Meta many-valued logic programming for incomplete and locally inconsistent databases

Zoran Majkić
Dipartimento di Informatica e Sistemistica, University of Roma “La Sapienza”
Via Salaria 113, I-00198 Rome, Italy
majkic@dis.uniroma1.it
http://www.dis.uniroma1.it/~majkic/

Abstract. Large databases obtained by the data integration of different source databases can be incomplete and inconsistent in many ways. Classical logic is not the appropriate formalism for reasoning about inconsistent databases. Certain local inconsistencies should not be allowed to significantly alter the intended meaning of such logic programs. The variety of semantical approaches that have been invented for logic programs is quite broad. Especially, we are interested in many-valued logics with negation, based on bilattices. We present a 2-valued logic, called EMV (Encapsulation of Many-Valued Logic Programming). This new approach introduces the trilattice (of interpretations) with the third truth dimension for this ‘meta’ logic over a bilattice of the many-valued ‘object’ logic. We defined a Model theoretic approach for Herbrand interpretations of an encapsulated logic program: they constitute a complete semilattice under knowledge preorder, so that is possible to apply the Knaster-Tarski theorem for monotonic ‘immediate consequence operator’ in order to define the least fixpoint semantics for encapsulated logic programs.

1 Introduction

Semantics of logic programs are generally based on a classical 2-valued logic by means of stable models, [1,2]. Under these circumstances not every program has a stable model. Generally, three-valued, or partial model semantics has had an extensive development for logic programs, [3,4]. Przymusinski extended the notion of the stable model to allow 3-valued, or partial, stable models, [5], and showed that every program has at least one partial stable model, and that the well-founded model is the smallest among them, [6]. Once one has made the transition from classical to partial models allowing incomplete information, it is a small step towards allowing models admitting inconsistent information. Doing so provides a natural framework for the semantic understanding of logic programs that are distributed over several sites, with possible conflicting information coming from different places. As classical logic semantics decrees that inconsistent theories have no models, classical logic is not the appropriate formalism for reasoning about inconsistent databases: certain "localizable" inconsistencies should not be allowed to significantly alter the intended meaning of such databases. So far, research in many-valued logic programming has proceeded along different directions: Signed logics [7,8,9] and Annotated logic programming [10,11,12] which can
be embedded into the first, *Bilattice-based* logics, [13,14], and *Quantitative rule-sets*, [15,16]. One of the key insights behind bilattices was the interplay between the truth values assigned to sentences and the (non classic) notion of *implication*. The problem was to study how truth values should be propagated "across" implications. Roughly, these frameworks can be classified into *implication based* (IB) and *annotation based* (AB).

In the IB approach a rule is of the form \( A \leftarrow \alpha \land B_1, \ldots, B_n \), which says that the certainty associated with the implication \( B_1 \land \cdots \land B_n \rightarrow A \) is \( \alpha \). Computationally, given an assignment \( I \) of logical values to the \( B_i \)'s, the logical value of \( A \) is computed by taking the "conjunction" of logical values \( I(B_i) \) and then somehow "propagating" it to the rule head \( A \). In the AB approach a rule is of the form \( A : f(\beta_1, \ldots, \beta_n) \leftarrow B_1 : \beta_1, \ldots, B_n : \beta_n \), which asserts "the certainty of the atom \( A \) is at least (or is in) \( f(\beta_1, \ldots, \beta_n) \), whenever the certainty of the atom \( B_i \) is at least (or is in) \( \beta_i, 1 \leq i \leq n \)". Where \( f \) is an \( n \)-ary computable function and \( \beta_i \) is either constant or a variable ranging over many-valued logic values.

It is believed that IB approach is easier to use and is more amenable for efficient implementations, but also annotated syntax (but with IB semantics) is useful: the syntax of new encapsulated many-valued logic (in some sense 'meta'-logic for a many-valued bilattice logic) will be 2-valued and can be syntactically seen as a kind of very simple annotated syntax. Thus the implication (and classical negation also), not present in a bilattice algebra operators, will have a natural semantic interpretation in this enriched framework.

The plan of this paper is the following: Section 2 introduce the Belnap’s bilattice concepts and the particular 4-valued version, \( B_4 \), used in this paper. Section 3 defines the syntax and the *model theoretic* Herbrand semantics for the encapsulated many-valued (EMV) logic programmes, together with a simple example of local inconsistency considerations: built-in predicates have a particular importance in a 4-valued logic programming. Section 4 develops the theory of a trilattice, \( B^A \), of interpretations for an encapsulated logic program \( P^A \), which generalize the bilattice structure of a functional space of Herbrand interpretations for (non encapsulated) logic programs.

## 2 Bilattice

In [17], Belnap introduced a logic intended to deal in a useful way with inconsistent or incomplete information. It is the simplest example of a non-trivial bilattice and it illustrates many of the basic ideas concerning them. We denote the four values as \( \{ t, f, \top, \perp \} \), where \( t \) is true, \( f \) is false, \( \top \) is inconsistent (both true and false) or possible, and \( \perp \) is unknown. As Belnap observed, these values can be given two natural orders: *truth* order, \( \leq_t \), and *knowledge* order, \( \leq_k \), such that \( f \leq_t \top \leq_t t \), \( f \leq_t \perp \leq_t t \), and \( \perp \leq_k f \leq_k \top \), \( \perp \leq_k t \leq_k \top \). This two orderings define corresponding equivalences \( =_t \) and \( =_k \). Thus any two members \( \alpha, \beta \) in a bilattice are equal, \( \alpha = \beta \), if and only if (shortly ’iff’) \( \alpha =_t \beta \) and \( \alpha =_k \beta \).

Meet and join operators under \( \leq_t \) are denoted \( \land \) and \( \lor \); they are natural generalizations of the usual conjunction and disjunction notions. Meet and join under \( \leq_k \) are denoted \( \odot \) (*consensus*), because it produces the most information that two truth values can agree
on) and \( \oplus \) (gullibility, it accepts anything it's told), such that hold:
\[
 f \otimes t = \bot, \quad f \otimes t = \top, \quad f \wedge \bot = f \quad \text{and} \quad f \vee \bot = t.
\]

There is a natural notion of truth negation, denoted \( \sim \), (reverses the \( \leq_k \) ordering, while preserving the \( \leq_k \) ordering): switching \( f \) and \( t \), leaving \( \bot \) and \( \top \), and corresponding knowledge negation, denoted \( \neg \), (reverses the \( \leq_k \) ordering, while preserving the \( \leq_k \) ordering): switching \( \bot \) and \( \top \), leaving \( f \) and \( t \). These two kinds of negation commute:
\[
 \neg \sim x = \sim \neg x \quad \text{for every member} \ x \ \text{of a bilattice.}
\]

It turns out that the operations \( \land, \lor \) and \( \sim \) are exactly those of Kleene’s strong 3-valued logic. Any bilattice \( \langle B, \leq, \otimes, \oplus, \sim, \top, \bot \rangle \) is:

1. **Interlaced**, if each of the operations \( \land, \lor \otimes \) and \( \oplus \) is monotone with respect to both orderings (for instance, \( x \leq_t y \) implies \( x \otimes z \leq_t y \otimes z \), \( x \leq_k y \) implies \( x \land z \leq_k y \land z \)).

2. **Infiniarily interlaced**, if it is complete and four infiniarily meet and join operations are monotone with respect to both orderings.

3. **Distributive**, if all 12 distributive laws connecting \( \land, \lor \otimes \) and \( \oplus \) are valid.

4. **Infiniarily distributive**, if it is complete and infiniarily, as well as finitary, distributive laws are valid. (Note that a bilattice is complete if all meets and joins exist, w.r.t. both orderings. We denote infiniary meet and join w.r.t. \( \leq_k \) by \( \land_k \) and \( \lor_k \), and by \( \land \) and \( \lor \) for the \( \leq_k \) ordering; for example, the distributive low for \( \otimes \) and \( \oplus \) may be given by:
\[
x \lor \land_k y_i = \bigwedge_k (x \otimes y_i).
\]

A more general information about bilattice may be found in [18]; he also defines exact members of a bilattice, when \( x = \neg \neg x \) (they are 2-valued consistent), and consistent members, when \( x \leq_k \neg \neg x \) (they are 3-valued consistent), but a specific 4-valued consistency will be analyzed in the following paragraphs.

The Belnap’s 4-valued bilattice is infiniarily distributive. In the rest of this paper we denote by \( B \) a special case of the Belnap’s bilattice, when also holds \( t = \top \), \( \top \not\leq_k f \) and \( f \not\leq_k t \) and, consequently, \( \top \land \bot = \bot \) and \( \top \lor \bot = \top \). In this way we consider the possible value as weak true value.

For any 4-valued quantified predicate logic, \( L_B \), based on this bilattice \( B \) (with propositional connectives limited to bilattice operations define above), which can be used to define the bodies of logic programming clauses (remember that this logic based on bilattice does not have implication connective), we are able to define a valuation, that is, a mapping from pure ground (variable free) atoms of \( L_B \) into the space of truth values \( B \). Thus, each Herbrand interpretation with a domain \( I_U \) is a valuation. Valuations can be extended to maps from the set of all ground (variable free) formulas to \( B \) in the following way:

**Definition 1.** A valuation \( v_B \) determines a unique map, also denoted \( v_B \), on the set of all ground formulas, according to the following conditions:

1. \( v_B(\neg x) = \neg v_B(x) \)
2. \( v_B(x \otimes y) = v_B(x) \otimes v_B(y), \, \text{where} \otimes \in \{ \land, \lor, \otimes, \oplus \} \)
3. \( v_B(\forall x)P(x)) = \bigwedge_{d \in I_U} v_B(P(d)) \), \( v_B(\exists x)P(x)) = \bigvee_{d \in I_U} v_B(P(d)) \)
4. \( v_B(\forall x)P(x)) = \prod_{d \in I_U} v_B(P(d)) \), \( v_B(\exists x)P(x)) = \prod_{d \in I_U} v_B(P(d)) \)

If \( P \) is a many-valued logic program with the Herbrand base \( H_P \) (set of all ground atoms of \( P \)), then the ordering relations and operations in a bilattice \( B \) are propagated to the function space \( B^H_P \), that is the set of all Herbrand interpretations (functions), \( I = v_B : H_P \rightarrow B \), as follows:
Definition 3. Ordering relations are defined on the Function space $\mathcal{B}_4^{HP}$ pointwise, as follows: for any two Herbrand interpretations $v_B, w_B \in \mathcal{B}_4^{HP}$

1. $v_B \leq_t w_B$ if $v_B(A) \leq_t v_B(w)$ for all $A \in H_P$.
2. $v_B \leq_k w_B$ if $v_B(A) \leq_k v_B(w)$ for all $A \in H_P$.
3. $\sim v_B$ is the interpretation such that $(\sim v_B)(A) = \neg(v_B(A))$.
4. $\neg v_B$ is the interpretation such that $(\neg v_B)(A) = \neg(v_B(A))$.

It is straightforward [18] that this makes a function space $\mathcal{B}_4^{HP}$ itself a complete infinitary distributive bilattice.

3 Encapsulation of a many-valued logic

The many-valued ground atoms of a bilattice-based logical language $\mathcal{L}_B$ can be transformed in ‘encapsulated’ atoms of a 2-valued logic in the following simple way: the original (many-valued) fact that the ground atom $A = p(c_1, \ldots, c_n)$, of the n-ary predicate $p$, has a value $\alpha$ in $\mathcal{B}_4$, we transform in annotated syntax $A : \alpha$ with meaning “it is true that $A$ has a value $\alpha$”. This syntax annotation is used for brevity: indeed, what we do is to replace the original n-ary predicate $p(x_1, \ldots, x_n)$ with n+1-ary predicate $p^\alpha(x_1, \ldots, x_n, \alpha)$, with the added logic-attribute $\alpha$ called an annotation also.

So, $A : \alpha$ represents the atom $p^\alpha(c_1, \ldots, c_n, \alpha)$. It is easy to verify that for any given many-valued valuation $v_B$, every ground atom $A : \alpha = p^\alpha(c_1, \ldots, c_n, \alpha)$ is true (when $\alpha = v_B(p(c_1, \ldots, c_n))$) or false. Let EMV denote this new 2-valued encapsulation of many-valued logic for logic programming, such that $\mathcal{L}_B$ is its sublanguage.

We assume that the Herbrand universe is $\Gamma_U = \Gamma \cup \Omega$, where $\Gamma$ is ordinary domain of database constants, and $\Omega$ is an infinite enumerable set of marked null values, $\Omega = \{\omega_0, \omega_1, \ldots\}$, and for a given logic program $P$, the Herbrand base, $H_P$, is the set of all ground (i.e., variable free) atoms in $\mathcal{L}_B$. A (ordinary) Herbrand interpretation for any bilattice $\mathcal{B}$ (thus for $\mathcal{B}_4$ also) is a many-valued mapping $I : H_P \rightarrow \mathbb{B}$.

The syntax of the language EMV (in [19]) is similar to that of predicate calculus.

Definition 3. (Annotation [19]) Let $P$ be an ‘object’ many-valued logic program. An annotation for an atom $A = p(x_1, \ldots, x_n)$, of $P$, is $\alpha = \kappa_P(x_1, \ldots, x_n)$ given by a function $\kappa_P : F^{\text{arity}(P)} \rightarrow \mathcal{B}$. The translation in the annotated syntax version is as follows:

1. If $A$ is a positive literal in $P$, $A \Rightarrow A : \alpha = p^\alpha(x_1, \ldots, x_n, \alpha)$;
2. If $\sim A$ is a negative literal in $P$, $A \sim \Rightarrow A : \sim \alpha = p^\alpha(x_1, \ldots, x_n, \sim \alpha)$;

where ‘$\sim$’ is truth negation in a bilattice $\mathcal{B}$. We define also an inverse mapping for atoms, which forgets annotations, such that for any $A : \alpha$, $f_{\sim}(A : \alpha) = A$ and $f_\alpha(A : \sim \alpha) = \sim A$ : its extension to all annotated formulas is denoted by $I_{\sim}$.

When $\alpha$ is a bilattice constant, then $A : \alpha$ is constant-annotated (c-annotated), otherwise it is variable-annotated (v-annotated).

3. Each clause in $P$, $A \leftarrow B_1, \ldots, B_n$, is translated into a clause $A : \alpha \leftarrow B_1 : \beta_1, \ldots, B_n : \beta_n$, with the head, $A : \alpha$, and the body, $B_1 : \beta_1, \ldots, B_n : \beta_n$.

A strictly ground instance of clause is any ground instance of clause which contains only c-annotations.

By this translation of the ‘object’ logic program $P$ we obtain an annotated logic program, $P^A$, with its annotated Herbrand base.
\[ H_P^B = \{ A : \alpha \mid A \text{ is a ground atom of a program } P \text{ and } \alpha \in B \} \]
\[ = \{ p^A(c_1, \ldots, c_n, \alpha) \mid p(c_1, \ldots, c_n) \in H_P \text{ and } \alpha \in B \} \]

Note, that with the transformation of the original ‘object’ logic program \( P \) into its annotated ‘meta’ version program \( P^A \) we obtain always positive logic program; thus the syntax for the annotated logic does not need negation at the ‘meta’ (annotated) level and the annotated logic programs have the unique minimal Herbrand models.

An annotated Herbrand interpretation (a-interpretation) of \( P^A \) is a 2-valued mapping \( I^A : H_P^B \rightarrow \{ t, f \} \). We denote by \( \{ t, f \} H_P^A \) the set of all a-interpretations from \( H_P^B \) into \( \{ t, f \} \), and by \( B^{HR} \) the set of all consistent Herbrand many-valued interpretations, from \( H_P \) to the bilattice \( B \).

It is important, that if assume that all facts (ground atoms) of a database are true, then we possibly obtain locally inconsistent database, i.e., a database without consistent models. The way to overcome this problem, as we will see in the example 1, is to relativize this to much strong assumption, by the weaker assumption where logical values of facts can be true or possible. As we will see, the way to obtain such weaker assumption in the annotated logic program version (and explains why it is so useful), is to use ‘variables’ (by function symbols \( \bullet \)) also for ground facts; the assignment of the true or possible value to such function symbols, for a given ground fact, will be left to the immediate consequence operator of the annotated program, during the calculation of its minimal Herbrand model.

We are ready now to define the semantics of encapsulation of many-valued logic programs into 2-valued logic programs.

**Definition 4.** The encapsulation of an ‘object’ logic program \( P \) into an annotated ‘meta’ program \( P^A \) means that, for any consistent many-valued Herbrand interpretation \( I \in B^{HR} \), the interpretation of the function symbols \( \kappa_p \) is compatible to it, i.e. for any tuple \( c \in \Gamma^U_{\text{arr}(p)} \), \( \kappa_p(c) = I(p(c)) \).

The encapsulation defines a mapping from many-valued Herbrand interpretations into a-interpretations, \( \Theta : B^{HR} \rightarrow \{ t, f \} H_P^A \), such that for each Herbrand interpretation \( I \in B^{HR} \) of \( P \), we obtain the a-interpretation, \( I^A = \Theta(I) \in \{ t, f \} H_P^A \) of \( P^A \), as follows: for any \( c \)-annotated ground atom \( A : \alpha \), \( \Theta(I)(A : \alpha) = t \), if \( \alpha = I(A) : f \) otherwise.

The subset, \( \text{Imm} \Theta \subseteq \{ t, f \} H_P^A \), of a-interpretations with such a property are denominated encapsulated (c-interpretations).

Try to resume now what really is an encapsulated ‘meta’-logic program \( P^A \) obtained by the original ‘object’ logic program \( P \): it is a simple transformation when each original atom \( p(x_1, \ldots, x_n) \) replaced by the its annotated predicate version \( p^A(x_1, \ldots, x_n, \kappa_p(x_1, \ldots, x_n)) \) with the introduction of a family of functional symbols \( \kappa_p : \Gamma^U_{\text{arr}(p)} \rightarrow B \). The meaning of the encapsulation of this ‘object’ logic program \( P^A \) into this ‘meta’ logic program \( P^A \) is fixed into the kind of interpretation to give to such new introduced functional symbols: in fact we want that they represent (encapsulate) the semantics of the ‘object’ level logic program \( P \).

**Definition 5.** (Satisfaction \( [19] \)) Let \( I : H_P^A \rightarrow B \) be a many-valued Herbrand interpretation of a program \( P \) and \( v_B \) its valuation extension to all ground formulas.
Let \( g \) be a variable assignment which assigns values from \( \Gamma_U \) to object variables. We extent it to atoms with variables, so that \( g(p(x_1,\ldots,x_n)) = p(g(x_1),\ldots,g(x_n)) \). Then all annotated formulas in the usual way: \( \psi / g \) denotes a ground formula obtained from \( \psi \) by assignment \( g \), then

1. \( I^A \models_g \psi \) iff \( \kappa_p \) is compatible to \( I \), i.e., \( I^A = \Theta(I) \).
2. \( I^A \models_g \phi \land \psi \) iff \( I^A \models_g \phi \) and \( I^A \models_g \psi \).
3. \( I^A \models_g \phi \lor \psi \) iff \( I^A \models_g \phi \) or \( I^A \models_g \psi \).
4. Standard implication: \( I^A \models_g \kappa_p(g(x_1),\ldots,g(x_n)) \iff I^A \models \kappa_p(x_1,\ldots,x_n) \).

Inconsistency acceptance: if \( p \) is a built-in predicate, this clause is satisfied also when \( \kappa_p(g(x_1),\ldots,g(x_n)) \leq_k v_B(\mathcal{F}_p/\psi) \).
define the integrity constraints we can have more than one minimal Herbrand model of such programs. Let us see the example below where we use the key integrity constraint in the virtual database of a simple data integration system and such integrity constraints is not satisfied by source databases.

**Example 1:** The built-in predicates (ex. =, ≤, ≥, ..) may be used for integrity constraints: let \( P(x, y) \) be a predicate for a source database relation and we define the key-constraint for attributes in \( x \). The clause of a program for such key constraint is given by \( (y = z) \leftarrow P(x, y), P(x, z) \), where the atom \( y = z \) is based on the built-in predicate ‘\( = \)’. Let consider a program \( P \), and its translation \( P^A \).

<table>
<thead>
<tr>
<th>Program ( P )</th>
<th>Encapsulated program ( P^A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r(a, b) \leftarrow t )</td>
<td>( r(a, b) : t \leftarrow t )</td>
</tr>
<tr>
<td>( r(a, c) \leftarrow t )</td>
<td>( r(a, c) : t \leftarrow t )</td>
</tr>
<tr>
<td>( p(x, y) \leftarrow r(x, y) )</td>
<td>( p(x, y) : s_3 \leftarrow r(x, y) : s_4 )</td>
</tr>
<tr>
<td>( (y = z) \leftarrow p(x, y), p(x, z) )</td>
<td>( (y = z) : s_5 \leftarrow p(x, y) : s_6, p(x, z) : s_7 )</td>
</tr>
</tbody>
</table>

where \( r \) is a source database relation, and \( p \) is a virtual relation of this database with key constraint, \( x, y, z \) are object variables and \( s_i, i = 3, \ldots, 7 \) annotation variables. The built-in predicates have the same prefixed extension in all models of a logic program, and that their ground atoms are true or false. In the extensional part \( r \) of this database we have two tuples, \( (a, b), (a, c) \): such facts are mutually inconsistent for \( p \) because of key constraint, thus only one of them may be true in any stable model of this logic program. In fact, for \( g(x) = a, g(y) = b, g(z) = c \), thus \( s_5 = \kappa_m(b, c) = f \); let \( s_6 = \kappa_p(g(x), g(y)) = t \) and \( s_7 = \kappa_p(g(x), g(z)) = t \), then \( v_B(p(a, b) \land p(a, c)) = t \), and from \( E_4 \) \( f \not\leq t \); thus we have that \( I^A \not\models (y = z) : s_5 \leftarrow p(x, y) : s_6, p(x, z) : s_7 \) for such assignment \( g \). But if we assign the ‘possible’ value, \( s_7 = \kappa_p(g(x), g(z)) = \top \), then \( v_B(p(a, b) \land p(a, c)) = \top \) and the clause is satisfied, \( I^A \models (y = z) : s_5 \leftarrow p(x, y) : s_6, p(x, z) : s_7 \).

Note that in this way, for this simple data integration of a source database \( r \) into a virtual schema \( p \) with integrity key constraint, we can’t use the query rewriting: In fact the same tuple \( (a, c) \) in virtual predicate \( p \) has possible logic value, while in the source relation \( r \) it is true. In order to be able to answer to queries using rewriting technics, we have to assign to each ground fact (source database) the variable annotation, i.e., \( r(a, b) : s_1 \leftarrow t \) and \( r(a, c) : s_2 \leftarrow t \) in the example above. In that case, for such program we are able to assign them true or possible value also (in our billatice holds \( \tau = t \)). Thus, we can, as first attempt, assign to a ground fact the value \( t \) (true); if such assignment does not satisfy this program, then we can assign the possible value \( \tau \), in order to overcome mutually-inconsistent source data. In such an approach, the tuples in the virtual and in the source database have the same logical value, thus, we can use the query-rewriting technics, which rewrite the original query on the virtual database into the equivalent query on source databases, to answer to user queries.

It is easy to see, that the obtained minimal Herbrand model of the program above is not unique. In fact there are other two models, one where \( s_6 = \kappa_p(g(x), g(y)) = \tau \), and \( s_7 = \kappa_p(g(x), g(z)) = t \), the other when \( s_6 = \kappa_p(g(x), g(y)) = \tau \) and \( s_7 = \kappa_p(g(x), g(z)) = \tau \). The approach described above will give, for any set of mutually-inconsistent facts, only one true and all other possible facts. The natural way to obtain an unique canonical model which maximize the knowledge, is such where all facts, in
some set of mutually-inconsistent facts, are forced to have the possible logic value (the last of the three models above), because there is no any other rule to declare some of them true and all other possible only.

4 The trilattice of EMV interpretations

Why we need a trilattice? Is it not enough the bilattice of Herbrand interpretations (as in Fitting) in order to consider the models for annotated ‘meta’ logic programs? Well, it is well known that the semantics of ‘object’ many-valued logic programs with negations can be defined as a least fixpoint w.r.t. the knowledge ordering in the bilattice of Herbrand interpretations (for positive logic programs it can be defined as least fixpoint w.r.t. the truth ordering also). What we have now to investigate is which ordering in the EMV Herbrand interpretations of encapsulated logic programs we need to define the least fixpoint as its semantics. The answer comes directly from Proposition 1: encapsulated (annotated) logic programs, with negation also, are always positive programs.

Let call meta-truth ordering the truth ordering for annotated programs (to distinct it from the truth ordering at ‘object’ level): from the fact that the annotated programs are syntactically ordinary positive logic programs we conclude that their semantics can be defined as a least fixpoint w.r.t. the meta-truth ordering.

Thus, we need to enrich the original ‘object’ level bilattice also with this 3-th meta-truth ordering in order to obtain a trilattice (for more details see [19]).

Any a-interpretation \( I^A : H^A \rightarrow \{t, f\} \) of an annotated logical program \( P^A \) may be equivalently represented by the set of true annotated atoms, \( m^A =_{df} \{ A : \alpha \mid I^A(A : \alpha) = t \} \in \mathcal{P}(H^A) \), where \( \mathcal{P}(H^A) \) is the set of all subsets of \( H^A \). In the rest of this section we will prove that the set of all a-interpretations of a program \( P^A \) is a complete trilattice with an embedded infinitely distributive bilattice.

**Proposition 2** Given an encapsulated logic program \( P^A \) over a many-valued bilattice \( B \), with \( \alpha, \beta \in B \), then there is a family of bilattice vector-functions \( f_\alpha : \mathcal{P}(H_P) \rightarrow \mathcal{P}(H^A) \), such that for any \( m \in \mathcal{P}(H_P) \), \( f_\alpha(m) =_{df} \{ A : \alpha \mid A \in m \} \), and there is a family of projection vector-functions \( f_\beta : \mathcal{P}(H^A) \rightarrow \mathcal{P}(H_P) \), such that for any \( m^A \in \mathcal{P}(H^A) \), \( f_\beta(m^A) =_{df} \{ A : \beta \mid A \in m^A \} \), which are orthogonal and idempotent.

Let us denote by \( f_P : \mathcal{P}(H^A) \rightarrow \mathcal{P}(H_P) \) the reduction function such that for any \( m^A \), \( f_P(m^A) = \{ A : \alpha \in m^A \} \), and by \( h_P : \mathcal{P}(H_P) \rightarrow \mathcal{P}(H^A) \) the constant function, such that for any \( m^A \), \( h_P(m^A) = H_P = f_P(H_P) \).

In order to give an intuitive understanding of the formal definition of a trilattice, let forget for the moment the unknown atoms and consider any \( m^A \) as a triple of sets \( [T, P, F] \), with \( T = f_\theta^A(m^A) \) true, \( P = f_\delta^A(m^A) \) possible, and \( F = f_j^A(m^A) \) false atoms in \( H_P \). Then given two a-interpretations \( m^A, m^A_1 \), represented by triples \( [T, P, F] \) and \([T_1, P_1, F_1] \) respectively, we can introduce the following three preorders:

1. Set inclusion lattice preorder (truth preorder at ‘meta’ level), \( m^A \subseteq m^A_1 \).
2. Truth lattice preorder, \( m^A \leq^A_1 m^A \) \iff \( T \subseteq T_1 \) and \( F \supseteq F_1 \).
3. Knowledge lattice preorder, \( m^A \leq^A_k m^A \) \iff \( T \subseteq T_1, P \subseteq P_1 \) and \( F \subseteq F_1 \).
Now we can introduce the following lattice operators (meet, join and negation):
1. Set intersection, set union and set complement, for the inclusion lattice preorder.
2. For the truth preorder ('object' level) lattice:
   \[ [T, P, F] \land [T_1, P_1, F_1] =_{df} [T \cap T_1, P \cap P_1, F \cup F_1], \]
   \[ [T, P, F] \lor [T_1, P_1, F_1] =_{df} [T \cup T_1, P \cap P_1, F \cap F_1], \]
   and
   \[ \lnot [T, P, F] =_{df} [F, P, T], \]

3. For the knowledge preorder ('object' level) lattice:
   \[ [T, P, F] \otimes [T_1, P_1, F_1] =_{df} [T \cap T_1, P \cup P_1, F \cap F_1], \]
   \[ [T, P, F] \odot [T_1, P_1, F_1] =_{df} [T \cup T_1, P \cap P_1, F \cup F_1], \]
   \[ \lnot [T, P, F] =_{df} [T, P, F], \]
   where \( S \) denotes set complement.

In this simplified framework, when unknown atoms are neglected, we obtain that knowl-
edge and set-inclusion preorders and operators coincide, but in the general framework,
described below, that is not the case:

**Definition 7.** (Trilattice) If \( P^A \) is an annotated logic program based on a 4-valued bi-
lattice \( B_A \), then the trilattice \( B^A =_{df} \langle \mathcal{P}(H^A), \subseteq, \preceq, \preceq \rangle \) of all \( a \)-interpretations is
defined as follows, for \( S = \{ t, \top, f \} \). \( S_F = \{ \neg\neg f, \neg f, f, f \} \) and for any \( m^A_1, m^A_2 \in \mathcal{P}(H^A) \):

1. The EMV meta-truth partial ordering, \( \subseteq \), is the set inclusion, with meet and join operators, \( \cap \) (set intersection) and \( \cup \) (set union), respectively. The strong equivalence \( \equiv \), is the set equivalence, \( \backslash \) is the set difference operator. Empty set, \( m^A_\bot \), of if its bot-
tom element, and \( m^A_\top = H^A \) is its top element.

2. The many-valued truth partial ordering \( \preceq^A \)
   \[ m^A_1 \preceq^A m^A_2 \iff \neg\neg f (m^A_1) \subseteq f (m^A_2) \quad \text{and} \quad f (m^A_1) \supseteq f (m^A_2). \]
   1. The meet and join operators w.r.t. the \( \preceq^A \) ordering
   \[ m^A_1 \land m^A_2 =_{df} f (m^A_1 \cap m^A_2) \cup \neg\neg f (m^A_1) \cup \neg\neg f (m^A_2). \]
   \[ m^A_1 \lor m^A_2 =_{df} f (m^A_1 \cup m^A_2) \cup \neg\neg f (m^A_1) \cup \neg\neg f (m^A_2). \]

2. The truth negation operator
   \[ \lnot^A =_{df} \bigcup_{g \in S_F} g (m^A). \]

3. The many-valued knowledge partial ordering \( \preceq^A \)
   \[ m^A_1 \preceq^A m^A_2 \iff \neg\neg f (m^A_1) \subseteq f (m^A_2), \]
   for all \( a \in S \).

2.1 The meet and join operators w.r.t. the \( \preceq^A \) ordering are
   \[ \otimes =_{df} \cap \subseteq \cap \quad \text{and} \quad \odot =_{df} \cup \subseteq \cup \]

3.2 The knowledge negation operator w.r.t. truth partial ordering
   \[ \lnot^A =_{df} \bigcup_{a \in S} (h_P \setminus f_P \lnot a (m^A)). \]

Empty set, \( m^A_\bot \), of its bottom element, and each \( m^A \), such that \( \lnot^A \]

4. The unary operations, \( n_\bot \) (the change the truth in possibility), and \( p_\bot \) (change the falseh-
hood in possibility), are defined as follows: for any \( m^A \in B^A \)

\[ n_\bot (m^A) =_{df} \bigcup_{a \in S_1} g (m^A), \]
\[ p_\bot (m^A) =_{df} \bigcup_{a \in S_2} g (m^A), \]
with
\[ n_\bot (\bot) = \bot, \quad n_\bot (f) = n_\bot (\top) = f, \quad p_\bot (\bot) = p_\bot (\top) = \bot, \quad p_\bot (f) = p_\bot (\bot) = \bot. \]
There is also another weak equivalence in a trilattice $B^A$, generated by two introduced many-valued orderings.

**Definition 8.** The trilattice weak equivalence relation "\(\sim\)" is defined by: \(m^A \sim m^A\) iff \(m^A_i =_l m^A_j\) and \(m^A_k =_l m^A_l\). This equivalence determines equivalence classes of members in $B^A$. In each equivalence class of members \(\{m^A_1, \ldots, m^A_n\}\), with \(m^A_i \approx m^A_j\) for any \(1 \leq i, j \leq n\), the subset of ground atoms annotated by \(\bot\) of each member \(m^A_i\) is left to be free.

1. We define a representative member of any equivalence class, a member with \(\overline{f} = \overline{f}(m^A) = \overline{f}_L(h_P \setminus f_P(\bigcup_{a \in \{t, f, \top\}} f^A_a))\). We also introduce the function \(f_R\) which for any \(m^A\), returns with its representative:

\[
m^A \approx f_R(m^A) = \overline{f}_L(h_P \setminus f_P(\bigcup_{a \in S} (\bigcap_{\{t, f, \top\}} f^A_a)))\.
\]

2. We define minimal members \(m^A \in B^A\), such that \(\overline{f}_L = \{\}\); it is easy to see that each trilattice operator, different from true meet and join (\(\cap\) and \(\cup\)), returns with a minimal member.

3. We define minimal consistent members \(m^A \in B^A\), such that they are minimal and \(\bigcap_{a \neq \beta} f^A_a \cap f^A_\beta(m^A) = \{\}\).

It is easy to verify that a representative \(f_R(m^A)\), of any minimal consistent \(m^A\), is a $e$-interpretation such that \(m^A \subseteq f_R(m^A)\).

**Proposition 3** \(\langle P(H^A), \leq_t, \leq_k, \leq_\bot \rangle\) is a complete infinitarily distributive bilattice: its knowledge negation, \(\sim\), coincides with the truth negation of the truth ordering \(\leq_\bot\). Thus, \(\sim\) reverses the \(\leq_t\) ordering, while preserving the \(\leq_k\) and \(\leq_\bot\) orderings; \(\sim\) reverses the \(\leq_k\) and \(\leq_\bot\), while preserving the \(\leq_t\) ordering.

We have seen (Definition 3) that each consistent many-valued Herbrand interpretation \(I : H_P \rightarrow B^A\) determines a consistent 2-valued $a$-interpretation \(m^A \in B^A\)(i.e., \(I^A = \Theta(I) : H^A_P \rightarrow \{t, f\}\)), but not vice versa; only members \(m^A\) in a bilattice of $e$-interpretations, of an encapsulated program $P^A$, correspond to a consistent many-valued Herbrand interpretations of a program $P$.

Now we will define a sublattice of $B^A$ which corresponds to consistent many-valued interpretations \(I : H_P \rightarrow B^A\) of a program $P$.

Recall that each representative member \(m^A\) is not partial w.r.t. the Herbrand base, i.e., that \(f_P(m^A) = H_P\), and that such completeness is obtained by assigning the default value unknown, \(\bot\), to all remaining ground atoms.

**Proposition 4** (Consistency) If $P^A$ is an annotated logic program based on a 4-valued bilattice $B^A$, and $B^A = f_P(\langle P(H^A_P), \leq_t, \leq_k, \leq_\bot \rangle)$ is the trilattice of all its $a$-interpretations with $H_P = f_P(\langle H^A_P \rangle)$, then for any \(m^A \in B^A\)

1. if \(m^A =_A \neg m^A\), then it is representative member and a 2-valued consistent (or "exact") Herbrand interpretation \(I : H_P \rightarrow \{t, f\}\), \((t, f, \top\) are exact). For any \(m^A\), the member \(m^A \vee \neg m^A\) is "exact".

2. if \(m^A \leq_\bot \neg m^A\), then its representative is a 3-valued consistent Herbrand interpretation \(I : H_P \rightarrow \{t, f, \bot\}\), \((t, f, \bot\) are 3-valued consistent).
3. if \( \nu_4(m^A) \oplus p_2(m^A) \leq_k^A -m^A \) then its representative is a 4-valued consistent Herbrand interpretation \( \Gamma : P \rightarrow \mathcal{B}_k \). All values in \( \mathcal{B}_k \) are 4-valued consistent.

Proposition 5 The 4-valued consistent members in \( \mathcal{B}^A \), i.e., \( e \)-interpretations, are closed under \( \sim \), \( \otimes \), and under \( \wedge \), \( \vee \) and their infinitary versions. The 4-valued consistent members, or \( e \)-interpretations, constitute a complete semilattice under \( \leq_k^4 \). \( \mathcal{B}_C^4 \) = \{ \( \mu(m^A) \mid \nu_4(m^A) \oplus p_2(m^A) \leq_k^A -m^A \) \}, being closed under \( \Pi \) and under directed \( \Sigma \) (applied to directed sets).

The Knaster-Tarski like theorem for monotonic functions in a complete sublattice is partially valid (the greatest fixpoints may not exist as in complete lattices): the least fixpoint exists for each monotonic function (‘truth revision’) w.r.t. \( \leq_k^A \) ordering. All annotated logic programs have a nice property: their Herbrand instantiation are positive annotated logic programs w.r.t. \( e \)-interpretations; such \( e \)-interpretations define a complete semilattice, \( \mathcal{B}^A \) under \( \leq \). Such a property, analogously as for ordinary positive logic programs, induces that the monotonic ‘immediate consequence operator’, \( T_P \), of a program \( P^4 \) has the least fixpoint (the least fixpoint correspond to the least annotated Herbrand model of \( P^4 \)). In [20] is given the coalgebraic semantics for such logic programs also, while in [21] is given the example for fixpoint semantics in data integration systems with infinite canonical Herbrand models.

5 Conclusion

We have presented a programming logic capable of handling inconsistent beliefs, based on the 4-valued Belnap’s bilattice, which has clear model theory and fixed point semantics. In the process of the encapsulation we distinguish two levels: the encapsulated or ‘object’ many-valued level of ordinary logic programs with epistemic negation based on a bilattice operators (without implication operator), and the annotated or ‘meta’ level of encapsulated logic programs also with implication operator. In this approach, ‘inconsistent’ logic programs (which minimal stable models contain at least an ‘inconsistent’ ground atom) at object level are also classic consistent logic programs at ‘meta’ (annotated) level. In such an abstraction we obtain some kind of a minimal Annotated Predicate Logic where fixpoint ‘immediate consequence’ operator is always continuous, and which is computationally equivalent to standard Fitting’s fixpoint semantics. A clear contribution, in an example, of built-in predicates is also given in order to manage local inconsistencies.

A contribution from the theoretical point of view is given in understanding why (when we pass from a 2-valued to many-valued logic programming) we pass from truth to knowledge ordering. The knowledge ordering at ‘object’ many-valued level is homomorphic to the truth ordering at ‘meta’ level of logic programming.

References


5. T. Przymusinski, “Every logic program has a natural stratification and an iterated fixed point model,” *In Eighth ACM Symposium on Principles of Databases Systems*, pp. 11–21, 1989.


