Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 225 (2009) 141–159
www.elsevier.com/locate/entcs

Sound and Complete SLD-Resolution for Bilattice-Based Annotated Logic Programs

Ekaterina Komendantskaya [1,2]

Department of Mathematics
University College Cork
Cork, Ireland

Anthony Karel Seda[1,3]

Department of Mathematics
University College Cork
Cork, Ireland

Abstract

We introduce the class of normal bilattice-based annotated first-order logic programs (BAPs) and develop declarative and operational semantics for them. In particular, SLD-resolution for these programs is defined and its soundness and completeness established.

Keywords: Annotated logic programming, bilattices, declarative and operational semantics, SLD-resolution.

1 Introduction

Since their introduction by Ginsberg [9], bilattices have become a well-known algebraic structure for reasoning about the sort of inconsistencies which arise when one formalises the process of accumulating information from different sources. In particular, Fitting [6,7,8] introduced quite general consequence operators for logic programs whose semantics are based on four-valued bilattices, and derived their basic properties.

Annotated (sometimes called signed) languages are an alternative formal tool for handling, for example, the semantics of logic programs over many-valued logics

[1] The authors thank the Boole Centre for Research in Informatics at University College Cork for substantial support during the preparation of this paper.

[2] Email: komendantskaya@gmail.com

[3] Email: a.seda@ucc.ie

doi:10.1016/j.entcs.2008.12.071

and probabilistic programs, see [4,11,15,23,26,31]. Their use, however, gives rise to the obvious question of how annotated logic programming compares with logic programming based on (bi)lattices, see [14,15,21,24,25,28]. In this paper, we contribute to this discussion by combining the two approaches in that we make use of bilattice structures within the framework of annotated logic programs. Specifically, we introduce bilattice-based annotated logic programs (BAPs) and establish declarative and operational semantics for them. BAPs, being many-valued and quantitative in nature, enable us to work with statistical knowledge, and with databases which can be incomplete or inconsistent, and thereby introduce monotonic extensions of two-valued negation. But what is particularly important is that they possess all the desirable properties of classical logic programming. Thus, we define monotonic and continuous semantic operators for BAPs and develop sound and complete proof procedures for them based on classical binary resolution.

The results which we obtain here can be seen as a further development of lattice-based logic programming for handling inconsistencies, see [4,8,14,15,26,28], for example. However, there are several original results obtained in the paper, as follows.

First of all, we show that logic programs based on bilattices (and on lattices whose elements are not linearly ordered) may have logical consequences which cannot be computed by the semantic operators defined in the papers just cited. In fact, most authors who have considered lattice- and bilattice-based logic programs follow the tradition of classical logic programming and require their semantic operators to work by propagating interpretation from the body of each clause to its head. However, the authors of [14] extended and enriched resolution for lattice-based logics by adding new rules reflecting the non-linear ordering of their lattices. This work distinguished proof procedures for lattice-based logics from those for many-valued logics interpreted by linearly-ordered structures, for example, fuzzy logics. The work of [14] inspired us to extend these ideas to logic programs and their semantic operators, and to SLD-resolution. As a result, we define a new semantic operator which guarantees computation of all the logical consequences of a bilattice-based annotated logic program. Unlike the case of [14], we allow infinite (but countable) interpretations for logic programs.

Throughout, we work with first-order bilattice-based logic programs interpreted by arbitrary (possibly infinite) distributive bilattices with finite joins, and this framework considerably enriches that of propositional-based logic programs [4,21,26,28] based on finite sets or finite lattices [2,10,14,23,24,25,31]. Moreover, we allow annotations to be variables or to contain function symbols unlike, for example, the case of [2,10,14,23,24,25,31].

We use the fact proven in [20] that the semantic operator we introduce is continuous. As usual, continuity of the semantic operator ensures that it reaches its least fixed point in at most ω iterations. This property is crucial for the whole theory. It makes possible computer implementations of the operator and it also makes it possible to introduce sound and complete proof procedure for BAPs. Note that the semantic operators introduced earlier for annotated logic programs of this generality do not possess this important property, see [4,15].

Finally, we establish sound and complete SLD-resolution for BAPs. As far as we know, this is the first sound and complete proof procedure for first-order infinitely interpreted (bi)lattice-based annotated logic programs. Compare, for example, our results with those obtained for constrained resolution for GAPs, which was shown to be incomplete, see [15], or with sound and complete (SLD)- resolutions for finitely-interpreted annotated logic programs (these logic programs do not contain annotation variables and annotation functions) [2,10,14,23,24,25,31]. [4]

Thus, in summary, we fully describe a broad class of first-order, infinitely-valued, bilattice-based annotated logic programs and propose suitable proof procedures and implementations for them.

The structure of the paper is as follows. In §2, we describe bilattices, closely following Ginsberg [9] in our presentation. In §3, we give full details of the syntax and semantics of bilattice-based languages, and of BAPs in particular. We also develop a declarative semantics for BAPs in §3. In §4, we define SLD-resolution for BAPs and prove its soundness and completeness. Finally, in §5, we summarise our results and discuss further work yet to be done.

2 Bilattices

In this section, we briefly describe bilattices, following closely Ginsberg [9] and Fitting [6,7,8].

Definition 2.1 A *bilattice* $\mathbf{B}$ is a sextuple $(\mathbf{B}, \vee, \wedge, \oplus, \otimes, \neg)$ such that $(\mathbf{B}, \vee, \wedge)$ and $(\mathbf{B}, \oplus, \otimes)$ are both complete lattices, and $\neg : \mathbf{B} \to \mathbf{B}$ is a mapping satisfying the following three properties:

- $\neg^2 = \mathrm{Id}_{\mathbf{B}}$,
- $\neg$ is a dual lattice homomorphism from $(\mathbf{B}, \vee, \wedge)$ to $(\mathbf{B}, \wedge, \vee)$, and
- $\neg$ is a lattice homomorphism from $(\mathbf{B}, \oplus, \otimes)$ to itself.

Certain kinds of bilattices can be obtained by taking a product of two lattices, and this property makes it easy to describe and investigate these structures.

Let $L_1 = (\mathcal{L}_1, \leq_1)$ and $L_2 = (\mathcal{L}_2, \leq_2)$ be two lattices, let x_1, x_2 denote arbitrary elements of the lattice L_1, and let y_1, y_2 denote arbitrary elements of the lattice L_2. Let $\cap_1, \cup_1$ denote the meet respectively join defined in the lattice L_1, and let $\cap_2, \cup_2$ denote the meet respectively join defined in the lattice L_2.

Two familiar bilattice orderings can be defined on $\mathcal{L}_1 \times \mathcal{L}_2$. We define the *truth* ordering $\leq_t$ and the *knowledge* ordering $\leq_k$ as follows.
(1) $\langle x_1, y_1 \rangle \leq_t \langle x_2, y_2 \rangle$ if and only if $x_1 \leq_1 x_2$ and $y_2 \leq_2 y_1$.
(2) $\langle x_1, y_1 \rangle \leq_k \langle x_2, y_2 \rangle$ if and only if $x_1 \leq_1 x_2$ and $y_1 \leq_2 y_2$.

[4] Note also that [28] introduced computer programs which compute consequence operators for (propositional) bilattice-based logic programs. However, this work cannot be seen as a proof procedure for the simple reason that the semantic operator of [28] is capable only of producing sets of logical consequences of a program. But we look for a proof procedure that can respond to certain goals and give correct answers together with proper substitutions for individual and/or annotation variables.

We denote the resulting structure by $L_1 \odot L_2 = (\mathcal{L}_1 \times \mathcal{L}_2, \leq_t, \leq_k) = (\mathbf{B}, \leq_t, \leq_k)$, where $\mathbf{B}$ denotes $\mathcal{L}_1 \times \mathcal{L}_2$.

Having defined $L_1 \odot L_2$, we define bilattice operations on it as follows. The four bilattice operations associated with $\leq_t$ and $\leq_k$ are:

$$\langle x_1, y_1\rangle \wedge \langle x_2, y_2\rangle = \langle x_1 \cap_1 x_2, y_1 \cup_2 y_2\rangle,$$
$$\langle x_1, y_1\rangle \vee \langle x_2, y_2\rangle = \langle x_1 \cup_1 x_2, y_1 \cap_2 y_2\rangle$$
$$\langle x_1, y_1\rangle \otimes \langle x_2, y_2\rangle = \langle x_1 \cap_1 x_2, y_1 \cap_2 y_2\rangle$$
$$\langle x_1, y_1\rangle \oplus \langle x_2, y_2\rangle = \langle x_1 \cup_1 x_2, y_1 \cup_2 y_2\rangle.$$

Negation is defined as follows: $\neg\langle x_1, y_1\rangle = \langle y_1, x_1\rangle$; this latter definition assumes that $L_1 = L_2$.

Similarly, we can define infinite meet and join with respect to the t- and k-orderings, and we denote them respectively by $\bigwedge$, $\bigvee$, $\prod$, and $\sum$.

Proposition 2.2 *[6,9] Suppose* $\mathbf{B}$ *is a distributive bilattice. Then there are distributive lattices* L_1 *and* L_2 *such that* $\mathbf{B}$ *is isomorphic to* $L_1 \odot L_2$.

Thus, every distributive bilattice can be represented as a product of two lattices. In the present article, we consider only logic programs over distributive bilattices and therefore the underlying bilattice of any program we consider will always be formed as a product of two lattices.

Example 2.3 Consider the bilattice $\mathbf{B}_{25} = L_1 \odot L_2$, where $L_1 = L_2 = (\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}, \leq)$, with $0 \leq \frac{1}{4} \leq \frac{1}{2} \leq \frac{3}{4} \leq 1$. The set $\mathcal{B}_{25} = \mathcal{L}_1 \times \mathcal{L}_2$ contains the following elements: $\langle 0,0\rangle, \langle 0, \frac{1}{4}\rangle, \langle 0, \frac{1}{2}\rangle, \langle 0, \frac{3}{4}\rangle, \langle 0,1\rangle, \langle \frac{1}{4}, 0\rangle, \langle \frac{1}{4}, \frac{1}{4}\rangle, \langle \frac{1}{4}, \frac{1}{2}\rangle, \langle \frac{1}{4}, \frac{3}{4}\rangle, \langle \frac{1}{4}, 1\rangle, \langle \frac{1}{2}, 0\rangle, \langle \frac{1}{2}, \frac{1}{4}\rangle, \langle \frac{1}{2}, \frac{1}{2}\rangle, \langle \frac{1}{2}, \frac{3}{4}\rangle, \langle \frac{1}{2}, 1\rangle, \langle \frac{3}{4}, 0\rangle, \langle \frac{3}{4}, \frac{1}{4}\rangle, \langle \frac{3}{4}, \frac{1}{2}\rangle, \langle \frac{3}{4}, \frac{3}{4}\rangle, \langle \frac{3}{4}, 1\rangle, \langle 1, 0\rangle, \langle 1, \frac{1}{4}\rangle, \langle 1, \frac{1}{2}\rangle, \langle 1, \frac{3}{4}\rangle, \langle 1, 1\rangle$. The two orderings $\leq_k$ and $\leq_t$ are determined according to the definition of $L_1 \odot L_2$, see Figure 1.

Informally speaking, these pairs of numbers will allow us to formalise, in the next section, the degrees of belief and doubt we assign to statements. The first element of a pair will reflect the degree of belief, and the second element will reflect the degree of doubt.

3 Annotated Logic Programs Based on Bilattice Structures

In this section, we define annotated bilattice-based logic programs. Our definition extends previous definitions on this topic in various ways and, in particular, it extends annotated (or signed) languages (see [2,4,15,23,24,25,31] and others) to the case of distributive bilattices. The languages we introduce allow variables and functions in annotations, unlike [14,23,24,25] and others, for example; they generalize some of the propositional lattice-based languages [4,28] to the first-order case, and allow negations unlike, for example, [4,23] and others.

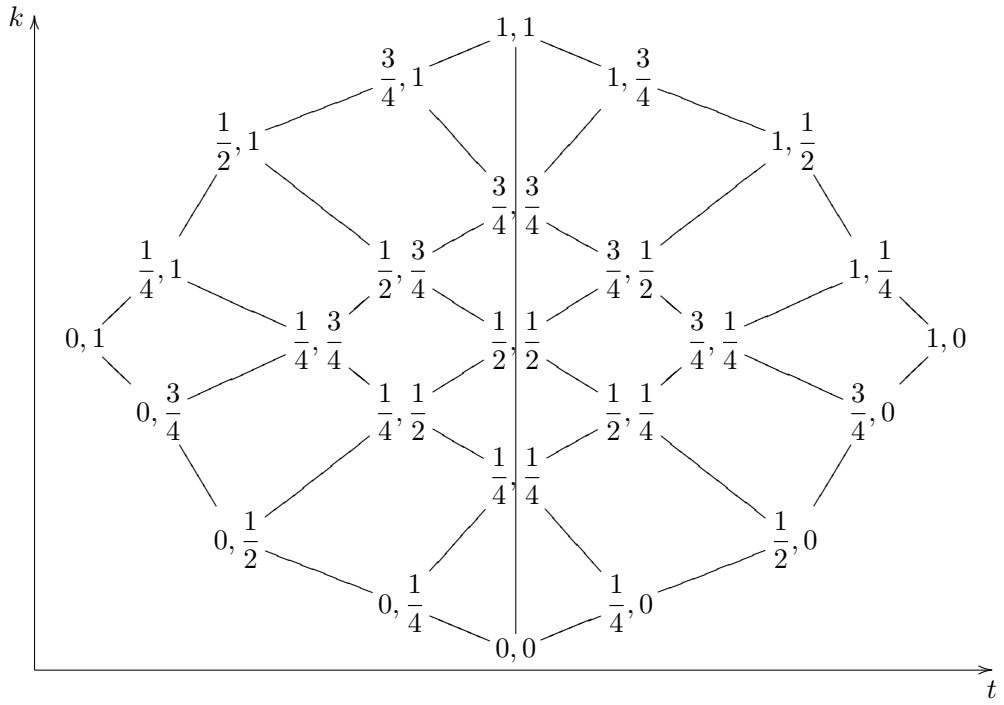

Fig. 1. Bilattice $\mathbf{B}_{25}$

Let $\mathbf{B} = L_1 \odot L_2$ denote a bilattice given as the product of two complete lattices L_1, L_2 each of which is a sublattice of the lattice $([0,1], \leq)$, where $[0,1]$ is the unit interval of real numbers and $\leq$ is the usual linear ordering on it. For the rest of the paper, we restrict our attention to bilattices with finite joins.

We define an annotated bilattice-based language $\mathcal{L}$ over $\mathbf{B}$ to consist of individual variables denoted $x_1, x_2, \ldots$, constants $a_1, a_2, \ldots$, functions $f_1, f_2, \ldots$ and predicate symbols $R_1, R_2, \ldots$ together with *annotation terms* which can consist of variables $(\mu_1, \nu_1), (\mu_2, \nu_2), \ldots$, constants $(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots$, and functions over annotation variables and constants. We allow six connectives and four quantifiers, as follows: $\oplus, \otimes, \vee, \wedge, \neg, \sim, \Sigma, \Pi, \exists, \forall$. The full fragment of bilattice-based annotated logic (BAL) is considered in [18]. Here, we restrict our attention only to Horn-clause fragments of it.

An *annotated formula* is defined inductively as follows: if R is an n-ary predicate symbol, $t_1, \ldots, t_n$ are terms, and (τ) is an annotation term, then $R(t_1, \ldots, t_n) : (\tau)$ is an *annotated formula* (called an *annotated atom*). Annotated atoms can be combined to form complex formulae using the connectives and quantifiers.

Example 3.1 Consider a binary predicate `connected`, which describes the fact of existence of an edge in a probabilistic graph. These graphs can be used to describe the behaviour of Internet connections, for example. Then $\texttt{connected}(a, b) : (\frac{1}{3}, \frac{2}{3})$ will describe the fact that the probability of establishing a connection between nodes

a and b is equal to $\frac{1}{3}$, while the probability of losing this connection is $\frac{2}{3}$.

A *bilattice-based annotated logic program (BAP)* P consists of a finite set of (annotated) *program clauses* of the form

$$A : (\tau) \leftarrow L_1 : (\tau_1), \ldots, L_n : (\tau_n),$$

where $A : (\tau)$ denotes an annotated atom called the *head* of the clause, and $L_1 : (\tau_1), \ldots, L_n : (\tau_n)$ denotes $L_1 : (\tau_1) \otimes \ldots \otimes L_n : (\tau_n)$ and is called the *body* of the clause; each $L_i : (\tau_i)$ is an annotated literal called an *annotated body literal* of the clause.

By allowing each $L_i : (\tau_i)$ to be a literal, we assume that the connective $\neg$ can be applied to the atoms appearing in the bodies. However, due to the definition of $\neg$, each negated atom $\neg F : (\alpha, \beta)$ transforms easily into the positive atom $F : (\beta, \alpha)$. This is why, for the rest of the paper, we develop BAPs assuming that all the literals in the clauses are brought into positive form. A detailed analysis of properties of negation in BAPs can be found in [17,18].

Note that we use $\otimes$ to connect literals in the bodies of annotated clauses, and thus give priority to the lattice based on $\leq_k$. This permits us to use monotonicity of this lattice relative to the negation of the language. Individual and annotation variables in the body are thought of as being existentially quantified using Σ. We showed in [20] how the remaining bilattice connectives can be introduced into annotated clauses, but we will not address this issue in this paper.

We mention here that any logic program contains only a finite set of annotation constants. This is why there are classes of programs which can be interpreted by finite bilattices, for example, logic programs which do not contain annotation variables and/or annotation functions, and logic programs containing only functions which do not generate new annotation constants through the process of computing.

Example 3.2 Consider the following infinitely-interpreted logic program:

$$\begin{aligned} R_1(a_1) : (1, 0.5) &\leftarrow \\ R_2(f(x)) : (\frac{\mu}{2}, \frac{\nu}{3}) &\leftarrow R_1(x) : (\mu, \nu) \\ R_1(f(x)) : (\frac{\mu}{3}, \frac{\nu}{3}) &\leftarrow R_2(x) : (\mu, \nu) \end{aligned}$$

This program receives its interpretations from the countable bilattice whose underlying set of elements contains 0, 1, 0.5 and all the numbers which can be generated from 0, 1, 0.5 by iterating the functions $\frac{\mu}{2}$, $\frac{\mu}{3}$, $\frac{\nu}{3}$.

Example 3.3 Suppose we want to implement a program P which is able to make decisions about delaying and cancelling airport flights without human supervision. Let this program work with databases which are obtained through accumulating information from weather forecasts taken from different sources. These sources may give inconsistent or incomplete information which is collected in the database of P. The first and the second elements of each annotation denote evidence for, respectively against, a given fact. Let $\mathbf{B}_{25}$ from Example 2.3 be the chosen bilattice. Let individual variables x, y receive, respectively, values from the set

{Monday, Tuesday} and the set with the numbers of flights $\{1, 2\}$. A suitable program for processing the database may contain the following fragment:

$$\begin{aligned}
\text{Storm}(\text{Monday}) : (\frac{3}{4}, \frac{1}{2}) \leftarrow \\
\text{Storm}(x) : (\frac{1}{2}, \frac{3}{4}) \leftarrow \\
\text{Delay}(y, x) : ((\frac{3}{4}, \frac{1}{2}) \vee (\frac{1}{2}, \frac{1}{2})) \leftarrow \text{Storm}(x) : (\frac{3}{4}, \frac{3}{4}), \text{Storm}(\text{Tuesday}) : (\frac{1}{2}, \frac{1}{2}) \\
\text{Cancel}(y, x) : (1, 0) \leftarrow \text{Delay}(y, x) : (\frac{3}{4}, \frac{1}{2})
\end{aligned}$$

Note that this program is able to make quantitative (fuzzy) conclusions, as in the third clause, as well as qualitative (two-valued) conclusions, as in the last clause. It can work with conflicting sources of information, see, for example, the two unit clauses.

In this section, we have defined Bilattice-Based Annotated Logic Programs (BAPs), and considered some examples. We proceed by giving a semantic characterization of BAPs.

4 Declarative Semantics and the $\mathcal{T}_P$-Operator

In this section, we define interpretations and Herbrand interpretations for BAPs, and investigate their properties. We define the semantic operator $\mathcal{T}_P$ and show that it is continuous. We establish that $\mathcal{T}_P$ computes the minimal Herbrand models for BAPs.

Let D and v denote respectively a domain of interpretation and a variable assignment for a given language $\mathcal{L}$, see [22]. An interpretation I for $\mathcal{L}$ consists of the following mappings. The first mapping $\mathcal{I}$ assigns $|R|_v : D^n \longrightarrow \mathbf{B}$ to each n-ary predicate symbol R in $\mathcal{L}$. Further, for each element $\langle \alpha, \beta \rangle$ of $\mathbf{B}$, we define a mapping $\chi_{\langle \alpha,\beta \rangle} : \mathbf{B} \longrightarrow \mathbf{B}$, where $\chi_{\langle \alpha,\beta \rangle}(\langle \alpha', \beta' \rangle) = \langle 1, 0 \rangle$ if $\langle \alpha, \beta \rangle \leq_k \langle \alpha', \beta' \rangle$ and $\chi_{\langle \alpha,\beta \rangle}(\langle \alpha', \beta' \rangle) = \langle 0, 1 \rangle$ otherwise. The mapping χ is used to evaluate annotated formulae.

We will use the two functions $\mathcal{I}$ and χ to define interpretation I for annotated atoms. If F is an annotated atom $R(t_1, \ldots, t_n) : (\tau)$, then the value of F is given by $I(F) = \chi_{\langle \tau \rangle}(|R|_v(|t_1|_v, \ldots, |t_n|_v))$, as follows. Given an annotated atom $F : (\alpha', \beta')$ with constant annotation (α', β'), an interpretation $\mathcal{I}$ for a first-order formula F, and a value $\langle \alpha, \beta \rangle$ from $\mathbf{B}$ assigned to F, we use χ as follows: if the value $\langle \alpha, \beta \rangle \geq_k \langle \alpha', \beta' \rangle$, then $I(F : (\alpha', \beta')) = \langle 1, 0 \rangle$, and $I(F : (\alpha', \beta')) = \langle 0, 1 \rangle$ otherwise. If the annotated term τ attached to an annotated atom $F : \tau$ contains variables $\overline{\mu}, \overline{\nu}$, we use the existential quantifier Σ when applying χ as follows: $\chi_\tau(\langle \alpha, \beta \rangle) = \langle 1, 0 \rangle$ if $\Sigma(\overline{\mu}, \overline{\nu})(\tau \geq \langle \alpha, \beta \rangle)$.

Furthermore, we can proceed and give interpretation to complex annotated formulae in the standard way [15,20,17], using the operations defined on $\mathbf{B}$ to evaluate connectives and quantifiers. All the connectives of the language are put into correspondence with bilattice operations, and in particular quantifiers correspond to

infinite bilattice operations. We call the composition of the two mappings $\mathcal{I}$ and χ an *interpretation* for the bilattice-based annotated language $\mathcal{L}$ and for simplicity of notation denote it by I. Indeed, the interpretations of BAPs possess some remarkable properties which make the study of BAPs worthwhile, as follows.

Proposition 4.1 *[20]*

(i) *Let F be a formula, and fix the value $I(F)$. If $I(F : (\alpha, \beta)) = \langle 1, 0\rangle$, then $I(F : (\alpha', \beta')) = \langle 1, 0\rangle$ for all $\langle \alpha', \beta'\rangle \leq_k \langle \alpha, \beta\rangle$.*

(ii) *$I(F_1 : (\tau_1) \otimes \ldots \otimes F_k : (\tau_k)) = \langle 1, 0\rangle \iff I(F_1 : (\tau_1) \oplus \ldots \oplus F_k : (\tau_k)) = \langle 1, 0\rangle \iff I(F_1 : (\tau_1) \wedge \ldots \wedge F_k : (\tau_k)) = \langle 1, 0\rangle \iff I(F_i : (\tau_i)) = \langle 1, 0\rangle$ for each $i \in \{1, \ldots, k\}$.*

(iii) *If $I(F_1 : (\tau_1) \odot \ldots \odot F_k : (\tau_k)) = \langle 1, 0\rangle$, then $I((F_1 \odot \ldots \odot F_k) : ((\tau_1) \odot \ldots \odot (\tau_k))) = \langle 1, 0\rangle$, where $\odot$ is any one of the connectives $\otimes, \oplus, \wedge$.*

(iv) *For every formula F, $I(F : (0, 0)) = \langle 1, 0\rangle$.*

These properties generalise easily to infinite bilattice operations $\Sigma, \Pi, \exists, \forall$; and they influence the models for BAPs.

Let I be an interpretation for $\mathcal{L}$ and let F be a closed annotated formula of $\mathcal{L}$. Then I is a *model* for F if $I(F) = \langle 1, 0\rangle$. We say that I is a model for a set S of annotated formulae if I is a model for each annotated formula of S. We say that F *is a logical consequence of* S if, for every interpretation I of $\mathcal{L}$, I is a model for S implies I is a model for F.

Let B_P denote the annotation Herbrand base for a program P, namely, the set of all ground annotated atoms which can be formed out of the symbols of P. An *annotation Herbrand interpretation* HI *for* P is the assignment of a mapping I from B_P into $\mathbf{B}$. Following the convention of logic programming, each Herbrand interpretation HI for P can be identified with the subset $\{R(t_1, \ldots, t_k) : (\alpha, \beta) \in B_P | R(t_1, \ldots, t_k) : (\alpha, \beta)$ receives the value $\langle 1, 0\rangle$ with respect to $I\}$ of B_P, where $R(t_1, \ldots, t_k) : (\alpha, \beta)$ denotes a typical element of B_P. This set constitutes an *annotation Herbrand model for* P. Finally, we let $\mathrm{HI}_{P,\mathbf{B}}$ denote the set of all annotation Herbrand interpretations for P.

In [20], we introduced a semantic operator $\mathcal{T}_P$ for BAPs, proved its continuity and showed that it computes the least Herbrand model M_P for a given BAP P. Detailed analysis of some of the properties of $\mathcal{T}_P$ can be found in [17]. We define $\mathcal{T}_P$ next.

Definition 4.2 We define the mapping $\mathcal{T}_P : \mathrm{HI}_{P,\mathbf{B}} \to \mathrm{HI}_{P,\mathbf{B}}$ as follows: $\mathcal{T}_P(\mathrm{HI}) = = \{A : (\tau) \in B_P$ such that either

(i) $A : (\tau) \leftarrow L_1 : (\tau_1), \ldots, L_n : (\tau_n)$ is a strictly ground instance of a clause in P and $\{L_1 : (\tau_1'), \ldots, L_n : (\tau_n')\} \subseteq \mathrm{HI}$, and for each (τ_i'),

$$(\tau_i) \leq_k (\tau_i'),$$

or

(ii) there are annotated strictly ground atoms $A : (\tau_1^*), \ldots, A : (\tau_k^*) \in \mathrm{HI}$ such that $(\tau) \leq_k (\tau_1^*) \oplus \ldots \oplus (\tau_k^*)\}$.

Item (i) above reflects the property stated in item (i) of Proposition 4.1. Item (ii) above reflects the properties captured in items (ii) and (iii) of Proposition 4.1. Note that according to item (i) of Proposition 4.1, whenever $F : (\tau) \in HI$ and $(\tau') \leq_k (\tau)$, then $F : (\tau') \in HI$. Also, item (iv) of Proposition 4.1 ensures that for each formula F, $F : (0,0) \in HI$, and we assume this property of HI throughout the paper. Thus, we always assume that each HI contains all ground formulae of the type $F : (0,0)$ prior to any implementations of the semantic operator.

Semantic operators defined for many logic programs in the style of [4,6,7,15,31] use only some form of item (i) from Definition 4.2. However, this condition is not sufficient for the computation of the (least) Herbrand models for (bi)lattice-based logic programs, as we see next.

Let $\widehat{\mathcal{T}_P}$ denote a form of semantic operator $\mathcal{T}_P$ with no item (ii) from Definition 4.2. The next example displays the difference between $\widehat{\mathcal{T}_P}$ and $\mathcal{T}_P$. Symbols [*i*] and [*ii*] refer to items of Definition 4.2.

Example 4.3 Consider the logic program from Example 3.3, and the least fixed points of $\mathcal{T}_P$ and $\widehat{\mathcal{T}_P}$ [5].

Iteration	$\mathcal{T}_P$	$\widehat{\mathcal{T}_P}$
1	Storm(Monday) : $(\frac{3}{4}, \frac{1}{2})$[i], Storm(Monday) : $(\frac{1}{2}, \frac{3}{4})$[i], Storm(Tuesday) : $(\frac{1}{2}, \frac{3}{4})$[i]	Storm(Monday) : $(\frac{3}{4}, \frac{1}{2})$, Storm(Monday) : $(\frac{1}{2}, \frac{3}{4})$, Storm(Tuesday) : $(\frac{1}{2}, \frac{3}{4})$
2	Storm(Monday) : $(\frac{3}{4}, \frac{3}{4})$[ii]	—
3	Delay(1, Monday) : $((\frac{3}{4}, \frac{1}{2}) \vee (\frac{1}{2}, \frac{1}{2}))$[i], Delay(2, Monday) : $((\frac{3}{4}, \frac{1}{2}) \vee (\frac{1}{2}, \frac{1}{2}))$[i]	—
4	Cancel(1, Monday) : $(1,0)$[i], Cancel(2, Monday) : $(1,0)$[i]	—

Thus, in this example we can see that the program will definitely cancel both flights 1 and 2 on Monday.

The following example shows the fixed point of the semantic operators reached in ω steps.

Example 4.4 Consider the logic program given in Example 3.2. The least fixed point of this program (for $\widehat{\mathcal{T}_P}$ and for $\mathcal{T}_P$) is
$\{R_1(a_1) : (1, 0.5), R_2(f(a_1)) : (\vartheta_1(1), \vartheta_2(0.5)), R_1(f(f(a_1))) :$
$(\vartheta_3(\vartheta_1(1)), \vartheta_4(\vartheta_2(0.5))), \ldots, R_1(f^{n-1}(a_1)) : (\vartheta_3^n(\vartheta_1^{n-1} \ldots (\ldots((1))\ldots)),$
$\vartheta_4^n(\vartheta_2^{n-1} \ldots (\ldots((0.5))\ldots))), R_2(f^n(a_1)) : (\mu', \nu'), \ldots\}$, where $n \in \omega$. Here, μ' de-

[5] Note that we use the fact that $((\frac{3}{4}, \frac{1}{2}) \vee (\frac{1}{2}, \frac{1}{2})) = (\frac{3}{4}, \frac{1}{2})$ and $((\frac{3}{4}, \frac{1}{2}) \oplus (\frac{1}{2}, \frac{3}{4})) = (\frac{3}{4}, \frac{3}{4})$.

notes $\vartheta_1^n(\vartheta_3^{n-1}(\vartheta_1^{n-2}\ldots(\ldots((1))\ldots)))$, ν' denotes $\vartheta_2^n(\vartheta_4^{n-1}(\vartheta_2^{n-2}\ldots(\ldots((0.5))\ldots)))$, and ϑ_1, ϑ_2, ϑ_3, ϑ_4 stand for the functions $\frac{\mu}{2}$, $\frac{\nu}{3}$, $\frac{\mu}{3}$, $\frac{\nu}{3}$ respectively.

The following theorem is important and plays a fundamental role in considering the computation of the least fixed points of $\mathcal{T}_P$.

Theorem 4.5 *The mapping $\mathcal{T}_P$ is continuous.*

The proof of this theorem can be found in [17,20]. Note that $\mathcal{T}_P$ works with sets of strictly ground formulae, and not with interpretations themselves and this, in conjunction with the requirement that **B** has only finite joins, guarantees that $\mathcal{T}_P$ is continuous. Note that analogous semantic operators defined in [4,15] and elsewhere do not possess this property.

Now, using Kleene's theorem and Theorem 4.5, we may assert that $\mathrm{lfp}(\mathcal{T}_P) = \mathcal{T}_P \uparrow \omega$. Indeed, we have the following generalization of a well-known theorem due to van Emden and Kowalski [30].

Theorem 4.6 *[20] For a BAP P, we have $M_P = lfp(\mathcal{T}_P) = \mathcal{T}_P \uparrow \omega$.*

To summarize this section, we have defined interpretations and Herbrand interpretations for BAPs, and showed how to characterize the minimal models of BAPs using the semantic operator $\mathcal{T}_P$.

We finish the section by giving a useful definition that will provide a link between the declarative semantics of this section and the operational semantics we are going to consider in the next section.

Definition 4.7 Let P be a BAP and let G be a goal $\leftarrow A_1 : (\tau_1), \ldots, A_k : (\tau_k)$. An *answer* for $P \cup \{G\}$ is a substitution $\theta\lambda$ for individual and annotation variables of G. We say that $\theta\lambda$ is a *correct answer* for $P \cup \{G\}$ if $\Pi((A_1 : (\tau_1), \ldots, A_k : (\tau_k))\theta\lambda)$ is a logical consequence of P.

We will give a proof theoretic counterpart for the notion of the correct answer next.

5 SLD-Resolution

The resolution method was first introduced by Robinson, and was implemented (as SLD-resolution) in two-valued logic programming by Colmerauer et al. A detailed exposition of this can be found in [22], and for many-valued resolution procedures see [12,14,15,23,27,29], for example. Kifer and Lozinskii show in [14] that unlike classical refutation procedures, where only the resolution rule is applied, lattice-based theories need to have four procedures: resolution, factorisation, reduction and elimination in order to be sound and complete. This enriches resolution for many-valued logics which have linearly ordered sets of values, as it was defined, for example, in [12,31]. Some very interesting ideas about the relationship between resolutions for languages with ordered and non-ordered annotations can be found

in [24,25]. Comparing with these papers, we allow variables and functions in annotations and adopt the additional refutation rules (which correspond to some rules of [14]) in order to obtain soundness and completeness of SLD-resolution for BAPs. Note that in [14,24,25] only constant annotations are allowed in the language and therefore each logic program becomes finitely interpreted in these settings. We extend all our results to infinitely interpreted programs with functions and variables in annotations. [6] Finally, we establish an operational semantics for BAPs and prove its soundness and completeness.

Throughout this section, we denote a BAP by P, and a BAP goal by G. We refer to [22] for a description of the unification process (originally defined by Herbrand and later refined by Clark [3]) for classical logic programming. Here, we follow this development, but involve annotation variables in the process of unification, see also [17]. The unification process for the individual variables remains unchanged, and, as in two-sorted languages, unification for the first-order and annotation parts of an annotated formula are handled independently. Following conventional notation, we denote the disagreement set by S, and substitutions by $\theta\lambda$, possibly with subscripts, where θ denotes a first-order substitution, and λ denotes a substitution involving annotations. We will use the abbreviation *mgu* when talking about *most general unifiers* defined in [22] or [17], for example.

Definition 5.1 [SLD-derivation] Let G_i be the annotated goal $\leftarrow B_1 : (\tau_1), \ldots, B_k : (\tau_k)$, and let $C_1, \ldots, C_l$ be the annotated clauses $A_1 : (\tau_1^*) \leftarrow \mathrm{body}_1, \ldots, A_l : (\tau_l^*) \leftarrow \mathrm{body}_l$, where each body_i from $\mathrm{body}_1, \ldots, \mathrm{body}_l$ denotes the body of the clause C_i. Then the goal G_{i+1} is *derived* from G_i and $C_1, \ldots, C_l$ using mgu $\theta\lambda$ if the following conditions hold:

(i) $B_m : (\tau_m)$ is an annotated atom, called the *selected atom*, in G_i and
  (a) θ is an *mgu* of B_m and A_1, and one of the following conditions holds: either λ is an *mgu* of (τ_m) and (τ_1^*); or $(\tau_m)\lambda$ and $(\tau_1^*)\lambda$ receive constant values such that $(\tau_m)\lambda \leq_k (\tau_1^*)\lambda$;
    or
  (b) θ is an *mgu* of B_m and $A_1, \ldots, A_l$, and either λ is an *mgu* of (τ_m) and $(\tau_1^*), \ldots, (\tau_l^*)$ or $(\tau_m)\lambda$ and $(\tau^*)\lambda, \ldots, (\tau_l^*)\lambda$ receive constant values such that $(\tau_m)\lambda \leq_k ((\tau_1^*)\lambda \oplus \ldots \oplus (\tau_l^*)\lambda)$.

(ii) in case (a), G_{i+1} is the goal $(\leftarrow B_1 : (\tau_1), \ldots, B_{m-1} : (\tau_{m-1}), \mathrm{body}_1, B_{m+1} : (\tau_{m+1}), \ldots, B_k : (\tau_k))\theta\lambda$. In this case, G_{i+1} is said to be derived from G_i and C_1 using $\theta\lambda$.

(iii) in case (b), G_{i+1} is the goal $(\leftarrow B_1 : (\tau_1), \ldots, B_{m-1} : (\tau_{m-1}), \mathrm{body}_1, \ldots, \mathrm{body}_l, B_{m+1} : (\tau_{m+1}), \ldots, B_k : (\tau_k))\theta\lambda$. In this case, G_{i+1} is said to be derived from G_i, $C_1, \ldots, C_l$ using $\theta\lambda$.

(iv) Whenever a goal G_i contains a formula of the form $F : (0, 0)$, then remove $F : (0, 0)$ from the goal and form the next goal G_{i+1} that is G_i except that it does not contain $F : (0, 0)$.

[6] An SLD-resolution for lattice-based logic programs with annotation functions and annotation variables was first introduced in [15] and was shown to be incomplete.

Note that certain items in the definition of derivation correspond to certain items in Definition 4.2 of $\mathcal{T}_P$. For example, item (a) corresponds to item (i) in Definition 4.2, and item (ii) corresponds to item (ii) in Definition 4.2. And, as we have noted before in relation to the definition of $\mathcal{T}_P$, all these items serve to reflect the model properties of BAPs captured in Proposition 4.1.

Definition 5.2 Suppose that P is a BAP and G_0 is a goal. An *SLD-derivation* of $P \cup \{G_0\}$ consists of a sequence $G_0, G_1, G_2 \ldots$ of BAP goals, a sequence of finite sets $S_1, S_2, \ldots$ of BAP clauses and a sequence $\theta_1\lambda_1, \theta_2\lambda_2, \ldots$ of *mgu*s such that each G_{i+1} is derived from G_i and C_{i+1} using $\theta_{i+1}\lambda_{i+1}$.

Note that $S_1, S_2, \ldots$ is defined to be a sequence of *finite sets* of clauses, and not just a sequence of clauses as in the case of classical SLD-resolution. This happens because item (b) admits the use of a finite set of clauses at each step of the derivation. This item was not included in the classical definition of SLD-resolution.

In [16], we gave a many-sorted representation of BAPs. This translation allowed us to apply the classical definition of SLD-resolution to the many-sorted translation of BAPs. This result showed that in principle, one can use just *sequences of clauses*, and not *sequences of finite sets of clauses* when defining many-sorted *SLD-derivation* for BAPs; but this would require many-sorted non-annotated translation.

Definition 5.3 An *SLD-refutation* of $P \cup G_0$ is a finite SLD-derivation of $P \cup G_0$ which has the empty clause $\Box$ as the last goal of the derivation. If $G_n = \Box$, we say that the refutation has *length* n.

Definition 5.4 The success set of P is the set of all $A : (\mu, \nu) \in B_P$ such that $P \cup \{\leftarrow A : (\mu, \nu)\}$ has an SLD-refutation.

Definition 5.5 A *computed answer* $\theta\lambda$ for $P \cup \{G_0\}$ is the substitution obtained by restricting the composition of $\theta_1, \ldots, \theta_n, \lambda_1, \ldots, \lambda_k$ to the variables of G_0, where $\theta_1, \ldots, \theta_n, \lambda_1, \ldots, \lambda_k$ is the sequence of *mgu*s used in the SLD-refutation of $P \cup \{G_0\}$.

Example 5.6 Let P be the program from Example 3.3 and let G_0 be the goal $\leftarrow \mathrm{Cancel}(1, \mathrm{Monday}) : (\mu, \nu)$, that is, we want to know the probability of cancelling flight number 1 on Monday.

(i) We have a clause $\mathrm{Cancel}(y, x) : (1, 0) \leftarrow \mathrm{Delay}(y, x) : (\frac{3}{4}, \frac{1}{2})$ in P. Form the set $S = \{\mathrm{Cancel}(1, \mathrm{Monday}) : (\mu, \nu), \mathrm{Cancel}(y, x) : (1, 0)\}$, find its disagreement set, and apply item (a) from Definition 5.1 to get its *mgu*: $\theta_0\lambda_0 = \{y/1, x/\mathrm{Monday}, \mu/1, \nu/0\}$.

(ii) Now G_1 is $\leftarrow \mathrm{Delay}(y, x) : (\frac{3}{4}, \frac{1}{2})\theta_0\lambda_0 = \mathrm{Delay}(1, \mathrm{Monday}) : (\frac{3}{4}, \frac{1}{2})$. We have a clause $\mathrm{Delay}(y, x) : ((\frac{3}{4}, \frac{1}{2}) \vee (\frac{1}{2}, \frac{1}{2})) \leftarrow \mathrm{Storm}(x) : (\frac{3}{4}, \frac{3}{4}), \mathrm{Storm}(\mathrm{Tuesday}) : (\frac{1}{2}, \frac{1}{2})$, whose head contains annotations which are unifiable with $(\frac{3}{4}, \frac{1}{2})$. This means that item (a) from Definition 5.1 can be applied here.

(iii) $G_2 = \leftarrow (\mathrm{Storm}(\mathrm{Monday}) : (\frac{3}{4}, \frac{3}{4}), \mathrm{Storm}(\mathrm{Tuesday}) : (\frac{1}{2}, \frac{1}{2}))\theta_1\lambda_1 = \leftarrow \mathrm{Storm}(\mathrm{Monday}) : (\frac{3}{4}, \frac{1}{2}), \mathrm{Storm}(\mathrm{Tuesday}) : (\frac{1}{2}, \frac{1}{2})$.
Let $\mathrm{Storm}(\mathrm{Monday}) : (\frac{3}{4}, \frac{3}{4})$ be the selected atom. We see that it is not unifiable with any input clause so we apply item (b) from Definition 5.1. So, we find the

mgu θ_2 for Storm(x) : $(\frac{1}{2}, \frac{3}{4})$, Storm(Monday) : $(\frac{3}{4}, \frac{1}{2})$ and Storm(Monday) : $(\frac{3}{4}, \frac{3}{4})$ such that $\theta_2 = \{x/\text{Monday}\}$, and $(\frac{3}{4}, \frac{3}{4}) \leq_k (\frac{1}{2}, \frac{3}{4}) \oplus (\frac{3}{4}, \frac{1}{2})$. So, we can form the next goal according to item (iii) from Definition 5.1.

(iv) The goal $G_3 = \leftarrow$ Storm(Tuesday) : $(\frac{1}{2}, \frac{1}{2})$. Now Storm(Tuesday) : $(\frac{1}{2}, \frac{1}{2})$ is a selected atom and, choosing the input clause Storm(x) : $(\frac{1}{2}, \frac{3}{4}) \leftarrow$, we apply item (a) from Definition 5.1: the *mgu* for Storm(Tuesday) and Storm(x) is $\theta_2 = x/\text{Tuesday}$, and $(\frac{1}{2}, \frac{1}{2}) \leq_k (\frac{1}{2}, \frac{3}{4})$.

(v) Use item (ii) and form the last goal $G_4 = \Box$.

Thus, we conclude that we have obtained an SLD-refutation of $P \cup \{G_0\}$ of length 4 with computed answer $\theta_0\lambda_0\theta_1\lambda_1\theta_2\lambda_2$ ($\lambda_2 = \lambda_1 = \varepsilon$), which is restricted to the variables of G_0 (in our case, the computed answer is $\{\mu/1, \nu/0\}$, that is, the flight number 1 will definitely be cancelled on Monday). Moreover, we conclude that Cancel$(1, \text{Monday}) : (1, 0)$ is in the success set of P. As can be seen from Example 4.3, this formula is contained in the least fixed point of the $\mathcal{T}_P$ operator applied to P.

The next theorem shows that the computations performed by the SLD resolution algorithm are correct.

Theorem 5.7 (Soundness of SLD-resolution for BAPs) *Let P be a BAP. Then every computed answer for $P \cup \{G\}$ is a correct answer for $P \cup \{G\}$.*

Proof. Let G be $\leftarrow B_1 : (\tau_1), \ldots, B_k : (\tau_k)$ and $\theta_1\lambda_1, \ldots, \theta_n\lambda_n$ be the sequence of *mgu*s used in a refutation of $P \cup \{G\}$. We have to show that $\Pi(B_1(\tau_1) \otimes \ldots \otimes B_k : (\tau_k))\theta_1\lambda_1 \ldots \theta_n\lambda_n$ is a logical consequence of P.

We prove this by induction on the length n of the refutation.

Basis step Suppose first that $n = 1$. This means that G is a goal of the form $\leftarrow B_1 : (\tau_1)$, and one of the following conditions holds:

(i) P has a unit program clause of the form $A_1 : (\tau_1^*) \leftarrow$ and $B_1\theta = A_1\theta$ and
  (a) either $(\tau_1)\lambda = (\tau_1^*)\lambda_1$,
  (b) or $(\tau_1)\lambda$ and $(\tau_1^*)\lambda$ receive constant values and $(\tau_1^*)\lambda \leq_k (\tau_1)\lambda$.

(ii) According to the definition of refutation, we have the third case in which P has clauses $A_1 : (\tau_1^*) \leftarrow$, …, $A_l : (\tau_l^*) \leftarrow$ such that $B_1\theta = A_1\theta = \ldots = A_l\theta$, and
  (a) either $(\tau_1)\lambda = (\tau_1^*)\lambda = \ldots = (\tau_l^*)\lambda$,
  (b) or $(\tau_1)\lambda$ and $(\tau_1^*)\lambda, \ldots, (\tau_l^*)\lambda$ are constants, and $(\tau_1)\lambda \leq_k ((\tau_1^*)\lambda \oplus \ldots \oplus (\tau_l^*))\lambda$.

(iii) $G = \leftarrow A_1 : (0, 0)$.

Suppose (a) holds. Since $(A_1 : (\tau_1^*))\theta\lambda$ is an instance of a unit clause in P, we conclude that $(\Pi(B_1 : (\tau_1)))\theta\lambda$ is a logical consequence of $P \cup \{G\}$.

Suppose (b) holds. Since $A_1\theta = B_1\theta$, $(\tau_1)\lambda \leq_k (\tau_1^*)\lambda$, and $A_1 : (\tau_1^*)\theta\lambda$ is an instance of a unit clause in P and, using Proposition 4.1, we conclude that $(\Pi(B_1 : (\tau_1)))\theta\lambda$ is a logical consequence of P.

Suppose (ii) holds. The case (a) can be proved analogously to the proof of (a).

Consider case (b). Since all $(A_1 : (\tau_1^*) \leftarrow)\theta\lambda, \ldots, (A_l : (\tau_l^*) \leftarrow)\theta\lambda$ are instances of unit clauses in P, all these formulae are logical consequences of P. But then, using the fact that $A_1^*\theta = \ldots = A_l^*\theta$ and Proposition 4.1, we conclude that $A((\tau_1^*) \oplus \ldots \oplus (\tau_l^*))$ is a logical consequence of P. Now, using Proposition 4.1 and the fact that $B_1\theta = A_1\theta$, we have that $(\Pi(B_1 : (\tau_1)))\theta\lambda$ is a logical consequence of P.

Suppose (iii) holds. According to Proposition 4.1, $\Pi(A_1 : (0, 0))$ is a logical consequence of P.

Inductive step. Suppose that the result holds for computed answers which come from refutations of length $n-1$. Suppose $\theta_1\lambda_1, \ldots, \theta_n\lambda_n$ is the sequence of *mgu*s used in a refutation of $P \cup \{G\}$ of length n. Suppose further that the first refutation step in the refutation of length n was made in accordance with item (a) in Definition 5.1, and let $A_1 : (\tau_1^*) \leftarrow \text{body}_1$ be the first input clause, such that $A_1 : (\tau_1^*)\theta_1 \ldots \theta_n = B_m : (\tau_m)\theta_1 \ldots \theta_n$ for some $B_m : (\tau_m)$ in G. By the induction hypothesis, $\Pi(B_1 : (\tau_1) \otimes \ldots \otimes B_{m-1} : (\tau_{m-1}) \otimes \text{body}_1 \otimes B_{m+1} : (\tau_{m+1}) \otimes \ldots \otimes B_k : (\tau_k))\theta_1\lambda_1 \ldots \theta_n\lambda_n$ is a logical consequence of P. (This is because only $n-1$ steps are needed to obtain a refutation for the latter formula.) But then $(\text{body}_1)\theta_1\lambda_1 \ldots \theta_n\lambda_n$ is a logical consequence of P. This means that $A_1 : (\tau_1^*)\theta_1\lambda_1 \ldots \theta_n\lambda_n$ is a logical consequence of P. Additionally, we use the fact that $(\tau_m)\lambda_1 \ldots \lambda_n \leq_k (\tau_1^*)\lambda_1 \ldots \lambda_n$ and apply Proposition 4.1 to conclude that $B_m : (\tau_m)\theta_1\lambda_1 \ldots \theta_n\lambda_n$ is a logical consequence of P.

Suppose the first refutation step in the refutation of length n was taken in accordance with item (b) in Definition 5.1. Consider input clauses $A_1 : (\tau_1^*) \leftarrow \text{body}_1, \ldots, A_l : (\tau_l^*) \leftarrow \text{body}_l$; and selected atom $B_m : (\tau_m)$ of G at this step of the refutation. By the induction hypothesis, $\Pi(B_1 : (\tau_1) \otimes \ldots \otimes B_{m-1} : (\tau_{m-1}) \otimes (\text{body}_1) \otimes \ldots \otimes (\text{body}_l) \otimes B_{m+1} : (\tau_{m+1}) \otimes \ldots \otimes B_k : (\tau_k))\theta_1\lambda_1 \ldots \theta_n\lambda_n$ is a logical consequence of P. (This is because only $n-1$ steps are needed to obtain a refutation for the latter formula.) Consequently, $(\text{body}_1), \ldots, (\text{body}_l))\theta_1\lambda_1 \ldots \theta_n\lambda_n$ is a logical consequence of P, which means that $(A_1 : (\tau^*))\theta_1\lambda_1 \ldots \theta_n\lambda_n, \ldots, A_l : (\tau_l^*))\theta_1\lambda_1 \ldots \theta_n\lambda_n$ are logical consequences of P. But since $A_1\theta_1 \ldots \theta_n = \ldots = A_l\theta_1 \ldots \theta_n = B_m\theta_1 \ldots \theta_n$, using Proposition 4.1, we see that $(A_1 : ((\tau_1^*) \oplus \ldots \oplus (\tau_l^*)))\theta_1 \ldots \theta_n$ is a logical consequence of P. Moreover, according to Definition 5.1, item (b) which we have taken as our assumption, we have $(\tau_m)\lambda_1 \ldots \lambda_n \leq_k (\tau_1^*)\lambda_1 \ldots \lambda_n, \oplus \ldots \oplus (\tau_l^*)\lambda_1 \ldots \lambda_n$. Now, using Proposition 4.1, we conclude that $(B_m : (\tau_m))\theta_1\lambda_1 \ldots \theta_n\lambda_n$ is a logical consequence of P.

We can apply the same arguments for the rest of the atoms $(B_1 : (\tau_1), \ldots, B_k : (\tau_k))$ from G. Thus, $\Pi((B_1 : (\tau_1), \ldots, B_k : (\tau_k)))\theta_1\lambda_1 \ldots \theta_n\lambda_n$ is a logical consequence of P. □

Corollary 5.8 *The success set of P is contained in its least annotation Herbrand model.*

Proof. Let $A : (\tau) \in B_P$ and suppose that $P \cup \{\leftarrow A : (\tau)\}$ has a refutation. By Theorem 5.7, $A : (\tau)$ is a logical consequence of P. Thus $A : (\tau)$ is in the least annotation Herbrand model for P. □

We have proved that the algorithm of SLD resolution we defined for BAPs is

sound. We now wish to prove that it is complete.

6 Completeness of SLD-Resolution for BAPs

This section finishes the formal discussion of BAPs by showing that the SLD resolution algorithm for BAPs is complete.

Some proofs in this section require the use of lemmas which are straightforward generalizations of the so-called *mgu lemma* and *lifting lemma* [22] to the case when unification is allowed with respect to annotation variables as well as individual variables. The lemmas and proofs can be found in [17].

The following completeness theorem extends the corresponding theorem for two-valued propositional logic programming due to Apt and Van Emden.

Theorem 6.1 *Let P be a BAP. The success set of P is equal to its least annotation Herbrand model.*

Proof. By Corollary 5.8, it suffices to show that the least Herbrand model for P is contained in its success set.

Suppose that $A : (\tau)$ is in the least annotation Herbrand model for P. By Theorem 4.6, $A : (\tau) \in \mathcal{T}_P \uparrow n$, for some $n \in \omega$. We claim that $A : (\tau) \in \mathcal{T}_P \uparrow n$ implies that $P \cup \{\leftarrow A : (\tau)\}$ has a refutation, and hence that $A : (\tau)$ is in the success set; we prove this claim by induction on n.

Basis step. $n = 1$. Then $A : (\tau) \in \mathcal{T}_P \uparrow 1$, which means that either $A : (\tau) \leftarrow$ is a strictly ground instance of a clause in P or $(\tau) = (0, 0)$. And in both cases $P \cup \{\leftarrow A : (\tau)\}$ has a refutation (see items (a) and (iv) in Definition 5.1).

Inductive step. Suppose the claim holds for $n - 1$. Let $A : (\tau) \in \mathcal{T}_P \uparrow n$. By the definition of $\mathcal{T}_P$, one of the following holds:

(i) there exists a strictly ground instance of a clause $B : (\tau') \leftarrow B_1 : (\tau_1), \ldots, B_k : (\tau_k)$ such that $A = B\theta$, $(\tau) = (\tau')\lambda$, and $B_1 : (\tau_1')\theta, \ldots, B_k : (\tau_k')\theta \in \mathcal{T}_P \uparrow (n-1)$ with $\tau_1', \ldots, \tau_k'$ such that:

$$(\tau_i)\lambda \leq_k (\tau_i'), \text{ for } i \in \{1, \ldots, k\}.$$

(ii) There are strictly ground atoms $A : (\tau_1), \ldots, A : (\tau_k) \in \mathcal{T}_P \uparrow (n-1)$ such that $(\tau) \leq_k ((\tau_1) \oplus \ldots \oplus (\tau_k))$.

Suppose (i) holds. By the induction hypothesis, each $P \cup \{\leftarrow B_i : (\tau_i')\theta\}$ has a refutation, for $i \in \{1, \ldots, k\}$. We want to show that then $P \cup \{\leftarrow B_i : (\tau_i)\theta\lambda\}$ has a refutation, for $i \in \{1, \ldots, k\}$.

Consider the refutation of $G_0 = \{\leftarrow B_i : (\tau_i')\theta\}$. According to Definition 5.1, there are two ways in which $\{\leftarrow B_i : (\tau_i')\theta\}$ can be derived.

Case 1.

There is a clause $C : (\tau^*) \leftarrow$ body in P such that $B_i\theta = C\theta$, and $(\tau_i') \leq_k (\tau^*)\lambda$. Taking into account that $(\tau_i)\lambda \leq_k (\tau_i')$, by transitivity of $\leq_k$, we conclude that $(\tau_i)\lambda \leq_k (\tau^*)\lambda$. But then, by Definition 5.1, item (a), the goal $\leftarrow B_i(\tau_i)\theta\lambda$ will receive a refutation.

Case 2.

There are clauses $C_1 : (\tau_1^*) \leftarrow \text{body}_1, \ldots, C_m : (\tau_1^*) \leftarrow \text{body}_m$ in P such that $B_i\theta = C_1\theta = \ldots = C_m\theta$, and $\tau_i' \leq_k (\tau_1^*\lambda \oplus \ldots \oplus \tau_m^*\lambda)$. But then, because $(\tau_i)\lambda \leq_k (\tau_i')$, we have $\tau_i\lambda \leq_k (\tau_1^*\lambda \oplus \ldots \oplus \tau_m^*\lambda)$. So, by Definition 5.1, item (b) the goal $\leftarrow (B_i : \tau_i)\theta\lambda$ will receive refutation as well.

Suppose (ii) holds, that is, there are strictly ground atoms $A : (\tau_1), \ldots, A : (\tau_k) \in \mathcal{T}_P \uparrow (n-1)$ such that $(\tau) \leq_k ((\tau_1) \oplus \ldots \oplus (\tau_k))$. We want to show that then $A : (\tau)$ has a refutation.

Using the induction hypothesis, each of $P \cup \{\leftarrow A : (\tau_1)\}, \ldots, P \cup \{\leftarrow A : (\tau_k)\}$ has a refutation. This means that, according to Definition 5.1, items (a) and (b), there are clauses $A : (\tau_1^*) \leftarrow \text{body}_1^*, \ldots, A : (\tau_n^*) \leftarrow \text{body}_n^*$, such that for each $A : (\tau_i)$ one of the following holds:

- $(\tau_i)\lambda \leq_k (\tau_j^*)\lambda$. In this case, $P \cup \{\leftarrow \text{body}_j^*\}$ has a refutation.
- •• $(\tau_i)\lambda \leq_k ((\tau_j^*) \oplus \ldots \oplus (\tau_l^*))$. Then $P \cup \{\leftarrow \text{body}_j^*, \ldots, \text{body}_l^*\}$ has a refutation, for some $j, l \in \{1, \ldots, n\}$.

Combining all these refutations for $i \in \{1, \ldots, n\}$, we obtain a refutation of $P \cup \{\leftarrow (\text{body}_1^*, \ldots, \text{body}_n^*)\theta\lambda\}$. But then, according to Definition 5.1, items (a) and (b), there is a refutation for $P \cup \{\leftarrow A : (\tau')\theta\lambda\}$, where $\tau'\lambda$ is some annotation such that

$$(\tau')\lambda \leq_k ((\tau_1^*)\lambda \oplus \ldots \oplus (\tau_n^*)\lambda) \quad (*)$$

for each $(\tau')\lambda \leq_k (\tau_j^*)\lambda$ (in case of •) or for each $(\tau')\lambda \leq_k ((\tau_j^*) \oplus \ldots \oplus (\tau_l^*))$ (in case of ••), for $j, l \in \{1, \ldots, n\}$. Now, having that either each $(\tau_i)\lambda \leq_k (\tau_j^*)\lambda$ or each $(\tau_i)\lambda \leq_k ((\tau_j^*) \oplus \ldots \oplus (\tau_l^*))$, for $i \in \{1, \ldots, k\}$ and $j, l \in \{1, \ldots, n\}$, we conclude, by monotonicity of $\oplus$, that

$$((\tau_1)\lambda \oplus \ldots \oplus (\tau_k)\lambda) \leq_k ((\tau_1^*)\lambda \oplus \ldots \oplus (\tau_n^*)\lambda).$$

But then, by Definition 5.1, item (a), the condition $(*)$ implies that, for each annotation constant $(\tau^\Diamond)$ such that $(\tau^\Diamond) \leq_k ((\tau_1^*) \oplus \ldots \oplus (\tau_n^*))\lambda$, there exists a refutation for $P \cup \{\leftarrow A : (\tau^\Diamond)\theta\lambda\}$. This holds in particular for all the $\tau^\Diamond$ such that $(\tau^\Diamond) \leq_k ((\tau_1) \oplus \ldots \oplus (\tau_k))\lambda$. According to item (2), among such $(\tau^\Diamond)$ will be $(\tau^\Diamond) = (\tau)\lambda$. Thus, we conclude that $P \cup \{\leftarrow A : (\tau)\theta\lambda\}$ has an unrestricted refutation. Finally, we apply the *mgu* lemma to obtain a refutation for $P \cup \{\leftarrow A : (\tau)\}$.□

Thus, the SLD resolution algorithm is complete in that, for every BAP P, the success set of P is equal to its least annotation Herbrand model.

Next, we need to obtain the completeness with respect to correct answers. As in classical two-valued logic programming, it is impossible to prove the exact converse of Theorem 5.7. However, we can extend the classical result that every correct answer is an instance of a computed answer to the case of BAPs. The next lemma and associated theorem are straightforward (the proofs can be found in [17]), recall that we allow refutation to work over individual and annotation variables independently.

Lemma 6.2 *Let $\Pi(A : (\mu, \nu))$ be a logical consequence of P. Then there exists an SLD-refutation of $P \cup \{\leftarrow A : (\mu, \nu)\}$ with the identity substitution as a computed answer.*

The next completeness result is a generalization of the analogous theorem of Clark [3], its proof can be found in [17].

Theorem 6.3 *For every correct answer $\theta\lambda$ for $P \cup \{G\}$, there exist a computed answer $\theta^*\lambda^*$ for $P \cup \{G\}$ and substitutions φ, ψ such that $\theta = \theta^*\varphi$ and $\lambda = \lambda^*\psi$.*

This theorem concludes our discussion of the SLD resolution algorithm.

7 Conclusions and further work

We have carefully examined the declarative and operational semantics of bilattice-based annotated logic programs. In particular, we have shown that unlike the usual approach to many-valued logic programming semantics (see, for example, [7,8,15,29] and many others), the immediate consequence operator for bilattice-based annotated logic programs cannot be obtained as a simple extension of the classical semantic operator. We have given some examples displaying the immediate consequence operators as defined in [7,8,15], and shown that these operators do not compute all the logical consequences of a program, but only certain of them. We have proposed an original definition of the immediate consequence operator computing all the logical consequences of a given bilattice-based annotated logic program, and showed that it is continuous.

The declarative semantics for BAPs allows us to propose an SLD-resolution for BAPs and prove its soundness and completeness relative to our semantics. Like the resolution procedures given in [14] for lattice-based logics, this SLD-resolution is enriched with additional rules reflecting the properties of the extended semantic operator for BAPs, and is an alternative to the constrained resolution for the general annotated logic programs of Kifer and Subrahmanian, see [15] and to resolutions for logics which are interpreted by linearly ordered sets [12,23,31] and/or finite sets [15,23,24,25].

We show in a companion paper [19] that the semantic operator for the logic programs we have introduced here can be computed by learning artificial neural networks in the style of [13], but with learning functions embedded into connections between the layers. This shows that the automated proof procedure (SLD-resolution) we have introduced has its counterpart in the field of neural computation.

Further work to be done includes the elimination of annotations from the language in the same way as this was carried out in [16]. In particular, further investigation of the possible computational effects of the translation of BAPs into non-annotated sorted programs [16] may be interesting. Another field of possible extension of our results is to relate BAPs to probabilistic logic programs, as they were defined and studied, for example, in [1,5] and other papers. Finally, it would be interesting to show how we can extend BAPs to logic programs with interval-based annotations, and thereby establish linear programming for them. Such work

would relate to [21,26] and others and would make use of many results established by these authors.

References

[1] F. Bacchus. Lp, a logic for representing and reasoning with statistical knowledge. *Computational Intelligence*, 6:209–231, 1990.

[2] J. Calmet, J. J. Lu, M. Rodriguez, and J. Schü. Signed formula logic programming: Operational semantics and applications. In *Proceedings of the Ninth International Symposium on Foundations of Intelligent Systems*, volume 1079 of *Lecture Notes in Artificial Intelligence*, pages 202–211, Berlin, June 9-13 1996. Springer.

[3] K. Clark. Predicate logic as a computational formalism. Technical Report DOC 79/59, Department of Computing, Imperial College, 1979.

[4] C. V. Damásio and L. M. Pereira. Sorted monotonic logic programs and their embeddings. In *Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-04)*, pages 807–814, 2004.

[5] R. Fagin, J. Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1,2):78–128, 1990.

[6] M. Fitting. Bilattices in logic programming. In G. Epstein, editor, *The twentieth International Symposium on Multiple-Valued Logic*, pages 238–246. IEEE, 1990.

[7] M. Fitting. Bilattices and the semantics of logic programming. *Journal of logic programming*, 11:91–116, 1991.

[8] M. Fitting. Fixpoint semantics for logic programming — a survey. *Theoretical computer science*, 278(1-2):25–51, 2002.

[9] M. L. Ginsberg. Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[10] R. Hähnle. Towards an efficient tableaux proof procedure for multiple-valued logics. In *Workshop in Computer Science and Logic, Heildelberg*, volume 533 of *lecture Notes in Computer Science*, pages 248–260, 1990.

[11] R. Hähnle. *Automated Deduction in Multiple-Valued Logics*, volume 10 of *International Series of Monographs in Computer Science*. Oxford University Press, 1994.

[12] R. Hähnle and G. Escalado-Imaz. Deduction in many-valued logics: a survey. *Mathware and soft computing*, IV(2):69–97, 1997.

[13] S. Hölldobler, Y. Kalinke, and H. P. Storr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.

[14] M. Kifer and E. L. Lozinskii. RI: A logic for reasoning with inconsistency. In *Proceedings of the 4th IEEE Symposium on Logic in Computer Science (LICS)*, pages 253–262, Asilomar, 1989. IEEE Computer Press.

[15] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of logic programming*, 12:335–367, 1991.

[16] E. Komendantskaya. A many-sorted semantics for many-valued annotated logic programs. In *Proc. 4th Irish Conf. on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT)*, pages 225–229, Cork, Ireland, August 1– August 5 2006.

[17] E. Komendantskaya. *Learning and Deduction in Neural Networks and Logic*. PhD thesis, Department of Mathematics, University College Cork, Ireland, 2007.

[18] E. Komendantskaya. A sequent calculus for bilattice-based logic and its many-sorted representation. In N. Olivetti, editor, *Proc. Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX'07*, volume 4548 of *LNAI*, pages 165–182, Aix en Provance, 3– 6 July, 2007. Springer.

[19] E. Komendantskaya and A. K. Seda. Logic programs with uncertainty: neural computations and automated reasoning. In *Proc. CiE'06*, pages 170–182, Swansea, Wales, June 30– July 5 2006.

[20] E. Komendantskaya, A. K. Seda, and V. Komendantsky. On approximation of the semantic operators determined by bilattice-based logic programs. In *Proc. 7th Int. Workshop on First-Order Theorem Proving (FTP'05)*, pages 112–130, Koblenz, Germany, September 15–17 2005.

[21] L. V. S. Lakshmanan and F. Sadri. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Programming*, 1(1):5–42, January 2001.

[22] J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.

[23] J. J. Lu. Logic programming with signs and annotations. *Journal of Logic and Computation*, 6(6):755–778, 1996.

[24] J. J. Lu, N. V. Murray, and E. Rosenthal. A framework for automated reasoning in multiple-valued logics. *Journal of Automated Reasoning*, 21(1):39–67, 1998.

[25] J. J. Lu, N. V. Murray, and E. Rosenthal. Deduction and search strategies for regular multiple-valued logics. *Journal of Multiple-valued logic and soft computing*, 11:375–406, 2005.

[26] R. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and computation*, 101(2):150–201, 1992.

[27] M. I. Sessa. Approximate reasoning by similarity-based SLD-resolution. *Theoretical computer science*, 275:389–426, 2002.

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

[29] M. van Emden. Quantitative deduction and fixpoint theory. *Journal of Logic Programming*, 3:37–53, 1986.

[30] M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the Assoc. for Comp. Mach.*, 23:733–742, 1976.

[31] P. Vojtás and L. Paulík. Soundness and completeness of non-classical extended sld-resolution. In R. Dyckhoff, H. Herre, and P. Shroeder-Heister, editors, *Extensions of Logic Programming, 5th International Workshop ELP'96, Leipzig, Germany, March 28-30, 1996*, volume 1050 of *Lecture notes in Computer Science*, pages 289–301. Springer, 1996.