in \mathcal{P}^*$. 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 $I_\bot$ and is attained after at most $\omega$ iterations.

**Example 2** Consider the bilattice $\mathcal{K}(\langle 0, 1 \rangle \cap \mathbb{Q})$, the function $f \in \mathcal{F}$, $f((x, 1)) = \langle \frac{x + a}{2}, 1 \rangle$ for $0 < a \leq 1$, $a \in \mathbb{Q}$, and the logic program $\mathcal{P} = \{ A \leftarrow f(A) \}$. Then the Kripke-Kleene model is attained after $\omega$ steps of $\Phi_\mathcal{P}$ iterations over $I_\bot = \langle 0, 1 \rangle$ 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 $\{f, t\}$ 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 $\{ (p \leftarrow p), (q \leftarrow \bot), (r \leftarrow \bot \land \top) \}$, 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 $\mathcal{P}^{I_5}$ (= $\mathcal{P}^{I_3}$), is $I_5$, so $I_5$ 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 $(\mathcal{I}(\mathcal{B}), \preceq_t, \preceq_k)$. The notion of pseudo-interpretation $I \triangle J$ over

<table>
<thead>
<tr>
<th>$I_i \models \mathcal{P}$</th>
<th>$p$</th>
<th>$q$</th>
<th>$r$</th>
<th>$KK$</th>
<th>$WF$</th>
<th>stable models</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_2$</td>
<td>⊥</td>
<td>t</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_3$</td>
<td>t</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_4$</td>
<td>t</td>
<td>⊥</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_5$</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_6$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_7$</td>
<td>t</td>
<td>⊥</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_8$</td>
<td>⊤</td>
<td>t</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_9$</td>
<td>⊤</td>
<td>⊤</td>
<td>⊤</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

³Recall that all atoms are head of exactly one rule in $\mathcal{P}^*$.³
the bilattice is defined as follows ($I$ gives meaning to positive literals, while $J$ gives meaning to negative literals): for a pure ground atom $A$:

$$(I \triangle J)(A) = I(A)$$

$$(I \triangle J)(\neg A) = \neg J(A).$$

Pseudo-interpretations are extended to non-literals in the obvious way. For instance, $(I \triangle J)(f(\neg A \land B)) = f((I \triangle J)(\neg A) \land (I \triangle J)(B)) = f(\neg J(A) \land I(B)).$

We can now define $\Psi_P$ as follows. For $I, J \in \mathcal{I}(\mathcal{B})$, $\Psi_P(I, J)$ is the interpretation, which for any ground atom $A$ is such that

$$\Psi_P(I, J)(A) = (I \triangle J)(\varphi),$$

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

**Proposition 3** In the space of interpretations the operator $\Psi_P$ is $\preceq_k$-continuous in both arguments, $\preceq_k$-continuous in its first argument and $\preceq_k$-antitone in its second argument.

To define the stable model semantics, Fitting [20] introduces the $\Psi_P$ operator, whose fixed-points will be the stable models of a program. For any interpretation $I$, $\Psi_P(I)$ is the $\preceq_k$-least fixed-point of the operator $\lambda x. \Psi_P(x, I)$, i.e.

$$\Psi_P(I) = \text{lfp}_{\preceq_k}(\lambda x. \Psi_P(x, I)).$$

Due to the $\preceq_k$-continuity of $\Psi_P$ on its first argument, $\Psi_P$ is well defined. $\Psi_P(I)$ can be computed by iterating $\Psi_P(x, I)$ over $\mathcal{I}_I$ and the limit is attained in at most $\omega$ iterations. Additionally, it can be shown that

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

A stable model for a logic program $\mathcal{P}$ is a fixed-point of $\Psi_P$. Therefore, the set of fixed-points of $\Psi_P$, i.e. the set of stable models of $\mathcal{P}$, is a complete lattice under $\preceq_k$ and, thus, $\Psi_P$ 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. $WF(\mathcal{P}) = \min_{\preceq_k}(\{I: I \text{ stable model of } \mathcal{P}\})$.

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

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

$$A \leftarrow A \lor B$$

$$B \leftarrow (\neg C \land A) \lor (0.3, 0.5)$$

$$C \leftarrow \neg B \lor (0.2, 0.4)$$

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

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$K_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(0, 1)$</td>
<td>$(0, 1)$</td>
<td>$(0, 1)$</td>
<td>$K_0$</td>
</tr>
<tr>
<td>$(0, 1)$</td>
<td>$(0.3, 1)$</td>
<td>$(0.2, 1)$</td>
<td>$K_1$</td>
</tr>
<tr>
<td>$(0.3, 1)$</td>
<td>$(0.3, 0.8)$</td>
<td>$(0.2, 0.7)$</td>
<td>$K_2$</td>
</tr>
<tr>
<td>$(0.3, 1)$</td>
<td>$(0.3, 0.8)$</td>
<td>$(0.2, 0.7)$</td>
<td>$K_3 = K_2 = KK(\mathcal{P})$</td>
</tr>
</tbody>
</table>

\[Note that negation may appear in front of a literal only.\]
Note that knowledge increases during the computation as the intervals becomes more precise, i.e. \(K_i \preceq K_{i+1}\).

The following table shows us the computation of the well-founded semantics of \(\mathcal{P}\), \(WF(\mathcal{P})\), as \(\preceq^W\)-least fixed-point of \(\Psi_P\).

<table>
<thead>
<tr>
<th>(v_i^P)</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(v_i^P)</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(W_{i+1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(v_0^P)</td>
<td>(0,0)</td>
<td>(0,0)</td>
<td>(0,0)</td>
<td>(0,1)</td>
<td>(0,1)</td>
<td>(0,1)</td>
<td>(W_0)</td>
<td></td>
</tr>
<tr>
<td>(v_0^P)</td>
<td>(0,0)</td>
<td>(0.3,0.5)</td>
<td>(0.1)</td>
<td>(0.3,0.5)</td>
<td>(0.3,0.5)</td>
<td>(0.1)</td>
<td>(W_1)</td>
<td></td>
</tr>
<tr>
<td>(v_1^P)</td>
<td>(0.3,0.5)</td>
<td>(0.3,0.5)</td>
<td>(0.1)</td>
<td>(0.3,0.5)</td>
<td>(0.3,0.5)</td>
<td>(0.1)</td>
<td>(W_2)</td>
<td></td>
</tr>
<tr>
<td>(v_2^P)</td>
<td>(0.3,0.5)</td>
<td>(0.3,0.5)</td>
<td>(0.1)</td>
<td>(0.3,0.5)</td>
<td>(0.3,0.5)</td>
<td>(0.1)</td>
<td>(W_3 = W_2 = WF(\mathcal{P}))</td>
<td></td>
</tr>
</tbody>
</table>

Notice that \(W_i \preceq W_{i+1}\) and \(KK(\mathcal{P}) \preceq WF(\mathcal{P})\), 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 \(I(\mathcal{P})\) 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 \((B, \preceq_1, \preceq_2)\) 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 \(\mathcal{P}^\ast\) 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 \(B\) (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 \(x_1, \ldots, x_n\). For an immediate consequence operator \(O\), e.g. \(\Phi_P\), 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 \(B\) of the system of equations of the following form:

\[
x_1 = f_1(x_{11}, \ldots, x_{1n})
\]

\[
x_n = f_n(x_{n1}, \ldots, x_{nn})
\]

(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 \(B\), 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 \(\Phi_\rho\) operator are the solutions over a bilattice of the system of equations \((p \mapsto x_1, q \mapsto x_2, r \mapsto x_3)\)

\[
\begin{align*}
  x_1 &= x_1, \\
  x_2 &= x_3, \\
  x_3 &= x_2 \land \neg x_1.
\end{align*}
\]

(3)

It is easily verified that all nine interpretations \(I_i\) in Example 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 \(\preceq\)-least solution and corresponds to the Kripke-Kleene model of \(P\).

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 \(L = \langle L, \preceq \rangle\). Then, we will customize it to the particular case of the Kripke-Kleene semantics and the Well-founded semantics.

We use some auxiliary functions. \(s(x)\) denotes the set of sons of \(x\), i.e.

\[
s(x_i) = \{x_{i_1}, \ldots, x_{i_{a_i}}\}.
\]

\(p(x)\) denotes the set of parents of \(x\), i.e. the set

\[
p(x) = \{x_i : x \in s(x_i)\}.
\]

In the general case, we assume that each function \(f_i : L^{a_i} \rightarrow L\) in Equation (2) is \(\preceq\)-monotone. We also use \(f_i\) in place of \(f_i\), for \(x = x_i\). We refer to the monotone system as in Equation (2) as the tuple \(S = \langle L, V, \vec{f} \rangle\), where \(L\) is a lattice, \(V = \{x_1, \ldots, x_n\}\) are the variables and \(\vec{f} = (f_1, \ldots, f_n)\) is the tuple of functions.

As it is well known, a monotonic equation system as (2) has a \(\preceq\)-least solution, \(\text{lfp}_\preceq(\vec{f})\), 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

\[
\begin{align*}
  \vec{x}_0 &= \bot, \\
  \vec{x}_{i+1} &= \vec{f}(\vec{x}_i).
\end{align*}
\]

(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 \(v(x_i)\), 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 \(A\), 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
Given a monotone system of equations to (2), it can be shown that the above algorithm behaves correctly.

Example 4

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 $S = \langle L, V, \vec{f} \rangle$, where $L = \langle L, \preceq \rangle$, let $h(L)$ 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 $v_1, ..., v_n, ...$ is the cardinal $|\{v_1, ..., v_n, ...\}|$. The cardinal of a countable set $X$ is the least ordinal $\alpha$ such that $\alpha$ and $X$ are equipollent, i.e. there is a bijection from $\alpha$ to $X$. For instance, $h(\text{FOU}(R)) = 2$ w.r.t. $\preceq$ as well as w.r.t. $\preceq_t$, while $h(\chi([0, 1] \cap \mathbb{Q})) = \omega$. Likewise to (2), it can be shown that the above algorithm behaves correctly.

Proposition 5

Given a monotone system of equations $S = \langle L, V, \vec{f} \rangle$, then there is a limit ordinal $\lambda$ such that after $|\lambda|$ steps Solve($S, Q$) determines a set $B \subseteq V$ with $Q \subseteq B$ such that the mapping $v$ equals $\text{lfp}_<(\vec{f})$ on $B$, i.e. $v_B = \text{lfp}_<(\vec{f})_B$.

Example 4

Let us consider Example 3 and the associated equational system $S$ 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. $A = \{x_A\}, x_i = x_A, k = \emptyset, dg = \{x_A, x_B\}, r = \perp, \text{exp}(x_A) = \text{true}, A = \{x_A, x_B\}, in = \{x_A, x_B\}$
2. $x_i = x_B, A = \{x_A\}, dg = \{x_A, x_B, x_C\}, r = (0.3, 1), v(x_B) = (0.3, 1), A = \{x_A, x_C\}, \text{exp}(x_B) = \text{true}, \text{in} = \{x_A, x_B, x_C\}$
3. $x_i = x_C, A = \{x_A\}, r = (0.2, 0.7), v(x_C) = (0.2, 0.7), A = \{x_A, x_B\}, \text{exp}(x_C) = \text{true}$
4. $x_i = x_B, A = \{x_A\}, r = (0.3, 0.8), v(x_B) = (0.3, 0.8), A = \{x_A, x_C\}$
5. $x_i = x_C, A = \{x_A\}, r = (0.2, 0.7)$
6. $x_i = x_A, A = \emptyset, r = (0.3, 1), v(x_A) = (0.3, 1), A = \{x_A, x_B\}$
7. $x_i = x_B, A = \{x_A\}, r = (0.3, 0.8)$
8. $x_i = x_A, A = \emptyset, r = (0.3, 1)$
9. stop return $v(x_A, x_B, x_C) = (0.3, 1, 0.3, 1, 0.3, 0.8, 0.2, 0.7)$
Note that $\text{Solve}(S, \{x_A\})$ answers w.r.t. the Kripke-Kleene semantics as we considered the $\preceq_h$ ordering.

From a Computational point of view, it is easily verified that by means of appropriate data structures, the operations on $\wedge$, $\vee$, $\text{dg}$, $\text{in}$, $\exp$, $\text{p}$ and $\text{son}$ 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

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

(5)

As a consequence,

**Proposition 6** Given a monotone system of equations $S = \langle \mathcal{L}, V, \vec{f} \rangle$. 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 $\text{Solve}$ 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 $\text{Solve}$. We exploit two of such conditions. Consider a monotone system of equations $S = \langle \mathcal{L}, V, \vec{f} \rangle$. Consider a function $f : L \rightarrow L$, 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) if $\{\bot\}f$ 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 $S = \langle \mathcal{L}, V, \vec{f} \rangle$. If $\vec{f}$ has a finite generation, then $\{\bot\}f$ is finite. That is, $\{\bot\}, f(\{\bot\}), f^2(\{\bot\}), \ldots$ is finite. In particular, on induction on the computation of the $\preceq$-least fixed-point of $S$ 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 $\{\bot\}f$. Therefore, the height of $\{\bot\}f$, $h(\{\bot\}f)$, is finite. On the other hand, it can easily be seen that $\text{Solve}$ terminates if the sequence defined by Equation (4), $\{\bot\}, f(\{\bot\}), f^2(\{\bot\}), \ldots$ converges after a finite number of steps. Therefore,

**Proposition 7** Given a monotone system of equations $S = \langle \mathcal{L}, V, \vec{f} \rangle$. Then $\text{Solve}$ 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 $\text{Solve}$ is $O(|V|cah)$, where $h$ is the height of $\{\bot\}f$.

**Example 5** Consider $S = \langle \mathcal{L}, V, \vec{f} \rangle$, where $\mathcal{L}$ has a finite height $h$. Consider a chain $x_1 \preceq x_2 \preceq \ldots \preceq x_h$ 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 $\leq i \leq h$). Then $\vec{f}$ has a finite generation, the variable $x$ enters $h$ times into the $A$ list, and $\text{Solve}$ terminates after $h$ iterations.
The second condition on the functions, which guarantees the termination of $\text{Solve}$, is inspired directly by [12, 14] and is a special case of above. On bilattices, we say that a function $f : B^n \rightarrow B$ is bounded iff $f(x_1, \ldots, x_n) \preceq_k \circ_i x_i$. Now, consider a monotone system of equations $S = \langle L, V, \vec{f} \rangle$. 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 $B$ or one of $\lor, \land, \land$ and $\sim$. For instance, the function in Example 2 is not bounded, while $f_1((x, y)) = \langle \max(0, x + y - 1), 1 \rangle \land \langle 0.3, 0.4 \rangle$ over $K([0, 1] \cap \mathbb{Q})$ is. The idea is to prevent the existence of an infinite ascending chain of the form $\vec{1} \prec_k \vec{f}(\vec{1}) \prec_k \ldots \prec_k \vec{f}^n(\vec{1}) \prec_k \ldots$. 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 $B$ or functions among $\lor, \land, \land$ and $\sim$. Then $\vec{1} \preceq_k \vec{f}(\vec{1}) = \vec{g} \circ \vec{h}(\vec{1}) = \vec{g}(\vec{h}(\vec{1})) \preceq_k \vec{h}(\vec{1})$. 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 $S = \langle L, V, \vec{f} \rangle$, where $\vec{f}$ is bounded. Then $\text{Solve}$ terminates. The cost is as for Proposition 7.

Note that for bounded functions $\vec{f} = \vec{g} \circ \vec{h}$, the height of $\langle \vec{1} \rangle$ is given by the height of $\langle \vec{1} \rangle_U$. We believe that this latter height is bounded by the number $n = |V|$ as we conjecture that $h^n(\vec{1}) = \vec{h}^{n+1}(\vec{1})$ (this is compatible with [12, 14]). This would imply that the worst-case complexity of the algorithm $\text{Solve}$ is $O(|V|^2 \cdot \text{ca})$ 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 $P$ be a logic program and consider $P^*$. As already pointed out, each atom appear exactly once in the head of a rule in $P^*$. The system of equations that we build from $P^*$ is straightforward. Assign to each atom $A$ a variable $x_A$ and substitute in $P^*$ each occurrence of $A$ with $x_A$. Finally, substitute each occurrence of $\leftarrow$ with $\equiv$ and let $S_{KK}(P) = \langle L, V, \vec{f}_P \rangle$ be the resulting equational system (see Equation 3). Of course, $|V| = |B^P|$, $|S_{KK}(P)|$ can be computed in time $O(|P|)$ and all functions in $S_{KK}(P)$ are $\preceq_k$-continuous. As $f_P$ is one to one related to $\Phi_P$, it follows that the $\preceq_k$-least fixed-point of $S_{KK}(P)$ corresponds to the Kripke-Kleene semantics of $P$. The algorithm $\text{Solve}_{KK}(P, ?A)$, first computes $S_{KK}(P)$ and then calls $\text{Solve}(S_{KK}(P), \{x_A \})$ and returns the output $\nu$ on the query variable, where $\nu$ is the output of the call to $\text{Solve}$. It can be shown that $\text{Solve}_{KK}$ behaves correctly (see Example 4).

**Proposition 9** Let $P$ and $?A$ be a logic program and a query, respectively. Then $KK(P)(A) = \text{Solve}_{KK}(P, \{?A \})$.

The extension to a set of query atoms is straightforward.

From a computational point of view, we can avoid the cost of translating $P$ into $S_{KK}(P)$ as we can directly operate on $P$. So the cost $O(|P|)$ 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 $P$ is $O(|B_P| \cdot \text{ca})$.

Furthermore, note that is reasonable to assume that the cost of computing each of the functions of $f_P$ is in $O(1)$. By observing that $|B_P| \cdot \text{ca}$ is in $O(|P|)$ 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 $P$ is $O(|P|h)$.
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 \( S_{KK}(\mathcal{P}) \) guarantees the termination after at most \( \omega \) 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 \( \preceq_s \)-least fixed-point of the operator \( \Psi^s \), defined as

\[
\Psi^s(I) = \text{Ifp}_{\preceq_s}(\lambda x. \Psi^s(I, x)),
\]

where \( \Psi^s(I, J)(A) = (I \triangle J)(\varphi) \). 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 \textit{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 \( r := \lnot x_2 \land \lnot x_1 \), we have to compute the values of \( x_1 \) and \( x_2 \) w.r.t. the \( \preceq_s \)-least fixed-point of another equational system, where the current partial evaluation \( \nu \) acts as the interpretation \( I \). This means that we have to make a call to another instance of the \textit{Solve} algorithm, which computes the values of \( x_1 \) and \( x_2 \) w.r.t. to the current evaluation \( \nu(x_1, x_2, x_3) \). 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 \( \lnot x_1, \lnot x_2 \) and \( \lnot x_3 \) with \( \nu(x_1) \) and \( \nu(x_2) \), and \( \nu(x_3) \) 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 \( r := \lnot x_2 \land \lnot x_1 \) 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 \( S(\mathcal{P}^I) \) the equational system obtained from \( S_{KK}(\mathcal{P}) \) in which all occurrences of \( \lnot x \) have been replaced with \( \lnot I(x) \), except that \( S(\mathcal{P}^I) \) is based on the \( \preceq_s \) order.

It can be shown that

Proposition 12 \( \text{Solve}(S(\mathcal{P}^I), Q) \) outputs a set \( B \subseteq V \), with \( Q \subseteq B \), s.t. the mapping \( \nu \) equals to the \( \preceq_s \)-least fixed-point on \( B \) of the functions in \( S(\mathcal{P}^I) \) and, thus, \( \nu_B = \Psi(p)(I)_B \).

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

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

4.1. \( Q := s(x_l); I := \nu; \)
4.2. \( \nu' := \text{Solve}(S(\mathcal{P}^I), Q); \)
4.3. \( r := f_i(\nu'(x_{l_1}), ..., \nu'(x_{l_{n_i}})). \)

It can be shown that

Proposition 13 Let \( \mathcal{P} \) and ?\( A \) be a logic program and a query, respectively. Then \( WF(\mathcal{P})(A) = \text{Solve}_{WF}(\mathcal{P}, ?A). \)
Example 6 Let us consider Example 3 and the associated equational system $S$ with query variable $x_A$. Below is a sequence of $\text{Solve}_{WF}(P, ?A)$ 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. $A := \{x_A\}$, $x_i := x_A$, $A := \emptyset$, $dg := \{x_A, x_B\}$, $Q := \{x_A, x_B\}$, $v' := \langle(0.3, 0.5), (0.3, 0.5), (0, 1)\rangle$, $r := (0.3, 0.5), v(x_A) := (0.3, 0.5), A := \{x_A, x_B\}, \exp(x_A) := \text{true}, \text{in} := \{x_A, x_B\}$
2. $x_i := x_B$, $A := \{x_A\}$, $dg := \{x_A, x_C\}$, $Q := \{x_A, x_C\}$, $v' := \langle(0.3, 0.5), (0.3, 0.5), (0.5, 0.7)\rangle$, $r := (0.3, 0.5), v(x_B) := (0.3, 0.5), A := \{x_A, x_C\}, \exp(x_B) := \text{true}, A := \{x_A, x_C\}$, $\text{in} := \{x_A, x_B, x_C\}$
3. $x_i := x_C$, $A := \{x_A\}$, $Q := \{x_B\}$, $v' := \langle(0.3, 0.5), (0.3, 0.5), (0.5, 0.7)\rangle$, $r := (0.3, 0.5), v_C := (0.5, 0.7), A := \{x_A, x_B\}, \exp(x_C) := \text{true}$
4. $x_i := x_B$, $A := \{x_A\}$, $Q := \{x_A, x_C\}$, $v' := \langle(0.3, 0.5), (0.3, 0.5), (0.5, 0.7)\rangle$, $r := (0.3, 0.5)$
5. $x_i := x_A$, $A := \emptyset$, $Q := \{x_A, x_B\}$, $v' := \langle(0.3, 0.5), (0.3, 0.5), (0.5, 0.7)\rangle$, $r := (0.3, 0.5)$
6. stop, return $v(x_A, x_B, x_C)$

We conclude with addressing the computational complexity of $\text{Solve}_{WF}$. The analysis parallels the one we have made for $\text{Solve}_{KK}$. We first address the case the height of a bilattice is finite. Like $\text{Solve}_{KK}$, each variable $x_j$ will appear in $\lambda$ at most $a_j \cdot (h(\lambda) + 1)$ times and, thus, the worst-case complexity is $O(\sum_{x_j \in V} (c(f_j) \cdot (a_j \cdot (h(\lambda) + 1))))$. But now, the cost of $c(f_j)$ is the cost of a recursive call to $\text{Solve}$, which analogously to Proposition 10 is $O(|B_P| c ah)$. Therefore, $\text{Solve}_{WF}$ runs in time $O(|B_P|^2 a^2 h^2 c)$. Therefore,

Proposition 14 The worst-case complexity for top-down query answering under the well-founded semantics of a logic program $P$ over bilattices is $O(|P|^2 h^2 c)$, 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 $O(|P|^2)$. However, note that for the classical case we can do better, as worked out in [36]. It remains open whether those results extend to our case as well.

In the case the height of a bilattice is not finite, the continuity of the functions $f \in F$ guarantees that each recursive call to $\text{Solve}$ requires at most $\omega$ steps. Thus, we have at most $\omega^\omega$ steps for $\text{Solve}_{WF}$. 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 $\text{Solve}_{WF}$ inspired by [36] or by [58] may give advantages from a computational point of view.

References


