Paraconsistent Declarative Semantics for Extended Logic Programs

Ofer Arieli *

Department of Computer Science, The Academic College of Tel-Aviv
Antokolsky 4, P.O.Box 16131, Tel-Aviv 61161, Israel.
E-mail: oarieli@mta.ac.il

We introduce a fixpoint semantics for logic programs with two kinds of negation: an explicit negation and a negation-by-failure. The programs may also be prioritized, that is, their clauses may be arranged in a partial order that reflects preferences among the corresponding rules. This yields a robust framework for representing knowledge in logic programs with a considerable expressive power. The declarative semantics for such programs is particularly suitable for reasoning with uncertainty, in the sense that it pinpoints the incomplete and inconsistent parts of the data, and regards the remaining information as classically consistent. As such, this semantics allows to draw conclusions in a non-trivial way, even in cases that the logic programs under consideration are not consistent. Finally, we show that this formalism may be regarded as a simple and flexible process for belief revision.

Keywords: Logic programming, Fixpoint semantics, Paraconsistency, Multi-valued logics.

AMS Subject classification: Primary 68N17, 68T37; Secondary 03B50, 03B53.

1. Introduction

Logic programming is a combination of logic as a representation language and the theory of (constructive) automated deduction. However, it has long been claimed that standard logic programs are neither sufficiently expressive for formalizing various informal considerations (such as making preferences among different assertions, exception handling, completion of partial knowledge in a "rational" way, etc.) nor they are capable of properly imitating common-sense reasoning. This was partly explained by the limited syntactical structure of such programs, which in particular does not support a proper representation of negative information.

Various formalisms have been considered in order to overcome this limitation

* This work was prepared while the author was visiting the Department of Computer Science, University of Leuven, Belgium.

(see, e.g., [1,25,27,31,37,38,42,48] and a survey in [10, Section 3]). Most of them do so by extending the expressive power of the programs under consideration (and, of-course, provide some appropriate semantics that captures the "intended" meaning of the extended logic programs that are obtained). Two syntactical modifications are usually considered in this context. First, negations may appear not only in the bodies of the rules, but also in their heads. Second, *two* different operators are used for representing two different types of negative information. One kind of negation, denoted here by $\neg$, corresponds to an "explicit" negative data. Its role, like that of negation in classical logic, is to represent counter-information. The other kind of negation, denoted here by `not`, may be associated with a more "implicit" way of representing negative data. It is usually used for expressing the fact that the corresponding assertion cannot be proved or verified on the basis of the available information. It is therefore usual to associate this connective with "negation-as-failure" (to prove the corresponding assertion). The different nature of the two kinds of negations is demonstrated by the following example:

Example 1.1. Consider a rule that expresses the fact that "If someone is innocent (s)he cannot be guilty". This rule may be represented by the following implication:

$$\neg\texttt{guilty}(x) \leftarrow \texttt{innocent}(x)$$

I.e., innocence must entail no guilt. On the other hand, a rule like the following one:

$$\texttt{innocent}(x) \leftarrow \texttt{not guilty}(x)$$

is somewhat less strict. It may be understood as stating that "someone is innocent as long as it has not been proven that (s)he is guilty".

It follows, then, that these two negation operators should be used in different contexts. This is further illustrated by the following example (borrowed from [27]):

Example 1.2. Consider a rule that states that "a school bus may cross railway tracks if there is no crossing train". This rule may be represented by the following implication:

$$\texttt{cross\_railway\_tracks} \leftarrow \neg\texttt{train\_is\_comming}$$

However, it should *not* be expressed as follows:

$$\texttt{cross\_railway\_tracks} \leftarrow \texttt{not train\_is\_comming}$$

The reason for this is that the condition in the latter clause holds in cases that there is no information available about a presence of a train. This is a weaker condition than that of the former clause, which is satisfied only if there is an *explicit* evidence that no train is approaching the railway tracks.

Clearly, extending logic programs with two kinds of negations and allowing the appearance of negative data in the rule heads, have far-reaching impacts on the way knowledge is represented and processed. For instance, query evaluation becomes more accurate since it is possible to distinguish between a query that fails because it *does not succeed*, and a query that fails in a stronger sense, that its *negation succeeds*. Moreover, these extensions of standard logic programs offer some new opportunities that were not available before. A particularly important one (on which we focus the attention in this paper) is the ability to *represent and reason with uncertain information*. More specifically, in the representation level this means the ability to express the following kinds of knowledge:

- *Inconsistent belief.* I.e., a representation of contradictory data *within the language* (unlike, e.g., positive logic programs, the syntax of which rules out any possibility of representing contradictions.[1])
- *Partial knowledge.* I.e., the ability to deal *directly* with incomplete information by *explicitly* pointing to cases in which the data (or the knowledge) is incomplete.
- *(Hierarchy of) exceptions.* I.e., the ability to disregard some piece of information in the presence of another. More generally, preferring certain rules over others. Such preferences may be represented either in the program language itself or in a "meta-language" (as an additional information, not necessarily represented in a clausal form, and sometimes not even specified by first-order formulae).

In what follows we shall see how all these different types of knowledge are represented in our framework. For the time being we remain on the intuitive level and just note that the strong negation $\neg$ will usually be useful for representing contradictory data, while the negation-by-failure operator `not` will be useful for representing incomplete data. In addition, we will also allow some additional information, expressed as a "meta-knowledge", which exhibits preferences of certain pieces of information over others.

A proper way of handling incomplete and inconsistent information means also an adequate formalism for *processing* (i.e., reasoning with) this kind of data. The two main requirements in this respect are the following:

[1] Furthermore, even some formalisms that do allow negations in the clause bodies and heads (e.g., [27,31,42]), treat atomic formulae and their negations as two different ways of representing atomic information, so practically a representation of inconsistent information is not fully supported in this case as well. We shall return to this issue in what follows.

- *Non-monotonicity.* I.e., the ability to modify the set of conclusions in the light of new data. This is an important property of any formalism that deals with partial information and applies default assumptions to it.
- *Paraconsistency* [15]. The semantics of inconsistent logic programs should not be trivial, that is, inconsistent information should *not* entail every conclusion.

The following examples demonstrate these properties:

Example 1.3. Consider the following logic program, where p and q are two atomic formulae, and $\mathtt{t}$ is a propositional constant that corresponds to the classical truth value that represents true assertions.

$$\mathcal{P} \;=\; \{q \leftarrow \mathtt{t},\;\; p \leftarrow \mathtt{t},\;\; \neg p \leftarrow \mathtt{not}\ \neg q\}$$

Intuitively, $\mathcal{P}$ may be understood such that both p and q are known to be true, and $\neg p$ is also true, provided that it cannot be shown that the negation of q holds. In this interpretation $\mathcal{P}$ clearly contains inconsistent information regarding p. However, a paraconsistent formalism should not attach to $\mathcal{P}$ a trivial fixpoint semantics, since some part of it ($\{q \leftarrow \mathtt{t}\}$ in our case) is not related whatsoever to any contradictory information in $\mathcal{P}$ (and therefore it should have a consistent interpretation).[2] Moreover, a plausible semantics for such programs should make a clear distinction between the "robust" part of the program (i.e., those clauses that are not based on any contradictory information) and the "spoiled" one (i.e., rules that are defined in terms of inconsistent data).

Suppose now that a new datum arrives, and it indicates that if p holds then $\neg q$ must hold as well. The new program is therefore the following:

$$\mathcal{P}' \;=\; \mathcal{P} \;\cup\; \{\neg q \leftarrow p\}$$

Now, the information regarding p becomes consistent (as the condition for concluding $\neg p$ does not hold anymore), while the data regarding q is now inconsistent. A non-monotonic formalism should adapt itself to the new situation. In particular, while the query $\neg p$ should succeed where $\mathcal{P}$ is the underlying program, it should fail w.r.t. $\mathcal{P}'$.

Example 1.4. A robust formalism for reasoning with uncertainty should also be able to handle incomplete information in a plausible way. This is demonstrated by the following example:

$$\mathcal{P} \;=\; \{q \leftarrow \mathtt{t},\;\; p \leftarrow \mathtt{not}\ p\}$$

This time $\mathcal{P}$ contains incomplete information regarding p. Unlike some formalisms that do not provide any model for this program (e.g., the stable model semantics [26]), we claim that a proper semantics for $\mathcal{P}$ should distinguish between the

[2] Note that since the data regarding p is inconsistent, the other two clauses contain "unreliable" information, and thus may have inconsistent interpretations.

meaningful data in $\mathcal{P}$ ($\{q \leftarrow \mathtt{t}\}$), and the meaningless data ($\{p \leftarrow \mathtt{not}\ p\}$). Note that the well-founded semantics [47] does provide a plausible solution to this case. In what follows (Section 3.2) we shall use this property of the well-founded semantics for defining our way of handling incomplete information.

Many of the well-known formalisms for giving semantics to extended logic programs (e.g., [27,37]) reduce to triviality in the presence of contradictory data, and so they are *not* paraconsistent. Our approach, on the other hand, accepts contradictory data and tries to cope with it. As classical logic is neither non-monotonic nor paraconsistent, this approach must be non-classical in nature.[3] In our case we use *multiple-valued semantics* in which there are particular truth values that correspond to different degrees of contradictions and partial information. The use of multiple-valued logics for giving semantics to logic programs is discussed and justified in Sections 2.2 and 4.2.

The rest of this paper is organized as follows: in the next section we represent our framework. In particular, we consider some semantical aspects such as showing that Belnap four-valued structure [11,12] is particularly suitable for representing the kind of information we intend to decode in logic programs. In Section 3 we introduce our fixpoint theory, first for logic programs without negation-as-failure, and then for the general case. In Section 4 we further generalize our formalism to cases in which the logic programs under consideration are prioritized, i.e., every clause has its own relative priority over the other clauses. For extending our semantics to the prioritized case we consider a generalization of the four-valued semantical structure to a larger family of multiple-valued algebraic structures, called *bilattices* [20,28,29]. The semantics that is obtained is then discussed and some of its properties are illustrated. In Section 5 we summarize the main properties of our formalism (with respect to other related fixpoint semantics), and conclude.

2. Preliminaries

2.1. Logic programs

In what follows p, q, r denote atomic formulae, $l, l_1, l_2, \ldots$ denote literals (i.e., atomic formulae that may be preceded by $\neg$), and $e, e_1, e_2, \ldots$ denote extended literals (i.e. literals that may be preceded by $\mathtt{not}$). The complement of a literal l is denoted by $\bar{l}$ (that is, if $l = p$ for some atom p then $\bar{l} = \neg p$, and if $l = \neg p$ then $\bar{l} = p$). As usual in the context of logic programming, we shall deal with formulae in a clausal form, as defined below:

[3] See, e.g., [5,6,11,12,32,34,39] for some non-classical methods for reasoning with partial or contradictory information.

Definition 2.1. Let $n \geq m \geq 0$.

- A *positive clause* is a formula of the form $p \leftarrow p_1, \ldots, p_n$ [4]
- A *standard clause* is a formula of the form $p \leftarrow p_1, \ldots, p_m, \texttt{not}\ p_{m+1}, \ldots, \texttt{not}\ p_n$
- A *normal clause* is a formula of the form $p \leftarrow l_1, \ldots, l_n$
- A *general clause* is a formula of the form $l \leftarrow l_1, \ldots, l_n$
- An *extended clause* is a formula of the form $l \leftarrow e_1, \ldots, e_n$

Given a clause $l \leftarrow e_1, e_2, \ldots, e_n$, we say that l is the cause *head*, and $e_1, \ldots, e_n$ is the clause *body* (sometimes abbreviated by *Body*). The clause head is also called the *conclusion* (of the clause), and each element in the clause body is called a *condition* (of the clause). The set of the (extended) literals that appear in *Body* is denoted by $\mathcal{L}(Body)$.

A (possibly infinite) set $\mathcal{P}$ of extended (respectively: positive, standard, normal, general) clauses is called an *extended* (respectively: *positive*, *standard*, *normal*, *general*) *logic program*.

2.2. Some semantical considerations

A. The underlying multi-valued structure

First, we should decide what underlying semantics is most suitable for our intended formalism. It is well-accepted that two-valued semantics is an appropriate semantical framework for positive logic programs. This is so since every positive logic program $\mathcal{P}$ has a unique least Herbrand model, which is identical to the least fixpoint of van-Emden and Kowalski's immediate consequence operator [46] of $\mathcal{P}$. It follows, therefore, that the "intended" semantics of positive logic programs can be captured within the two-valued setting.

Things are getting more complicated when negations may appear in the clause bodies. In such cases a least two-valued model does not always exist (Consider, e.g., $\mathcal{P} = \{p \leftarrow \texttt{not}\ p\}$), and there are cases in which several minimal two-valued models exist (For instance, $\mathcal{P} = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$ has two minimal Herbrand models. In one of them p is true and q is false, and in the other one q is true and p is false). One common way to overcome these problems is to consider a minimization w.r.t. a three-valued semantics: Fitting's operator [19,23], based on Kripke/Kleene 3-valued semantics [30] always yields a least fixpoint when applied to normal logic programs, and this is also the case with the three-valued well-founded semantics [47], applied to standard logic programs. Under some further assumptions on the syntactical structure of the logic programs under consideration, some other 2-valued and 3-valued fixpoint semantics are uniquely

[4] Such formulae are also called *definite* clauses.

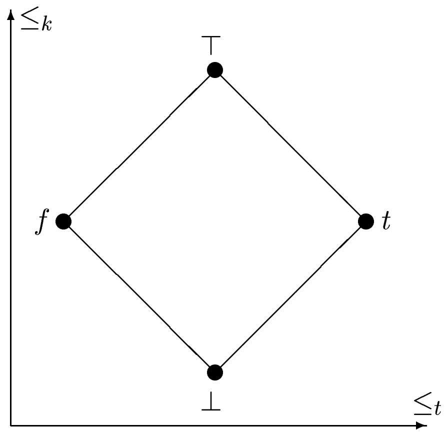

Figure 1. Belnap four-valued structure, $\mathcal{FOUR}$

determined. For instance, as shown in [40,41], every standard logic program that is weakly stratified [40] has a unique weakly perfect model [40], which coincides with its unique stable model [26] and its unique well-founded model [47].

When negations may also appear in the clause heads, the logic programs may be inconsistent, and so unless inconsistency reduces to triviality, neither 2-valued nor 3-valued models can capture the semantics of such programs anymore. Briefly, this is due to the fact that if $\mathcal{P}$ is some general (or extended) logic program, then an "appropriate" semantics for it should be able to distinguish among the following four different cases:

1. $p \leftarrow \mathtt{t} \in \mathcal{P}$, $\neg p \leftarrow \mathtt{t} \notin \mathcal{P}$
2. $p \leftarrow \mathtt{t} \notin \mathcal{P}$, $\neg p \leftarrow \mathtt{t} \in \mathcal{P}$
3. $p \leftarrow \mathtt{t} \in \mathcal{P}$, $\neg p \leftarrow \mathtt{t} \in \mathcal{P}$
4. $p \leftarrow \mathtt{t} \notin \mathcal{P}$, $\neg p \leftarrow \mathtt{t} \notin \mathcal{P}$

Assuming that neither p nor $\neg p$ appears in any clause head in $\mathcal{P}$ other than those mentioned above, one expects that in the former two cases p would have a classical value (t in the first case and f in the second case), and in the latter two cases *two other values* should be attached to p: one for denoting that the data regarding p is inconsistent (as in case 3 above), and the other for denoting that there is insufficient information regarding p (as in case 4 above).

It follows that in order to capture these four different cases on the semantical level, a semantical structure for general or extended logic programs should contain (at least) four different elements. Probably the best-known structure with this property is Belnap's $\mathcal{FOUR}$ (Figure 1).

Belnap's algebraic structure was introduced in [11,12] as a semantical tool for representing different states of a reasoner's knowledge (or belief). This structure consists of four truth values: the classical ones (t, f), a truth value $(\bot)$ that intuitively represents lack of information, and a truth value $(\top)$ that may intuitively be understood as representing contradictions. These four elements are simultaneously arranged in two partial orders. In one of them (denoted here by $\leq_t$), f is the minimal element, t is the maximal one, and $\bot, \top$ are two intermediate values that are incomparable. This partial order may be intuitively understood as representing differences in the amount of *truth* of each element. We denote by $\wedge$ and $\vee$ the meet and join operations w.r.t $\leq_t$ (hence, e.g., $\top \vee \bot = t$). In the other partial order (denoted here by $\leq_k$), $\bot$ is the minimal element, $\top$ is the maximal one, and t, f are two intermediate values. This partial order intuitively represents differences in the amount of *knowledge* (or information) that each element exhibits. We denote by $\otimes$ and $\oplus$ the meet and join operations w.r.t $\leq_k$ (hence, e.g., $t \oplus f = \top$). Another useful operator on $\mathcal{FOUR}$ is the negation (denoted here by $\neg$) that is order reversing w.r.t. $\leq_t$ and order preserving w.r.t. $\leq_k$, i.e., $\neg t = f$, $\neg f = t$, $\neg \top = \top$, and $\neg \bot = \bot$.

The various semantical notions are defined on $\mathcal{FOUR}$ as natural generalizations of similar classical ones: a *valuation* ν is a function that assigns a truth value in $\mathcal{FOUR}$ to each atomic formula. In what follows we shall sometimes write $\nu = \{p : x, q : y\}$ instead of $\nu(p) = x$, $\nu(q) = y$. Any valuation is extended to complex formulae in the obvious way. The set of the four-valued valuations is denoted by V^4.

$\mathcal{D} = \{t, \top\}$ is the set of the *designated* elements of $\mathcal{FOUR}$, i.e., the set of elements in $\mathcal{FOUR}$ that represent true assertions. Hence, we say that a valuation ν *satisfies* a formula ψ iff $\nu(\psi) \in \mathcal{D}$. Note that $\mathcal{D}$ is a prime filter in $\mathcal{FOUR}$ (w.r.t. both $\leq_t$ and $\leq_k$) that consists of the elements that are $\leq_k$-greater than or equal to t. This corresponds to Belnap's observation that the designated elements of $\mathcal{FOUR}$ should be those that are "at least true" (see [12, Page 36]).

A valuation that assigns a designated value to every clause in a logic program $\mathcal{P}$ is a *model* of $\mathcal{P}$.

Next we define a useful partial order on the elements of V^4 in terms of the partial order $\leq_k$ of $\mathcal{FOUR}$:

Definition 2.2.

a) A valuation $\nu_1 \in V^4$ is *k-smaller* than another valuation $\nu_2 \in V^4$ if for every atomic formula p, $\nu_1(p) \leq_k \nu_2(p)$.

b) A valuation $\nu \in V^4$ is a *k-minimal* element in a set $S \subseteq V^4$ if there is no other element in S that is k-smaller than ν. If there is a single k-minimal element in S, we shall sometimes call it the *k-least* element of S.

It is easy to see that Definition 2.2(a) induces a lattice structure on the set of the four-valued valuations: $\mathcal{V}^4 = (V^4, \leq_k)$. Another useful way of (pre-)ordering the elements in V^4 is the following:

Definition 2.3 [5,6].

a) A valuation $\nu_1 \in V^4$ is *more consistent* than another valuation $\nu_2 \in V^4$ if $\{p \mid \nu_1(p) = \top\} \subset \{p \mid \nu_2(p) = \top\}$.
b) A valuation $\nu \in V^4$ is a *maximally consistent* element in a set $S \subseteq V^4$ if there is no other element in S that is more consistent than ν. [5]

Clearly, the interesting cases of Definitions 2.2(b) and 2.3(b) are obtained when S is the set of the models of the logic program $\mathcal{P}$ under consideration. Two important sets of models are obtained in these cases: The *k-minimal models* of $\mathcal{P}$, and the *maximally consistent models* of $\mathcal{P}$. We shall reconsider these models in what follows.

B. The meaning of the implication connective

Let $\mathcal{P}$ be a general logic program. The connectives that appear in the bodies or the heads of the clauses in $\mathcal{P}$, i.e.: conjunctions (,)[6] and negations ($\neg$), should be regarded, respectively, as the greatest lower bound and the order-reversing operation w.r.t. the $\leq_t$-partial order of $\mathcal{FOUR}$.[7] This corresponds to the natural extensions to the multiple-valued case of the 2-valued definitions of these connectives. However, as has already been observed in [10,27], the implication connective $\leftarrow$ of the program's clauses should *not* be taken as the material implication $\hookleftarrow$, where $p \hookleftarrow q \equiv p \vee \neg q$. This is so since, for instance, the intuitive meaning of $\{\neg p \leftarrow \mathtt{t},\ p \leftarrow \neg q\}$ is different than the intuitive meaning of $\{\neg p \leftarrow \mathtt{t},\ q \leftarrow \neg p\}$,[8] thus a plausible semantics for $\mathcal{P}$ cannot be 'contrapositive' w.r.t. $\leftarrow$ and $\neg$. Moreover, in the multi-valued setting, the material implication $\hookleftarrow$ is not suitable for representing entailment anymore. This is mainly due to the following reasons:

1. $p \hookleftarrow p$ does not always hold in the four-valued setting, since excluded-middle is not a four-valued tautology (Note that $\nu(p \vee \neg p) = \bot$ when $\nu(p) = \bot$).

[5] In [5,6] a valuation with the same property is called a *most consistent* element of S. Here we have changed the terminology, so that it would not suggest uniqueness.
[6] Commas are used here also as a separator among clauses in the same program. This will not cause any ambiguity.
[7] We shall discuss the semantics of the negation as failure operator (`not`) in a later stage.
[8] Intuitively, in the former program there is no explicit information on the validity of q, and so q should not have a designated value, while in the latter program the condition in the rule that defines q is satisfied, and so this time q should be assigned t.

2. $\hookleftarrow$ does not have a deductive nature in $\mathcal{FOUR}$. For instance, the fact that every four-valued model of some conjunction $Body$ is also a model of a literal l does *not* imply that $l \hookleftarrow Body$ is true in every four-valued valuation (Consider, e.g., the case in which $Body = l$).

We therefore consider an alternative definition for the implication connective, according to which it does function as an entailment in the four-valued setting:

Definition 2.4 [5,8]. Let $x, y \in \mathcal{FOUR}$. Define:

$$x \leftarrow y = \begin{cases} x & \text{if } y \in \mathcal{D} \\ t & \text{otherwise} \end{cases}$$

Note that on $\{t, f\}$ the material implication and the new implication are identical, and both of them are generalizations of the classical implication. However, unlike the material implication, the implication connective defined in 2.4 does preserve both properties of entailment that were mentioned above.

In what follows we therefore use the implication connective of Definition 2.4 for representing the entailment of the program's clauses. Note that the semantics of this implication is in accordance with the following standard way of understanding entailment in logic programs:

Proposition 2.5. For every valuation $\nu \in V^4$ we have that $\nu(l \leftarrow Body) \in \mathcal{D}$ iff either $\nu(l) \in \mathcal{D}$ or $\nu(Body) \notin \mathcal{D}$.

Proof. Immediately follows from Definition 2.4. □

2.3. The language and its extension to the first-order case

The language of the logic programs considered here is based on the implication connective $\leftarrow$, the meaning of which was discussed in the previous section, conjunction that correspond to the $\leq_t$-join operator in $\mathcal{FOUR}$, two negation operators $\neg$ and `not`, and four propositional constants `t`, `f`, `c`, `u`, that are respectively associated with the elements t, f, $\top$, $\bot$ in $\mathcal{FOUR}$. We therefore remain, basically, on the propositional level. However, as first-order clauses are considered as universally quantified, first order logic programs may be handled within our framework as well. We do so by considering their ground instances; every non-grounded clause is viewed as representing the corresponding set of ground clauses, formed by substituting every variable that appear in this clause with every possible element of the corresponding domain. Formally, let ρ be a ground substitution from the variables of every clause $\mathcal{C}$ in $\mathcal{P}$ to the individuals of the set of the closed terms H of $\mathcal{P}$. Then we shall consider programs of the form

$$\mathcal{P}^H = \{\rho(\mathcal{C}) \mid \mathcal{C} \in \mathcal{P},\ \rho : var(\mathcal{C}) \to H\}.$$

In what follows we shall abbreviate $\mathcal{P}$ for $\mathcal{P}^H$.

3. Paraconsistent declarative semantics for logic programs

We are now ready to introduce our fixpoint semantics for logic programs. First, we treat general logic programs (i.e., programs without negation-as-failure), and then we consider extended logic programs.

3.1. Semantics for general logic programs

Definition 3.1. Given a general logic program $\mathcal{P}$, define for every $i \geq 1$ and every literal l the following valuations:

$\nu_0^{\mathcal{P}}(l) = \bot$.

$$\mathsf{val}_i^{\mathcal{P}}(l) = \begin{cases} t & \text{if there is a clause } l \leftarrow Body \in \mathcal{P} \text{ s.t. } \nu_{i-1}^{\mathcal{P}}(Body) \in \mathcal{D}, \text{ [9]} \\ \bot & \text{otherwise.} \end{cases}$$

$\nu_i^{\mathcal{P}}(l) = \mathsf{val}_i^{\mathcal{P}}(l) \oplus \neg\mathsf{val}_i^{\mathcal{P}}(\bar{l})$. [10]

For a limit ordinal λ we define:

$$\mathsf{val}_\lambda^{\mathcal{P}}(l) = max_{\leq_k}\{\mathsf{val}_\alpha^{\mathcal{P}}(l) \mid \alpha < \lambda\}, \qquad \nu_\lambda^{\mathcal{P}}(l) = \mathsf{val}_\lambda^{\mathcal{P}}(l) \oplus \neg\mathsf{val}_\lambda^{\mathcal{P}}(\bar{l}).$$

Also, for every propositional constant $\mathtt{x} \in \{\mathtt{t}, \mathtt{f}, \mathtt{c}, \mathtt{u}\}$ that is associated with an element $x \in \{t, f, \top, \bot\}$ in $\mathcal{FOUR}$, we define $\nu_i^{\mathcal{P}}(\mathtt{x}) = \mathsf{val}_i^{\mathcal{P}}(\mathtt{x}) = x$ $(i = 0, 1, \ldots)$.

The basic operator used here for handling contradictory information is the $\leq_k$-join, $\oplus$. As it is noted in [21,24], this operator may be associated with a "gullibility" ("accept all") function that computes the combined knowledge of its arguments. The choice of this operator may be intuitively justified, then, by the need to do the following: (a) record cases in which there is an evidence for a specific assertion and for the complementary assertion as the same time, and (b) pinpoint the contradictory knowledge (or belief).

Note also that for every i, $\nu_i^{\mathcal{P}}$ behaves as expected w.r.t. negation: since for every $x, y \in \mathcal{FOUR}$, $\neg(x \oplus y) = \neg x \oplus \neg y$, we have that

$$\neg\nu_i^{\mathcal{P}}(l) = \neg(\mathsf{val}_i^{\mathcal{P}}(l) \oplus \neg\mathsf{val}_i^{\mathcal{P}}(\bar{l})) = \neg\mathsf{val}_i^{\mathcal{P}}(l) \oplus \mathsf{val}_i^{\mathcal{P}}(\bar{l}) = \nu_i^{\mathcal{P}}(\bar{l}).$$

[9] $\nu_j^{\mathcal{P}}$ is defined on conjunctive formulae in the usual way: $\nu_j^{\mathcal{P}}(Body) = \bigwedge_{l_i \in \mathcal{L}(Body)} \nu_j^{\mathcal{P}}(l_i)$. Thus, $\nu_j^{\mathcal{P}}(Body) \in \mathcal{D}$ iff $\forall l_i \in \mathcal{L}(Body)$ $\nu_j^{\mathcal{P}}(l_i) \in \mathcal{D}$.

[10] Recall that $\bar{l}$ is the complement of l, and $\oplus$ is the $\leq_k$-join operation in $\mathcal{FOUR}$.

Proposition 3.2. The sequence $\nu_0^{\mathcal{P}}, \nu_1^{\mathcal{P}}, \nu_2^{\mathcal{P}}, \ldots$, defined in 3.1 for a general logic program $\mathcal{P}$, is $\leq_k$-monotonic in $\mathcal{V}^4$.

Proof. Since the set $\mathcal{D}$ of the designated values is upwards-closed w.r.t. $\leq_k$, it easily follows from Definition 3.1 that for a given general logic program $\mathcal{P}$, the sequences $\{\mathsf{val}_i^{\mathcal{P}}\}$ and $\{\nu_i^{\mathcal{P}}\}$ are both $\leq_k$-monotonic in i. □

By Knaster-Tarski theorem [45], it follows from Proposition 3.2 that the sequence $\{\nu_i^{\mathcal{P}}\}$ has a $\leq_k$-least fixpoint. Denote this fixpoint by $\nu^{\mathcal{P}}$. An induced consequence relation $\mid\!\sim_\nu$ may now be defined as follows: $\mathcal{P} \mid\!\sim_\nu \psi$ iff $\nu^{\mathcal{P}}(\psi) \in \mathcal{D}$ (Thus, a query ψ *follows* from a logic program $\mathcal{P}$ if $\nu^{\mathcal{P}}(\psi)$ is designated).

Proposition 3.3. Let $\mathcal{P}$ be a general logic program. Then $\nu^{\mathcal{P}}$ is the k-minimal four-valued model of $\mathcal{P}$. Moreover, it is at least as consistent as any other model of $\mathcal{P}$,[11] and the consequence relation induced by it is paraconsistent.

Proof. See Appendix A. □

Corollary 3.4. Let $\mathcal{P}$ be a general logic program. Then $\nu^{\mathcal{P}}$ is the k-least model and a maximally consistent model of $\mathcal{P}$.

Proof. Immediately follows from Proposition 3.3 and its proof. □

Being the k-least model of $\mathcal{P}$, $\nu^{\mathcal{P}}$ minimizes the amount of knowledge that is pre-supposed, i.e. it does not assume anything that is not *really* known. This property is further discussed in Remark 3.6 below. Moreover, the last corollary also indicates that $\nu^{\mathcal{P}}$ minimizes the amount of inconsistent belief in the set of clauses.[12] This is in accordance with the intuition that while one has to deal with conflicts in a nontrivial way, contradictory data corresponds to inadequate information about the real world, and therefore it should be minimized (see also [6] for a discussion on maximally consistent models of general theories).

Note that the last corollary does *not* imply that $\nu^{\mathcal{P}}$ is the only maximally consistent model of $\mathcal{P}$. Indeed, consider for instance the following program:

$$\mathcal{P} = \{p \leftarrow \mathtt{t}, \quad \neg p \leftarrow \mathtt{t}, \quad p \leftarrow q\}$$

Beside $\nu^{\mathcal{P}} = \{p\!:\!\top,\ q\!:\!\bot\}$, $\mathcal{P}$ has two other maximally consistent models, namely $\{p\!:\!\top,\ q\!:\!t\}$ and $\{p\!:\!\top,\ q\!:\!f\}$. However, by Corollary 3.4 we have the following result that shows that $\nu^{\mathcal{P}}$ provides the most compact way of representing the most consistent knowledge:

[11] In the sense that the set of the atoms that are assigned $\top$ by $\nu^{\mathcal{P}}$ does not properly contain any similar set of another model of $\mathcal{P}$.

[12] Recall Definition 2.3.

Corollary 3.5. $\nu^{\mathcal{P}}$ is the k-least model among the maximally consistent model of $\mathcal{P}$.

Remark 3.6. Syntactically, normal logic programs are special cases of general logic programs. Still, there are *semantical differences* between a set of rules viewed as a normal program, and the same set of rules viewed as a general (or extended) program. For instance, most of the semantics for normal logic programs assign f to any atom that does not appear in (any clause head in) the program. However, in general or extended logic programs this can be inferred from general rules that are stated in the program itself (e.g., by adding rules for closed word assumption; see Example 3.14a below), and so the absence of an atom p from a general logic program indicates that p does not hold. Hence, if nothing is stated regarding $\neg p$ as well, in the corresponding semantics p should be *unknown*.[13]

For another example on the semantical differences, consider the following (general) program:

$$\mathcal{P} = \{p \leftarrow \neg q,\ q \leftarrow \mathtt{t}\}.$$

Treated as a normal program, some of the 2-valued and the 3-valued fixpoint semantics for $\mathcal{P}$ assign t to q and f to p. According to those semantics the head of a clause program is associated with its corresponding clause bodies, and therefore the truth value attached to a clause head should be the same as the (least upper-bound of) the value(s) of its clause body(ies). This, however, is not the case in our semantics, which assigns t to q and $\bot$ to p. This is justified by the fact that $\mathcal{P}$ is now considered as a *general* logic program, and so had one wanted to identify the information regarding p with that of its clause body, (s)he should have added to $\mathcal{P}$ also the converse implication (in trms of p), i.e., $\neg p \leftarrow q$.[14] In the absence of such clause our four-valued semantics correctly indicates that one should *not* conclude here that p is false! [15]

Once again, this example demonstrates our slogan: our semantics always assumes *as minimal knowledge as reasonably possible.* Thus, if one wishes to introduce more assumptions (e.g., to apply Clark's completion [14,33] to specific predicates), (s)he just has to add the appropriate clauses. In the absence of such information the program may have other meaning than those that are imposed by some of the 2-valued or 3-valued semantics for normal logic programs. Moreover, since it is not always possible to distinguish in standard or normal logic programs among the various possibilities offered by general (or extended) logic programs, such refinements sometimes cannot even be captured by the 2-valued or the 3-valued semantics for standard/normal logic programs!

[13] See also a remark on this matter in [27, Pages 591–592].

[14] Indeed, as shown in Example B.3 in Appendix B, our 4-valued fixpoint semantics of $\mathcal{P} \cup \{\neg p \leftarrow q\}$ assigns t to q and f to p.

[15] This is so since the condition of the rule that defines p does not hold, and so one cannot infer anything meaningful about p.

In the rest of this section we show that in certain cases it is possible to restore from our 4-valued fixpoint formalism some other 2-valued or 3-valued fixpoint formalisms, if so one wishes.

Proposition 3.7. Let $\mathcal{P}$ be a positive logic program, and let $\mathcal{P}$' be the positive program obtained from $\mathcal{P}$ by replacing every implication connective by a material implication. Denote by $\nu^{\mathcal{P}}_{f/\bot}$ the valuation that is obtained from $\nu^{\mathcal{P}}$ by changing the $\bot$-assignments to f-assignments (I.e., for every atom p, if $\nu^{\mathcal{P}}(p) = \bot$ then $\nu^{\mathcal{P}}_{f/\bot}(p) = f$). Then:

1. $\mathcal{P}$ and $\mathcal{P}'$ have the same classical models (and thus the same least Herbrand model), and
2. $\nu^{\mathcal{P}}_{f/\bot}$ is the (unique) 2-valued minimal Herbrand model of $\mathcal{P}$ and $\mathcal{P}$'.

Proof. See Appendix A. □

The process of restoring Fitting's $\leq_k$-minimal 3-valued Kripke/Kleene semantics for normal logic programs [19] is somewhat more complicated than the process of restoring the 2-valued semantics of positive logic programs, described in Proposition 3.7 above. Note, however, that if $\mathcal{P}$ is a normal logic program, the following properties hold:

1. For every atom p, $\nu^{\mathcal{P}}(p) \in \{t, \bot\}$.

   Proof. For every i and p, $\mathsf{val}_i(p) \in \{t, \bot\}$, and since $\mathcal{P}$ is a normal program, $\mathsf{val}_i(\neg p) = \bot$. Thus, for every i, $\nu^{\mathcal{P}}_i(p) = \mathsf{val}_i(p) \in \{t, \bot\}$, and so $\nu^{\mathcal{P}}(p) \in \{t, \bot\}$ as well.

2. $\nu^{\mathcal{P}} \leq_k \Psi^{\mathcal{P}}$, where $\Psi^{\mathcal{P}}$ is the $\leq_k$-least fixpoint of Fitting's operator for $\mathcal{P}$.

   Proof. By the fact that $\Psi^{\mathcal{P}}$ is a model of $\mathcal{P}$ and $\nu^{\mathcal{P}}$ is the $\leq_k$-least model of $\mathcal{P}$.[16]

It follows, therefore, that $\nu^{\mathcal{P}}$ can be viewed as an "approximation" of $\Psi^{\mathcal{P}}$: if $\nu^{\mathcal{P}}$ assigns t to some atomic formulae, then so is $\Psi^{\mathcal{P}}$, and if $\nu^{\mathcal{P}}$ assigns $\bot$ to some atom, then $\Psi^{\mathcal{P}}$ assigns either $\bot$ or f to this atom. Thus, in order to restore from $\nu^{\mathcal{P}}$ Fitting's 3-valued fixpoint semantics for $\mathcal{P}$, it is possible to apply Fitting's operator on $\nu^{\mathcal{P}}$ (rather than to start the iterations with a valuation that assigns $\bot$ to every atom), and then to proceed until a fixpoint is reached. This fixpoint coincides with Fitting 3-valued Kripke/Kleene semantics for $\mathcal{P}$.

An alternative way of computing Fitting's 3-valued semantics from our 4-valued semantics is described in Appendix B.

[16] Note that according to our semantics, the set of models of $\mathcal{P}$ contains the set of models w.r.t. Fitting's semantics. Thus (as illustrated in Remark 3.6), although $\Psi^{\mathcal{P}}$ is the $\leq_k$-least model in Fitting's semantics, it is not necessarily the $\leq_k$-least one in our case.

3.2. Semantics for extended logic programs

In this section we extend the fixpoint semantics for general logic programs, considered in the previous section, to extended logic programs. So now, in addition to the explicit negation ($\neg$), the negation-as-failure operator (`not`) may also appear in the clauses bodies.

One way of understanding `not` in the four-valued setting is the following: if we don't know anything about p, i.e. we cannot prove either p or $\neg p$, then we cannot say anything about `not` p as well. Otherwise, if p has a designated value in the intended semantics (i.e., p is provable), then `not` p does not hold, and if p does not have a designated value (i.e., it is not provable), then `not` p holds. It follows, then, that `not` $t = f$, `not` $\top = f$, `not` $f = t$, and `not` $\bot = \bot$.[17]

This interpretation of `not` is a natural generalization to the four-valued case of the way `not` is interpreted by the well-founded semantics [47]. We thus give semantics to logic programs in which `not` may appear in the clause bodies by using a transformation, which is similar to that of the well-founded semantics, for reducing extended logic programs to general logic programs. Then we use the machinery of the previous section for giving semantics to the general logic programs that are obtained. Below we formalize this idea.

Definition 3.8. Let ν be a four-valued valuation. The set S_ν of literals that is *associated* with ν is the smallest set that satisfies the following conditions: [18]

$$\text{if } \nu(l)=t \text{ then } l\in S_\nu, \quad \text{if } \nu(l)=f \text{ then } \bar{l}\in S_\nu, \quad \text{if } \nu(l)=\top \text{ then } \{l,\bar{l}\}\subseteq S_\nu.$$

Obviously, one can define the converse transformation as well. A four-valued valuation ν_S may be constructed from a set S of literals as follows: for every atom p,

$$\nu_S(p) = \begin{cases} t & \text{if } p\in S \text{ and } \neg p\notin S \\ f & \text{if } p\notin S \text{ and } \neg p\in S \\ \top & \text{if } p\in S \text{ and } \neg p\in S \\ \bot & \text{if } p\notin S \text{ and } \neg p\notin S \end{cases}$$

[17] It is interesting to note that in this interpretation, `not` may be represented as a conjunction of two other negation operators: `not` $p = \neg p \wedge \sim p$, where $\sim p$ is an abbreviation of $p \to f$, i.e., $\sim p = f$ if $p\in\mathcal{D}$, and $\sim p = t$ if $p\notin\mathcal{D}$.

[18] Such sets are sometimes called *answer sets* (for ν). We shall not use this terminology here, since it is usual to require that if an answer set contains a pair of complementary literals, then it should contain *every* literal. Since our formalism does not reduce to triviality in the presence of inconsistent information, this requirement should obviously not hold here.

Definition 3.9. Let $\mathcal{P}$ be an extended program and let S be a set of literals. The *reduction* of $\mathcal{P}$ w.r.t. S is the general logic program $\mathcal{P}{\downarrow}S$, obtained from $\mathcal{P}$ as follows:

1. Each clause that has a condition of the form $\mathtt{not}\ l$ for some $l \in S$, is deleted from $\mathcal{P}$.
2. Every occurrence of $\mathtt{not}\ l$, where $\bar{l} \in S$, is eliminated from the (bodies of the) remaining clauses.[19]
3. Every occurrence of $\mathtt{not}\ l$ in the (bodies of the) remaining clauses is replaced by the propositional constant $\mathtt{u}$.[20]

Now we are ready to define our fixpoint semantics for extended logic programs. Recall that $\nu^{\mathcal{P}}$ denotes the fixpoint semantics for a general logic program $\mathcal{P}$.

Definition 3.10. A valuation $\mu \in V^4$ is a *plausible model* of an extended logic program $\mathcal{P}$, if it coincides with the fixpoint semantics of the general logic program obtained by reducing $\mathcal{P}$ w.r.t. the set that is associated with μ. In other words, μ is a plausible model of $\mathcal{P}$ iff

$$\mu = \nu^{\mathcal{P}\downarrow S_\mu}.$$

Remark 3.11. If the only negation operator that appears in $\mathcal{P}$ is $\neg$, then $\mathcal{P}$ is a general logic program, and so its unique plausible model is $\nu^{\mathcal{P}}$. It follows, in particular, that the notion of plausible models of extended logic programs is a generalization of the definition of fixpoint semantics for general logic programs.

Proposition 3.12. A plausible model of $\mathcal{P}$ is indeed a model of $\mathcal{P}$.

Proof. Let $\mu = \nu^{\mathcal{P}\downarrow S_\mu}$ be a plausible model of $\mathcal{P}$. Since $\mathcal{P}{\downarrow}S_\mu$ is a general logic program, by Proposition 3.3 μ is a model of $\mathcal{P}{\downarrow}S_\mu$. But by the semantics of $\mathtt{not}$ and the definition of reduction it is clear that if M is a model of $\mathcal{P}{\downarrow}S_M$ then M is also a model of $\mathcal{P}$. One concludes, then, that μ is a model of $\mathcal{P}$. $\square$

As it is shown in Example 3.14 below, an extended logic program may have more than one plausible model, and so one may use different preference criteria for choosing the best models among the plausible ones. In the case of general logic programs we have chosen $\leq_k$-minimization as the criterion for preferring the "best" model among the fixpoint valuations. This was justified by the fact that general logic programs may contain contradictory data, and so we want to

[19] If a clause body becomes empty by this transformation, we treat this body as if it consists of the propositional constant $\mathtt{t}$.

[20] Note that for this l, necessarily $l \notin S$ and $\bar{l} \notin S$.

minimize the redundant information as much as possible. In the present case we rather use the opposite methodology: since the negation-as-failure operator is associated with incomplete information, we are dealing here with a lack of data, so this time we should try to restrict the effect of the negation-as-failure operator only to those cases in which indeed there is not enough data available. It follows, therefore, that now we should seek for a *maximal knowledge* (among the plausible models). [21]

Definition 3.13. μ is an *adequate model* of $\mathcal{P}$ if it is a $\leq_k$-maximal element among the plausible models of $\mathcal{P}$. [22]

Example 3.14. Below we consider our semantics for some inconsistent and/or incomplete logic programs.

1. $\mathcal{P} = \{\neg p \leftarrow \texttt{not}\ p\}$.
   Intuitively, $\mathcal{P}$ represents a closed word assumption (CWA, [43]) regarding p: in the absence of any evidence for p, assume that $\neg p$ holds. $\mathcal{P}$ has two plausible models $\mu_1 = \{p : \bot\}$ and $\mu_2 = \{p : f\}$. But $\mu_2 >_k \mu_1$, and so μ_2 is the adequate model of $\mathcal{P}$.

2. (Example 1.4, revisited) $\mathcal{P} = \{p \leftarrow \texttt{not}\ p,\ q \leftarrow \texttt{t}\}$.
   The adequate model of $\mathcal{P}$ here is $\{p : \bot,\ q : t\}$. This indeed seems to be the only reasonable interpretation here (see the discussion in Example 1.4), and it coincides with the well-founded model [47] (for standard logic programs) of $\mathcal{P}$. Two-valued semantics, such as Gelfond-Lifschitz stable model semantics [26], do not provide any model for $\mathcal{P}$.

3. (Example 1.3, revisited) $\mathcal{P} = \{q \leftarrow \texttt{t},\ p \leftarrow \texttt{t},\ \neg p \leftarrow \texttt{not}\ \neg q\}$.
   The adequate model here is $\{p : \top,\ q : t\}$. It reflects our expectation that since $\neg q$ does not follow from $\mathcal{P}$, the knowledge about p is contradictory. Note that according to the semantics given in [27,37], $\mathcal{P}$ does not have any model, since it contains contradictory information.

We postpone to a later stage (Section 5) some further discussions on adequate models and other formalisms for giving semantics to extended logic programs. First we complete the presentation of our formalism also for the prioritized case.

[21] Informally, we use here a "min/max strategy": knowledge minimization on the contradictory components of the program, and knowledge maximization on its incomplete components.

[22] I.e., μ is a plausible model of $\mathcal{P}$, and there is no other plausible model of $\mathcal{P}$ that is strictly $\leq_k$-bigger than μ. Note also that by Proposition 3.12, μ is indeed a model of $\mathcal{P}$.

4. Prioritized logic programs

4.1. Motivation

As Proposition 3.3 and Corollary 3.4 imply, the four-valued fixpoint semantics considered in the previous section has several appealing properties. However, there might be cases in which one would like to refine the inference mechanism induced by this fixpoint. To see this, consider the following example.

Example 4.1 (Tweety dilemma). Consider the following well-known program:

$$\mathcal{P} = \left\{ \begin{array}{ll} \mathtt{fly}(x) \leftarrow \mathtt{bird}(x) & \neg\mathtt{reptile}(x) \leftarrow \mathtt{bird}(x) \\ \mathtt{bird}(x) \leftarrow \mathtt{penguin}(x) & \neg\mathtt{fly}(x) \leftarrow \mathtt{penguin}(x) \\ \mathtt{bird}(\mathtt{Tweety}) \leftarrow \mathtt{t} & \mathtt{penguin}(\mathtt{Tweety}) \leftarrow \mathtt{t} \end{array} \right\}$$

The fixpoint semantics of $\mathcal{P}$ is the following:

$$\begin{array}{ll} \nu^{\mathcal{P}}(\mathtt{bird}(\mathtt{Tweety})) = t, & \nu^{\mathcal{P}}(\mathtt{penguin}(\mathtt{Tweety})) = t, \\ \nu^{\mathcal{P}}(\mathtt{reptile}(\mathtt{Tweety})) = f, & \nu^{\mathcal{P}}(\mathtt{has\_feathers}(\mathtt{Tweety})) = \bot, \\ \nu^{\mathcal{P}}(\mathtt{fly}(\mathtt{Tweety})) = \top. & \end{array}$$

While the truth values that are assigned to $\mathtt{bird}(\mathtt{Tweety})$, $\mathtt{penguin}(\mathtt{Tweety})$, $\mathtt{reptile}(\mathtt{Tweety})$ and $\mathtt{has\_feathers}(\mathtt{Tweety})$ correspond to the intuitive expectations in this case,[23] one usually tends to conclude from $\mathcal{P}$ that Tweety *cannot* fly. However, this conclusion is based on some *further, implicit knowledge, that is not represented in the program*. Such knowledge is, e.g., the fact that the rule "birds can fly" has exceptions that should "override" the default rule. Another kind of knowledge that is not encoded in this program is the fact that the information that Tweety is a penguin is more specific than the statement that it is a bird, therefore the former data should have a higher priority than the latter one, in case of "collisions" between the two.

The above inaccurate conclusion about the flying ability of Tweety is therefore an outcome of the limited way that knowledge is represented here, rather than a consequence of a shortcoming of the reasoning process. A general method to improve knowledge representation is to provide a way to prefer a certain data over the other. In example 4.1, for instance, such mechanism will allow us to indicate that the clause that states that "penguins cannot fly" should get a precedence over the one that states that "birds can fly".

Several methodologies for making such preferences have been proposed in the literature. In [31], for instance, rules with negative conclusions are viewed as

[23] The assignment of $\bot$ to $\mathtt{has\_feathers}(\mathtt{Tweety})$ is justified by the fact that nothing is mentioned in the program about the property "has_feathers".

representing exceptions of rules with the positive counterparts as their conclusions. As such, the former rules are given higher priorities over the latter rules. Thus, for instance, in the semantics of [31] for $\mathcal{P}=\{q \leftarrow \mathtt{t},\ p \leftarrow \mathtt{t},\ \neg p \leftarrow \mathtt{not}\ \neg q\}$, p is *false* (cf. Example 3.14, Item 3).

According to the formalism proposed by Pereira et al. in [38], preferences of different rules are encoded within the language itself. According to this approach the conflict regarding the flying ability of Tweety in Example 4.1 is resolved by stating that birds can fly unless they are "abnormal birds". Thus, in the program of that example, $\mathtt{fly}(x) \leftarrow \mathtt{bird}(x)$ should be replaced by the following two rules:

$$\mathtt{fly}(x) \leftarrow \mathtt{bird}(x), \mathtt{not}\ \mathtt{abnormal\_bird}(x)$$

$$\mathtt{abnormal\_bird}(x) \leftarrow \mathtt{bird}(x), \neg\mathtt{fly}(x)$$

In the same paper, Pereira et al. also propose to associate a different 'label' to each program rule, and to insert this label as another condition to the body of the rule. This enables an easy way to represent a hierarchy of rules in the language itself. For instance, the fact that under the conditions specified in *Body* one should apply a rule labeled by l_1 instead of a rule labeled by l_2, is encoded by the following special preference rule: $\neg l_2 \leftarrow Body, l_1$.

The formalisms mentioned above, although being elegant ones, have their own limitations. First, they rule out any representation of contradictions in the reasoner's belief. Such contradictions do occur in practical problems, and it may be useful to use a methodology to trace them and to represent their effect. Second, as already observed in [38], because of the inherent asymmetry in the representation of the hierarchy of exceptions, each time that exceptions to exceptions are specified, some rules in the program should be changed. Third, the rule labeling and the need to maintain the preferences and the exceptions with special additional rules, require a lot of overhead; in practical cases this might yield awkward programs, in which it would be difficult to grasp the essence from the whole data.

Here we consider another way of making preferences among programs clauses, which has a more quantitative nature. The idea is to attach, in a meta-language, different priorities to different clauses. We do so by assigning to every clause a 'confidence factor' that reflects its relative priority over the other clauses. For this, we consider algebraic structures that generalize Belnap's four-valued structure. In particular, we extend the four-valued semantics to a more general semantics that is based on arbitrarily many truth values. In the next section we review the basic notions that are related to these structures, and in Section 4.3 we use them for giving semantics to prioritized (extended) logic programs.

4.2. Bilattices and logical bilattices – An overview

Definition 4.2 [28,29]. A *bilattice* is a structure $\mathcal{B} = (B, \leq_t, \leq_k, \neg)$ such that B is a nonempty set containing at least two elements, $(B, \leq_t)$ and $(B, \leq_k)$ are complete lattices, and $\neg$ is a unary operation on B that has the following properties: (i) if $x \leq_t y$ then $\neg x \geq_t \neg y$, (ii) if $x \leq_k y$ then $\neg x \leq_k \neg y$, (iii) $\neg\neg x = x$.

The original motivation of Ginsberg [29] for using bilattices was to provide a uniform approach for a diversity of applications in artificial intelligence. In particular, he treated first order theories and their consequences, truth maintenance systems, and formalisms for default reasoning. The algebraic structure of bilattices has been further investigated by Fitting [21,24] and Avron [9]. In a series of paper Fitting has also shown that bilattices are very useful tools for providing semantics for logic programs. He proposed an extension of Smullyan's tableaux-style proof method to bilattice-valued programs, and showed that this method is sound and complete with respect to a natural generalization of van-Emden and Kowalski's operator [20,22]. Fitting also introduced a multi-valued fixpoint operator for providing bilattice-based stable models and well-founded semantics for logic programs [23]. A well-founded semantics for logic programs that is based on a specific bilattice (denote here by *NINE*, see Figure 2 below) is also considered in [16]. Bilattices have also been found useful for model-based diagnostics [29], computational linguistics [36], reasoning with inconsistent knowledge-bases [5,44], and processing of distributed knowledge [35].

As in the four-valued case, we shall continue to denote by $\wedge, \vee, \neg$ the meet, join, and the involution operations w.r.t. $\leq_t$, and by $\otimes, \oplus$ the meet and the join operations w.r.t. $\leq_k$. The $\leq_t$-maximal (respectively, $\leq_t$-minimal) element is denoted by t (respectively, f), and the $\leq_k$-maximal (respectively, $\leq_k$-minimal) element is denoted by $\top$ (respectively, $\bot$).

Definition 4.3 [5]. Let $\mathcal{B} = (B, \leq_t, \leq_k, \neg)$ be a bilattice.

a) A *bifilter* of $\mathcal{B}$ is a nonempty proper subset $\mathcal{D} \subset B$, such that:
   (i) $x \wedge y \in \mathcal{D}$ iff $x \in \mathcal{D}$ and $y \in \mathcal{D}$, (ii) $x \otimes y \in \mathcal{D}$ iff $x \in \mathcal{D}$ and $y \in \mathcal{D}$.

b) A bifilter $\mathcal{D}$ is called *prime*, if it also satisfies the following conditions:
   (i) $x \vee y \in \mathcal{D}$ iff $x \in \mathcal{D}$ or $y \in \mathcal{D}$, (ii) $x \oplus y \in \mathcal{D}$ iff $x \in \mathcal{D}$ or $y \in \mathcal{D}$.

Clearly, for every prime bifilter $\mathcal{D}$ we have that $t, \top \in \mathcal{D}$, while $f, \bot \notin \mathcal{D}$.

Definition 4.4 [5]. A *logical bilattice* is a pair $(\mathcal{B}, \mathcal{D})$, in which $\mathcal{B}$ is a bilattice and $\mathcal{D}$ is a prime bifilter of $\mathcal{B}$.

The basic semantical notions of the four-valued case can easily be extended to the bilattice-valued case. For instance, given a logical bilattice $(\mathcal{B}, \mathcal{D})$, the notions of valuations, models, etc. are the same as in the four-valued case. The definition of the implication connective $\leftarrow$ also remains the same: for every $x, y, \in B$ the value of $x \leftarrow y$ is x if $y \in \mathcal{D}$, and it is t otherwise. The only difference is that instead of taking $\mathcal{D} = \{t, \top\}$ as the set of the designated values, we now allow that any prime bifilter in B would be the set of the designated values.

The minimal logical bilattice is $\mathcal{FOUR}$ with $\mathcal{D} = \{t, \top\}$. Next we describe a general way of constructing logical bilattices with arbitrarily many elements:

Definition 4.5 [29]. Let $(L, \leq_L)$ be a complete lattice. The structure $L \odot L = (L \times L, \leq_t, \leq_k, \neg)$ is defined as follows:
- $(x_1, y_1) \geq_t (x_2, y_2)$ iff $x_1 \geq_L x_2$ and $y_1 \leq_L y_2$,
- $(x_1, y_1) \geq_k (x_2, y_2)$ iff $x_1 \geq_L x_2$ and $y_1 \geq_L y_2$,
- $\neg(x, y) = (y, x)$.

A pair $(x, y) \in L \odot L$ may intuitively be understood so that x represents the amount of belief *for* some assertion, and y is the amount of belief *against* it.

Definition 4.6. Let $(x, y) \in L \odot L$. Denote: $[(x, y)]_T = x$ and $[(x, y)]_F = y$.

Example 4.7. Let $\mathcal{TWO} = (\{0, 1\}, 0 < 1\}$ be the classical (two-valued) lattice. Then in the notations of Definition 4.5, Belnap's bilattice $\mathcal{FOUR}$ is isomorphic to $\mathcal{TWO} \odot \mathcal{TWO}$ by the following isomorphism: t corresponds to $(1, 0)$, f corresponds to $(0, 1)$, $\bot$ corresponds to $(0, 0)$, and $\top$ corresponds to $(1, 1)$.

Proposition 4.8 [20,29]. For every complete lattice $(L, \leq_L)$, the structure $L \odot L$ is a bilattice.

Proof. (Outlines) Given a lattice L with a meet operation $\sqcap$ and a join operation $\sqcup$, the bilattice operators are defined as follows:

$$(x_1, y_1) \vee (x_2, y_2) = (x_1 \sqcup x_2, y_1 \sqcap y_2), \qquad (x_1, y_1) \wedge (x_2, y_2) = (x_1 \sqcap x_2, y_1 \sqcup y_2),$$

$$(x_1, y_1) \oplus (x_2, y_2) = (x_1 \sqcup x_2, y_1 \sqcup y_2), \qquad (x_1, y_1) \otimes (x_2, y_2) = (x_1 \sqcap x_2, y_1 \sqcap y_2),$$

$$\neg(x, y) = (y, x).$$

It is easy to verify that for every two elements $x, y, \in L \times L$, $x \vee y$ (respectively, $x \wedge y$) is the least upper bound (respectively, the greatest lower bound) of x and y w.r.t. the $\leq_t$-partial order (Definition 4.5). Similarly, $x \oplus y$ (respectively, $x \otimes y$) is the least upper bound (respectively, the greatest lower bound) of x and y w.r.t. the $\leq_k$-partial order (Definition 4.5), and $\neg$ satisfies all the properties of a negation operator (Definition 4.2). $\square$

Proposition 4.9 [5]. Let $(L, \leq_L)$ be a complete lattice with a maximal element,

m. Then the smallest[24] (prime) bifilter in $L \odot L$ is of the form $\{(m,x) \mid x \in L\}$. We shall denote this bifilter by $\mathcal{D}_{L\odot L}$.

Corollary 4.10. Let $(L,\leq_L)$ be a complete bounded lattice. Then

a) $(L\odot L, \mathcal{D}_{L\odot L})$ is a logical bilattice.

b) Every complete bounded lattice induces a logical bilattice.

Proof. Part (a) follows from Propositions 4.8 and 4.9. Part (b) follows from part (a). ☐

4.3. Bilattice-based semantics for prioritized logic programs

For giving semantics to prioritized logic programs, we have found it useful to concentrate on bilattices of the form $\{0,1,\ldots,m\}\odot\{0,1,\ldots,m\}$. We shall denote these bilattices by $\mathcal{B}^m$.

As in the non-prioritized case, we start by considering a semantics for prioritized programs without negation-as-failure, and then consider the general case.

Definition 4.11. An *m-prioritized general logic program* is a set of quantitative general clauses, i.e., a set of formulae of the form $l \overset{n}{\leftarrow} Body$, where l is a literal, $Body$ is a conjunction of literals, and n is a number between 1 and m.

The quantitative values of the clauses (the n's) may be intuitively understood as representing "confidence factors" or "threshold values" of (the belief in) the corresponding clauses. The idea is that a head of a quantitative clause is evaluated only if there is a "sufficient" evidence in favor of the clause's body, and the evidence for the complement of the clause head does not exceed the clause's threshold value. Next we formalize this intuition:

Definition 4.12. Given a logical bilattice $(\mathcal{B}^m,\mathcal{D})$ and an m-prioritized general logic program $\mathcal{P}$, consider for every literal l and $i\geq 1$ the following functions:

$\nu_0^{\mathcal{P}}(l) = \mathsf{val}_0^{\mathcal{P}}(l) = \bot$.

$\mathsf{threshold}_i^{\mathcal{P}}(l) = lub_{\leq_k}\{\nu_{i-1}^{\mathcal{P}}(Body) \mid l \overset{n}{\leftarrow} Body \in \mathcal{P}\}$. [25]

$\mathsf{belief}_i^{\mathcal{P}}(l) = lub_{\leq_k}\{\nu_{i-1}^{\mathcal{P}}(Body) \mid l \overset{n}{\leftarrow} Body \in \mathcal{P},\ [\mathsf{threshold}_i^{\mathcal{P}}(\bar{l})]_T \leq n\}$. [26]

$\mathsf{val}_i^{\mathcal{P}}(l) = lub_{\leq_k}(\mathsf{val}_{i-1}^{\mathcal{P}}(l), \mathsf{belief}_i^{\mathcal{P}}(l))$.

[24] With respect to set inclusion.

[25] If there is no clause of the form $l \overset{n}{\leftarrow} Body$ in $\mathcal{P}$, define $\mathsf{threshold}_i^{\mathcal{P}}(l)=\bot$.

[26] If no clause of the form $l \overset{n}{\leftarrow} Body$ appears in $\mathcal{P}$, or $[\mathsf{threshold}_i^{\mathcal{P}}(\bar{l})]_T > \max\{n \mid l \overset{n}{\leftarrow} Body \in \mathcal{P}\}$, define $\mathsf{belief}_i^{\mathcal{P}}(l)=\bot$.

$\nu_i^{\mathcal{P}}(l) = ([\mathsf{val}_i^{\mathcal{P}}(l)]_T, [\mathsf{val}_i^{\mathcal{P}}(\bar{l})]_T)$.

For a limit ordinal λ we define:

$\mathsf{threshold}_\lambda^{\mathcal{P}}(l) = lub_{\leq_k}\{\mathsf{threshold}_\alpha^{\mathcal{P}}(l) \mid \alpha < \lambda\}$.

$\mathsf{belief}_\lambda^{\mathcal{P}}(l) = lub_{\leq_k}\{\mathsf{belief}_\alpha^{\mathcal{P}}(l) \mid \alpha < \lambda\}$.

$\mathsf{val}_\lambda^{\mathcal{P}}(l) = lub_{\leq_k}\{\mathsf{val}_\alpha^{\mathcal{P}}(l) \mid \alpha < \lambda\}$.

$\nu_\lambda^{\mathcal{P}}(l) = ([\mathsf{val}_\lambda^{\mathcal{P}}(l)]_T, [\mathsf{val}_\lambda^{\mathcal{P}}(\bar{l})]_T)$.

Again, for every propositional constant $\mathtt{x}$ that corresponds to an element $x \in \mathcal{B}^m$ we define, for every $i \geq 0$, $\nu_i^{\mathcal{P}}(\mathtt{x}) = \mathsf{val}_i^{\mathcal{P}}(\mathtt{x}) = x$.

In each iteration we therefore compute, for each literal, its 'threshold value', which is the least upper bound (w.r.t. $\leq_k$) of the values attached to the relevant clause bodies at the previous iteration.[27] Then we compute the amount of the 'belief' in a certain literal l during the current iteration. Again, the required value is obtained by considering the relevant clause bodies, but this time we take into account only those clauses with a 'sufficiently high' confidence factor, i.e., those clauses with l as their head, and with a confidence value that is not smaller than the threshold value of l's complement. Now, $\mathsf{val}_i^{\mathcal{P}}(\cdot)$ is a $\leq_k$-monotonic function that is based on these belief values, and finally — as in the four-valued case — we use $\mathsf{val}_i^{\mathcal{P}}(\cdot)$ for constructing the $\leq_k$-monotonic sequence of valuations $\nu_i^{\mathcal{P}}$ that yields, eventually, the fixpoint semantics for $\mathcal{P}$.

Remark 4.13. As in the four-valued case, for every i and l, $\nu_i^{\mathcal{P}}(\bar{l}) = \neg\nu_i^{\mathcal{P}}(l)$.

As the next proposition shows, the semantics for non-prioritized programs (Definition 3.1) is a particular case of the semantics for prioritized programs (Definition 4.12).

Proposition 4.14. Let $\mathcal{P}$ be a general logic program, and let $\mathcal{P}'$ be the 1-prioritized logic program obtained from $\mathcal{P}$ by assigning the quantitative factor $n = 1$ to every general clause in $\mathcal{P}$. Then, for every i, $\nu_i^{\mathcal{P}}$ (defined in 3.1) is the same as $\nu_i^{\mathcal{P}'}$ (defined in 4.12).

Proof. See Appendix A. □

As in the four-valued case, the partial order $\leq_k$ on $\mathcal{B}$ can be used for defining a partial order on the set $V^{\mathcal{B}}$ of the B-valued valuations: a valuation $\nu_1 \in V^{\mathcal{B}}$ is

[27] This value may be intuitively understood as an 'a priori' belief in the literal under consideration, since the confidence factors of the clauses are not taken into account in the calculation of this value.

k-smaller than another valuation $\nu_2 \in V^{\mathcal{B}}$ (notation: $\nu_1 <_k \nu_2$) if $\nu_1(p) \leq_k \nu_2(p)$ for every atom p. The pair $\mathcal{V}^{\mathcal{B}} = (V^{\mathcal{B}}, \leq_k)$ is clearly a lattice. Moreover,

Proposition 4.15. Let $\mathcal{P}$ be an m-prioritized general logic program, and let $\mathcal{B} = \{0, 1, \ldots, m\} \odot \{0, 1 \ldots, m\}$. The sequence $\nu_0^{\mathcal{P}}, \nu_1^{\mathcal{P}}, \nu_2^{\mathcal{P}}, \ldots$ is $\leq_k$-monotonic in $\mathcal{V}^{\mathcal{B}}$.

Proof. By Definition 4.12, for every $\alpha > \beta$ and every literal l, $\mathsf{val}_\alpha^{\mathcal{P}}(l) \geq_k \mathsf{val}_\beta^{\mathcal{P}}(l)$. Thus $\nu_\alpha^{\mathcal{P}}(l) \geq_k \nu_\beta^{\mathcal{P}}(l)$, and so $\nu_\alpha^{\mathcal{P}} \geq_k \nu_\beta^{\mathcal{P}}$. $\square$

Again, by Knaster-Tarski theorem [45], it follows that $\{\nu_i^{\mathcal{P}}\}$ has a $\leq_k$-least fixpoint. We denote it by $\nu^{\mathcal{P}}$.

The following result is an immediate consequence of Proposition 4.14.

Proposition 4.16. Let $\mathcal{P}$ be a general logic program, and let $\mathcal{P}'$ be the 1-prioritized program obtained from $\mathcal{P}$ by setting $n = 1$ as the quantitative factor of every general clause in $\mathcal{P}$. Then $\nu^{\mathcal{P}}$ (the $\leq_k$-least fixpoint of the $\nu_i^{\mathcal{P}}$-sequence, defined in 3.1) is the same as $\nu^{\mathcal{P}'}$ (the $\leq_k$-least fixpoint of the $\nu_i^{\mathcal{P}'}$-sequence, defined in 4.12).

We now generalize our formalism to m-prioritized *extended* logic programs. We do so in a way which is completely analogous to the way we generalized the fixpoint semantics for non-prioritized general logic programs to non-prioritized extended logic programs. I.e., we use a transformation like that of the well-founded semantics to eliminate the negation-as-failure operators from the clause bodies. What remains are m-prioritized general logic programs, to which we give semantics in the way described above. The following definitions formalize this process.

Definition 4.17. An *m-prioritized extended logic program* is a set of quantitative extended clauses, i.e.: a set of formulae of the form $l \stackrel{n}{\leftarrow} Body$, where l is a literal, $Body$ is a conjunction of extended literals, and n is a number between 1 and m.

Definition 4.18. Let $\mathcal{P}$ be an m-prioritized extended logic program.

a) A valuation $\mu \in V^4$ is a *plausible model* of $\mathcal{P}$, if it coincides with the fixpoint semantics of the m-prioritized general logic program, obtained by reducing $\mathcal{P}$ w.r.t. the set that is associated with μ. I.e., $\mu = \nu^{\mathcal{P} \downarrow S_\mu}$.
b) An *adequate model* of $\mathcal{P}$ is a $\leq_k$-maximal plausible model of $\mathcal{P}$.

By Proposition 4.14 and Definition 4.18 we have the following result:

Proposition 4.19. Let $\mathcal{P}$ be an extended logic program, and let $\mathcal{P}'$ be the 1-prioritized extended logic program obtained from $\mathcal{P}$ by assigning the quantitative factor $n=1$ to every extended clause in $\mathcal{P}$. Then:

a) μ is a plausible model of $\mathcal{P}$ (according to Definition 3.10) iff it is a plausible model of $\mathcal{P}'$ (according to Definition 4.18).
b) μ is an adequate model of $\mathcal{P}$ iff it is an adequate model of $\mathcal{P}'$.

4.4. Tweety dilemma, revisited

Consider again the logic program for Tweety dilemma, given in Example 4.1. The corresponding 1-prioritized program is the following:

$$\mathcal{P}_1 = \left\{ \begin{array}{ll} \texttt{fly}(x) \overset{1}{\leftarrow} \texttt{bird}(x) & \neg\texttt{reptile}(x) \overset{1}{\leftarrow} \texttt{bird}(x) \\ \texttt{bird}(x) \overset{1}{\leftarrow} \texttt{penguin}(x) & \neg\texttt{fly}(x) \overset{1}{\leftarrow} \texttt{penguin}(x) \\ \texttt{bird(Tweety)} \overset{1}{\leftarrow} \texttt{t} & \texttt{penguin(Tweety)} \overset{1}{\leftarrow} \texttt{t} \end{array} \right\}$$

Since $\mathcal{P}_1$ is a "flat" program (each clause is assigned the same priority), then by Proposition 4.16 its fixpoint semantics coincides with that of the non-prioritized case. In particular, we still have that $\nu^{\mathcal{P}_1}(\texttt{fly(Tweety)}) = \top$. However, now it is possible to give different priorities to different clauses. As we have noted during the previous discussion on this example, the clause $\texttt{fly}(x) \overset{1}{\leftarrow} \texttt{bird}(x)$ describes only a default property of birds, while the other clauses of the program describe rules that do not have exceptions. We therefore attach to $\texttt{fly}(x) \overset{1}{\leftarrow} \texttt{bird}(x)$ a lower priority (confidence factor) than the other assertions. The logic program that is obtained is the following:

$$\mathcal{P}_2 = \left\{ \begin{array}{ll} \texttt{fly}(x) \overset{1}{\leftarrow} \texttt{bird}(x) & \neg\texttt{reptile}(x) \overset{2}{\leftarrow} \texttt{bird}(x) \\ \texttt{bird}(x) \overset{2}{\leftarrow} \texttt{penguin}(x) & \neg\texttt{fly}(x) \overset{2}{\leftarrow} \texttt{penguin}(x) \\ \texttt{bird(Tweety)} \overset{2}{\leftarrow} \texttt{t} & \texttt{penguin(Tweety)} \overset{2}{\leftarrow} \texttt{t} \end{array} \right\}$$

The corresponding bilattice, *NINE* $= \{0,1,2\} \odot \{0,1,2\}$ is displayed in Figure 2 (see also [5,6,44]). We abbreviate its elements by the following notations:

$$\begin{array}{lllll} \bot=(0,0) & \mathit{df}=(0,1) & \mathit{dt}=(1,0) & f=(0,2) & t=(2,0) \\ d\top=(1,1) & \mathit{of}=(1,2) & \mathit{ot}=(2,1) & \top=(2,2) & \end{array}$$

Note that *NINE* has *two* prime bifilters: $\mathcal{D}_t = \{x \mid x \geq_k t\}$ and $\mathcal{D}_{dt} = \{x \mid x \geq_k dt\}$. Consequently, two corresponding logical bilattices may be considered: $\mathcal{NINE}_t = (\mathit{NINE}, \mathcal{D}_t)$ and $\mathcal{NINE}_{dt} = (\mathit{NINE}, \mathcal{D}_{dt})$. As $\mathcal{D}_t \subset \mathcal{D}_{dt}$, the former logical bilattice may be used for a more skeptical reasoning process, while the latter one provides a more liberal approach for a query evaluation (we shall

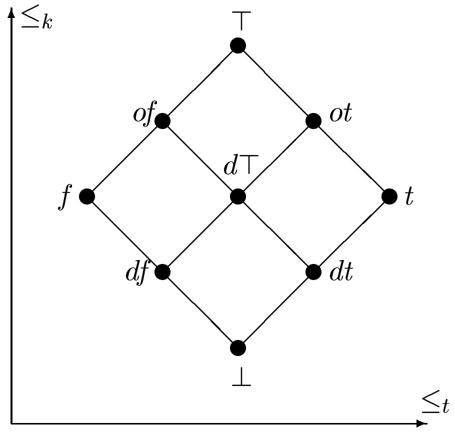

Figure 2. *NINE*

demonstrate this distinction in what follows).

Table 1 presents the process of constructing $\nu^{\mathcal{P}_2}$.

Table 1
iterative construction of $\nu^{\mathcal{P}_2}$

function	bird	¬bird	penguin	¬penguin	fly	¬fly	reptile	¬reptile
$\mathsf{threshold}_1$	t	$\bot$	t	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
belief_1	t	$\bot$	t	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
val_1	t	$\bot$	t	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
ν_1	t	f	t	f	$\bot$	$\bot$	$\bot$	$\bot$
$\mathsf{threshold}_2$	t	$\bot$	t	$\bot$	t	t	$\bot$	t
belief_2	t	$\bot$	t	$\bot$	$\bot$	t	$\bot$	t
val_2	t	$\bot$	t	$\bot$	$\bot$	t	$\bot$	t
ν_2	t	f	t	f	f	t	f	t

It is easy to see that for every $i \geq 2$, $\nu_{i+1}^{\mathcal{P}_2} = \nu_i^{\mathcal{P}_2}$, thus the $\leq_k$-least fixpoint of $\mathcal{P}_2$ is the following:

$$\nu^{\mathcal{P}_2}(\texttt{bird(Tweety)}) = t, \qquad \nu^{\mathcal{P}_2}(\texttt{penguin(Tweety)}) = t,$$
$$\nu^{\mathcal{P}_2}(\texttt{reptile(Tweety)}) = f, \qquad \nu^{\mathcal{P}_2}(\texttt{has\_feathers(Tweety)}) = \bot,$$
$$\nu^{\mathcal{P}_2}(\texttt{fly(Tweety)}) = f.$$

So the intuitive conclusion regarding the flying ability of Tweety is obtained, and the other literal conclusions remain as in the non-prioritized case, as expected.

It is interesting to note that our approach supports a very flexible process of belief revision. To see this, suppose that another datum arrives, and we are informed that Tweety might fly after all. Suppose further that our resource is not so sure about this information, or that this resource is not a reliable one. There are two options to express this uncertainty in our program: one way is to attach to the new information a low priority. Alternatively, we can put an attenuation factor in the clause body. The impact of the former option is that in case of conflicts we prefer the complementary information and ignore the new data altogether, while the effect of the latter option is that we always consider the new data, but give it a lower weight when we draw our conclusions. According to the second option, the modified program may be the following: [28]

$$\mathcal{P}_3 = \mathcal{P}_2 \cup \{ \ \texttt{fly(Tweety)} \overset{2}{\leftarrow} \texttt{dt} \ \}$$

Table 2 describes the iterative construction of $\nu^{\mathcal{P}_3}$.

Table 2
iterative construction of $\nu^{\mathcal{P}_3}$

function	bird	¬bird	penguin	¬penguin	fly	¬fly	reptile	¬reptile
$\mathsf{threshold}_1$	t	$\bot$	t	$\bot$	dt	$\bot$	$\bot$	$\bot$
belief_1	t	$\bot$	t	$\bot$	dt	$\bot$	$\bot$	$\bot$
val_1	t	$\bot$	t	$\bot$	dt	$\bot$	$\bot$	$\bot$
ν_1	t	f	t	f	dt	df	$\bot$	$\bot$
$\mathsf{threshold}_2$	t	$\bot$	t	$\bot$	t	t	$\bot$	t
belief_2	t	$\bot$	t	$\bot$	dt	t	$\bot$	t
val_2	t	$\bot$	t	$\bot$	dt	t	$\bot$	t
ν_2	t	f	t	f	of	ot	f	t

Again, after two iterations we reach a fixpoint, in which $\nu^{\mathcal{P}_3}(\texttt{fly(Tweety)}) = of$. The interpretation of this result depends on the logical bilattice under consideration (i.e., the choice of the prime bifilter in *NINE*):

- In $\mathcal{NINE}_{ot}$ the fixpoint values of fly(Tweety) and of ¬fly(Tweety) are both designated. This means that the new datum, although being somewhat unreliable, causes an inconsistent belief regarding the flying ability of Tweety. Nevertheless, the fact that fly(Tweety) is assigned *of* rather than $\top$ reflects the fact that $\mathcal{P}_3$ contains more evidence in favour of ¬fly(Tweety) rather than in favour of fly(Tweety).
- In $\mathcal{NINE}_t$ the fixpoint value of ¬fly(Tweety) is designated, while the fixpoint value of fly(Tweety) is not. This means that despite the new datum we actually still believe that Tweety cannot fly. However, because of the new information, we are less certain than before (thus fly(Tweety) is assigned *of* rather than f).

[28] Where dt is a propositional constant that corresponds to the truth value dt in *NINE* (intuitively understood as "true by default").

5. Some concluding remarks

We conclude with a summary of the main properties of the formalisms considered here, and some further remarks regarding related semantics.

Reasoning with incomplete and inconsistent data

One of the main drawbacks of some of the fixpoint semantics for extended logic program (like those of [27] and [37]) is that they are reduced to triviality in the presence of contradictions. As such, these formalisms inherit one of the well-known shortcomings of classical logic. We do believe that since inconsistent knowledge can and may be represented by extended logic programs, a plausible semantics for such programs should be able to handle inconsistent situations in a non-trivial way. That is, one should be able to draw meaningful conclusions (and reject others) despite the inconsistency. The fixpoint semantics considered here has such capabilities: it pinpoints the inconsistent and the incomplete parts of the data, and regards the rest of the information as classically consistent.

Consider, for instance, the following program, which is an extended variant of the program considered in Example 1.3 (see also item 3 of Example 3.14):

$$\mathcal{P} = \left\{ \begin{array}{lll} p \leftarrow \mathtt{t} & q \leftarrow \mathtt{not}\ q & \neg p \leftarrow \mathtt{not}\ \neg r_1 \\ r_1 \leftarrow \mathtt{t} & r_2 \leftarrow \mathtt{not}\ \neg r_1 & \neg r_3 \leftarrow r_1, r_2 \end{array} \right\}$$

This program provides a complete information regarding the truth or the falsity of r_i, $i=1,2,3$. Moreover, the information regarding these atoms is not affected by either p, q, or their negations. The fact that the data regarding q is incomplete and the data regarding p is inconsistent should be *localized* (i.e., restricted only to those literals whose definitions depend on p or q), and it should *not* affect the values of the r_i's. Thus, the inconsistent data in $\mathcal{P}$ should not spoil the whole piece of information that is represented by this program. The fixpoint semantics that was considered here follows these guidelines; the unique plausible model for $\mathcal{P}$ (and so its adequate model) is the following:

$$\nu^{\mathcal{P}} = \{\ p:\top,\ \ q:\bot,\ \ r_1:t,\ \ r_2:t,\ \ r_3:f\ \}.$$

It follows that the complete information in $\mathcal{P}$ (the one that concerns with r_i, $i=1,2,3$) is preserved. In addition, the reasoner may realize that the data about p is contradictory, and the data about q is incomplete.

Relating negative data to its positive counterpart

Another major difference between the semantics introduced here and some other semantics for extended logic programs (e.g, [25,27,31,42]) concerns with the way a negative data is related to its positive counterpart. While the formalisms of [25,27,31,42] treat p and $\neg p$ as two different *atomic formulae*, we preserve the relation between an atomic formula and its negated atom. To see the importance of this, consider the following program (also considered in [10, Example 3.3.6] and [37, Example 1]):

$$\mathcal{P} \;=\; \{\; p \leftarrow \texttt{not}\; q \qquad q \leftarrow \texttt{not}\; p \qquad \neg p \leftarrow \texttt{t} \;\}$$

According to the approaches that treat $\neg p$ as (a strange way of writing) an atomic formula, the well-founded semantics would assign here t to $\neg p$, $\bot$ to p, and $\bot$ to q. So even though $\mathcal{P}$ is classically consistent, the distinction between p and $\neg p$ causes a counter-intuitive result here. In contrast, our approach yields a semantics that seems to reflect the intuitive expectation in this case: the adequate model of $\mathcal{P}$ (which is also the only plausible model of $\mathcal{P}$ is this case) assigns f to p and t to q.

For another example, consider the following logic program [37, Example 6]:

$$\mathcal{P} \;=\; \{\; r \leftarrow \texttt{not}\; q \qquad q \leftarrow \texttt{not}\; p \qquad p \leftarrow \texttt{not}\; p \qquad \neg q \leftarrow \texttt{t} \;\}$$

The unique plausible model of $\mathcal{P}$ (and so its adequate model) is $\{p\!:\!\bot,\ q\!:\!f,\ r\!:\!t\}$ (which is the same as the one that is obtained in [37]). By considering $\neg q$ as a new atom, this program would have a single extended stable model, in which $\neg q$ is true and all the other atomic formulae (p, q, r) are unknown. This seems to be a counter-intuitive result in this case, since one expects here that r would follow from $\mathcal{P}$.

Paraconsistent and coherent approaches to inconsistency

The formalisms that we have described here for giving semantics to extended logic programs are paraconsistent in nature. I.e., they accept contradictions within the theory and try to cope with them. Another common approach to handle contradictions (sometimes called *coherent* or *conservative* [13,48]) first detects and eliminates the inconsistent part of the theory. Then, when consistency is restored, some classical formalism is used for drawing plausible conclusions from the "recovered" data. In [31], for instance, clauses with negative literals in their heads are getting higher priorities than clauses with positive literals in their head. The latter ones are ignored in case of contradictions with their negated counterparts. This approach assures a contradictions-free semantics [31, Theorem 2]. In [38] contradictions are excluded already in the level of knowledge representation, since clauses for default rules have the form $l \leftarrow Body, \texttt{not}\; \bar{l}$. Thus, in order to derive l, one has to verify first that its complement, $\bar{l}$, is not provable. Other coherent formalisms for managing inconsistent information are considered, e.g., in [2–4,7,13,18].

Belief revision

The need to alter the set of conclusions according to an input that is frequently modified is not an unusual phenomenon in common-sense reasoning in general and logic programming in particular. Thus, the plausibility of different formalisms in these areas is often determined by the way they handle revised information. To see that consider, e.g., the following example (anonymous author):

> "A man fell from a plane. Fortunately, he was wearing a parachute. Unfortunately, the parachute didn't open. Fortunately, he fell from the plane at a low altitude over a large haystack. Unfortunately, there was a pitchfork in the haystack. Fortunately, he missed the pitchfork. Unfortunately, he missed the haystack ...".

After each sentence in this example there is a tendency to jump back and forth between opposite conclusions regarding the ultimate fate of the skydriver. In Tweety dilemma, considered in Section 4.4, we faced the same phenomenon when we had to change our mind several times regarding Tweety's ability to fly in light of the new data that had arrived. Indeed, in the notations used in that section,

- `fly(Tweety)` follows from $\mathcal{P}_2 \setminus \{\texttt{penguin(Tweety)} \overset{2}{\leftarrow} \texttt{t}\}$,
- `fly(Tweety)` does not follow from $\mathcal{P}_2$,
- `fly(Tweety)` does not follow from $\mathcal{P}_2 \cup \{\texttt{fly(Tweety)} \overset{2}{\leftarrow} \texttt{dt}\}$ when the underlying semantics is induced by $\mathcal{NINE}_t$ (i.e., by a skeptical reasoning), but it does follow from $\mathcal{P}_2 \cup \{\texttt{fly(Tweety)} \overset{2}{\leftarrow} \texttt{dt}\}$ when the underlying semantics is induced by $\mathcal{NINE}_{ot}$ (i.e., by a more liberal reasoning).

As demonstrated in Section 4.4, the flexibility of the process for belief revision in our case is reflected both on the semantical level (different choices of logical bilattices yield different conclusions), and on the syntactical level (by enhancing the expressive power of the logic programs under consideration, thus allowing various ways to represent knowledge, either in the program language itself, or in a meta-language that reflects the reasoner's preferences).

Acknowledgements

I would like to thank Maurice Bruynooghe, Marc Denecker, and the anonymous reviewers for their helpful comments. This work was supported by the visiting postdoctoral fellowship FWO Flanders.

Appendix

A. Proofs

Proposition 3.3. Let $\mathcal{P}$ be a general logic program. Then $\nu^{\mathcal{P}}$ is the k-minimal four-valued model of $\mathcal{P}$. Moreover, it is at least as consistent as any other model of $\mathcal{P}$, and the consequence relation induced by it is paraconsistent.

Proof. This proposition contains several claims. We divide the proof accordingly.

1. *$\nu^{\mathcal{P}}$ is a model of $\mathcal{P}$*:
   Suppose not. Then there is a clause $l \leftarrow Body$ in $\mathcal{P}$ s.t. $\nu^{\mathcal{P}}(Body) \in \mathcal{D}$ while $\nu^{\mathcal{P}}(l) \notin \mathcal{D}$. In particular, there is an α s.t. for every $\beta \geq \alpha$, $\nu^{\mathcal{P}}_{\beta}(Body) \in \mathcal{D}$ while $\nu^{\mathcal{P}}_{\beta}(l) \notin \mathcal{D}$. But since $\nu^{\mathcal{P}}_{\alpha}(Body) \in \mathcal{D}$, for every $\beta > \alpha$, $\mathsf{val}^{\mathcal{P}}_{\beta}(l) = t$,[29] which implies that $\nu^{\mathcal{P}}_{\beta}(l) \geq_k t$, and so $\nu^{\mathcal{P}}_{\beta}(l) \in \mathcal{D}$. This contradicts the assumption that $\nu^{\mathcal{P}}_{\beta}(l) \notin \mathcal{D}$.

2. *$\nu^{\mathcal{P}}$ induces a paraconsistent consequence relation*:
   Consider, e.g., $\mathcal{P} = \{p \leftarrow \mathtt{t},\ \neg p \leftarrow \mathtt{t}\}$. Here $\nu^{\mathcal{P}}(p) = \top$ while $\nu^{\mathcal{P}}(q) = \bot$ for every atom $q \neq p$. Thus $\mathcal{P} \not\models_{\nu} q$ for every atom $q \neq p$, which means that trivial reasoning from an inconsistent set of premises is not allowed.

3. *$\nu^{\mathcal{P}}$ is $\leq_k$-smaller than any other model of $\mathcal{P}$*:
   For a valuation ν denote $\mathsf{Sat}(\nu) = \{l \mid \nu(l) \in \mathcal{D}\}$. Let α be the fixpoint ordinal of $\nu^{\mathcal{P}}$ (i.e., the minimal ϵ s.t. $\nu^{\mathcal{P}}_{\beta'} = \nu^{\mathcal{P}}_{\beta}$ for every $\beta' \geq \beta \geq \epsilon$). We show that for every model M of $\mathcal{P}$, $\mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha}) \subseteq \mathsf{Sat}(M)$. This immediately implies that $\nu^{\mathcal{P}} \leq_k M$, since in this case, for every atom p, we have that

   - If $\nu^{\mathcal{P}}(p) = \top$, then since α is the fixpoint ordinal of $\nu^{\mathcal{P}}$, $\nu^{\mathcal{P}}_{\alpha}(p) = \top$. Hence $p, \neg p \in \mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha})$. By our assumption this implies that $p, \neg p \in \mathsf{Sat}(M)$, and so $M(p) = \top$ as well.
   - If $\nu^{\mathcal{P}}(p) = t$, then again by the definition of α, $p \in \mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha})$, and so $p \in \mathsf{Sat}(M)$. Thus $M(p) \in \{t, \top\}$, which implies that $M(p) \geq_k \nu^{\mathcal{P}}(p)$.
   - If $\nu^{\mathcal{P}}(p) = f$, then $\nu^{\mathcal{P}}_{\alpha}(p) = f$ and since $\nu^{\mathcal{P}}_{\alpha}(\neg p) = \neg \nu^{\mathcal{P}}_{\alpha}(p)$, we have that $\neg p \in \mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha})$. By our assumption, then, $\neg p \in \mathsf{Sat}(M)$, thus $M(\neg p) \in \{t, \top\}$. It follows that $M(p) = \neg M(\neg p) \in \{f, \top\}$ and so $M(p) \geq_k \nu^{\mathcal{P}}(p)$.
   - If $\nu^{\mathcal{P}}(p) = \bot$ then clearly $M(p) \geq_k \nu^{\mathcal{P}}(p)$.

   It remains to show, therefore, that for every model M of $\mathcal{P}$, $\mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha}) \subseteq \mathsf{Sat}(M)$. We show this by a transfinite induction on α.

[29] for successor ordinals this follows from the definition of $\mathsf{val}^{\mathcal{P}}_{\beta}(l)$, and for limit ordinals this follows from the fact that for every l', $\mathsf{val}^{\mathcal{P}}_{\beta}(l') \in \{t, \bot\}$.

The case $\alpha=0$ is obvious, since $\mathsf{Sat}(\nu_0^{\mathcal{P}})=\{\mathtt{t},\mathtt{c}\}$.[30] For a successor ordinal $\alpha>0$, let $l\in\mathsf{Sat}(\nu_\alpha^{\mathcal{P}})$. Then $\nu_\alpha^{\mathcal{P}}(l)\in\{t,\top\}$, and so $\mathsf{val}_\alpha^{\mathcal{P}}(l)\oplus\neg\mathsf{val}_\alpha^{\mathcal{P}}(\bar{l})\in\{t,\top\}$. This means that either $\mathsf{val}_\alpha^{\mathcal{P}}(l)\in\{t,\top\}$, or $\neg\mathsf{val}_\alpha^{\mathcal{P}}(\bar{l})\in\{t,\top\}$ (i.e., $\mathsf{val}_\alpha^{\mathcal{P}}(\bar{l})\in\{f,\top\}$). But since for every literal l', $\mathsf{val}_\alpha^{\mathcal{P}}(l')\in\{t,\bot\}$, this means that in our case necessarily $\mathsf{val}_\alpha^{\mathcal{P}}(l)=t$. Hence, there is a general clause of the form $l\leftarrow Body$, for which $\nu_{\alpha-1}^{\mathcal{P}}(Body)\in\{t,\top\}$. Thus, for every $l_i\in Body$, $l_i\in\mathsf{Sat}(\nu_{\alpha-1}^{\mathcal{P}})$. By the induction hypothesis, for every $l_i\in Body$, $l_i\in\mathsf{Sat}(M)$, and so $M(Body)\in\{t,\top\}$ as well. But M is a model of $\mathcal{P}$, and so necessarily $M(l)\in\{t,\top\}$, i.e. $l\in\mathsf{Sat}(M)$.
If α is a limit ordinal and $l\in\mathsf{Sat}(\nu_\alpha^{\mathcal{P}})$, then by the same arguments as above, $\mathsf{val}_\alpha^{\mathcal{P}}(l)=t$. This implies that there exists an ordinal $\beta<\alpha$, for which $\mathsf{val}_\beta^{\mathcal{P}}(l)=t$ as well. Thus $l\in\mathsf{Sat}(\nu_\beta^{\mathcal{P}})$, and by the induction hypothesis we are done.

4. *$\nu^{\mathcal{P}}$ is at least as consistent as any other model of $\mathcal{P}$*:
Denote again by α the fixpoint ordinal of $\nu^{\mathcal{P}}$, and let $\mathsf{Sat}(\nu)=\{l\mid\nu(l)\in\mathcal{D}\}$. Suppose that $\nu^{\mathcal{P}}(p)=\top$ for some atom p. Then $p,\neg p\in\mathsf{Sat}(\nu^{\mathcal{P}})$, and so $p,\neg p\in\mathsf{Sat}(\nu_\alpha^{\mathcal{P}})$. By the proof of the previous item, for every model M of $\mathcal{P}$, $\mathsf{Sat}(\nu_\alpha^{\mathcal{P}})\subseteq\mathsf{Sat}(M)$. Thus $p,\neg p\in\mathsf{Sat}(M)$, and so $M(p)=\top$ as well.

□

Proposition 3.7. Let $\mathcal{P}$ be a positive logic program, and let $\mathcal{P}$' be the positive program obtained from $\mathcal{P}$ by replacing every implication connective by a material implication. Denote by $\nu_{f/\bot}^{\mathcal{P}}$ the valuation that is obtained from $\nu^{\mathcal{P}}$ by changing the $\bot$-assignments to f-assignments. Then:

1. $\mathcal{P}$ and $\mathcal{P}'$ have the same classical models (and thus the same least Herbrand model), and
2. $\nu_{f/\bot}^{\mathcal{P}}$ is the (unique) 2-valued minimal Herbrand model of $\mathcal{P}$ and $\mathcal{P}$'.

Proof. The first claim simply follows from the fact that the implication connective of Definition 2.4 is the same as the material implication on $\{t,f\}$. Regarding the other part, note first that since only atomic formulae appear in the clauses heads, for every atom p and for every i we have that $\mathsf{val}_i^{\mathcal{P}}(\neg p)=\bot$, and therefore $\nu_i^{\mathcal{P}}(p)=\mathsf{val}_i^{\mathcal{P}}(p)\in\{t,\bot\}$. It follows that $\nu^{\mathcal{P}}$ assigns only values in $\{t,\bot\}$ to the atomic formulae (and so, for every literal l, $\nu^{\mathcal{P}}(l)\in\{t,f,\bot\}$). Now, let $p\leftarrow Body$ be a clause in $\mathcal{P}$. Since $\mathcal{P}$ is positive, then $\nu^{\mathcal{P}}(Body)=t$ iff all the atoms in $Body$ are assigned t by $\nu^{\mathcal{P}}$, iff all the atoms in $Body$ are assigned t by $\nu_{f/\bot}^{\mathcal{P}}$, iff $\nu_{f/\bot}^{\mathcal{P}}(Body)=t$. Similarly, $\nu^{\mathcal{P}}(Body)\in\{f,\bot\}$ iff there is an atomic formula in $Body$ that is assigned either $\bot$ or f by $\nu^{\mathcal{P}}$, iff there is an atomic formula in $Body$

[30] Recall that $\mathtt{t}$ and $\mathtt{c}$ are the propositional constants that are associated with t and $\top$, respectively (and so they are elements of $\mathsf{Sat}(\nu)$ for every ν).

that is assigned f by $\nu^{\mathcal{P}}_{f/\bot}$, iff $\nu^{\mathcal{P}}_{f/\bot}(Body) = f$. It follows that for every clause $\mathcal{C}$ in $\mathcal{P}$, $\nu^{\mathcal{P}}(\mathcal{C}) \in \mathcal{D}$ iff $\nu^{\mathcal{P}}_{f/\bot}(\mathcal{C}) \in \mathcal{D}$. Thus $\nu^{\mathcal{P}}_{f/\bot}$ is a 2-valued model of $\mathcal{P}$.

It remains to show that $\nu^{\mathcal{P}}_{f/\bot}$ is $\leq_t$-minimal among the classical models of $\mathcal{P}$. Indeed, this follows from the fact that $\nu^{\mathcal{P}}$ is $\leq_k$-smaller than any other model of $\mathcal{P}$ (Proposition 3.3), and so the set of atomic formulae that are assigned t by $\nu^{\mathcal{P}}$ does not properly contain any corresponding set of a classical model of $\mathcal{P}$.[31] Thus, the set of atomic formulae that are assigned t by $\nu^{\mathcal{P}}_{f/\bot}$ does not properly contain any corresponding set of a classical model of $\mathcal{P}$ either, and so $\nu^{\mathcal{P}}_{f/\bot}$ is a $\leq_t$-minimal model among the classical models of $\mathcal{P}$. Since $\mathcal{P}$ has the same classical models as those of $\mathcal{P}'$ (item 1 of this proposition), we conclude that $\nu^{\mathcal{P}}_{f/\bot}$ is also a $\leq_t$-minimal model among the classical models of $\mathcal{P}'$. But being positive, $\mathcal{P}$' has only *one* $\leq_t$-minimal classical model (which is its least Herbrand model), and so $\nu^{\mathcal{P}}_{f/\bot}$ coincides with this model. $\square$

Proposition 4.14. Let $\mathcal{P}$ be a general logic program, and let $\mathcal{P}'$ be the 1-prioritized logic program obtained from $\mathcal{P}$ by assigning the quantitative factor $n=1$ to every general clause in $\mathcal{P}$. Then, for every i, $\nu^{\mathcal{P}}_i$ (defined in 3.1) is the same as $\nu^{\mathcal{P}'}_i$ (defined in 4.12).

Proof. First, as noted in Example 4.7, the bilattice $\mathcal{B}=\{0,1\}\odot\{0,1\}$ that gives semantics to the 1-prioritized program $\mathcal{P}'$ is isomorphic to the bilattice $\mathcal{FOUR}$ used in Section 2 to give semantics to the "flat" (non-prioritized) program $\mathcal{P}$. In what follows we shall use both representations to denote the same elements.

By the definition of the $\leq_t$-operations and the $\leq_k$-operations in the bilattice $\{0,1\}\odot\{0,1\}$, we have that [32]

$$\begin{aligned}\nu^{\mathcal{P}}_i(l) &= \mathsf{val}^{\mathcal{P}}_i(l) \oplus \neg\mathsf{val}^{\mathcal{P}}_i(\bar{l}) = ([\mathsf{val}^{\mathcal{P}}_i(l)]_T, [\mathsf{val}^{\mathcal{P}}_i(l)]_F) \oplus ([\mathsf{val}^{\mathcal{P}}_i(\bar{l})]_F, [\mathsf{val}^{\mathcal{P}}_i(\bar{l})]_T) \\ &= (max([\mathsf{val}^{\mathcal{P}}_i(l)]_T, [\mathsf{val}^{\mathcal{P}}_i(\bar{l})]_F),\ max([\mathsf{val}^{\mathcal{P}}_i(l)]_F, [\mathsf{val}^{\mathcal{P}}_i(\bar{l})]_T)).\end{aligned}$$

But since $\mathsf{val}^{\mathcal{P}}_i(\cdot) \in \{t,\bot\} = \{(1,0),(0,0)\}$, we have that $[\mathsf{val}^{\mathcal{P}}_i(\cdot)]_F = 0$, thus

$$\nu^{\mathcal{P}}_i(l) = ([\mathsf{val}^{\mathcal{P}}_i(l)]_T, [\mathsf{val}^{\mathcal{P}}_i(\bar{l})]_T).$$

Since $\nu^{\mathcal{P}'}_i(l) = ([\mathsf{val}^{\mathcal{P}'}_i(l)]_T, [\mathsf{val}^{\mathcal{P}'}_i(\bar{l})]_T)$, it remains to show that for every i and l, $[\mathsf{val}^{\mathcal{P}}_i(l)]_T = [\mathsf{val}^{\mathcal{P}'}_i(l)]_T$. We shall consider here the case in which i is a successor ordinal, leaving the other case to the reader.

Indeed, recall that all the clauses in $\mathcal{P}'$ are assigned the maximal confidence factor (which is 1 in our case), and so for every i and l, $[\mathsf{threshold}^{\mathcal{P}}_i(\bar{l})]_T \leq 1$. It follows, then, that

[31] Indeed, if M is a classical model of $\mathcal{P}$ and p is an atom s.t. $\nu^{\mathcal{P}}(p) = t$ while $M(p) \neq t$, then $M(p) = f$, and so $\nu^{\mathcal{P}} \not\leq_k M$.

[32] See the proof of Proposition 4.8 for the definitions of the relevant operations.

$$\begin{aligned}\mathsf{val}_i^{\mathcal{P}'}(l) &= lub_{\leq_k}(\mathsf{val}_{i-1}^{\mathcal{P}'}(l),\ \mathsf{belief}_i^{\mathcal{P}'}(l))\\ &= lub_{\leq_k}(\mathsf{val}_{i-1}^{\mathcal{P}'}(l),\ lub_{\leq_k}\{\nu_{i-1}^{\mathcal{P}'}(Body_j) \mid l \xleftarrow{1} Body_j \in \mathcal{P}'\}).\end{aligned}$$

Using again the fact that $\nu_i^{\mathcal{P}'}(l) = ([\mathsf{val}_i^{\mathcal{P}'}(l)]_T, [\mathsf{val}_i^{\mathcal{P}'}(\bar{l})]_T)$, we have that

$$\begin{aligned}[\mathsf{val}_i^{\mathcal{P}'}(l)]_T &= \max([\mathsf{val}_{i-1}^{\mathcal{P}'}(l)]_T,\ \max\{[\nu_{i-1}^{\mathcal{P}'}(Body_j)]_T \mid l \xleftarrow{1} Body_j \in \mathcal{P}'\})\\ &= \max([\nu_{i-1}^{\mathcal{P}'}(l)]_T,\ \max\{[\nu_{i-1}^{\mathcal{P}'}(Body_j)]_T \mid l \xleftarrow{1} Body_j \in \mathcal{P}'\}).\end{aligned}$$

Since in our case $\mathcal{B} = \{0,1\} \odot \{0,1\}$, it follows that

$$[\mathsf{val}_i^{\mathcal{P}'}(l)]_T = 1 \text{ if } \exists(l \xleftarrow{1} Body) \in \mathcal{P}' \text{ and } \nu_{i-1}^{\mathcal{P}'}(Body) \in \mathcal{D}_{\{0,1\}\odot\{0,1\}},$$

$$[\mathsf{val}_i^{\mathcal{P}'}(l)]_T = 0 \text{ otherwise.}$$

On the other hand, by the definition of $\mathsf{val}_i^{\mathcal{P}}$,

$$[\mathsf{val}_i^{\mathcal{P}}(l)]_T = 1 \text{ if } \mathsf{val}_i^{\mathcal{P}}(l) = t, \text{ if } \exists(l \leftarrow Body) \in \mathcal{P} \text{ and } \nu_{i-1}^{\mathcal{P}}(Body) \in \mathcal{D}_{\mathcal{FOUR}},$$

$$[\mathsf{val}_i^{\mathcal{P}}(l)]_T = 0 \text{ otherwise.}$$

It follows, therefore, that $[\mathsf{val}_i^{\mathcal{P}}(l)]_T = [\mathsf{val}_i^{\mathcal{P}'}(l)]_T$, as required. □

B. Embedding in Fitting's semantics

In what follows we consider some cases in which our 4-valued semantics may be transformed into Fitting's 3-valued semantics.

Definition B.1. Given a normal logic program $\mathcal{P}$, consider the following general logic program:

$$\begin{aligned}\mathcal{P}^* = \mathcal{P}\ \cup\ &\{\neg p \leftarrow \bar{l}\ \mid p \leftarrow Body \in \mathcal{P},\ \nu^{\mathcal{P}}(p) = \bot,\ l \in \mathcal{L}(Body)\} \cup\\ &\{\neg p \leftarrow \mathtt{t} \mid p \leftarrow \mathtt{f} \in \mathcal{P},\ \nu^{\mathcal{P}}(p) = \bot\}.\end{aligned}$$

Intuitively, $\mathcal{P}^*$ is obtained from $\mathcal{P}$ by adding rules that explicitly formalize what is implicitly assumed by Fitting's semantics. For instance, in Fitting's semantics the meaning of $p \leftarrow \mathtt{f}$ is that p is false. Here, since negations may also appear in the clause heads, we must also explicitly declare that we mean that $\neg p$ should hold, and so we add the statement $\neg p \leftarrow \mathtt{t}$.

Proposition B.2. Let $\mathcal{P}$ be a normal logic program in which each atomic formula appears at most once in a clause head. Let also $\Psi^{\mathcal{P}}$ be Fitting fixpoint semantics for $\mathcal{P}$. Then $\Psi^{\mathcal{P}} = \nu^{\mathcal{P}^*}$.

Example B.3.

1. Let $\mathcal{P} = \{p \leftarrow q\}$. According to Fitting's semantics this is an abbreviation of $\mathcal{P}' = \{p \leftarrow q,\ q \leftarrow \mathtt{f}\}$ (an atom that occurs in $\mathcal{P}$ but does not appear

in any clause head in $\mathcal{P}$ is considered as false, see [19, Section 5]). Fitting's least fixpoint semantics for $\mathcal{P}'$ assigns f to p and q. Now, by Definition B.1, $(\mathcal{P}')^* = \{p \leftarrow q,\ q \leftarrow \mathtt{f},\ \neg p \leftarrow \neg q,\ \neg q \leftarrow \mathtt{t}\}$. It is easy to verify that our four-valued semantics for $(\mathcal{P}')^*$ also assigns f to both p and q.

2. Let $\mathcal{P} = \{p \leftarrow p\}$. Then $\mathcal{P}^* = \{p \leftarrow p,\ \neg p \leftarrow \neg p\}$. This is a natural extension of $\mathcal{P}$ to an extended logic program that states that both p and its negation depend only on themselves. Clearly, then, $\Psi^{\mathcal{P}} = \nu^{\mathcal{P}^*} = \{p : \bot\}$.
3. Consider again the logic program $\mathcal{P}$ of Remark 3.6. By Definition B.1 we have that $\mathcal{P}^* = \{p \leftarrow \neg q,\ \neg p \leftarrow q,\ q \leftarrow \mathtt{t}\}$. By Proposition B.2, our four-valued semantics for $\mathcal{P}^*$ is the same as that of Fitting 3-valued fixpoint operator for $\mathcal{P}$. Both of them assign t to q and f to p.

Remark B.4. The requirement in Proposition B.2 that every atomic formula should not appear more than once in a clause head is indeed necessary. To see that consider, e.g., the following program:

$$\mathcal{P} = \{q \leftarrow p,\ \ q \leftarrow \neg r,\ \ r \leftarrow \mathtt{t}\}.$$

Then $\mathcal{P}^* = \mathcal{P} \cup \{\neg q \leftarrow \neg p,\ \neg q \leftarrow r\}$, and while $\Psi^{\mathcal{P}}(q) = \Psi^{\mathcal{P}}(p) \vee \Psi^{\mathcal{P}}(\neg r) = \bot$, we have that $\nu^{\mathcal{P}^*}(q) = f$.

Proof of Proposition B.2. By Definition B.1, $\mathcal{P}^* = \mathcal{P} \cup \mathcal{P}_\neg$ where

$$\mathcal{P}_\neg = \{\neg p \leftarrow \bar{l} \mid p \leftarrow Body \in \mathcal{P},\ \nu^{\mathcal{P}}(p) = \bot,\ l \in \mathcal{L}(Body)\} \cup \{\neg p \leftarrow \mathtt{t} \mid p \leftarrow \mathtt{f} \in \mathcal{P},\ \nu^{\mathcal{P}}(p) = \bot\}.$$

• Suppose first that for some atom q, $\Psi^{\mathcal{P}}(q) = t$. We show that in this case $\nu^{\mathcal{P}}(q) = t$ as well. Assuming this, then by the definition of $\mathcal{P}_\neg$, $\neg q$ cannot appear in the head of any clause of $\mathcal{P}_\neg$, and so $\neg q$ cannot appear in the head of any clause of $\mathcal{P}^*$. It follows, then, that for every α, $\mathsf{val}_\alpha^{\mathcal{P}^*}(\neg q) = \bot$, and so $\nu_\alpha^{\mathcal{P}^*}(q) = \mathsf{val}_\alpha^{\mathcal{P}^*}(q) \in \{t, \bot\}$. Thus $\nu^{\mathcal{P}^*}(q) \in \{t, \bot\}$. On the other hand, if $\mathcal{P}_1 \subseteq \mathcal{P}_2$ then $\nu^{\mathcal{P}_2} \geq_k \nu^{\mathcal{P}_1}$, thus $\nu^{\mathcal{P}^*}(q) \geq_k \nu^{\mathcal{P}}(q) = t$. It follows, then, that $\nu^{\mathcal{P}^*}(q) = t$, and so $\nu^{\mathcal{P}^*}(q) = \Psi^{\mathcal{P}}(q)$ in this case.

To complete the proof for the first case it remains therefore to show that for every atom q, if $\Psi^{\mathcal{P}}(q) = t$ then $\nu^{\mathcal{P}}(q) = t$ as well. Let $\{\Psi_0^{\mathcal{P}}, \Psi_1^{\mathcal{P}}, \ldots\}$ be the $\leq_k$-monotonic iterative sequence of valuations used for constructing $\Psi^{\mathcal{P}}$. Since $\neg q$ does not appear in any clause head in $\mathcal{P}$, we have that for every α $\mathsf{val}_\alpha^{\mathcal{P}}(\neg q) = \bot$, and so $\nu_\alpha^{\mathcal{P}}(p) = \mathsf{val}_\alpha^{\mathcal{P}}(q) = t$. Thus, for showing that if $\Psi^{\mathcal{P}}(q) = t$ then $\nu^{\mathcal{P}}(q) = t$, it is sufficient to show that for every α and atom q s.t. $\Psi_\alpha^{\mathcal{P}}(q) = t$, $\mathsf{val}_\alpha^{\mathcal{P}}(q) = t$ as well. We show this by a transfinite induction on α. For $\alpha = 0$ we have that $\Psi_0^{\mathcal{P}}(q) = \mathsf{val}_\alpha^{\mathcal{P}}(q) = \bot$, so the condition is vacuously met. For $\alpha = 1$, $\Psi_1^{\mathcal{P}}(q) = t$ iff $q \leftarrow \mathtt{t} \in \mathcal{P}$ iff $\mathsf{val}_1^{\mathcal{P}}(q) = t$. For a successor ordinal $\alpha > 1$, $\Psi_\alpha^{\mathcal{P}}(q) = t$ iff there is a clause of the form $q \leftarrow Body$ in $\mathcal{P}$ and $\Psi_{\alpha-1}^{\mathcal{P}}(Body) = t$, iff $\forall l_i \in \mathcal{L}(Body)$ $\Psi_{\alpha-1}^{\mathcal{P}}(l_i) = t$, iff (induction hypothesis) $\forall l_i \in \mathcal{L}(Body)$ $\mathsf{val}_{\alpha-1}^{\mathcal{P}}(l_i) = t$. Thus $\mathsf{val}_{\alpha-1}^{\mathcal{P}}(Body) = t$, which implies that $\nu^{\mathcal{P}}_{\alpha-1}(Body)\in\mathcal{D}$, and so $\mathsf{val}^{\mathcal{P}}_{\alpha}(q)=t$. Finally, if α is a limit ordinal, $\Psi^{\mathcal{P}}_{\alpha}(q)=max_{\preceq_k}\{\Psi^{\mathcal{P}}_{\beta}(q)\mid\beta<\alpha\}$, where $\preceq_k$ is the three-valued analogue of $\leq_k$.[33] Thus, the assumption that $\Psi^{\mathcal{P}}_{\alpha}(q)=t$ implies that $\Psi^{\mathcal{P}}_{\beta}(q)=t$ for some $\beta<\alpha$. By the induction hypothesis, then, $\mathsf{val}^{\mathcal{P}}_{\beta}(q)=t$. Thus $\mathsf{val}^{\mathcal{P}}_{\alpha}(q)=max_{\leq_k}\{\mathsf{val}^{\mathcal{P}}_{\beta}(l)\mid\beta<\alpha\}\in\{t,\top\}$. But we have shown that $\mathsf{val}^{\mathcal{P}}_{\alpha}(q)\in\{t,\bot\}$, thus $\mathsf{val}^{\mathcal{P}}_{\alpha}(q)=t$.

• Suppose now that $\Psi^{\mathcal{P}}(q)=f$. Let $\{\Psi^{\mathcal{P}}_0,\Psi^{\mathcal{P}}_1,\ldots\}$ be the $\leq_k$-monotonic iterative sequence of valuations used for constructing $\Psi^{\mathcal{P}}$. We show that for every α and literal l s.t. $\Psi^{\mathcal{P}}_{\alpha}(l)=f$, $\mathsf{val}^{\mathcal{P}^*}_{\alpha}(\overline{l})=t$. Assuming this, we are able to show that $\nu^{\mathcal{P}^*}(q)=\Psi^{\mathcal{P}}(q)$ in this case as well, since the fact that $\Psi^{\mathcal{P}}(q)=f$ implies that there exists some ϵ s.t. for every $\beta\geq\epsilon$ $\Psi^{\mathcal{P}}_{\beta}(q)=f$, and so by our assumption, $\mathsf{val}^{\mathcal{P}^*}_{\beta}(\neg q)=t$. Note also that by Proposition 3.3, $\nu^{\mathcal{P}}$ is the $\leq_k$-least model of $\mathcal{P}$, thus, since $\Psi^{\mathcal{P}}$ is a model of $\mathcal{P}$, and since $f=\Psi^{\mathcal{P}}(q)\geq_k\nu^{\mathcal{P}}(q)\in\{t,\bot\}$, necessarily $\nu^{\mathcal{P}}(q)=\bot$. Thus, for every α, $\mathsf{val}^{\mathcal{P}}_{\alpha}(q)=\bot$. Since q does not appear in the head of clauses in $\mathcal{P}_{\neg}$, this means that for every α, $\mathsf{val}^{\mathcal{P}^*}_{\alpha}(q)=\bot$ as well. It follows, then, that for every $\beta\geq\epsilon$ $\nu^{\mathcal{P}^*}_{\beta}(q)=\mathsf{val}^{\mathcal{P}^*}_{\beta}(q)\oplus\neg\mathsf{val}^{\mathcal{P}^*}_{\beta}(\neg q)=\bot\oplus\neg t=f$. One thus concludes that $\nu^{\mathcal{P}^*}(q)=f=\Psi^{\mathcal{P}}(q)$.

For this case of the proof it remains, therefore, to show that for every α and a literal l s.t. $\Psi^{\mathcal{P}}_{\alpha}(l)=f$, $\mathsf{val}^{\mathcal{P}^*}_{\alpha}(\overline{l})=t$. We show it by a transfinite induction on α. For $\alpha=0$ the condition is vacuously met, since $\Psi^{\mathcal{P}}_0(l)=\bot$ for every l. For $\alpha=1$, the fact that $\Psi^{\mathcal{P}}_1(l)=f$ entails that $l\leftarrow\mathtt{f}$ appears in $\mathcal{P}$. Since $\mathcal{P}$ is a normal program, l must be an atom in this case. Moreover, by our assumption on $\mathcal{P}$, this is the only clause which contains l as its head, and so $\nu^{\mathcal{P}}(l)=\bot$. Thus $\overline{l}\leftarrow\mathtt{t}$ appears in $\mathcal{P}_{\neg}$, and so $\mathsf{val}^{\mathcal{P}^*}_1(\overline{l})=t$. Suppose now that for some successor ordinal $\alpha>1$, $\Psi^{\mathcal{P}}_{\alpha}(l)=f$. By the construction of the $\Psi^{\mathcal{P}}_j$-s, and by our assumption on $\mathcal{P}$, it follows that for the only clause of the form $l\leftarrow Body$ that appears in $\mathcal{P}$, $\Psi^{\mathcal{P}}_{\alpha-1}(Body)=f$. This means that there is some $l'\in\mathcal{L}(Body)$ s.t. $\Psi^{\mathcal{P}}_{\alpha-1}(l')=f$. By induction hypothesis, then, $\mathsf{val}^{\mathcal{P}^*}_{\alpha-1}(\overline{l'})=t$. Now, $\Psi^{\mathcal{P}}(l)\geq_k\Psi^{\mathcal{P}}_{\alpha}(l)=f$ thus $\Psi^{\mathcal{P}}(l)=f$. On the other-hand, using again the fact that $\nu^{\mathcal{P}}$ is the $\leq_k$-least model of $\mathcal{P}$ and that $\Psi^{\mathcal{P}}$ is a model of $\mathcal{P}$, $\Psi^{\mathcal{P}}(l)\geq_k\nu^{\mathcal{P}}(l)\in\{t,\bot\}$. Hence $\nu^{\mathcal{P}}(l)=\bot$. This means that $\overline{l}\leftarrow\overline{l'}$ appears in $\mathcal{P}_{\neg}$. But $\mathsf{val}^{\mathcal{P}^*}_{\alpha-1}(\overline{l'})=t$, and so $\nu^{\mathcal{P}^*}_{\alpha-1}(\overline{l'})$ is designated. Thus, $\mathsf{val}^{\mathcal{P}^*}_{\alpha}(\overline{l})=t$, as required. The proof for limit ordinals is the same as in the previous item.

• Finally, suppose that $\Psi^{\mathcal{P}}(q)=\bot$. Again, let $\{\Psi^{\mathcal{P}}_0,\Psi^{\mathcal{P}}_1,\ldots\}$ be the $\leq_k$-monotonic iterative sequence of valuations used for constructing $\Psi^{\mathcal{P}}$. This time we show that for every α and literal l s.t. $\Psi^{\mathcal{P}}_{\alpha}(l)=\bot$ we have that $\mathsf{val}^{\mathcal{P}^*}_{\alpha}(\overline{l})=\bot$ as well. This implies that $\nu^{\mathcal{P}^*}(q)=\Psi^{\mathcal{P}}(q)$ also in this case, since the fact that $\Psi^{\mathcal{P}}(q)=\bot$ implies that for every α, $\Psi^{\mathcal{P}}_{\alpha}(q)=\bot$, and so by our assumption, for every α we have that $\mathsf{val}^{\mathcal{P}^*}_{\alpha}(\neg q)=\bot$. Note also that since $\bot=\Psi^{\mathcal{P}}(q)\geq_k\nu^{\mathcal{P}}(q)$, necessarily $\nu^{\mathcal{P}}(q)=\bot$, and so $\mathsf{val}^{\mathcal{P}}_{\alpha}(q)=\bot$ for every α. Since q does not appear in the

[33] I.e., $\nu_1\preceq_k\nu_2$ iff for every atom p, $\nu_1(p)\preceq_k\nu_2(p)$, where $\bot\prec_k t$ and $\bot\prec_k f$.

head of clauses in $\mathcal{P}_\neg$, this means that for every α, $\mathsf{val}_\alpha^{\mathcal{P}^*}(q)=\bot$ as well. Thus $\nu^{\mathcal{P}^*}(q)=\mathsf{val}_\alpha^{\mathcal{P}^*}(q)\oplus\neg\mathsf{val}_\alpha^{\mathcal{P}^*}(\neg q)=\bot\oplus\neg\bot=\bot$. It follows that $\nu^{\mathcal{P}^*}(q)=\bot=\Psi^{\mathcal{P}}(q)$.

It remains to show that for every α and a literal l s.t. $\Psi_\alpha^{\mathcal{P}}(l)=\bot$, we have that also $\mathsf{val}_\alpha^{\mathcal{P}^*}(\bar{l})=\bot$. Again, we show it by a transfinite induction on α. For $\alpha=0$ this is obviously true, since by their definitions $\Psi_0^{\mathcal{P}}$ and $\mathsf{val}_0^{\mathcal{P}^*}$ are both identically $\bot$. For $\alpha=1$, $\Psi_1^{\mathcal{P}}(l)=\bot$ iff $\Psi_1^{\mathcal{P}}(p)=\bot$ where p is the atomic part of l, iff either p does not appear in the head of any clause of $\mathcal{P}$, or $p\leftarrow Body\in\mathcal{P}$ and $\mathcal{L}(Body)\neq\emptyset$. In the first case neither l nor $\bar{l}$ appear in the head of any clause of $\mathcal{P}^*$, and in the second case if a clause of the form $\bar{l}\leftarrow Body$ appears in $\mathcal{P}^*$, then $\mathcal{L}(Body)\neq\emptyset$. In both cases, therefore, $\mathsf{val}_1^{\mathcal{P}^*}(\bar{l})=\bot$. For a successor ordinal $\alpha>1$, $\Psi_\alpha^{\mathcal{P}}(l)=\bot$ means again that $\Psi_\alpha^{\mathcal{P}}(p)=\bot$, where p is the atomic part of l. This can happen if either p does not appear in the head of any clause of $\mathcal{P}$ (which again implies that $\mathsf{val}_\alpha^{\mathcal{P}^*}(\bar{l})=\bot$, as in the basis of the induction), or else — by our assumption on $\mathcal{P}$ — there is a single clause in $\mathcal{P}$ of the form $p\leftarrow Body$ and $\Psi_{\alpha-1}^{\mathcal{P}}(Body)=\bot$. This means that $\forall l'\in\mathcal{L}(Body)$ $\Psi_{\alpha-1}^{\mathcal{P}}(l')\in\{t,\bot\}$ (and $\exists l_0'\in\mathcal{L}(Body)$ s.t. $\Psi_{\alpha-1}^{\mathcal{P}}(l_0')=\bot$). By what we have shown in the first case of this proof (in case that $\Psi_{\alpha-1}^{\mathcal{P}}(l')=t$) and by the induction hypothesis (in case that $\Psi_{\alpha-1}^{\mathcal{P}}(l')=\bot$), $\mathsf{val}_{\alpha-1}^{\mathcal{P}^*}(\overline{l'})=\bot$ for every $l'\in\mathcal{L}(Body)$. Thus, $\nu_{\alpha-1}^{\mathcal{P}^*}(l')=\mathsf{val}_{\alpha-1}^{\mathcal{P}^*}(l')\in\{\bot,t\}$. In other words, $\nu_{\alpha-1}^{\mathcal{P}^*}(\overline{l'})\in\{\bot,f\}$ so $\nu_{\alpha-1}^{\mathcal{P}^*}(\overline{l'})$ is not designated. Since the only clauses in which $\neg p$ may appear as their head are of the form $\neg p\leftarrow\overline{l'}$, it follows that $\mathsf{val}_\alpha^{\mathcal{P}^*}(\neg p)=\bot$. Since p do not appear as a head of any clause in $\mathcal{P}_\neg$, we also have that $\mathsf{val}_\alpha^{\mathcal{P}^*}(p)=\mathsf{val}_\alpha^{\mathcal{P}}(p)\leq_k\nu_\alpha^{\mathcal{P}}(p)\leq_k\nu^{\mathcal{P}}(p)\leq_k\Psi^{\mathcal{P}}(p)=\bot$. Hence, both $\mathsf{val}_\alpha^{\mathcal{P}^*}(p)=\bot$ and $\mathsf{val}_\alpha^{\mathcal{P}^*}(\neg p)=\bot$. Thus, either if $l=p$ or $l=\neg p$, we have that $\mathsf{val}_\alpha^{\mathcal{P}^*}(\bar{l})=\bot$. The proof for limit ordinals is similar to the ones given in the previous items. □

References

[1] J.J.Alferes, H.Herre, and L.M.Pereira, Partial models of extended generalized logic programs, in: *Proc. 1st International Conference on Computational Logic (CL'2000), Lecture Notes in Artificial Intelligence 1861*, eds. J.Lloyd et al., Springer-Verlag, 2000, pp. 149–163.

[2] J.J.Alferes, J.A.Leite, L.M.Pereira, H.Przymusinska and T.C.Przymusinski, Dynamic updates of non-monotonic knowledge base, Journal of Logic Programming 45 (2000) 43–70.

[3] O.Arieli, Four-valued logics for reasoning with uncertainty in prioritized data, in: *Information, Uncertainty, Fusion,* eds. B.Bouchon-Meunier, R.R.Yager and L.A.Zadeh, Kluwer Academic Publishers, 1999, pp. 293–304.

[4] O.Arieli, An algorithmic approach to recover inconsistent knowledge-bases, *Proc. 7th European Workshop on Logics in Artificial Intelligence (JELIA'00), Lecture Notes in Artificial Intelligence 1919*, eds. M.Ojeda-Aciego, I.P.de Guzman, G.Brewka and L.M.Pereira, Springer-Verlag, 2000, pp. 148–162.

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

[6] O.Arieli and A.Avron, The value of the four values, Artificial Intelligence 102(1) (1998) 97–141.

[7] O.Arieli, B.Van Nuffelen, M.Denecker and M.Bruynooghe, Coherent composition of distributed knowledge-bases through abduction. *Proc. 8th International Conference on Logic Programming, Artificial Intelligence and Reasoning (LPAR'01), Lecture Notes in Computer Science 2250* eds. A.Nieuwenhius and A.Voronokov, Springer-Verlag, 2001, pp. 620–635.

[8] A.Avron, Simple consequence relations, Information and Computation 92 (1991) 105–139.

[9] A.Avron, The structure of interlaced bilattices, Mathematical Structures in Computer Science 6 (1996) 287–299.

[10] C.Baral and M.Gelfond, Logic programming and knowledge representation, Journal of Logic Programming 19–20 (1994) 73–148.

[11] N.D.Belnap, A useful four-valued logic, in: *Modern Uses of Multiple-Valued Logic,* eds. G.Epstein and J.M.Dunn, Reidel Publishing Company, 1977, pp. 7–37.

[12] N.D.Belnap, How computer should think, in: *Contemporary Aspects of Philosophy,* ed. G.Ryle, Oriel Press, 1977, pp. 30–56.

[13] S.Benferhat, D.Dubois and H.Prade, How to infer from inconsistent beliefs without revising? in: *Proc. International Joint Conference on Artificial Intelligence (IJCAI'95),* 1995, pp. 1449–1455.

[14] K.L.Clark, Negation as failure, in: *Logic and Databases,* eds. H.Gallaire and J.Minker, Plenum Press, 1978, pp. 293–322.

[15] N.C.A.da-Costa, On the theory of inconsistent formal systems, Notre Damm Journal of Formal Logic 15 (1974) 497–510.

[16] C.V.Damásio and L.M.Pereira, A model theory for paraconsistent logic programming, in: *Proc. 7th Portuguese Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence 990,* Springer-Verlag, 1995, pp. 409–418,

[17] M.Denecker, The well-founded semantics is the principle of inductive definitions, in: *Proc. 6th European Workshop on Logic in Artificial Intelligence (JELIA'98), Lecture Notes in Artificial Intelligence 1498,* eds. J.Dix, L.Fari nas del Cerro and U.Furbach, Springer-Verlag, 1998, pp. 1–16.

[18] D.Dubios, J.Lang and H.Prade, Possibilistic logic, in: *Handbook of Logic in Artificial Intelligence and Logic Programming* eds. D.Gabbay, C.Hogger and J.Robinson, Oxford Science Publications, 1994, pp. 439–513.

[19] M.Fitting, Kripke-Kleene semantics for logic programs, Journal of Logic Programming 2 (1985) 295–312.

[20] M.Fitting, Bilattices in logic programming, in: *Proc. 20th IEEE International Symposium on Multiple-valued Logics,* ed. G.Epstein, IEEE Press, 1990, pp. 238–246.

[21] M.Fitting, Kleene's logic, generalized, Logic and Computation 1 (1990) 797–810.

[22] M.Fitting, Bilattices and the semantics of logic programming, Journal of Logic Programming 11(2) (1991) 91–116.

[23] M.Fitting, The family of stable models, Journal of Logic Programming 17 (1993) 197–225.

[24] M.Fitting, Kleene's three-valued logics and their children, Fundamenta Informaticae 20 (1994) 113–131.

[25] A.J.García, G.R.Simari, and C.I.Chesñevar, An argumentative framework for reasoning with inconsistent and incomplete information, in: *ECAI'98 Workshop on Practical Reasoning and Rationality,* 1998.

[26] M.Gelfond and V.Lifschitz, The stable model semantics for logic programming, in: *Proc. 5th Logic Programming Symposium,* eds. R.Kowalski and K.Bowen, MIT Press, 1988, pp. 1070–1080.

[27] M.Gelfond and V.Lifschitz, Logic programs with classical negation, in: *Proc. 7th International Conference on Logic Programming (ICLP'90),* eds. D.Warren and P.Szeredi, 1990, pp. 579–597.

[28] M.L.Ginsberg, Multi-valued logics. in: *Readings in Non-Monotonic Reasoning,* ed. M.L.Ginsberg, 1987, pp. 251–258.

[29] M.L.Ginsberg, Multi-valued logics: A uniform approach to reasoning in artificial intelligence, Computer Intelligence 4 (1988) 256–316.

[30] S.C.Kleene, *Introduction to metamathematics.* Van Nostrand, 1950.

[31] R.A.Kowalski and F.Sadri, Logic programs with exceptions, in: *Proc. 7th International Conference on Logic Programming (ICLP'90),* eds. D.Warren, and P.Szeredi, 1990, pp. 598–613.

[32] S.Kraus, D.Lehmann and M.Magidor, Nonmonotonic reasoning, preferential models and cumulative logics, Artificial Intelligence 44 (1990) 167–207.

[33] J.W.Lloyd, *Foundations of logic programming.* Springer Verlag, 1987.

[34] D.Makinson, General patterns in nonmonotonic reasoning, in: *Handbook of Logic in Artificial Intelligence and Logic Programming 3,* eds. D.Gabbay, C.Hogger and J.Robinson, Oxford Science Publications, 1994, pp. 35–110.

[35] B.Messing, Combining knowledge with many-valued logics, Data and Knowledge Engineering 23 (1997) 297–315.

[36] R.Nelken and N.Francez, Bilattices and the semantics of natural language questions, Technical Report, Department of Computer Science, The Technion, Israel (1998).

[37] L.M.Pereira and J.J.Alferes, Well-founded semantics for logic programs with explicit negation, in: *Proc. 10th European Conference on Artificial Intelligence (ECAI'92),* ed. B.Neumann, 1992, pp. 102–106.

[38] L.M.Pereira, J.N.Aparício and J.J.Alferes, Nonmonotonic reasoning with well founded semantics, in: *Proc. 8th International Conference on Logic Programming (ICLP'91),* ed. K.Furukawa, 1991, pp. 475–489.

[39] G.Priest, Minimally Inconsistent LP, Studia Logica 50 (1991) 321–331.

[40] H.Przymusinska and T.Przymusinski, Weakly perfect model semantics for logic programs, in: *Proc. 5th Logic Programming Symposium,* eds. R.Kowalski and K.Bowen, MIT Press, 1988, pp. 1106–1122,.

[41] H.Przymusinska and T.Przymusinski, Semantic issues in deductive databases and logic programs, in: *Formal Techniques in Artificial Intelligence,* ed. R.B.Banejí, Elsevier Science Publishers, 1990, pp. 321–367.

[42] T.Przymusinski, Extended stable semantics for normal and disjunctive programs, in: *Proc. 7th International Conference on Logic Programming (ICLP'90),* eds. D.Warren and P.Szeredi, 1990, pp. 459–477.

[43] R.Reiter, On closed word data bases, in: *Logic and Data Bases,* eds. H.Gallaire and J.Minker, Plenum Press, 1978, pp. 119–140.

[44] A.Schoter, Evidential bilattice logic and lexical inferences, Journal of Logic, Language and Information 5(1) (1996) 65–105.

[45] A.Tarski, Lattice-theoretic fixpoint theorem and its applications, Pacific Journal of Mathematics 5 (1955) 285–309.

[46] M.van-Emden and R.A.Kowalski, The semantics of predicate logic as a programming language, Journal of the ACM 23(4) (1976) 733–742.

[47] A.Van Gelder, K.A.Ross and J.S.Schlipf, The well-founded semantics for general logic programs, Journal of the ACM 38(3) (1991) 620–650.

[48] G.Wagner. *Vivid logic.* Lecture Notes in Artificial intelligence 764, Springer-Verlag, 1994.