Query Answering under the Any-World Assumption for Normal Logic Programs

Umberto Straccia
ISTI - CNR
Via G. Moruzzi, 1
Pisa, ITALY
straccia@isti.cnr.it

Abstract

Recently, in (Loyer & Straccia 2005) the Any-World Assumption (AWA) has been introduced for normal logic programs as a generalization of the well-known notions of Closed World Assumption (CWA) and the Open World Assumption (OWA). The AWA allows *any* assignment (i.e. interpretation), over a *truth space* (bilattice), to be a default assumption and, thus, the CWA and OWA are just special cases.

While a declarative and a fixed-point characterization for normal logic programs under the AWA has been given in (Loyer & Straccia 2005), the topic of this paper is to provide a simple, yet general top-down query answering procedure for this setting.

Introduction

The *Any-World Assumption* (AWA) for normal logic programs (Loyer & Straccia 2005) is a generalization of the notions of the Closed World Assumption (CWA) (which asserts that by default the truth of an atom is *false*) and the Open World Assumption (OWA) (which asserts that the truth of the atoms is supposed to be *unknown* by default). Essentially, the AWA allows *any* interpretation over a *truth space* to be a default assumption. The truth spaces considered are so-called *bilattices* (Ginsberg 1988) and the semantics generalizes the notions of Kripke-Kleene, well-founded and stable model semantics (Fitting 1993; 2002; Gelfond & Lifschitz 1988; van Gelder, Ross, & Schlimpf 1991). The AWA has many applications (see (Loyer & Straccia 2005)), among which: (i) it covers *Extended Logic Programs* (ELPs) (see, e.g. (Alferes & Pereira 1992; Arieli 2002; Gelfond & Lifschitz 1991)); (ii) it covers many-valued logic programming with non-monotone negation (e.g. (Damásio & Pereira 2001; Straccia 2005)); and (iii) it allows to represent default rules by relying on the so-called *abnormality theory* (McCarthy 1980; 1986).

(Loyer & Straccia 2005) presents a declarative and a fixed-point characterization for the AWA. As a consequence, in order to answer queries we have to compute the intended model I of a logic program $\mathcal{P}$ by a bottom-up fixed-point computation and then answer with $I(A)$. This requires to compute a whole model, even if in order to determine $I(A)$, not all the atom's truth is required.

The topic of this paper is to complete our investigation about the AWA by presenting a simple, yet general top-down query answering procedure. It relies on the computation of just a part of an intended model and is based on a transformation of a program into a system of equations of monotonic functions over bilattices. The least solution of this equational system is one-to-one related with the intended model of a logic program. Our procedure allows to compute the truth of an atom in the least solution and, thus, in the intended model in a top-down style. The side effect is that we have a top-down query answering procedure for the above mentioned application areas as well.

We proceed as follows. In the next recall a minimum of definitions about the AWA (Loyer & Straccia 2005). Then we present our top-down query procedure.

Preliminaries

The truth spaces we consider are *bilattices* (Ginsberg 1988). Due to their interesting mathematical structure, bilattices play an important role in (especially in theoretical aspects of) logic programming, and in knowledge representation in general, allowing to develop unifying semantical frameworks (Fitting 1991; 1993; 2002).

Bilattice

A *bilattice* (Ginsberg 1988; Fitting 2002) is a structure $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$ where $\mathcal{B}$ is a non-empty 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*. *Meet (or greatest lower bound)* and *join (or least upper bound)* under $\preceq_t$ are denoted $\wedge$ and $\vee$, while *meet and join under* $\preceq_k$ are denoted $\otimes$ and $\oplus$. *Top and bottom under* $\preceq_t$ are denoted $\mathtt{t}$ and $\mathtt{f}$, and *top and bottom under* $\preceq_k$ are denoted $\top$ and $\bot$, respectively.

We 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 $\neg\neg x = x$ [1].

[1] The dual operation to negation is *conflation* i.e. an operator $\sim$ that reverses the $\preceq_k$ ordering, leaves unchanged the $\preceq_t$ ordering, and $\sim\sim x = x$. We do not deal with conflation in this paper.

We also provide a family $\mathcal{F}$ of $\preceq_k$ and $\preceq_t$-monotone n-ary functions over $\mathcal{B}$ to manipulate truth values.

Furthermore, we assume that bilattices are *infinitary distributive bilattices* in which all distributive laws connecting $\wedge, \vee, \otimes$ and $\oplus$ hold.

Finally, 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 (e.g. $x \preceq_t y$ and $x' \preceq_t y'$ implies $x \otimes x' \preceq_t y \otimes y'$).

Bilattices has been used in several ways. For instance, the simplest non-trivial bilattice, $\mathcal{FOUR}$ (Belnap 1977) (see Figure 1), allows to deal with incomplete and/or inconsistent information. (Arieli & Avron 1996; 1998) show that the use of four values is preferable to the use of two or three values. The algebraic work of Fitting's fixed-point characterisation of stable model semantics on bilattices has been the root of the work carried out by Denecker, Marek and Truszczyński (Denecker, Marek, & Truszczyński 1999; 2002; 2003), who extended Fitting's work to a more abstract context of fixed-points operators on lattices, by relying on bilattices. Denecker, Marek and Truszczyński showed ((Denecker, Marek, & Truszczyński 1999; 2003)) interesting connections between (two-valued and four-valued) Kripke-Kleene (Fitting 1985), well-founded and stable model semantics, as well as to Moore's autoepistemic logic (Moore 1984) and Reiter's default logic (Reiter 1980). Other well-established applications of bilattices can be found in the context of reasoning under paraconsistency, imprecision and uncertainty (see, e.g. (Alcantâra, Damásio, & Pereira 2002; Arieli 2002; Blair & Subrahmanian 1989; Damásio & Pereira 1998; 2001; Loyer & Straccia 2002a; 2002b; 2003a; 2003b; 2005; Straccia 2005)).

In practice, 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 (Fitting 1993; Ginsberg 1988)).

The first bilattice construction method comes from (Ginsberg 1988). Suppose we have two complete distributive lattices $\langle L_1, \preceq_1 \rangle$ and $\langle L_2, \preceq_2 \rangle$. Think of L_1 as a lattice of values we use when we measure the degree of belief of a statement, while think of L_2 as the lattice we use when we measure the degree of doubt of it. Now, we define the structure $L_1 \odot L_2$ as follows. The structure is $\langle L_1 \times L_2, \preceq_t, \preceq_k \rangle$, where

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

In $L_1 \odot L_2$ the idea is: knowledge goes up if both degree of belief and degree of doubt go up; truth goes up if the degree of belief goes up, while the degree of doubt goes down. It is easily verified that $L_1 \odot L_2$ is a bilattice. Furthermore, if $L_1 = L_2 = L$, i.e. we are measuring belief and doubt in the same way, then negation can be defined as $\neg\langle x, y \rangle = \langle y, x \rangle$. That is, negation switches the roles of belief and doubt. In Figure 1 we report the bilattice based on $L_1 = L_2 = \{\mathtt{f}, \bot, \mathtt{t}\}$ and order $\preceq_1 = \preceq_2 = \preceq$, where $\mathtt{f} \preceq \bot \preceq \mathtt{t}$.

The second construction method has been sketched in (Ginsberg 1988) and addressed in more details in (Fitting 1992), 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 (Hähnle & Escalada-Imaz 1997)). 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. For instance, in reasoning under uncertainty (see, e.g. (Loyer & Straccia 2002b; 2003a; 2003b)), L is the unit interval $[0, 1]$ with standard ordering, $L \times L$ is interpreted as the set of (closed) sub-intervals of $[0, 1]$, and the pair $\langle x, y \rangle$ is interpreted as a lower and an upper bound of the exact value of the certainty value. Formally, given a distributive lattice $\langle L, \preceq \rangle$, the *bilattice of intervals*, denoted $\mathcal{K}(L)$, is $\langle L \times L, \preceq_t, \preceq_k \rangle$, where:

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

The intuition of those orders is that truth increases if the interval contains greater values, whereas the knowledge increases when the interval becomes more precise. Negation can be defined as $\neg\langle x, y \rangle = \langle \neg y, \neg x \rangle$, where $\neg$ is a negation operator on L. As an example, in Figure 1 we report the bilattice $\mathcal{K}([0, 1])$.

Generalized logic programs

Fix a bilattice $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$. We extend logic programs (Lloyd 1987) to the case where *computable functions* $f \in \mathcal{F}$ are allowed to manipulate truth values (see (Straccia 2005)). [2] That is, we allow any $f \in \mathcal{F}$ to appear in the body of a rule to be used to combine the truth of the atoms appearing in the body. The language is sufficiently expressive to accommodate almost all frameworks on many-valued logic programming with or without negation (Straccia 2005).

A *term*, t, is either a variable or a constant symbol. An *atom*, A, is an expression of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol and all t_i are terms. A literal, L, is of the form A or $\neg A$, where A is an atom. A *formula*, φ, is an expression built up from the atoms, the truth values $b \in \mathcal{B}$ of the bilattice and the functions $f \in \mathcal{F}$. The members of the bilattice may appear in a formula, as well as functions $f \in \mathcal{F}$. For instance, e.g. in the interval bilattice $\mathcal{K}([0, 1])$ (Fitting 1993), the expression

$$\min(p, q) \cdot \max(\neg r, \langle 0.7, 0.9 \rangle) + v$$

is a formula φ, where p, q, r and v are atoms. The intuition here is that the truth value of the formula is obtained by determining the truth value of p, q, r and v and then to apply the arithmetic functions, $\min, \max, 1-$ and product $\cdot$ to determine the value of φ [3].

[2] With computable we mean that for any input, the value of f can be determined in finite time.

[3] The arithmetic functions are extended to intervals point wise, e.g. $\langle a, b \rangle + \langle c, d \rangle = \langle a + c, b + d \rangle$.

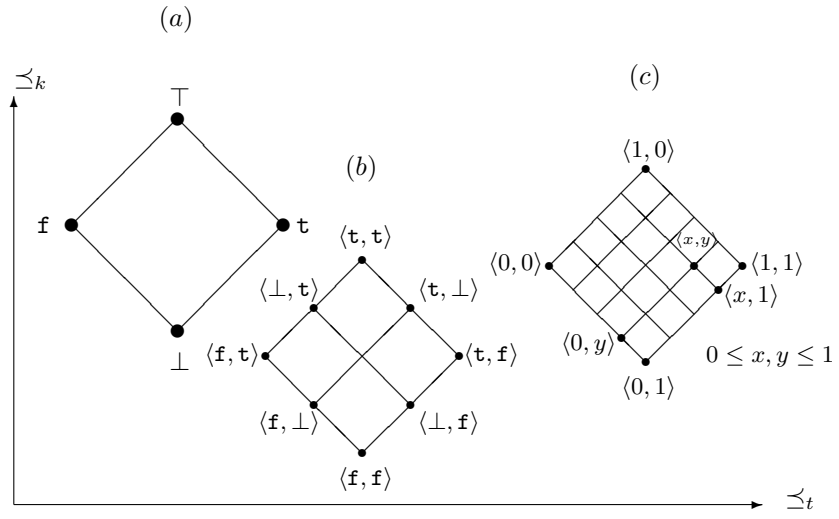

Figure 1: Bilattices. (a) $\mathcal{FOUR}$, (b) $\{\mathtt{f},\bot,\mathtt{t}\} \odot \{\mathtt{f},\bot,\mathtt{t}\}$ and (c) $\mathcal{K}([0,1])$.

A *rule* is of the form

$$A \leftarrow \varphi$$

where A is an atom and φ is a formula. For instance,

$$p \leftarrow max(0, q + r - 1)$$

is a rule dictating that p is at least as true as the conjunction of q and r with respect to the Lukasiewicz t-norm $x \wedge y = \max(0, x + y - 1)$ (Hájek 1998).

A *generalized normal logic program*, or simply *logic program*, $\mathcal{P}$, is a finite set of rules.

The notions of *Herbrand universe* $H_{\mathcal{P}}$ of $\mathcal{P}$ and *Herbrand base* (as the set of all ground atoms) $B_{\mathcal{P}}$ of $\mathcal{P}$ are as usual. Additionally, given $\mathcal{P}$, the generalized normal logic program $\mathcal{P}^*$ is constructed as follows:

1. set $\mathcal{P}^*$ to the set of all ground instantiations of rules in $\mathcal{P}$;
2. replace several rules in $\mathcal{P}^*$ having same head, $A \leftarrow \varphi_1$, $A \leftarrow \varphi_2, \ldots$ with $A \leftarrow \varphi_1 \vee \varphi_2 \vee \ldots$ (recall that $\vee$ is the join operator of the bilattice); and
3. if an atom A is not head of any rule in $\mathcal{P}^*$, then add the rule $A \leftarrow \mathtt{f}$ to $\mathcal{P}^*$ (it is a standard practice in logic programming to consider such atoms as *false*). This already acts as a kind of default assumption on non derivable facts. We will change this point once we allow any default value as assumption later one.

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

We next recall the usual semantics of logic programs over bilattices (cf. (Loyer & Straccia 2005)). For ease, we will rely on the following simple running example to illustrate the concepts we introduce in the paper.

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

$$p \leftarrow p \vee q$$
$$q \leftarrow \neg q\ .$$

In Table 1 we report the models I_i, the Kripke-Kleene, the well-founded model and stable models of $\mathcal{P}$, marked by bullets. The columns on the right hand side will be discussed later on.

Interpretations. An *interpretation* I on the bilattice $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$ is a mapping from atoms to members of $\mathcal{B}$. I is extended from atoms to formulae in the usual way: (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 $f(A)$, $I(f(A)) = f(I(A))$, and similarly for n-ary functions.

$\preceq_t, \preceq_k$ 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.

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). $\langle \mathcal{I}(\mathcal{B}), \preceq_t, \preceq_k \rangle$ is a bilattice as well.

It is easy to see that the space of interpretations $\langle \mathcal{I}(\mathcal{B}), \preceq_t, \preceq_k \rangle$ is an infinitary interlaced and distributive bilattice as well.

Models. I is a *model* of $\mathcal{P}$, denoted $I \models \mathcal{P}$, iff for all $A \leftarrow \varphi \in \mathcal{P}^*$, $I(A) = I(\varphi)$ 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. (Fitting 2002; Loyer & Straccia 2006; 2005)).

Among all the models, two models play a special role: namely the *Kripke-Kleene model* ($KK_{\mathcal{P}}$), which is the $\preceq_k$-least model of $\mathcal{P}$, and the *Well-Founded model* ($WF_{\mathcal{P}}$) (Fitting 1993; 2002; 1988; van Gelder, Ross, & Schlimpf 1991).

It is well-know that the $WF_{\mathcal{P}}$ is more informative (provides more knowledge) than $KK_{\mathcal{P}}$. For the definition of the well-founded semantics over bilattices refer to (Fitting 1993; 2002; Loyer & Straccia 2006). It's the generalization of the classical well-founded semantics to bilattices. We will ob-

$I_i \models \mathcal{P}$	I_i p	q	$KK(\mathcal{P})$	$WF(\mathcal{P})$	stable models	$s_{\mathcal{P}}^{\mathtt{I_f}}(I_i)$ p	q	$U_{\mathcal{P}}(I_i)$	H-models	H-closed models
I_1	$\bot$	$\bot$	•	•	•	$\bot$	$\bot$	$\emptyset$	•	•
I_2	$\mathtt{t}$	$\bot$				$\bot$	$\bot$	$\emptyset$	•	
I_3	$\top$	$\top$			•	$\mathtt{f}$	$\mathtt{f}$	$\{p, q\}$	•	•
I_4	$\mathtt{t}$	$\top$				$\mathtt{f}$	$\mathtt{f}$	$\{p, q\}$		

Table 1: Models, Kripke-Kleene, well-founded and stable models of $\mathcal{P}$.

tain it as a special case of the AWA, too.

Furthermore, for the sake of this paper, we note that the existence and uniqueness of $KK_{\mathcal{P}}$ is guaranteed by the fixed-point characterization based on the $\preceq_k$-monotone function $\Phi_{\mathcal{P}}$: for an interpretation I, for any ground atom A with (unique) $A \leftarrow \varphi \in \mathcal{P}^*$,

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

Then all models of $\mathcal{P}$ are fixed-points of $\Phi_{\mathcal{P}}$ and vice-versa, and $KK_{\mathcal{P}}$ can be computed in the usual way by iterating $\Phi_{\mathcal{P}}$ over $\mathtt{I}_\bot$.

Classical logic programs. In classical logic programs the body is a conjunction of literals, i.e. for $A \leftarrow \varphi \in \mathcal{P}^*$ (except for the case $A \leftarrow \mathtt{f} \in \mathcal{P}^*$) $\varphi = \varphi_1 \vee \ldots \vee \varphi_n$ and $\varphi_i = L_{i_1} \wedge \ldots \wedge L_{i_n}$.

For a set of literals X, with $\neg.X$ we indicate the set $\{\neg L\colon L \in X\}$, where for any atom A, $\neg\neg A$ is replaced with A. A classical interpretation (total or partial) can be represented as a consistent set of literals, i.e. $I \subseteq B_{\mathcal{P}} \cup \neg.B_{\mathcal{P}}$ and for all atoms A, $\{A, \neg A\} \not\subseteq I$. Of course, the opposite is also true, i.e. a consistent set of literals can straightforwardly be turned into an interpretation over $\mathcal{FOUR}$.

The classical WF semantics has been defined in terms of the well-known notion of *unfounded set* (see, e.g. (Leone, Rullo, & Scarcello 1997; van Gelder, Ross, & Schlimpf 1991)), which identifies the set of atoms that can safely be assumed false if the current information about $\mathcal{P}$ is given by an interpretation I. Indeed, given a partial classical interpretation I and a classical logic program $\mathcal{P}$, a set of ground atoms $X \subseteq B_{\mathcal{P}}$ is an *unfounded set* (i.e., the atoms in X can be assumed as false) for $\mathcal{P}$ w.r.t. I iff for each atom $A \in X$, if $A \leftarrow \varphi \in \mathcal{P}^*$, where $\varphi = \varphi_1 \vee \ldots \vee \varphi_n$ and $\varphi_i = L_{i_1} \wedge \ldots \wedge L_{i_n}$, then φ_i is false either w.r.t. I or w.r.t. $\neg.X$, for all $1 \leq i \leq n$. The *greatest unfounded set* for $\mathcal{P}$ w.r.t. I (which exists) is denoted by $U_{\mathcal{P}}(I)$.

Now, consider the well-founded operator (van Gelder, Ross, & Schlimpf 1991)

$$W_{\mathcal{P}}(I) = T_{\mathcal{P}}(I) \cup \neg.U_{\mathcal{P}}(I)$$

($T_{\mathcal{P}}$ is the usual immediate consequence operator (Lloyd 1987) for logic programs) then $W_{\mathcal{P}}(I)$ can be rewritten as

$$W_{\mathcal{P}}(I) = T_{\mathcal{P}}(I) \oplus \neg.U_{\mathcal{P}}(I) \ ,$$

by defining $\oplus = \cup, \otimes = \cap$ in the lattice $\langle 2^{B_{\mathcal{P}} \cup \neg.B_{\mathcal{P}}}, \subseteq \rangle$ (the partial order $\subseteq$ corresponds to $\preceq_k$).

Then, we recall that the well-founded semantics is defined to be the $\preceq_k$-least fixed-point of $W_{\mathcal{P}}$ in (van Gelder, Ross, & Schlimpf 1991) and is equivalent to:

$WF_{\mathcal{P}}$ = " $\preceq_k$-least model of $\mathcal{P}$ such that $\neg.U_{\mathcal{P}}(I) \subseteq I$" (Leone, Rullo, & Scarcello 1997).

As we will see next, the AWA generalizes this notion.

The AWA in logic programming

In the following a *hypothesis* (denoted H) is always an interpretation over a bilattice and represents our default assumption over the world.

The principle underlying the *Any-World Assumption* (AWA) is to regard an hypothesis H as an additional source of default information to be used to complete the implicit knowledge provided by a logic program. The AWA H dictates that any atom A, whose truth-value cannot be inferred from the facts and rules, is assigned to the default truth value $H(A)$.

For comparison, under the CWA, $H = \mathtt{I}_\mathtt{f}$ is assumed, while under the OWA, $H = \mathtt{I}_\bot$ is assumed.

Also note that any ground atom A not appearing in the head of any rule and, thus, not derivable, is mapped (up to now) into 'false'. Now, according to the AWA, any such atom A should be mapped into $H(A)$. *If not specified otherwise, we change Point 3. of the definition of $\mathcal{P}^*$ by adding $A \leftarrow H(A)$ to $\mathcal{P}^*$.* It should be noted that this implicitly affects also all definitions based on $\mathcal{P}^*$, e.g. the definitions of model, Kripke-Kleene model and that of $\Phi_{\mathcal{P}}$ (which now maps such atoms into $H(A)$ rather than into $\mathtt{f}$).

Now, we proceed in two steps.

The support

At first, we introduce the notion of *support*, denoted $s_{\mathcal{P}}^H(I)$. The support is a generalization of the notion of unfounded sets. Indeed, $s_{\mathcal{P}}^H(I)$ determines the amount of default information, taken from H, that can safely be joined to I. The support generalizes the notion of unfounded sets as it turns out that for classical logic programs $\mathcal{P}$ and $H = \mathtt{I}_\mathtt{f}$ (see Table 1), $s_{\mathcal{P}}^H(I) = \neg.U_{\mathcal{P}}(I)$ (Loyer & Straccia 2005).

The principle underlying the support can be explained as follows. Consider a ground atom A and the rule $A \leftarrow \varphi \in \mathcal{P}^*$, an interpretation I, which is our current knowledge about $\mathcal{P}$, and a hypothesis H. We would like to determine how much default knowledge can be 'safely' taken from H to complete I. So, let us assume that $J \preceq_k H$ amounts to the default knowledge taken from H. $J(A)$ is the default information provided by J to the atom A. The completion of I with J is the interpretation $I \oplus J$. In order to accept this completion, we have to ensure that at least the assumed knowledge $J(A)$ is entailed by $\mathcal{P}$ w.r.t.

the completed interpretation $I \oplus J$, i.e. for $A \leftarrow \varphi \in \mathcal{P}^*$, $J(A) \preceq_k (I \oplus J)(\varphi) = \Phi_\mathcal{P}(I \oplus J)(A)$ should hold.

Therefore, we say that an interpretation J is *safe* w.r.t. $\mathcal{P}$, I and H iff

$$J \preceq_k H \text{ and } J \preceq_k \Phi_\mathcal{P}(I \oplus J) .$$

Note that safe interpretations correspond to unfounded sets for classical logic programs (Loyer & Straccia 2005).

Furthermore, like for unfounded sets, among all possible safe interpretations, we are interested in the $\preceq_k$-maximal (which exists and is unique). The $\preceq_k$-greatest safe interpretation is called the *support* provided by H to $\mathcal{P}$ w.r.t. I and is denoted by $s^H_\mathcal{P}(I)$. Table 1 reports the support for the logic program of Example 1

Note that by definition under the OWA $H = \mathtt{I}_\bot$, $s^H_\mathcal{P}(I) = \mathtt{I}_\bot$ holds, as expected, while for classical logic programs $s^H_\mathcal{P}(I) = \neg.U_\mathcal{P}(I)$, for $H = \mathtt{I_f}$.

In summary, the support is an extension of the notion of unfounded sets (i) to logic programming over bilattices; and to (ii) arbitrary default assumptions H.

Finally, we also recall that the support can effectively be computed as the iterated fixed-point of the $\preceq_k$-monotone function

$$\sigma^{I,H}_\mathcal{P}(J) = H \otimes \Phi_\mathcal{P}(I \oplus J) .$$

Indeed, (Loyer & Straccia 2005) shows that the iterated sequence of interpretations $F^{I,H}_i$, where

$$\begin{array}{rcl} F^{I,H}_0 & = & H \\ F^{I,H}_{i+1} & = & \sigma^{I,H}_\mathcal{P}(F^{I,H}_i) . \end{array}$$

is $\preceq_k$-monotone decreasing and reaches a fixed-point $F^{I,H}_\lambda = s^H_\mathcal{P}(I)$, for a limit ordinal λ.

Interestingly, the above method gives us a simple method to compute the negation of the greatest unfounded set, $\neg.U_\mathcal{P}(I)$, by means of the iteration

$$\begin{array}{rcl} F_0 & = & \neg.B_\mathcal{P} \\ F_{i+1} & = & \neg.B_\mathcal{P} \cap \Phi_\mathcal{P}(I \cup F_i) . \end{array}$$

H-models

At second, among all models of a program $\mathcal{P}$, let us consider those models, which $\preceq_k$-subsume their own support. That is, we say that an interpretation I is a *H-model* of $\mathcal{P}$ iff

$$I \models \mathcal{P} \text{ and } s^H_\mathcal{P}(I) \preceq_k I .$$

The $\preceq_k$-least H-model is called *H-founded model*, and is denoted with $HF_\mathcal{P}$.

H-models have interesting properties (Loyer & Straccia 2005).

Proposition 1 ((Loyer & Straccia 2005)) *The following statements are equivalent:*

1. *I is a H-model of $\mathcal{P}$;*
2. *$I = \Phi_\mathcal{P}(I) \oplus s^H_\mathcal{P}(I)$;*
3. *$I = \Phi_\mathcal{P}(I \oplus s^H_\mathcal{P}(I))$.*

From a fixed-point characterization point of view, it follows immediately that the set of H-models can be identified by the fixed-points of at least two $\preceq_k$-monotone immediate consequence operators:

$$\begin{array}{rcl} \tilde{\Pi}^H_\mathcal{P}(I) & = & \Phi_\mathcal{P}(I) \oplus s^H_\mathcal{P}(I) \\ \Pi^H_\mathcal{P}(I) & = & \Phi_\mathcal{P}(I \oplus s^H_\mathcal{P}(I)) . \end{array}$$

This guarantees the existence and uniqueness of the H-founded model of a program $\mathcal{P}$.

Also, these operators have a quite interesting property, once we restrict our attention to the everywhere false hypothesis $H = \mathtt{I_f}$. Under this condition, it can be shown that the least fixed-point under $\preceq_k$ of $\Pi^H_\mathcal{P}$ and, thus, of $\tilde{\Pi}^H_\mathcal{P}$ coincides with the classical well-founded semantics. This is not surprising in the light of the fact that the $\tilde{\Pi}^H_\mathcal{P}$ operator is quite similar to the $W_\mathcal{P}$ operator for classical logic programs and interpretations.

Also note that the definition of H-founded model is nothing else than a generalization from the classical setting to bilattices of the notion of well-founded model (recall that the well-founded model is the least model satisfying $\neg.U_\mathcal{P}(I) \subseteq I$ (Leone, Rullo, & Scarcello 1997), which is a special case of the definition of H-founded model).

We conclude by remaking that (Loyer & Straccia 2005) also generalizes the stable model semantics to the AWA. Indeed, (Loyer & Straccia 2005) defines the notion of H-closed models: I is an *H-closed model* of $\mathcal{P}$ iff $I = KK_{\mathcal{P} \oplus s^H_\mathcal{P}(I)}$, where $\mathcal{P} \oplus I$ is obtained from $\mathcal{P}^*$ by replacing all rules $A \leftarrow \varphi \in \mathcal{P}^*$ with $A \leftarrow \varphi \oplus I(A)$. It is shown in (Loyer & Straccia 2006) that stable models coincide with H-closed models under the CWA $H = \mathtt{I_f}$.

Example 2 (running example cont.) *Consider Example 1. In Table 1 we also report the support and H-models. Note that the support and the negation of the greatest unfounded set coincide as well. Also the WF model coincides with the $\preceq_k$-smallest H-model, and stable models coincide with H-closed models.*

We also consider the AWA $H = \mathtt{I_t}$ (see Table 2). Note that now under $H = \mathtt{I_t}$, contrary to the case $H = \mathtt{I_f}$, I_2 and I_4 are H-closed models of which I_2 is both the Kripke-Kleene and the well-founded model w.r.t. H. Essentially, the difference is in the truth of p, which in the former case is always t*, while in the latter case is $\bot$ (I_1) and $\top$ (I_3), respectively.*

Some applications of the AWA

We recall some examples from (Loyer & Straccia 2005). Consider, for instance, a rule expressing the fact that

if someone is innocent (s)he cannot be guilty.

This may be represented by

$$\mathtt{Guilty(x)} \leftarrow \neg \mathtt{Innocent(x)} .$$

We assume (which is the rule in most countries) that anyone is by default innocent and not guilty i.e. $H(\mathtt{Innocent(a)}) = \mathtt{t}, H(\mathtt{Guilty(a)}) = \mathtt{f}$. Therefore, in order to charge someone, we have to provide a "proof" for being guilty (i.e. not innocent) from the facts.

Similarly, a rule expressing the fact that

$I_i \models \mathcal{P}$	I_i		$s^H_{\mathcal{P}}(I_i)$		$KK_{\mathcal{P}\oplus s^H_{\mathcal{P}}(I_i)}$		H-models	H-closed models
	p	q	p	q	p	q		
I_1	$\bot$	$\bot$	t	$\bot$	t	$\bot$		
I_2	t	$\bot$	t	$\bot$	t	$\bot$	•	•
I_3	$\top$	$\top$	t	t	t	$\top$	•	
I_4	t	$\top$	t	t	t	$\top$	•	•

Table 2: Models, H-models and H-closed models of $\mathcal{P}$ w.r.t. $H = \mathtt{I}_{\mathtt{t}}$.

a car may cross railway tracks if there is no crossing train

may be represented by

$$\texttt{Cross\_railway\_tracks} \leftarrow \neg\texttt{Train\_is\_comming}\ .$$

In this situation, in order to safely cross the railway there should be explicit evidence that the train is not coming and, thus, we may assume by default that $H(\texttt{Train\_is\_comming}) = \bot$ (i.e. the atom is interpreted according to OWA) together with the default $H(\texttt{Cross\_railway\_tracks}) = \texttt{f}$, for safety.

Another frequently source of defaults is the case where we also want to express *default statements* of the form

normally, unless something abnormal holds, then φ implies A.

Such statements were the main motivation for non-monotonic logics like *Default Logic* (Reiter 1980), Autoepistemic Logic (Doyle & McDermott 1980; Marek & Truszczyński 1991; McDermott 1982; Moore 1985) and *Circumscription* (McCarthy 1980; 1986) (see also (Ginsberg 1987)). We can formulate such a statement in a natural way, using *abnormality theories*, as

$$\begin{array}{l} A \leftarrow \varphi \wedge \neg Ab \\ Ab \leftarrow \neg A\ , \end{array} \tag{1}$$

where Ab stands for *abnormality*, and then consider the hypothesis $H(Ab) = \texttt{f}$, i.e. by default there are no abnormal objects. McCarty (McCarthy 1980; 1986) originated the concept of abnormality theory as a way to represent default reasoning in Circumscription. Defaults are represented with an introduced *abnormality* predicate: for instance, to say that normally birds fly, we would use the rule

$$\begin{array}{l} \texttt{Flies(x)} \leftarrow \texttt{Bird(x)} \wedge \neg\texttt{Ab}_1\texttt{(x)} \\ \texttt{Ab}_1\texttt{(x)} \leftarrow \neg\texttt{Flies(x)}\ . \end{array} \tag{2}$$

Here the meaning of $\texttt{Ab}_1(x)$ is something like "x is abnormal with respect to flying birds". Note that there can be many different kinds of abnormality, and they are indexed according to kind. Likewise in Circumscription, where abnormality predicates are minimized (allowing relevant predicates to vary - e.g. Flies), we have to assert that no object is abnormal by default, i.e. $H(\texttt{Ab}_1(t)) = \texttt{f}$, for all terms t over the Herbrand universe.

Note that, due to the generality of the notion of hypotheses, we may also deal with "(default) degrees of abnormality", rather than just consider the classical $\{\texttt{f}, \texttt{t}\}$ setting. So, for instance, we may take advantage of some statistical indication on the number of abnormal birds w.r.t. the property of flying: we may state that $H(\texttt{Ab}_1(t)) = \langle 0, 0.2\rangle$ in the interval bilattice, indicating that the degree of being a bird t a non-flyer is quite low.

Top-down query answering

A *query* is an expression of the form $?A$ (*query atom*), intended as a question about the truth of the atom A in the 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.

Our top-down procedure is based on a transformation of a program into a system of equations of monotonic functions over bilattices for which we can compute the least fixed-point in a top-down style. The procedure is a generalization of the one presented in (Straccia 2005). Note that (Straccia 2005) relies on Fitting's (Fitting 1993; 2002) formulation based on a generalization of the Gelfond-Lifschitz formulation of stable models (Gelfond & Lifschitz 1988) and it works for the CWA only. But, here we deal with arbitrary default assumptions, whose formalization is based on a generalization of the notion of unfounded sets and, thus, the computation is different.

In this paper we assume the bilattices are *finite*. The infinite case can be handled similarly to (Straccia 2005). From a practical point of view this is a limitation we can live with especially taking into account that computers have finite resources, and thus, only a finite set of truth degrees can be represented [4]. This will guarantee the termination of our procedures (otherwise the termination after a finite number of steps cannot be guaranteed always, see also (Straccia 2005)).

The idea is the following. Consider a complete lattice $\mathcal{L} = \langle L, \preceq\rangle$, a logic program $\mathcal{P}$, its Herbrand base $B_{\mathcal{P}} = \{A_1, \ldots, A_n\}$ and $\mathcal{P}^*$. Let us associate to each atom $A_i \in B_{\mathcal{P}}$ a variable x_i, which will take a value in the domain L (sometimes, we will refer to that variable with x_A as well). An interpretation I may be seen as an assignment of truth 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 $A_i \in B_P$, $I(A_i) = O(I)(A_i)$. Therefore, we may identify the fixed-points of O as the solutions over L of the system of equations of the following

[4] In particular, this includes also the usual case were we use the rational numbers in $[0, 1]$ under a given fixed decimal precision p.

form:

$$\begin{array}{rcl} x_1 & = & f_1(x_{1_1}, \ldots, x_{1_{a_1}}) \,, \\ & \vdots & \\ x_n & = & f_n(x_{n_1}, \ldots, x_{n_{a_n}}) \,, \end{array} \tag{3}$$

where for $1 \leq i \leq n$, $1 \leq k \leq a_i$, we have $1 \leq i_k \leq n$. Each variable x_{i_k} will take a value in the domain L, 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 $O(I)(A_i)$. The models of $\mathcal{P}$ are bijectively related to the solutions of the system (3) and the $\preceq_k$-least solution corresponds to the $\preceq_k$-least model of $\mathcal{P}$, i.e. $KK_{\mathcal{P}}$.

In the general case, we assume that each function $f_i \colon L^{a_i} \mapsto L$ in Equation (3) 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 (3) as the tuple $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $\mathcal{L}$ is a lattice, $V = \{x_1, ..., x_n\}$ are the variables and $\mathbf{f} = \langle f_1, ..., f_n \rangle$ is the tuple of functions. As it is well known, a monotonic equation system as (3) has a $\preceq$-least (greatest) solution, $\mathrm{lfp}(\mathbf{f})$ ($\mathrm{gfp}(\mathbf{f})$), the $\preceq$-least (greatest) fixed-point of $\mathbf{f}$ is given as the least upper bound (greatest lower bound) of the $\preceq$-monotone sequence, $\mathbf{y}_0, \ldots, \mathbf{y}_i, \ldots$, where

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

Example 3 *Consider* $[0, 1]$ *and the bilattice of intervals build from it. Consider the following logic program:*

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

Then the corresponding equational system is of the form

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

For ease of exposition we assume that the hypothesis is the CWA $H = \mathtt{I_f}$*. Note that the Kripke-Kleene model of* $\mathcal{P}$ *is*

$$KK_{\mathcal{P}} = \{A \colon \langle 0.3, 1 \rangle, B \colon \langle 0.3, 0.8 \rangle, C \colon \langle 0.2, 0.7 \rangle\} \,,$$

while the well-founded model is

$$WF_{\mathcal{P}} = \{A \colon \langle 0.3, 0.5 \rangle, B \colon \langle 0.3, 0.5 \rangle, C \colon \langle 0.5, 0.7 \rangle\} \,.$$

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

The $\preceq_k$*-least fixed-point computation is (the triples represent* $\langle x_A, x_B, x_C \rangle$*,*

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

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

Furthermore, if $\mathbf{c} \preceq \mathbf{f}(\mathbf{c})$ ($\mathbf{f}(\mathbf{c}) \preceq \mathbf{c}$) then the sequence

$$\begin{array}{rcl} \mathbf{y}_0 & = & \mathbf{c} \\ \mathbf{y}_{i+1} & = & \mathbf{f}(\mathbf{y}_i) \end{array}$$

is non-decreasing (non-increasing) and converging to the least (greatest) fixed-point $\mathrm{lfp}(\mathbf{f}, \mathbf{c})$ ($\mathrm{gfp}(\mathbf{f}, \mathbf{c})$) of $\mathbf{f}$ such that $\mathbf{c} \preceq \mathrm{lfp}(\mathbf{f}, \mathbf{c})$ ($\mathrm{gfp}(\mathbf{f}, \mathbf{c}) \preceq \mathbf{c}$). Of course, $\mathrm{lfp}(\mathbf{f}) = \mathrm{lfp}(\mathbf{f}, \bot)$ and $\mathrm{gfp}(\mathbf{f}) = \mathrm{gfp}(\mathbf{f}, \top)$.

Informally, our algorithm works as follows (see Table 3). Assume, we are interested in the value of x_0 in the fixed-point $\mathrm{lfp}(\mathbf{f}, \mathbf{c})$ ($\mathrm{gfp}(\mathbf{f}, \mathbf{c})$) of the system, where $\mathbf{c} \in L^n$. We call the procedure with $Solve(\mathcal{S}, \{x_0\}, \mathbf{c}, \uparrow)$ ($Solve(\mathcal{S}, \{x_0\}, \mathbf{c}, \downarrow)$)). 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 $\mathbf{c}$. 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 $\mathbf{c}$ 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 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 attentive reader will notice that the $Solve$ procedure described in (Straccia 2005) is a special case of the one presented here and it also a has commonalities with the so-called *tabulation* procedures, like (Chen & Warren 1996; Damásio, Medina, & Ojeda Aciego 2004).

$Solve(\mathcal{S}, Q, \mathbf{c}, d)$ uses some auxiliary functions and data structures: given the equational system (3), (i) $\mathtt{s}(x)$ 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); (ii) $\mathtt{p}(x)$ 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). (iii) the variable $\mathtt{dg}$ collects the variables that may influence the value of the query variables; (iv) the array variable $\mathtt{exp}$ traces the equations that has been "expanded" (the body variables are put into the active list); (v) 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).

Example 4 *Consider Example 3 and query variable* x_A*. Below is a sequence of* $Solve(\mathcal{S}, \{x_A\})$ *computation w.r.t.* $\preceq_k$ *and the CWA* $H = \mathtt{I_f}$*. Each line is a sequence of steps in the 'while loop'. What is left unchanged is not reported.*

Procedure $Solve(\mathcal{S}, Q, \mathbf{c}, d)$
Input: $\preceq$-monotonic system $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $Q \subseteq V$ is the set of query variables, a starting point $\mathbf{c}$ and a direction of the computation $d \in \{\uparrow, \downarrow\}$;
Output: A set $B \subseteq V$, with $Q \subseteq B$ such that the mapping $\mathtt{v}$ restricted to B, equals to $\text{lfp}(\mathbf{f}, \mathbf{c})$ if $d = \uparrow$, equals to $\text{gfp}(\mathbf{f}, \mathbf{c})$ (if $d = \downarrow$) otherwise.

1. $\mathtt{A} := Q$, $\mathtt{dg} := Q$, $\mathtt{in} := \emptyset$, **for all** $x \in V$ **do** $\mathtt{v}(x) = \mathbf{c}$, $\mathtt{exp}(x) = \mathtt{false}$
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. $r := f_i(\mathtt{v}(x_{i_1}), ..., \mathtt{v}(x_{i_{a_i}}))$
5.1 **if** $(d = \uparrow$ **and** $r \succ \mathtt{v}(x_i))$ **then** $\mathtt{v}(x_i) := r$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{p}(x_i) \cap \mathtt{dg})$ **fi**
5.2 **if** $(d = \downarrow$ **and** $r \prec \mathtt{v}(x_i))$ **then** $\mathtt{v}(x_i) := r$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{p}(x_i) \cap \mathtt{dg})$ **fi**
6. **if not** $\mathtt{exp}(x_i)$ **then** $\mathtt{exp}(x_i) = \mathtt{true}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{s}(x_i) \setminus \mathtt{in})$, $\mathtt{in} := \mathtt{in} \cup \mathtt{s}(x_i)$ **fi**
od

Table 3: General top-down algorithm.

1. $\mathtt{A} := \{x_A\}, x_i := x_A, \mathtt{A} := \emptyset, \mathtt{dg} := \{x_A, x_B\}, r := \bot,$ $\mathtt{exp}(x_A) := \mathtt{true}, \mathtt{A} := \{x_A, x_B\}, \mathtt{in} := \{x_A, x_B\}$
2. $x_i := x_B, \mathtt{A} := \{x_A\}, \mathtt{dg} := \{x_A, x_B, x_C\}, r := \langle 0.3, 1\rangle,$ $\mathtt{v}(x_B) := \langle 0.3, 1\rangle, \mathtt{A} := \{x_A, x_C\}, \mathtt{exp}(x_B) := \mathtt{true},$ $\mathtt{in} := \{x_A, x_B, x_C\}$
3. $x_i := x_C, \mathtt{A} := \{x_A\}, r := \langle 0.2, 0.7\rangle, \mathtt{v}(x_C) := \langle 0.2, 0.7\rangle,$ $\mathtt{A} := \{x_A, x_B\}, \mathtt{exp}(x_C) := \mathtt{true}$
4. $x_i := x_B, \mathtt{A} := \{x_A\}, r := \langle 0.3, 0.8\rangle, \mathtt{v}(x_B) := \langle 0.3, 0.8\rangle,$ $\mathtt{A} := \{x_A, x_C\}$
5. $x_i := x_C, \mathtt{A} := \{x_A\}, r := \langle 0.2, 0.7\rangle$
6. $x_i := x_A, \mathtt{A} := \emptyset, r := \langle 0.3, 1\rangle, \mathtt{v}(x_A) := \langle 0.3, 1\rangle,$ $\mathtt{A} := \{x_A, x_B\}$
7. $x_i := x_B, \mathtt{A} := \{x_A\}, r := \langle 0.3, 0.8\rangle,$
8. $x_i := x_A, \mathtt{A} := \emptyset, r := \langle 0.3, 1\rangle$
10. `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.

From a computational point of view, by means of appropriate data structures, the operations on `A`, `v`, `dg`, `in`, `exp`, `p` and `s` can be performed in constant time. Therefore, Step 1. is $O(|V|)$, all other steps, except Step 2. and Step 4. are $O(1)$. Let $c(f_x)$ be the maximal cost of evaluating function f_x on its arguments, so Step 4. is $O(c(f_x))$. It remains to determine the number of loops of Step 2. As the height $h(\mathcal{L})$ of $\mathcal{L}$ is finite (i.e. the length of the longest strictly $\preceq$-increasing chain in L minus 1), observe that any variable is increasing/decreasing in the $\preceq$ order (depending on the parameter d) as it enters in the `A` list (Step 5.1 or Step 5.2), 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 $O(\sum_{x_i \in V}(c(f_i) \cdot (a_i \cdot h(\mathcal{L}) + 1))$. We generalize an analogous result as in (Straccia 2005).

Theorem 1 *Consider a monotone system of equations $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$ and let $\mathbf{c} \in L^n$. (i) If $\mathbf{c} \preceq \mathbf{f}(\mathbf{c})$ ($\mathbf{f}(\mathbf{c}) \preceq \mathbf{c}$) then after a finite number of steps $Solve(\mathcal{S}, Q, \mathbf{c}, \uparrow)$ ($Solve(\mathcal{S}, Q, \mathbf{c}, \downarrow)$) determines a set $B \subseteq V$, with $Q \subseteq B$ such that the mapping $\mathtt{v}$ equals $lfp(\mathbf{f}, \mathbf{c})$ ($gfp(\mathbf{f}, \mathbf{c})$) on B, i.e. $\mathtt{v}_{|B} = lfp(\mathbf{f}, \mathbf{c})_{|B}$ ($\mathtt{v}_{|B} = gfp(\mathbf{f}, \mathbf{c})_{|B}$); and (ii) If the computing cost of each function in $\mathbf{f}$ is bounded by c, the arity bounded by a, and the height is bounded by h, then the worst-case complexity of the algorithm $Solve$ is $O(|V|cah)$.*

Of course, $\text{lfp}(\mathbf{f})(x) = Solve(\mathcal{S}, \{x\}, \bot, \uparrow)(x)$ ($\text{gfp}(\mathbf{f})(x) = Solve(\mathcal{S}, \{x\}, \top, \downarrow)(x)$). Note also that if case the height of a lattice is not finite, the computation may not terminate after a finite number of steps (see (Straccia 2005)).

Query answering: Kripke-Kleene semantics

Consider a logic program $\mathcal{P}$. The system of equations that we build from $\mathcal{P}^*$ is straightforward. Assign to each atom A a variable x_A and substitute in $\mathcal{P}^*$ each occurrence of A with x_A. Finally, substitute each occurrence of $\leftarrow$ with $=$ and let $\mathcal{S}_{KK}(\mathcal{P}) = \langle \mathcal{L}, V, \mathbf{f}_\mathcal{P} \rangle$ be the resulting equational system with the $\preceq_k$ order. Of course, $|V| = |B_\mathcal{P}|$, $|\mathcal{S}_{KK}(\mathcal{P})|$ can be computed in time $O(|\mathcal{P}^*|)$ and all functions in $\mathcal{S}_{KK}(\mathcal{P})$ are $\preceq_k$-monotone. As $\mathbf{f}_\mathcal{P}$ is one to one related to $\Phi_\mathcal{P}$, 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 builds the equational system $\mathcal{S}_{KK}(\mathcal{P})$ and then calls $Solve(\mathcal{S}_{KK}(\mathcal{P}), \{x_A\}, \bot, \uparrow)$ and returns the output `v` on the query variable, where `v` is the output of the call to $Solve$. $Solve_{KK}$ behaves correctly as for a query $?A$, $KK_\mathcal{P}(A) = Solve_{KK}(\mathcal{P}, \{?A\})$.

The extension of this property to a set of query atoms is straightforward.

From a computational point of view, we can avoid the cost of translating $\mathcal{P}^*$ into $\mathcal{S}_{KK}(\mathcal{P})$ as we can directly operate on $\mathcal{P}^*$. It follows that query answering under the Kripke-Kleene semantics is $O(|B_\mathcal{P}|cah)$. Furthermore, often the cost of computing each of the functions of $\mathbf{f}_\mathcal{P}$ is in $O(1)$. By observing that $|B_\mathcal{P}|a$ is in $O(|\mathcal{P}^*|)$, the complexity becomes $O(|\mathcal{P}^*|h)$. If the height is a fixed parameter, i.e. a constant, query answering becomes linear as for propositional logic programs (Dowling & Gallier 1984) (of course, $|\mathcal{P}^*|$ may be exponential with respect to $\mathcal{P}$, however).

Query answering: H-founded semantics

As we have seen, the H-founded model of a logic program $\mathcal{P}$ is the $\preceq_k$-least fixed-point of the operator

$$\Pi^H_{\mathcal{P}}(I) = \Phi_{\mathcal{P}}(I \oplus s^H_{\mathcal{P}}(I))$$

and the support $s^H_{\mathcal{P}}(I)$ coincides with the iterated fixed-point of the function $\sigma^{I,H}_{\mathcal{P}}(J)$ beginning the computation with H, where

$$\sigma^{I,H}_{\mathcal{P}}(J) = H \otimes \Phi_{\mathcal{P}}(I \oplus J)\ .$$

That is, $s^H_{\mathcal{P}}(I)$ coincides with the limit of the $\preceq_k$-non-increasing sequence

$$\begin{array}{rcl} F^{I,H}_0 & = & H \\ F^{I,H}_{i+1} & = & \sigma^{I,H}_{\mathcal{P}}(F^{I,H}_i)\ . \end{array}$$

In the following, we show how we use the $Solve$ procedure to compute the support. That is, we want a top-down procedure answering that, for a given atom $A \in B_{\mathcal{P}}$, answers with $s^H_{\mathcal{P}}(I)(A)$, i.e. the truth of A in the support of $\mathcal{P}$ w.r.t. I.

To this purpose, we build an equational system whose greatest fixed-point corresponds to the support. Indeed, consider $A \leftarrow f(B_1, \ldots, B_n) \in \mathcal{P}^*$. Let us introduce variables $x_A, x_{B_1}, ..., x_{B_n}$. The intended meaning of a variable is that of denoting the value of the atom in the support, e.g. x_A will hold the value $s^H_{\mathcal{P}}(I)(A)$. Given $A \leftarrow f(B_1, \ldots, B_n) \in \mathcal{P}^*$ we consider the equation

$$\begin{array}{rcll} x_A & = & H \otimes [f(I(B_1), ..., I(B_n)) \oplus f(x_{B_1}, ..., x_{B_n}))] & (4) \\ & = & f^I_A(x_{B_1}, ..., x_{B_n})\ . & \end{array}$$

The above equation is the result of applying $\sigma^{I,H}_{\mathcal{P}}(J_i)$ to all rules using the fact that

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

and then replace $J_i(B_j)$ with the variable x_{B_j} and $J_{i+1}(A)$ with the variable x_A, as at the limit J_i will be the support. We denote the equational system by using Equation 4 above as $Supp^I_{\mathcal{P}}$ (the order is $\preceq_k$) and with $\mathbf{f}^I$ the tuple of functions f^I_A in Equation 4.

Example 5 *Consider Example 3 and an interpretation I. Then the corresponding equational system for computing the support is*

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

It can then be shown that:

Theorem 2 *For any program $\mathcal{P}$ and interpretation I, $s^H_{\mathcal{P}}(I) = gfp(\mathbf{f}^I, H)$.*

By Theorem 1, we have a top-down procedure to compute the truth of an atom (or set of atoms) in the support $s^H_{\mathcal{P}}(I)$.

Theorem 3 *$Solve(Supp^I_{\mathcal{P}}, Q, H, \downarrow)$ outputs a set $B \subseteq V$, with $Q \subseteq B$, such that the mapping $\mathtt{v}$ equals to the support $s^H_{\mathcal{P}}(I)$ on B, i.e. $\mathtt{v}_{|B} = s^H_{\mathcal{P}}(I)_{|B}$.*

Example 6 *Consider Example 5 and interpretation $I = \mathtt{I}_{\bot}$. Then the corresponding equational system for computing the support is $Supp^I_{\mathcal{P}}$,*

$$\begin{array}{rcl} x_A & = & \mathtt{f} \otimes (x_A \vee x_B)\ , \\ x_B & = & \mathtt{f} \otimes (\langle 0.3, 1\rangle \oplus ((\neg x_C \wedge x_A) \vee \langle 0.3, 0.5\rangle))\ , \\ x_C & = & \mathtt{f} \otimes (\langle 0.2, 1\rangle \oplus (\neg x_B \vee \langle 0.3, 0.5\rangle))\ . \end{array}$$

The hypothesis is $H = \mathtt{I}_{\bot}$. It can be verified that $s^H_{\mathcal{P}}(I)(A) = s^H_{\mathcal{P}}(I)(B) = \langle 0, 0.5\rangle$. Below is a sequence of $Solve(Supp^I_{\mathcal{P}}, \{x_A, x_B\}, H, \downarrow)$ computation w.r.t. $\preceq_k$ and the CWA $H = \mathtt{I}_{\mathtt{f}}$, returning the expected values. Each line is a sequence of steps in the 'while loop'. What is left unchanged is not reported.

1. $\mathtt{A}{:=}\{x_A, x_B\}, x_i{:=}x_A, \mathtt{A}{:=}\{x_B\}, \mathtt{dg}{:=}\{x_A, x_B\}$, $r{:=}\mathtt{f}, \mathtt{exp}(x_A){:=}\mathtt{true}, \mathtt{A}{:=}\{x_A, x_B\}, \mathtt{in}{:=}\{x_A, x_B\}$
2. $x_i{:=}x_B, \mathtt{A}{:=}\{x_A\}, \mathtt{dg}{:=}\{x_A, x_B, x_C\}, r{:=}\langle 0, 0.5\rangle$, $\mathtt{v}(x_B){:=}\langle 0, 0.5\rangle, \mathtt{A}{:=}\{x_A, x_C\}, \mathtt{exp}(x_B){:=}\mathtt{true}$, $\mathtt{in}{:=}\{x_A, x_B, x_C\}$
3. $x_i{:=}x_C, \mathtt{A}{:=}\{x_A\}, r{:=}\langle 0, 1\rangle, \mathtt{v}(x_C){:=}\langle 0, 1\rangle$, $\mathtt{A}{:=}\{x_A, x_B\}, \mathtt{exp}(x_C){:=}\mathtt{true}$
4. $x_i{:=}x_B, \mathtt{A}{:=}\{x_A\}, r{:=}\langle 0, 0.5\rangle$
5. $x_i{:=}x_A, \mathtt{A}{:=}\emptyset, r{:=}\langle 0, 0.5\rangle, \mathtt{v}(x_A){:=}\langle 0, 0.5\rangle$, $\mathtt{A}{:=}\{x_A, x_B\}$
6. $x_i{:=}x_A, \mathtt{A}{:=}\{x_B\}, r{:=}\langle 0, 0.5\rangle$
7. $x_i{:=}x_B, \mathtt{A}{:=}\emptyset, r{:=}\langle 0, 0.5\rangle$
9. $\mathtt{stop}$.
   $\mathtt{return}\ \mathtt{v}(x_A, x_B) = \langle\langle 0, 0.5\rangle, \langle 0, 0.5\rangle\rangle$

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

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

These steps correspond to the application of the $\Pi_{\mathcal{P}}(I) = \Phi_{\mathcal{P}}(I \oplus s^H_{\mathcal{P}}(I))$ operator to x_i. Indeed, if $x_i = f_i(x_{i_1}, ..., x_{i_{a_i}})$ is the definition of x_i in the equational system derived from $\mathcal{P}^*$ then, at first we ask about the value of the variables $x_{i_1}, ..., x_{i_{a_i}}$ in the support w.r.t. the current interpretation $\mathtt{I}{:=}\mathtt{v}$ (Steps 4.1 - 4.3). The variable $\mathtt{v}'$ holds these values. Finally, we evaluate

$$\Phi_{\mathcal{P}}(I \oplus s^H_{\mathcal{P}}(I))(x_i) = f_i(\mathtt{v}(x_{i_1}) \oplus \mathtt{v}'(x_{i_1}), ..., \mathtt{v}(x_{i_{a_i}}) \oplus \mathtt{v}'(x_{i_{a_i}}))\ .$$

We have that:

Theorem 4 *Let $\mathcal{P}$ and $?A$ be a logic program and a query, respectively. Then $HF_{\mathcal{P}}(A) = Solve_{HF}(\mathcal{P}, ?A)$.*

Example 7 *Consider Example 3 and query variable x_A. Table 4 reports a sequence of $Solve_{HF}(\mathcal{P}, ?A)$ computation. It resembles the one we have seen in Example 4. Each line is a sequence of steps in the 'while loop'. What is left unchanged is not reported.*

Note that the answer to $?A$, namely $\langle 0.3, 0.5\rangle$, is now more precise than the one ($\langle 0.3, 1\rangle$) under the Kripke-Kleene model (See Example 4), as expected.

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

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

Table 4: Top-down computation under the H-founded semantics.

The computational complexity analysis of $Solve_{HF}$ parallels the one we have made for $Solve_{KK}$. As the height of a lattice is finite then, like $Solve_{KK}$, each variable x_j will appear in $\mathtt{A}$ 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, $Solve_{HF}$ runs in time $O(|B_\mathcal{P}|^2a^2h^2c)$. That is, $Solve_{HF}$ runs in time $O(|\mathcal{P}^*|^2h^2c)$. If the lattice is fixed, then the height parameter is a constant. Furthermore, often we can assume that c is $O(1)$ and, thus, the worst-case complexity reduces to $O(|\mathcal{P}^*|^2)$.

Conclusions

We have presented a simple, yet general top-down query answering procedure for generalized logic programs over bilattices under the AWA and, thus, under the CWA and OWA. This gives us immediately an effective query answering procedure for those applications that the AWA covers (as many-valued logic programming with non-monotone negation and the representation of default rules using abnormality). The solution we proposed is based on the reduction of a generalized logic program into a system of equations of monotonic functions over lattices, for which we presented a top-down style procedure allowing to compute the value of an atom, or the values of a set of atoms, in the least solution of the system.

References

Alcantâra, J.; Damásio, C. V.; and Pereira, L. M. 2002. Paraconsistent logic programs. In *Proc. of the 8th European Conference on Logics in Artificial Intelligence (JELIA-02)*, number 2424 in Lecture Notes in Computer Science, 345–356. Cosenza, Italy: Springer-Verlag.

Alferes, J. J., and Pereira, L. M. 1992. On logic program semantics with two kinds of negation. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, 574–588. Washington, USA: The MIT Press.

Arieli, O., and Avron, A. 1996. Reasoning with logical bilattices. *Journal of Logic, Language and Information* 5(1):25–63.

Arieli, O., and Avron, A. 1998. The value of the four values. *Artificial Intelligence Journal* 102(1):97–141.

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

Belnap, N. D. 1977. A useful four-valued logic. In Epstein, G., and Dunn, J. M., eds., *Modern uses of multiple-valued logic*. Dordrecht, NL: Reidel. 5–37.

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

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

Damásio, C. V., and Pereira, L. M. 1998. A survey of paraconsistent semantics for logic programs. In Gabbay, D., and Smets, P., eds., *Handbook of Defeasible Reasoning and Uncertainty Management Systems*. Kluwer. 241–320.

Damásio, C. V., and Pereira, L. M. 2001. Antitonic logic programs. In *Proceedings of the 6th European Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, number 2173 in Lecture Notes in Computer Science. Springer-Verlag.

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

Denecker, M.; Marek, V. W.; and Truszczyński, M. 1999. Approximating operators, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In Minker, J., ed., *NFS-workshop on Logic-based Artificial Intelligence*, 1–26.

Denecker, M.; Marek, V. W.; and Truszczyński, M. 2002. Ultimate approximations in nonmonotonic knowledge representation systems. In Fensel, D.; Giunchiglia, F.;

McGuinness, D.; and Williams, M., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the 8th International Conference*, 177–188. Morgan Kaufmann.

Denecker, M.; Marek, V. W.; and Truszczyński, M. 2003. Uniform semantic treatment of default and autoepistemic logics. *Artificial Intelligence Journal* 143:79–122.

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

Doyle, J., and McDermott, D. 1980. Nonmonotonic logic I. *Artificial Intelligence* 13:41–72.

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

Fitting, M. 1988. Logic programming on a topological bilattice. *Fundamentae Mathematicae* XI:209–218.

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

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

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

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

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K., eds., *Proceedings of the 5th International Conference on Logic Programming*, 1070–1080. Cambridge, Massachusetts: The MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3/4):365–386.

Ginsberg, M. L., ed. 1987. *Readings in nonmonotonic reasoning*. Los Altos, CA: Morgan Kaufmann.

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

Hähnle, R., and Escalada-Imaz, G. 1997. Deduction in many-valued logics: a survey. *Mathware and Soft Computing* IV(2):69–97.

Hájek, P. 1998. *Metamathematics of Fuzzy Logic*. Kluwer.

Leone, N.; Rullo, P.; and Scarcello, F. 1997. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation* 135(2):69–112.

Lloyd, J. W. 1987. *Foundations of Logic Programming*. Heidelberg, RG: Springer.

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

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

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

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

Loyer, Y., and Straccia, U. 2005. Any-world assumptions in logic programming. *Theoretical Computer Science* 342(2-3):351–381.

Loyer, Y., and Straccia, U. 2006. Epistemic foundation of stable model semantics. *Journal of Theory and Practice of Logic Programming*. To appear.

Marek, V. W., and Truszczyński, M. 1991. Autoepistemic logic. *Journal of the ACM* 38(3):587–618.

McCarthy, J. 1980. Circumscription - a form of nonmonotonic reasoning. *Artificial Intelligence* 13:27–39.

McCarthy, J. 1986. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence* 28:89–116.

McDermott, D. 1982. Nonmonotonic logic II. *journal of the ACM* 22:33–57.

Moore, R. C. 1984. Possible-world semantics for autoepistemic logic. In *Proceedings of the 1st International Workshop on Nonmonotonic Reasoning*, 344–354.

Moore, R. C. 1985. Semantical considerations on nonmonotonic logic. *Artificial Intelligence* 25:75–94.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

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

van Gelder, A.; Ross, K. A.; and Schlimpf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 38(3):620–650.