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

Umberto Straccia

ISTI - CNR
Via G. Moruzzi, 1
56124 Pisa ITALY
straccia@isti.cnr.it

Abstract. 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. To answer queries, we provide a novel and simple tabling-like top-down procedure, which focuses on computing all answers to a query.

1 Introduction

The *Any-World Assumption* (AWA) for normal logic programs [27] 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* [19] and the semantics generalizes the notions of Kripke-Kleene, well-founded and stable model semantics [13, 14, 16, 38].

The AWA has many applications (see [27]), among which: (i) it covers *Extended Logic Programs* (ELPs) (e.g., [1, 2, 17]); (ii) it covers many-valued logic programming with non-monotone negation (e.g., [8, 35]); and (iii) it allows to represent default rules by relying on the so-called *abnormality theory* [30, 31].

In [27] a declarative and a fixed-point characterization for the AWA is presented. 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 $I(A)$. [36] provides a top-down query answering procedure. However, it requires the grounding of the logic program. Furthermore, queries are ground atoms only. This approach is clearly not satisfactory in case we are looking for *all* answers to a query atom of the form $q(\mathbf{x})$. Indeed, the size of the grounded instance of a logic program as well as the number of query instances $q(\mathbf{c})$ to query may be large and generally exponential with respect to the size of the non-ground expressions.

The topic of this paper is to further improve the query answering procedure related to the AWA. We present a simple, yet general tabulation-like top-down query answering procedure, which focuses on computing *all* answers of a query. This is important as it is quite natural that a user would like the answers $\mathbf{c}$ to a query $q(\mathbf{x})$ be *ranked* according to the degree of $q(\mathbf{c})$. A distinguishing feature of our query answering procedure is that we do not determine all answers by discovering *all proofs*, but rather apply a variant of so-called *memoing* techniques developed for classical logic programming – e.g., [39] for an overview. Essentially, the basic idea of our procedure is to collect, during the computation, all correct answers incrementally together in a similar way as it is done for classical Datalog [3, 37, 39]. Hence, for instance, we do not rely on any notion of *atom unification*, but rather iteratively access relational tables using relational algebra. Besides being the procedure novel for the AWA, we get for free a novel top-down query procedure for many-valued normal logic programs. This is the first time the issue of computing all answers has been addressed for many-valued normal logic programs under the OWA, CWA or more generally under the AWA in a many-valued semantics setting.

We proceed as follows. In the next two sections we recall a minimum of definitions about the AWA. Then we present our top-down query procedure.

2 Preliminaries

Bilattice. The truth spaces we consider are *bilattices* [19]. 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 [13–15]. A *bilattice* [19, 14] is a structure $\mathcal{B} = \langle B, \preceq_t, \preceq_k \rangle$ where B is a non-empty set and $\preceq_t$ (the *truth order*) and $\preceq_k$ (the *knowledge order*) are both partial orderings giving 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]. We also provide a family $\mathcal{F}$ of $\preceq_k$ and $\preceq_t$-monotone n-ary functions over 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 have been used in several ways. For instance, the simplest non-trivial bilattice, $\mathcal{FOUR}$ [4] ($B = \{\mathtt{f}, \mathtt{t}, \bot, \top\}$, $\mathtt{f} \preceq_t \bot, \top \preceq_t \mathtt{t}, \bot \preceq_k \mathtt{f}, \mathtt{t} \preceq_k \top, \neg\mathtt{f} = \mathtt{t}, \neg\bot = \bot, \neg\top = \top$), allows to deal with incomplete and/or inconsistent information. Fitting provided a fixed-point characterisation of stable model semantics on bilattices [13, 14]. Other well-established applications of bilattices can be found in the context of reasoning under paraconsistency, imprecision and uncertainty (see, e.g. [2, 5, 7, 8, 23–27, 35]). In practice, bilattices come up in natural ways. Indeed, there are

[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.

two general, but different, construction methods, which allow to build a bilattice from a lattice and are widely used (see also [13, 19]).

Generalized logic programs. We extend logic programs where *computable functions* $f \in \mathcal{F}$ are allowed to manipulate truth values (see [35, 36]). [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 [35].

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 literals, the truth values $b \in B$ of the bilattice and the functions $f \in \mathcal{F}$. Note that the members of the bilattice may appear in a formula, as well as functions $f \in \mathcal{F}$. 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 $x \wedge y = \max(0, x+y-1)$. 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. [27]). For ease, we will rely on the following simple example to illustrate the concepts we introduce in the paper.

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

$$\begin{array}{l} q(x) \leftarrow q(x) \vee \neg r(x) \\ p(x) \leftarrow p(x) \\ r(a) \leftarrow \mathtt{t} \qquad r(b) \leftarrow \mathtt{f} \end{array}$$

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

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

	I_i							
	$q(a)$	$q(b)$	$r(a)$	$r(b)$	$p(a)$	$p(b)$	$KK(\mathcal{P})$	$WF(\mathcal{P})$
I_1	$\perp$	t	t	f	$\perp$	$\perp$	●	
I_2	f	t	t	f	f	f		●
I_3	t	t	t	f	t	f		

	H_i					
	$q(a)$	$q(b)$	$r(a)$	$r(b)$	$p(a)$	$p(b)$
H_1	f	f	f	f	f	f
H_2	f	f	f	f	f	f
H_3	t	f	f	f	t	f

	$s_{\mathcal{P}}^{H_i}(I_i)$						
	$q(a)$	$q(b)$	$r(a)$	$r(b)$	$p(a)$	$p(b)$	$U_{\mathcal{P}}(I_i)$
I_1	f	$\perp$	$\perp$	f	f	f	$\{q(a), r(b), p(a), p(b)\}$
I_2	f	$\perp$	$\perp$	f	f	f	$\{q(a), r(b), p(a), p(b)\}$
I_3	$\perp$	$\perp$	$\perp$	f	t	f	$-$

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

Interpretations. An *interpretation* I on the bilattice $\mathcal{B} = \langle B, \preceq_t, \preceq_k \rangle$ is a mapping from atoms to members of B. I is extended from atoms to formulae in the usual way: (i) for $b \in B$, $I(b) = b$; (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 B to the set $\mathcal{I}(B)$ 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 $I_{\mathtt{f}}$ and $I_\perp$ we denote the bottom interpretations under $\preceq_t$ and $\preceq_k$ respectively (they map any atom into f and $\perp$, respectively). $\langle \mathcal{I}(B), \preceq_t, \preceq_k \rangle$ is a 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., [14, 27, 28]). 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* ($WF_{\mathcal{P}}$) [13, 14, 38]. 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 [13, 14, 28]. It is the generalization of the classical well-founded semantics to bilattices. We obtain it as a special case of the AWA, too. Furthermore, 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 $I_\perp$.

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., [20, 38]), 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 $U_{\mathcal{P}}(I)$. Then, the well-founded semantics $WF_{\mathcal{P}}$ is defined to be [20]:

$$WF_{\mathcal{P}} = \text{“ } \preceq_k\text{-least model } I \text{ of } \mathcal{P} \text{ such that } \neg.U_{\mathcal{P}}(I) \subseteq I\text{”.}$$

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

3 The AWA in logic programming

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, $H = I_{\mathtt{f}}$ is assumed, while under the OWA, $H = I_{\perp}$ 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 and that of $\Phi_{\mathcal{P}}$ (which now maps such atoms into $H(A)$ rather than into $\mathtt{f}$). To emphasize the impact of H to $\Phi_{\mathcal{P}}$, we denote the immediate consequence operator with $\Phi^H_{\mathcal{P}}$ in place of $\Phi_{\mathcal{P}}$. Now, we proceed in two steps.

The support. At first, we introduce the notion of *support*, denoted $s^H_{\mathcal{P}}(I)$. The support is a generalization of the notion of unfounded sets. Indeed, $s^H_{\mathcal{P}}(I)$ 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 $H = I_{\mathtt{f}}$ (see Table 1), $s^H_{\mathcal{P}}(I) = \neg.U_{\mathcal{P}}(I)$ [27].

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 $A \leftarrow \varphi \in \mathcal{P}^*$, $J(A) \preceq_k (I \oplus J)(\varphi) = \Phi^H_{\mathcal{P}}(I \oplus J)(A)$ should hold. Therefore, we say that an interpretation J is *safe* w.r.t. $\mathcal{P}$, I and H iff

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

Note that safe interpretations correspond to unfounded sets for classical logic programs [27]. 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 $H = I_\perp$, $s^H_{\mathcal{P}}(I) = I_\perp$ holds, as expected, while for classical logic programs $s^H_{\mathcal{P}}(I) = \neg.U_{\mathcal{P}}(I)$, for $H = I_{\mathbf{f}}$. 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

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

Indeed, [27] shows that the iterated sequence of interpretations J_i below is $\preceq_k$-monotone decreasing and reaches a fixed-point, $J_\lambda = s^H_{\mathcal{P}}(I)$, for a limit ordinal λ.

$$\begin{array}{ll} J_0 & = H \\ J_{i+1} & = \sigma^{I,H}_{\mathcal{P}}(J_i) \\ J_\lambda & = \inf_{i<\lambda} \sigma^{I,H}_{\mathcal{P}}(J_i) \ . \end{array}$$

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 [27].

Proposition 1 ([27]). *I is a H-model of $\mathcal{P}$ iff $I = \Phi^H_{\mathcal{P}}(I \oplus s^H_{\mathcal{P}}(I))$.*

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 the $\preceq_k$-monotone immediate consequence operator:

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

This guarantees the existence and uniqueness of the $\preceq_k$-least fixed-point of $\Pi^H_{\mathcal{P}}(I)$, i.e., the H-founded model of a program $\mathcal{P}$.

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 $\neg.U_{\mathcal{P}}(I) \subseteq I$ [20], which is a special case of the definition of H-founded model). We conclude by remarking that [27] also generalizes the stable model semantics to the AWA.

Example 2. Consider Example 1 and Table 1. Given the hypothesis H_i described in the tables (note that $H_1 = H_2 = I_{\mathtt{f}}$, i.e., the CWA is assumed), we observe that $s_{\mathcal{P}}^{H_i}(I_i) \preceq_k I_i$ for $i = 2, 3$ and, thus, both I_2 and I_3 are H-models, while I_1 is not. Furthermore, it can be verified that both I_2 and I_3 are also H-founded models and that I_2 corresponds to the classical well-founded semantics, as expected.

We refer the reader to [27, 36] for some applications of the AWA. For the sake of illustrative purposes, we recall the following example: a rule expressing the fact that *a car may cross railway tracks if there is no crossing train* may be represented by

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

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

Another example 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* [34], Autoepistemic Logic [11, 29, 32, 33] and *Circumscription* [30, 31] (see also [18]). We can formulate such a statement in a natural way, using *abnormality theories*, as

$$\begin{array}{l} A \leftarrow \varphi \land \neg Ab \\ Ab \leftarrow \neg A\ , \end{array}$$

where Ab stands for *abnormality*, and then consider the hypothesis $H(Ab) = \mathtt{f}$, i.e., by default there are no abnormal objects.

4 Top-down query answering

A *query* is an atom Q (*query atom*) of the form $q(\mathbf{x})$, intended as a question about the truth degree of all the instances of Q in the intended model of $\mathcal{P}$. We also allow a query to be a *set* $\{Q_1, \ldots, Q_n\}$ of query atoms. In that latter case we ask about the truth degree of all instances of the atoms Q_i in the intended model.

The procedure we devise in this paper is a generalization of the procedure presented in [36]. We anticipate that the main reason why the procedure in [36] is not suitable to be used for computing all answers to a query Q, given $\mathcal{P}$, is that

- [36] relies on $\mathcal{P}$'s grounded version $\mathcal{P}^*$, which may be rather huge (exponential with respect to $|\mathcal{P}|$, in general) in applications with many facts;
- [36] answers ground queries only. Strictly speaking, [36] can compute all answers of a query atom $q(\mathbf{x})$ by submitting as query the set of all ground instances $q(\mathbf{c})$. This is clearly not feasible if the Herbrand universe is large.

The procedure presented here does not require grounding. In the following, we assume that a logic program $\mathcal{P}$ is made out of an *extensional database* (EDB), $\mathcal{P}_E$, and an *intensional database* (IDB), $\mathcal{P}_I$. The extensional database is a set of facts of the form

$$r(c_1, \ldots, c_n) \leftarrow b\ ,$$

where $r(c_1,\dots,c_n)$ is a ground atom and b is a truth value. For convenience, for each n-ary extensional predicate r, we represent the facts $r(c_1,\dots,c_n) \leftarrow b$ in $\mathcal{P}$ by means of a relational $n+1$-ary table tab_r, containing the records $\langle c_1,\dots,c_n,b\rangle$. Thus, the table contains all the instances of r together with their degrees. We assume that there cannot be two records $\langle c_1,\dots,c_n,b_1\rangle$ and $\langle c_1,\dots,c_n,b_2\rangle$ in tab_r with $b_1 \neq b_2$.

The intensional database is a set of rules for the form

$$p(\mathbf{x}) \leftarrow \varphi(\mathbf{x},\mathbf{y}) \tag{1}$$

in which the predicates occurring in the extensional database (called *extensional predicates*) do not occur in the head of rules of the intensional database. Essentially, we do not allow that the fact predicates occurring in $\mathcal{P}_E$ can be redefined by $\mathcal{P}_I$. We also assume that the *intensional predicate* symbol p occurs in the head of at most one rule in the intensional database. Due to the expressiveness of rule bodies, it is not difficult to see that, possibly defining an equality predicate $Eq(x,y)$, logic programs can be put into this form.

For an atom A of the form $p(\mathbf{x})$, an *answer* for p is a pair $\langle\theta,b\rangle$, where $\theta=\{\mathbf{x}/\mathbf{c}\}$ is a substitution of the variables $\mathbf{x}$ in $p(\mathbf{x})$ with the constants in $\mathbf{c}$ and $b\in L$ is a truth degree. We say that the answer $\langle\theta,b\rangle$ is *correct* for p with respect to the intended model I of $\mathcal{P}$ iff $I(p(\mathbf{c}))=b$. That is, by substituting the variables in $\mathbf{x}$ using θ, the evaluation of the query in the intended model is b. An *answer set* for p is a set of answers for p. Of course, our goal is to determine the set of all correct answers for the query Q. For a given n-ary predicate p and a set of answers Δ_p of p, for convenience we represent Δ_p as an $n+1$-ary table tab_{Δ_p}, containing the records $\langle c_1,\dots,c_n,b\rangle$.

Given two answers $\delta_1=\langle\theta,b_1\rangle$ and $\delta_2=\langle\theta,b_2\rangle$ for the same atom P, we define $\delta_1 \preceq_k \delta_2$ ($\delta_1 \succeq_k \delta_2$) iff $b_1 \preceq_k b_2$ ($b_1 \succeq_k b_2$). We write $\delta_1 \prec_k \delta_2$ ($\delta_1 \succ_k \delta_2$) iff $b_1 \prec_k b_2$ ($b_1 \succ_k b_2$). If Δ_p^1 and Δ_p^2 are two sets of answers for p, we write $\Delta_p^1 \preceq_k \Delta_p^2$ ($\Delta_p^1 \succeq_k \Delta_p^2$) iff for all $\delta_1\in\Delta_p^1$ there is $\delta_2\in\Delta_p^2$ such that $\delta_1 \preceq_k \delta_2$ ($\delta_1 \succeq_k \delta_2$). We write $\Delta_p^1 \prec_k \Delta_p^2$ ($\Delta_p^1 \succ_k \Delta_p^2$) iff $\Delta_p^1 \preceq_k \Delta_p^2$ ($\Delta_p^1 \succeq_k \Delta_p^2$) and there is $\delta_2\in\Delta_p^2$ such that for no $\delta_1\in\Delta_p^1$, $\delta_2 \preceq_k \delta_1$ ($\delta_2 \succeq_k \delta_1$) holds.

We present now our top-down tabling like procedure tailored to compute all correct answer of a query Q in the intended model. The basic idea of our procedure is to try to collect, during the computation, all correct answers incrementally together. The procedure can be related to the so-called *memoing* techniques (*tabling/magic sets*) developed for classical logic programming –see e.g., [39] for an overview.

At first, consider a general rule of the form $p(\mathbf{x}) \leftarrow \varphi(\mathbf{x},\mathbf{y})$. We note that $\varphi(\mathbf{x},\mathbf{y})$ depends on a computable function f and the predicates $p_1,\dots,p_k$, which occur in the rule body $\varphi(\mathbf{x},\mathbf{y})$. Assume that $\Delta_{p_1},\dots,\Delta_{p_k}$ are the answers collected so far for the predicates $p_1,\dots,p_k$. Let us consider a procedure $eval(p,\Delta_{p_1},\dots,\Delta_{p_k})$, which computes the set of answers $\langle\{\mathbf{x}/\mathbf{c}\},b\rangle$ of p, by evaluating the body $\varphi(\mathbf{x},\mathbf{y})$ over the data provided by $\Delta_{p_1},\dots,\Delta_{p_k}$. Formally, let H be a hypothesis, let I^H be an interpretation restricted to the predicates $p_1,\dots,p_k$ and tuples such that for all n_i-ary predicates p_i,

$$I^H(p_i(\mathbf{c})) = \begin{cases} b, & \text{if } \langle\mathbf{c},b\rangle \in tab_{\Delta_{p_i}} \\ H(p_i(\mathbf{c})) & \text{if } p_i \text{ is an extensional predicate and } \langle\mathbf{c},b\rangle \notin tab_{\Delta_{p_i}} \\ \bot & \text{otherwise .} \end{cases}$$

The intuition in the definition above is that to an atom $p_i(\mathbf{c})$ we assign the current truth value if this truth value is known. Otherwise, we assign to it the default truth value taken from the hypothesis (if p_i is an extensional predicate). Then

$$eval(p, H, \Delta_{p_1}, \ldots, \Delta_{p_k}) = \{\langle\{\mathbf{x}/\mathbf{c}\}, b\rangle \mid b = \bigvee_{\mathbf{c}'} I^H(\varphi(\mathbf{c}, \mathbf{c}')), b \neq \bot\} \;, \qquad (2)$$

where $\mathbf{c}'$ is a tuple of constants occurring in $\bigcup_i \Delta_{p_i}$. We omit to report the tuple whose degree is $\bot$. The disjunction $\bigvee_{\mathbf{c}'}$ is required as the free variables $\mathbf{y}$ in $\varphi(\mathbf{x}, \mathbf{y})$ may be seen as existentially quantified.

Example 3. Consider $\mathcal{P} = \{p(x) \leftarrow q(x, y), q(a, b) \leftarrow \mathtt{f}, q(a, c) \leftarrow \mathtt{t}\}$. Assume $\Delta_q = \{\langle(a, b), \mathtt{f}\rangle, \langle(a, c), \mathtt{t}\rangle\}$. Then $eval(p, \Delta_q) = \{\langle a, \mathtt{t}\rangle\}$, which amounts to evaluate $q(a, b) \vee q(a, c)$.

We are not going to further investigate the implementation details of the $eval(p, H, \Delta_{p_1}, \ldots, \Delta_{p_k})$ procedure, though it has to be carefully written to minimize the number of table look-ups and relational algebraic operations such as joins. It can be obtained by means of a combination of SQL statements over the tables and the application of the truth combination functions occurring in the rule body of p. We point out that $eval(p, H, \Delta_{p_1}, \ldots, \Delta_{p_k})$ can also be seen as a query to a database made out by the relations $tab_{\Delta_{p_1}}, \ldots, tab_{\Delta_{p_k}}$ and that any successive evaluation step corresponds to the execution of the *same* query over an updated database. We refer the reader to e.g., [9, 10, 22] concerning the problem of repeatedly evaluating the same query to a database that is being updated between successive query requests. In this situation, it may be possible to use the difference between successive database states and the answer to the query in one state to reduce the cost of evaluating the query in the next state.

4.1 Query answering: Kripke-Kleene semantics

We start showing how to compute all answers with respect to the Kripke-Kleene semantics, i.e., the $\preceq_k$-least fixed-point of $\Phi^H_{\mathcal{P}}$. The procedure is detailed in Table 2. Assume, we are interested in determining all correct answers of $q(\mathbf{x})$ w.r.t. the Kripke-Kleene semantics. We call the procedure with $Answer(\mathcal{P}, Q, H)$. We start with putting the predicate symbols $q \in Q$ in the *active* list of predicate symbols $\mathtt{A}$. At each iteration step (step 2) we select a new predicate p from the queue $\mathtt{A}$ and evaluate it using the $eval$ function with respect to the answers gathered so far (steps 4 or 5). If the evaluation leads to a better answer set for p (step 6), we update the current answer set $\mathtt{v}(p)$ and add all predicates p', whose rule body contains p (the parents of p), to the queue $\mathtt{A}$, i.e., all predicate symbols that might depend on p 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 all correct answers of $q(\mathbf{x})$. The procedure in Table 2 uses some auxiliary functions and data structures:

- for predicate symbol p_i, $\mathtt{s}(p_i)$ is the set of predicate symbols occurring in the rule body of p_i, i.e., the *sons* of p_i;
- for predicate symbol p_i, $\mathtt{p}(p_i) = \{p_j : p_i \in \mathtt{s}(p_j)\}$, i.e., the *parents* of p_i;

```
**Procedure** $Answer(\mathcal{P}, Q, H)$
**Input:** Logic program $\mathcal{P}$, set $Q$ of query predicate symbols, hypothesis $H$;
**Output:** Mapping $\mathtt{v}$ containing all correct answers of predicates in $Q$ w.r.t. lfp($\Phi^H_\mathcal{P}$)
1. $\mathtt{A} := Q$, $\mathtt{dg} := Q$, $\mathtt{in} := \emptyset$, for all predicate symbols $p$ in $\mathcal{P}$ do $\mathtt{v}(p) = \emptyset$, $\mathtt{exp}(p) = \mathtt{false}$
2. while $\mathtt{A} \neq \emptyset$ do
3.    select $p_i \in \mathtt{A}$, $\mathtt{A} := \mathtt{A} \setminus \{p_i\}$, $\mathtt{dg} := \mathtt{dg} \cup \mathtt{s}(p_i)$
4.    if ($p_i$ extensional predicate) $\wedge$ ($\mathtt{v}(p_i) = \emptyset$) then $\mathtt{v}(p_i) := tab_{p_i}$
5.    if ($p_i$ intensional predicate) then $\Delta_{p_i} := eval(p_i, H, \mathtt{v}(p_{i_1}), ..., \mathtt{v}(p_{i_{k_i}}))$
6.    if $\mathtt{v}(p_i) \prec_k \Delta_{p_i}$ then $\mathtt{v}(p_i) := \Delta_{p_i}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{p}(p_i) \cap \mathtt{dg})$
7.    if not $\mathtt{exp}(p_i)$ then $\mathtt{exp}(p_i) = \mathtt{true}$, $\mathtt{A} := \mathtt{A} \cup (\mathtt{s}(p_i) \setminus \mathtt{in})$, $\mathtt{in} := \mathtt{in} \cup \mathtt{s}(p_i)$
   endwhile
```

Table 2. General top-down algorithm.

- in step 5, $p_{i_1}, \ldots, p_{i_{k_i}}$ are all predicate symbols occurring in the rule body of p_i, i.e., the sons $\mathtt{s}(p_i) = \{p_{i_1}, \ldots, p_{i_{k_i}}\}$ of p_i;
- the variable $\mathtt{dg}$ collects the predicate symbols that may influence the result of the query predicates;
- the array variable $\mathtt{exp}$ traces the rule bodies that have been "expanded" (the predicate symbols occurring in the rule body are put into the active list);
- the variable $\mathtt{in}$ keeps track of the predicate symbols that have been put into the active list so far due to an expansion (to avoid, to put the same predicate symbol multiple times in the active list due to rule body expansion).

Example 4. Consider Example 1. Let us consider the hypothesis $H = I_\perp$ (i.e., the OWA). The extensional database is shown in the relational table $tab_r = \{\langle a, \mathtt{t}\rangle, \langle b, \mathtt{f}\rangle\}$. Of course, tab_r is also the set tab_{Δ_r} of correct answers of predicate r, while it can be verified (by a straightforward bottom-up fixed-point computation iterating $\Phi^H_\mathcal{P}$ over $I_\perp$) that the set of correct answers of predicate q is given by: $\Delta_q = \{\langle b, \mathtt{t}\rangle\}$. We do not report the tuple $\langle a, \perp\rangle$, as if **c** does not occur in an answer set Δ then its truth degree is assumed to be $\perp$.

We next show the computation of $Answer(\mathcal{P}, \{q\}, H)$. The execution is shown below reporting also Δ_{p_i} and $\mathtt{v}(p_i)$ at each iteration i. Each line is a sequence of steps in the 'while loop'. What is left unchanged is not reported.

```
1. $\mathtt{A} := \{q\}, p_i := q, \mathtt{A} := \emptyset, \mathtt{dg} := \{q, r\}, \Delta_q := \emptyset$
   $\mathtt{exp}(q) := 1, \mathtt{A} := \{q, r\}, \mathtt{in} := \{q, r\}$
2. $p_i := q, \mathtt{A} := \{r\}, \Delta_q := \emptyset$
3. $p_i := r, \mathtt{A} := \emptyset, \mathtt{v}(r) \prec_k \Delta_r, \mathtt{v}(r) := \Delta_r, \mathtt{A} := \{q\}, \mathtt{exp}(r) := 1$
4. $p_i := q, \mathtt{A} := \emptyset, \mathtt{v}(q) \prec_k \Delta_q, \mathtt{v}(q) := \Delta_q, \mathtt{A} := \{q\}$
5. $p_i := q, \mathtt{A} := \emptyset, \Delta_q = \mathtt{v}(q)$
6. stop. return $\mathtt{v}(q)$
```

Iter i	Δ_{p_i}	$\mathtt{v}(p_i)$
0.	–	$\mathtt{v}(p_i) = \emptyset$
1.	$\Delta_q = \emptyset$	–
2.	$\Delta_q = \emptyset$	–
3.	$\Delta_r = \{\langle a, \mathtt{t}\rangle, \langle b, \mathtt{f}\rangle\}$	$\mathtt{v}(r) = \Delta_r$
4.	$\Delta_q = \{\langle b, \mathtt{t}\rangle\}$	$\mathtt{v}(q) = \Delta_q$
5.	$\Delta_q = \{\langle b, \mathtt{t}\rangle\}$	–

It can be shown that the procedure $Answer$ behaves as expected.

Proposition 2. *There is a limit ordinal λ such that after $|\lambda|$ steps $Answer(\mathcal{P}, Q, H)$ returns the set of all correct answers of $\mathcal{P}$ with respect to the predicates in Q and the Kripke-Kleene semantics under hypothesis H.*

4.2 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 $\Pi^H_{\mathcal{P}}$ (see Proposition 1) 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. In the following, we show how we can slightly change the $Answer$ procedure to compute the support. That is, we want a top-down procedure that, for a set of atoms $p(\mathbf{x})$, computes all answers $\langle \{\mathbf{x}/\mathbf{c}\}, b\rangle$ such that $s^H_{\mathcal{P}}(I)(p(\mathbf{c})) = b$.

So, let $Support(\mathcal{P}, Q, H, I)$ be the procedure, which is as the $Answer$ procedure except that:

– Step 1 is replaced with

```
$\mathcal{P} := \mathcal{P}^H_I$, $\texttt{A} := Q$, $\texttt{dg} := Q$, $\texttt{in} := \emptyset$,
**for all** predicate symbols $p$ in $\mathcal{P}$ **do** $\texttt{v}(p) = \emptyset$, $\texttt{exp}(p) = \texttt{false}$
```

The logic program $\mathcal{P}^H_I$ is obtained from $\mathcal{P}$ in the following way:

- for each intensional predicate p in $\mathcal{P}$, replace the rule $p(\mathbf{x}) \leftarrow \varphi(\mathbf{x}, \mathbf{y})$ in $\mathcal{P}$ with the rule
$$p(\mathbf{x}) \leftarrow H(p)(\mathbf{x}) \otimes (I(p_\varphi)(\mathbf{x}) \oplus \varphi(\mathbf{x}, \mathbf{y})) \ . \tag{3}$$
With $H(p)(\mathbf{x})$ we mean a built-in predicate that given a substitution $\mathbf{c}$ for $\mathbf{x}$, returns $H(p(\mathbf{c}))$. This can easily be encoded in the semantics, which we omit. The case $I(p_\varphi)(\mathbf{x})$ is similar: $I(p_\varphi)(\mathbf{x})$ is a built-in predicate that given a substitution $\mathbf{c}$ for $\mathbf{x}$, returns $\bigvee_{\mathbf{c}'} I(\varphi(\mathbf{c}, \mathbf{c}'))$.
- for each extensional predicate r in $\mathcal{P}$, replace the rule $r(\mathbf{c}) \leftarrow b$ in $\mathcal{P}$ with the rule
$$r(\mathbf{c}) \leftarrow b' \ , \tag{4}$$
where b' is the truth value $b' = H(r(\mathbf{c})) \otimes b$.

We point out that the rules above are the result of applying $\sigma^{I,H}_{\mathcal{P}}$ to the support $s^H_{\mathcal{P}}(I)$ and to all rules:

$$\begin{aligned} s^H_{\mathcal{P}}(I)(p(\mathbf{c})) &= [H \otimes \Phi^H_{\mathcal{P}}(I \oplus s^H_{\mathcal{P}}(I))](p(\mathbf{c})) \\ &= H(p(\mathbf{c})) \otimes [I \oplus s^H_{\mathcal{P}}(I)](\textstyle\bigvee_{\mathbf{c}'} \varphi(\mathbf{c}, \mathbf{c}')) \\ &= H(p(\mathbf{c})) \otimes (I(\textstyle\bigvee_{\mathbf{c}'} \varphi(\mathbf{c}, \mathbf{c}')) \oplus s^H_{\mathcal{P}}(I)(\bigvee_{\mathbf{c}'} \varphi(\mathbf{c}, \mathbf{c}'))) \\ &= H(p(\mathbf{c})) \otimes (\textstyle\bigvee_{\mathbf{c}'} I(\varphi(\mathbf{c}, \mathbf{c}')) \oplus \bigvee_{\mathbf{c}'} s^H_{\mathcal{P}}(I)(\varphi(\mathbf{c}, \mathbf{c}'))) \ . \end{aligned}$$

Since the above equation holds for all predicates p and all $\mathbf{c}$, we get rule (3) and (4). Build-in predicates do not count as sons and, thus, do not appear in the $\texttt{A}, \texttt{s}, \texttt{p}, \texttt{v}, \texttt{in}, \texttt{dg}$ variables.

– Step 6 is replaced with

$$\textbf{if } \mathtt{v}(p_i) \succ_k \varDelta_{p_i} \textbf{ then } \mathtt{v}(p_i) := \varDelta_{p_i}, \mathtt{A} := \mathtt{A} \cup (\mathtt{p}(p_i) \cap \mathtt{dg}) \textbf{ fi}$$

Essentially, in Step 6 we replace $\prec_k$ with $\succ_k$. This modification is motivated by the fact that during the computation of the support, $\varDelta_{p_i}$ is now decreasing in the knowledge order $\preceq_k$.

Example 5. Consider Example 1, interpretation I_2 and hypothesis H_2. We have seen that I_2 is the H-founded model of $\mathcal{P}$ w.r.t. H_2 and corresponds to the well-founded semantics of $\mathcal{P}$. We next want to show the computation of $Support(\mathcal{P}, \{q, r\}, H_2, I_2)$. We first determine $\mathcal{P}_{I_2}^{H_2}$. As predicate p does not play any role in the computation, we report the modified rule for predicate q and r only. $\mathcal{P}_{I_2}^{H_2}$ related to q and r is

$$\begin{array}{l} q(x) \leftarrow H_2(q)(x) \otimes (I_2(q_\varphi)(x) \oplus (q(x) \vee \neg r(x))) \\ r(a) \leftarrow \bot \qquad r(b) \leftarrow \mathtt{f}\ . \end{array}$$

We recall that $H_2(q)(a) = H_2(q)(b) = \mathtt{f}$ and that $I_2(q_\varphi)(a) = I_2(q(a) \vee \neg r(a)) = \mathtt{f}$, while $I_2(q_\varphi)(b) = \mathtt{t}$. Then, it can be verified that (by a straightforward fixed-point computation iterating $\sigma_{\mathcal{P}}^{I,H}$ starting with H_2) that the set of correct answers of predicate q, r of $\mathcal{P}$ w.r.t. $s_{\mathcal{P}}^{H_2}(I_2)$ are: $\varDelta_q = \{\langle a, \mathtt{f}\rangle\}, \varDelta_r = \{\langle b, \mathtt{f}\rangle\}$.

Below is a sequence of $Support(\mathcal{P}, \{q, r\}, H_2, I_2)$, returning the expected values.

```
1. $\mathtt{A} := \{q, r\}, p_i := q, \mathtt{A} := \{r\}, \mathtt{dg} := \{q, r\}, \varDelta_q \succ_k \mathtt{v}(q),$
   $\mathtt{exp}(q) := \mathtt{1}, \mathtt{A} := \{r, q\}, \mathtt{in} := \{q, r\}$
2. $p_i := r, \mathtt{A} := \{q\}, \mathtt{v}(r) \succ_k \varDelta_r, \mathtt{v}(r) := \varDelta_r, \mathtt{exp}(r) := \mathtt{1}$
3. $p_i := q, \mathtt{A} := \emptyset, \varDelta_q = \mathtt{v}(q)$
4. stop. return $\mathtt{v}(q)$
```

Iter i	$\varDelta_{p_i}$	$\mathtt{v}(p_i)$
0.	–	$\mathtt{v}(p_i) = \emptyset$
1.	$\varDelta_q = \{\langle a, \mathtt{f}\rangle\}$	$\mathtt{v}(q) = \varDelta_q$
2.	$\varDelta_r = \{\langle b, \mathtt{f}\rangle\}$	$\mathtt{v}(r) = \varDelta_r$
3.	$\varDelta_q = \{\langle a, \mathtt{f}\rangle\}$	–

It can then be shown that:

Proposition 3. *There is a limit ordinal λ such that after $|\lambda|$ steps $Support(\mathcal{P}, Q, H, I)$ returns the set of all correct answers of $\mathcal{P}$ with respect to the predicates in Q and the support $s_{\mathcal{P}}^{H}(I)$.*

We are now ready to define the top-down procedure $Answer_{HF}(\mathcal{P}, Q, H)$, which computes all correct answers to a query Q under the H-founded semantics. We define $Answer_{HF}(\mathcal{P}, Q, H)$ as $Answer(\mathcal{P}, Q, H)$, except that Step 5 is replaced with the statements

```
5.    if ($p_i$ intensional predicate) then
5.1.      $\mathtt{Q}' := \mathtt{s}(p_i)$;
5.2.      $\mathtt{I} := \mathtt{v}$;
5.3.      $\mathtt{supp} := Support(\mathcal{P}, \mathtt{Q}', H, \mathtt{I})$;
5.4.      $\mathtt{v}' := \mathtt{I} \oplus \mathtt{supp}$;
5.5.      $\varDelta_{p_i} := eval(p_i, H, \mathtt{v}'(p_{i_1}), ..., \mathtt{v}'(p_{i_{k_i}}))$ fi
```

These steps correspond to the application of the $\Pi^H_{\mathcal{P}}(I) = \Phi^H_{\mathcal{P}}(I \oplus s^H_{\mathcal{P}}(I))$ operator to p_i. Indeed, at first we ask about all the correct answers of the predicates occurring in the body of p_i w.r.t. the support and the current interpretation $\mathtt{I}:\ = \mathtt{v}$ (Steps 5.1 - 5.3). The variable supp holds these answers. Then we join them with I, i.e., we compute $I \oplus s^H_{\mathcal{P}}(I)$ (Step 5.4), where this latter is defined pointwise: (i) $\mathtt{v}' = \mathtt{v}_1 \oplus \mathtt{v}_2$ iff for all p, $\mathtt{v}'(p) = \mathtt{v}_1(p) \oplus \mathtt{v}_2(p) = \{\langle \theta, b\rangle \mid \langle \theta, b_1\rangle \in \mathtt{v}_1(p), \langle \theta, b_2\rangle \in \mathtt{v}_2(p), b = b_1 \oplus b_2\}$ (if $\langle \theta, b_i\rangle \notin \mathtt{v_i}(p)$ then $b_i = \bot$ is assumed). Finally, we evaluate the body of p_i with respect to $I \oplus s^H_{\mathcal{P}}(I)$ (Step 5.5), i.e., apply $\Phi^H_{\mathcal{P}}(I \oplus s^H_{\mathcal{P}}(I))$.

Example 6. Consider Example 1 and hypothesis H_2 (i.e., the CWA). Let us compute all correct answers to the query $q(x)$ w.r.t. the well-founded semantics. As the interpretation I_2 in Example 6 is the well-founded model (i.e., H_2-founded model), we expect to retrieve $\Delta_q = \{\langle a, \mathtt{f}\rangle, \langle b, \mathtt{t}\rangle\}$. Below is the computation of $Answer_{HF}(\mathcal{P}, \{q\}, H_2)$.

```
1. $\mathtt{A} := \{q\}, p_i := q, \mathtt{A} := \emptyset, \mathtt{dg} := \{q, r\}, \mathtt{supp} := \{\langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle r(b), \mathtt{f}\rangle\},$
   $\mathtt{v}(q) \prec_k \Delta_q, \mathtt{exp}(q) := 1, \mathtt{A} := \{q, r\}, \mathtt{in} := \{q, r\}$
2. $p_i := q, \mathtt{A} := \{r\}, \mathtt{supp} := \{\langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle q(b), \mathtt{t}\rangle, \langle r(b), \mathtt{f}\rangle\}, \Delta_q = \mathtt{v}(q)$
3. $p_i := r, \mathtt{A} := \emptyset, \mathtt{supp} := \{\langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle q(b), \mathtt{t}\rangle, \langle r(b), \mathtt{f}\rangle\},$
   $\mathtt{v}(r) \prec_k \Delta_r, \mathtt{v}(r) := \Delta_r, \mathtt{A} := \{q\}, \mathtt{exp}(r) := 1$
4. $p_i := q, \mathtt{A} := \emptyset, \mathtt{supp} := \{\langle q(a), \mathtt{f}\rangle, \langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle q(a), \mathtt{f}\rangle, \langle q(b), \mathtt{t}\rangle, \langle r(a), \mathtt{t}\rangle, \langle r(b), \mathtt{f}\rangle\},$
   $\mathtt{v}(q) \prec_k \Delta_q, \mathtt{v}(q) := \Delta_q, \mathtt{A} := \{q\}$
5. $p_i := q, \mathtt{A} := \emptyset, \mathtt{supp} := \{\langle q(a), \mathtt{f}\rangle, \langle r(b), \mathtt{f}\rangle\}, \mathtt{v}' := \{\langle q(a), \mathtt{f}\rangle, \langle q(b), \mathtt{t}\rangle, \langle r(a), \mathtt{t}\rangle, \langle r(b), \mathtt{f}\rangle\},$
   $\Delta_q = \mathtt{v}(q)$
6. stop. return $\mathtt{v}(q)$
```

Iter i	Δ_{p_i}	$\mathtt{v}(p_i)$
0.	–	$\mathtt{v}(p_i) = \emptyset$
1.	$\Delta_q = \{\langle b, \mathtt{t}\rangle\}$	$\mathtt{v}(q) = \Delta_q$
2.	$\Delta_q = \{\langle b, \mathtt{t}\rangle\}$	–
3.	$\Delta_r = \{\langle a, \mathtt{t}\rangle, \langle b, \mathtt{f}\rangle\}$	$\mathtt{v}(r) = \Delta_r$
4.	$\Delta_q = \{\langle a, \mathtt{f}\rangle, \langle b, \mathtt{t}\rangle\}$	$\mathtt{v}(q) = \Delta_q$
5.	$\Delta_q = \{\langle a, \mathtt{f}\rangle, \langle b, \mathtt{t}\rangle\}$	–

Therefore, $Answer_{HF}(\mathcal{P}, \{q\}, H_2)$ returns $\Delta_q = \{\langle a, \mathtt{f}\rangle, \langle b, \mathtt{t}\rangle\}$ as expected.

It can then be shown that:

Proposition 4. *There is a limit ordinal λ such that after $|\lambda|$ steps $Answer_{HF}(\mathcal{P}, Q, H)$ returns the set of all correct answers of $\mathcal{P}$ with respect to the predicates in Q and the H-founded semantics.*

5 Conclusions

We have presented a simple, general, yet effective top-down algorithm to retrieve *all* correct answers to queries for normal logic programs under the AWA and, thus, under the CWA and OWA. To the best of our knowledge, this is the first time the problem of computing all answers has been addressed in this context, and under the CWA in particular, where arbitrary monotone functions in the body can manipulate truth values taken from a bilattice. We believe that its interest relies on its easiness for an effective implementation. Computing all answers is the first step towards top-k query answering, as it is developed in the context relational databases [6, 12, 21] and will be our primary topic of future research.

References

1. J. J. Alferes and L. M. Pereira. On logic program semantics with two kinds of negation. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 574–588, Washington, USA, 1992. The MIT Press.
2. Ofer Arieli. Paraconsistent declarative semantics for extended logic programs. *Annals of Mathematics and Artificial Intelligence*, 36(4):381–417, 2002.
3. Francois Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs (extended abstract). In *Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems (PODS-86)*, pages 1–15, New York, NY, USA, 1986. ACM Press.
4. Nuel D. Belnap. A useful four-valued logic. In Gunnar Epstein and J. Michael Dunn, editors, *Modern uses of multiple-valued logic*, pages 5–37. Reidel, Dordrecht, NL, 1977.
5. H. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.
6. Kevin Chen-Chuan Chang and Seung won Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD-02)*, pages 346–357, New York, NY, USA, 2002. ACM Press.
7. Carlos Viegas Damásio and Luís Moniz Pereira. A survey of paraconsistent semantics for logic programs. In D. Gabbay and P. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, pages 241–320. Kluwer, 1998.
8. Carlos Viegas Damásio and Luís Moniz Pereira. Antitonic logic programs. In *Proceedings of the 6th European Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, number 2173 in Lecture Notes in Computer Science. Springer-Verlag, 2001.
9. Guozhu Dong, Leonid Libkin, and Limsoon Wong. Incremental recomputation in local languages. *Inf. Comput.*, 181(2):88–98, 2003.
10. Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Annals of Mathematics and Artificial Intelligence*, 14(2-4):187–223, 1995.
11. Jon Doyle and Drew McDermott. Nonmonotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.
12. Ronald Fagin. Combining fuzzy information: an overview. *SIGMOD Rec.*, 31(2):109–118, 2002.
13. M. C. Fitting. The family of stable models. *Journal of Logic Programming*, 17:197–225, 1993.
14. M. C. Fitting. Fixpoint semantics for logic programming - a survey. *Theoretical Computer Science*, 21(3):25–51, 2002.
15. Melvin Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.
16. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
17. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.
18. Matthew L. Ginsberg, editor. *Readings in nonmonotonic reasoning*. Morgan Kaufmann, Los Altos, CA, 1987.
19. Matthew L. Ginsberg. Multi-valued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

20. Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation*, 135(2):69–112, 1997.
21. Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. RankSQL: query algebra and optimization for relational top-k queries. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD-05)*, pages 131–142, New York, NY, USA, 2005. ACM Press.
22. Leonid Libkin and Limsoon Wong. On the power of incremental evaluation in SQL-like languages. *Lecture Notes in Computer Science*, 1949:17–??, 2000.
23. Yann Loyer and Umberto Straccia. Uncertainty and partial non-uniform assumptions in parametric deductive databases. In *Proc. of the 8th European Conference on Logics in Artificial Intelligence (JELIA-02)*, number 2424 in Lecture Notes in Computer Science, pages 271–282, Cosenza, Italy, 2002. Springer-Verlag.
24. Yann Loyer and Umberto Straccia. The well-founded semantics in normal logic programs with uncertainty. In *Proc. of the 6th International Symposium on Functional and Logic Programming (FLOPS-2002)*, number 2441 in Lecture Notes in Computer Science, pages 152–166, Aizu, Japan, 2002. Springer-Verlag.
25. Yann Loyer and Umberto Straccia. The approximate well-founded semantics for logic programs with uncertainty. In *28th International Symposium on Mathematical Foundations of Computer Science (MFCS-2003)*, number 2747 in Lecture Notes in Computer Science, pages 541–550, Bratislava, Slovak Republic, 2003. Springer-Verlag.
26. Yann Loyer and Umberto Straccia. Default knowledge in logic programs with uncertainty. In *Proc. of the 19th Int. Conf. on Logic Programming (ICLP-03)*, number 2916 in Lecture Notes in Computer Science, pages 466–480, Mumbai, India, 2003. Springer Verlag.
27. Yann Loyer and Umberto Straccia. Any-world assumptions in logic programming. *Theoretical Computer Science*, 342(2-3):351–381, 2005.
28. Yann Loyer and Umberto Straccia. Epistemic foundation of stable model semantics. *Journal of Theory and Practice of Logic Programming*, 6:355–393, 2006.
29. Victor W. Marek and Mirosław Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3):587–618, 1991.
30. John McCarthy. Circumscription - a form of nonmonotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.
31. John McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–116, 1986.
32. Drew McDermott. Nonmonotonic logic II. *journal of the ACM*, 22:33–57, 1982.
33. Robert C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.
34. Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
35. Umberto Straccia. Query answering in normal logic programs under uncertainty. In *8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05)*, number 3571 in Lecture Notes in Computer Science, pages 687–700, Barcelona, Spain, 2005. Springer Verlag.
36. Umberto Straccia. Query answering under the any-world assumption for normal logic programs. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation (KR-06)*, pages 329–339. AAAI Press, 2006.
37. J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1,2. Computer Science Press, Potomac, Maryland, 1989.
38. Allen van Gelder, Kenneth A. Ross, and John S. Schlimpf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
39. David S. Warren. Memoing for logic programs. *Commun. ACM*, 35(3):93–111, 1992.